Culture-Aware Co-Speech Gestures Using Generative Adversarial Nerworks

Ariel Gjaci

DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering

University of Genova

Supervisor: Prof. Carmine Recchiuto

Co-Supervisor: Prof. Antonio Sgorbissa

In partial fulfillment of the requirements for the degree of

Master in Robotics Engineering

June 15, 2021

Acknowledgements

I need to thank many people for helping me to finish this project, starting from the supervisor Carmine Recchiuto who was always available for all the problems that I have encountered, I also thank the co-supervisor Antonio Sgorbissa which also helped me, all the colleagues and friends of the lab in which I worked for helping me to feel this work much less stressful, all the colleagues that worked with me during these two hard years and that gave me the opportunity to get to the end of this long path, my parents for supporting me and finally all the professors that taught me all the valuable and precious concepts.

To all the Master and PhD students of Robotics Engineering at the University of Genova.

Abstract

Co-speech gestures are commonly used by humans to interact with other people since they help them to emphasize the meaning of the words, to express feelings and even for showing intentions. Moreover, some studies demonstrated that they are not only dependent from the specific style of a person but they are also strongly connected with the culture, fact that sometimes can be easily noticed if we go out from our country and we interact with foreign people.
For all these reasons it can be very useful, in the Social Robotics field, to have humanoid robots that use custom culture-dependent gestures to enhance the expressiveness and the interaction with people.
In this project there will be shown a new method to achieve this goal by relying on Generative Adversarial Networks (GANs), which have the capability to learn the internal structure of data and generate new samples. GANs are subjectively regarded to produce better samples than all the other generative methods and here they will be used to map gestures to speech data.
In the proposed approach, the Dataset for training the Neural Network is built by taking data of different people belonging to the same culture. In particular, a custom Dataset was created using Youtube videos that show people belonging to the same culture (specifically, the Indian one) talking about their stories or their projects: in this kind of videos it is simple to detect both poses and voice of the main speakers that are necessary for the training. However, since the approach is based on features that depend from the frequency of speech audio, it is expected that different voices produce different features even if the people are saying the same words, making the Culture-Aware element very difficult to be learnt. For this reason it was used another Neural Network for many-to-one voice conversion. To show the results on a real social robot, a third Neural Network was used for mapping the poses from 2-Dimensional to 3-Dimensional space and the result was reproduced by the humanoid robot Pepper.

In conclusion, the Dataset has been used to generate co-speech gestures associated to a set of sentences pronounced by the robot, and the result of this approach has been compared to the embedded APIs for speech generation and with a procedure for generating random talking gestures. An online evaluation with human subjects, belonging and non belonging to the Indian culture, has been carried out, to assess if the proposed method may have some effective impact in the generation of culture-aware co-speech gesture, showing that the model successfully learnt how to map speech audio-features with gestures of the Indian culture. Also an objective evaluation was carried out, showing that the proposed GANs model trained with converted audio features generates more realistic gestures than the non-GANs approach, as well as the GANs method trained with non-converted audio features.

Contents

Chapter 1

Introduction

It is easy to notice that humans do not interact with others relying only on the speech capabilities, in fact, many times they unconsciously accompany the talk with some non-verbal movements that are called in many ways, e.g. Co-Speech or Non-verbal gestures. The importance of the non-verbal communication was confirmed by many studies (Krauss *et al.* (1996),Mcneill (1994),Alibali *et al.* (2000)) and some others confirmed their dependence from the culture (Archer (1997), Kita (2009)).
The gestures are commonly reproduced by movements of arms, hands, head but also by the remaining part of upper-body, and they are spontaneously created while people talk for many different reasons like helping the listeners to better understand what we want to say, reinforcing the meaning of the words, express feelings, show intentions and so on.
Mcneill (1994) introduced the most known classification of gestures: *metaphoric* for describing abstract content, *iconic* for illustrating physical actions or proprieties of an element (they both are related with the speech lexicon), *deictic* for pointing a specific object (so it is related to speech lexicon and spatial context in which the gesture is made), *beat* (rythmic) gestures which are in tune with the speech and do not carry any speech content, *adaptors* which are hand movements towards other parts of the body, and finally *emblems* or *symbolic* gestures which do not necessarily need to accompany the speech and which have conventional meaning that often depends from the culture.
Since, as said before, all these kind of gestures are necessary in our conversation, we may think that they are also needed in human-robot interaction. Especially in the social robotic field, we aim to have robots that interact with humans as naturally as possible, so we need agents that are not only good in the verbal communication but even in the non-verbal one.
Some studies confirmed that gestures performed by artificial agents help a listener to understand utterances and remember facts (Bremner *et al.* (2011)), but they

also improve the intimacy between the human and the robot (Wilson J.R. (2017)). For all these reasons and since actually there does not exist a suitable approach to reach this goal by taking into account also the culture, my work has focused on the implementation of a new method for learning and generating culture-aware co-speech gestures for a humanoid robot. In other words, the main aim of this thesis is the development of a novel strategy for mapping human speech (using audio features) and body poses, to autonomously generate cultural competent talking gestures for a humanoid robot. Among all the possible approaches, a detailed analysis of the relevant Literature has suggested the usage of Generative Adversarial Networks (GANs).
GANs are Deep Generative Networks which are able to learn the probability density of the training data and discover the internal structure (Goodfellow *et al.* (2014)). In the context of gesture generation, they can create in that way novel movements without changing their nature. GANs became very meaningful in the last years thanks to the Deepfake, term created in 2017 for indicating automated procedures for generating fake content that is harder and harder for human observers to detect thanks to the latest technological advances in artificial intelligence and machine learning (Kietzmann *et al.* (2020)). Nowadays they are widely used for many tasks, even for gestures generation, and they are subjectively regarded as producing better samples than other generative methods (Goodfellow (2017)).
This work has been mainly based on the previous work performed by Ginosar *et al.* (2019), which used GANS for mapping speech audio to gestures, and Sun *et al.* (2016).

The main assumption of this work was that since the method of Ginosar *et al.* (2019) demonstrated to successfully learn a mapping between speech audios and poses of an individual person relying on logarithmic-Mel spectrogram features, it can be also possible to learn a mapping of the common and non-common poses among a group of people of a certain culture if the speech audio features (*prosody*) are coming from a common voice for all of them. For this reason all the audios taken for the dataset were converted such that all the voices can look the same for all people: if they say a word in the same way, that word will produce the same audio features for all of them. This method has also some drawbacks since it also produces noise and it loses some of the source expressiveness while it maintains most of the speech articulation (which is fundamental in speech-poses mapping) thanks to the senones features which are a phonetic class that depends on the sounds produced in a speech and by their past and future contexts (for more details check 2).
In order to have a culture-related data for extracting speech and body poses, a new personalized dataset composed by 501 videos of Indian people taken from YouTube was created. Moreover, while the work of Ginosar *et al.* (2019) only

generated 2d animated gestures, this work was aimed at generating 3d speech gestures for a humanoid robot (specifically, the Pepper robot (Softbank (2018)). To put the poses in a 3-Dimensional space, a third neural network was trained (it is the same network used by (Yoon *et al.* (2018))) and a script for mapping the joint angles to the Pepper joint space was written.
Since these models require a lot of computations and resources, and since they were created using *Tensorflow1* (which exploits the old Cuda 9 Nvidia library) (Abadi *et al.* (2015)), all the code was written and adapted such that it can be run as much as possible by the couple *Google Colaboratory Notebooks - Google Drive Cloud* (Bisong (2019)): even if these have a large amount of limits, they can offer unlimited cloud space, powerful TPUs - Tesla GPUs for the computations, and all the libraries that are needed to run all the parts of this project, except for the script used for replicating movements and speech by Pepper: in fact, this requires a local network and Python 2.7 to work (not available on Google Colaboratory).
In order to assess the validity of the approach, an objective and subjective evaluation has been carried out. Concerning the first, metrics such as the mean jerk of the gestures, the L1 loss and the PCK (as shown in the work of Ginosar *et al.* (2019)) have been adopted. Concerning the second, some short videos employing a Pepper robot pronouncing a set of sentences using a) the proposed approach, b) Pepper's APIs for animated speech and c) a random movement have been created and a questionnaire based on Wolfert *et al.* (2021) has been designed. Finally, a number of subjects (Indian and non-Indian) has been recruited using the Amazon Mechanical Turk platform, and asked to watch three videos (with the robot talking with the three aforementioned approaches) and answer to the questionnaire.

The results suggest that the proposed method produces better gestures than the non-GANs one and the GANs one trained with original audio features; moreover, it was proved that the gestures generated with the Dataset composed by Indian people talking were more appreciated by people belonging to the Indian culture than non-Indian subjects.

1.1 Summary

This document is structured as follows:

- Chapter 2, i.e. the *State of the Art* chapter, shows what is the current state of the art on co-speech gesture generation as well as the projects and actual methods that have been exploited for this thesis.

- Chapter 3, i.e. the *Methodology* chapter, shows all the methods that were applied, the code that was written to reach the previously defined goals and all the computational issues that were faced.

- Chapter4, i.e. the *Evaluation* chapter, shows the objective and subjective evaluation methods that were carried out to test the presented Culture-Aware co-speech gesture generation approach.
- Finally Chapter 5, i.e. the *Conclusion* chapter, shows what are the conclusions that can be drawn from the evaluations that have been carried out, what are the limits of the proposed approach and how it can be improved.

Chapter 2

State of the Art

2.1 Summary

In this chapter there will be explained in details the state of the art that was analyzed and exploited to achieve the goal of generating Culture-aware co-speech gestures.
The Non-verbal gesture generation is a very difficult problem since machines must be able to understand the speech,the poses and the relationship among them. There are two main ways to address this challenge: the rule-based approach and the data-driven one.

2.2 Rule-based method

In the Rule-based method, the rules for mapping the speech to the gestures are defined by humans. It is used a lot in commercial robots (e.g. for the Pepper Robot (Softbank (2018))) since it is relatively a simple approach, it does not require data for learning, and it is possible to create very natural gestures if programmers define good rules based on human movements. But, as it can be imagined, it works only with a limited number of gestures and it requires a lot of effort if the wish is to have a robot that is able to reproduce a large amount of gestures. This is not always necessary since many of them (especially the commercial ones) are built to use only a limited number of sentences and movements, but if we want a robot that is able to communicate as humans, it is very laborious to create all the mapping rules. For this reason, and since the goal of this project is to generate an unlimited number of Cultural-aware gestures such that Pepper can be able to move as humans do, these kind of methods were not considered but they were instead exploited in the test phase to check what

people prefers (see the *Evaluation* chapter 4) .
An application example of this method can be found on the work Le *et al.* (2011): in this project it was addressed the problem of generating gestures for the NAO humanoid Robot (Softbank (2018))) while it tells story. To reach the goal, it was created a database called Lexicon which contains manually annotated gestures obtained from story telling videos and then these gestures were mapped to the robot considering also their expressiveness and timing with the speech. The number of gestures used in this approach as well as the conversational capabilities were expanded in the research of Meena *et al.* (2012), more specifically it was improved the turn-management and the presentation behaviours of NAO during conversational interactions.

2.3 Data-driven method

Data-driven approach is more suitable for the defined goals, and it can be of many kinds depending on what we want to reach.
Some of them are suited to work fine with a limited number of gestures and speech features (like the *Probabilistic* ones shown here) while some others (like the *End-to-end* ones) aim to learn the probability density of the training raw data without using intermediate representations, such as predefined unit of gestures, and discover the internal structure at the lowest level so that they can crate novel unlimited movements without changing their nature.
In the next sections there will be shown some examples of these two different approaches, giving particular attention to the ones that were exploited for this project.
Before describing in details the possible approaches for probabilistic and end-to-end gesture generation, some details about Generative Adversarial Networks (GANs) should be given to better understand their importance in this context.

2.3.1 GANs

GANs, which name stands for Generative Adversarial Networks, are generative Neuronal Networks (NN) discovered by (Goodfellow *et al.* (2014)) that, differently from any other Deep Generative model, are capable of learning how to generate samples that are similar to the given dataset instead of providing an approximation of its density function. They consists of two different Neural Networks: the Generator(G) whose role is to capture the data distribution, and the Discriminator (D) whose role is to estimate the probability that a sample came from G rather than training data. These two networks are trained to play an adversarial game where G tries to imitate as much as possible the training data to fool D, while D tries as much as possible to be not fooled by G, recognizing rather or not the generated data is real. This game, which can be also called *MinMax game*, can be formulated by the equation 2.1:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}))] + E_{z\sim p_z(\mathbf{z})}[log(1-D(G(\mathbf{z})))] \tag{2.1}$$

where x represents real data coming from dataset, z noise variables, $E(*)$ is the Expectation operator over distribution $*$, while p_{data} and p_z represent the distributions of x and z respectively. Since D must output the probability for a sample being real or not and since we want to train G for learning the distribution p_{data} of the data x (let's call it p_g) starting from a noisy distribution p_z of noisy variables z, we train D to maximize the probability of assigning the correct label to training examples by maximizing the term $E_{x\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}))]$ (i.e. maximizing the probability that a real sample is recognized as real) of 2.1, and by maximizing the term $E_{z\sim p_z(\mathbf{z})}[log(1-D(G(\mathbf{z})))]$ (i.e. maximizing the probability that a fake sample coming from G is recognized as fake). Then we simultaneously train G to minimize $E_{z\sim p_z(\mathbf{z})}[log(1-D(G(\mathbf{z})))]$ (i.e. we want to minimize the probability that the Discriminator recognizes data of G as fake). The *log* is used to avoid computational issues that can happen when 2.1 is computed.
As explained by Goodfellow *et al.* (2014), GANs work well if both G and D are Multilayer Perceptrons, if we train at each epoch D for more steps than G so that D is being maintained near its optimal solution as long as G changes slowly, and if we train G to maximize $logD(G(z))$ instead of minimizing $log(1-D(G(z)))$ so that in the first epochs G does not saturate because D rejects samples at the start with high confidence (G is poor and create samples that are totally different from ground truth).
Recently Gans were used for very different tasks and many versions were created for the specific case. One of the most important applications is the image translation (Isola *et al.* (2016)): in this project indeed it was created a system composed by an encoder/decoder for both G and D for translating images from one domain to another, relying on conditional Gans (cGans) which are a modified version of Gans where the D can also see the inputs x; this leads to the final formula 2.2.

$$L_{cGAN}(D,G) = E_{x,y}[log(D(\mathbf{x},\mathbf{y}))] + E_{x,z}[log(1-D(\mathbf{x},G(\mathbf{x},\mathbf{z})))] \tag{2.2}$$

Even the loss function was slightly changed, in fact it became 2.3: as it can be noticed, it was added a new component λL_{L1} that is defined in 2.4 and that encourages less blurring by minimizing the norm between ground truth and output generated by G.

$$O^* = arg \min_G \max_D \, , L_{cGAN}(D,G) + \lambda L_{L1} \tag{2.3}$$

$$\lambda L_{L1} = E_{x,y,z}[\|y - G(\mathbf{x},\mathbf{z})\|_1] \tag{2.4}$$

This technique has been useful for a lot of applications as the *Deepfake* (defined in the introduction), for video-to-video translations as in Chan *et al.* (2018), but also for some projects that were used for mapping speech text/speech audio to gestures that were exploited for the thesis.

2.3.2 Probabilistic approach

The probabilistic approach, as the name suggests, it is based on the creation and usage of a probabilistic model for the mapping between gestures and speech. For using some of these methods, as the ones shown here, it is necessary to have a predefined units of gestures and text (as in (Kipp (2003))) or audios (as in (Levine *et al.* (2010)); the model of the latter is shown in 2.1). As it is possible to notice from figure, the Gesture Controller model of Levine *et al.* (2010) consists of two layers: an inference layer, which analyzes vocal features and produces a distribution over a set of hidden states, and a control layer, which uses the inferred hidden state distribution and other available inputs to select the most appropriate gesture segments from a library of motion data exploiting a Markov Decision Model(MDP). The hidden-state representation of gesture motion is learned from some kinematic parameters, which constrain the learning process to only the stylistic content of the motion.
This method, even if it works well and online, it has some important limitations: firstly the training set for the inference layer must be extensive enough to contain a representative sample of significant prosody cues and prosody-gesture associations, then the gestures and audios features are ”quantized” so only a limited number of them can be generated, and finally everytime we want to generate gestures of a given library we have to train the model from the start. For all these reasons, the model is not suited for the goal of this project.
The model of Kipp (2003) was instead one of the first data-driven procedure where gestures and text were annotated from a long video and then mapped using a probabilistic model. This one, as many others which came later, has the limitation of the finite number of gestures that can be mapped to text (1056), and it requires an important human effort for the poses transcriptions as well as for text annotation (the last is no more a problem today since there are a lot of speech-to-text services available). Some other methods that used predefined units of gestures are the ones of Rodriguez *et al.* (2019) and Huang & Mutlu (2014) so the result, as it can be again expected, is restricted to a sequence of predefined unit gestures. Rodriguez *et al.* (2019), in particular, demonstrated the goodness

of the gesture generation task using GANs Neural Networks overcoming the state of the art methods. Even if in that project the goal was not the gesture-speech mapping, this can be a clue for using GANs in similar goals as the ones defined in this thesis.

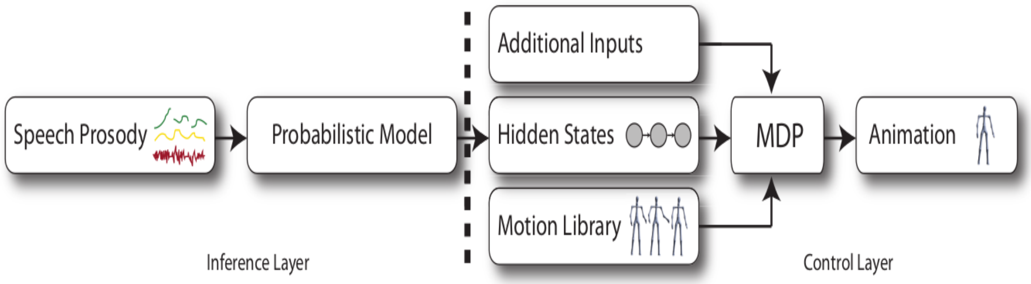


Figure 2.1: Gesture controllers consist of two layers: the inference layer, which infers a distribution over a set of intermediate hidden states, and the control layer, which selects appropriate gesture segments from a motion library.

2.3.3 End-to-end approach

The second kind of approach, i.e. the end-to-end approach, relies more on deep-learning models trained on raw data to generate gestures. Two of the latest end-to-end approaches are different from the others since they release the limits of gesture units and they map the movements with the speech audio (Ginosar *et al.* (2019)) (let's call that project *AudioToGesture*) and with the speech text (Yoon *et al.* (2018)) (let's call that project *WordToGesture*).
They both use a personalized dataset taken by exploiting Youtube videos, but, while the first one uses a specific-person dataset, the second one uses a more general dataset based on TED Talks videos: TED is a conference where people from all over the world talk about their stories and their ideas. A dataset based on TED talks has some advantages that have to be taken into account, also for reaching the main goals of the thesis:

- It can provide a very large dataset and the amount of Youtube videos are growing up more and more everyday.
- There can be thousands of unique speakers from all over the world and many of them speak the same language, i.e. English.

- The view of the speakers in the videos is often good and it is easy to detect their body using computer vision algorithms.
- For a large amount of time it is possible have a clear audio of the speech as well as the transcription (the last one is not available for all the videos).

So, by viewing the differences between the datasets used, it is possible to image also what can be some important differences between results: indeed, with one method we learn a mapping which is style-dependent while with the second one we learn a more general mapping that is style-indipendent.
What was done in the current project is basically the fusion of these two methods: it was exploited the approach of *AudioToGesture* but with a dataset composed by English Youtube TED Talks of people of the same culture. The original project of *AudioToGesture* is based on the log-mel-spectrogram features of a speech audio: thus even different speakers that pronounce the same words with the same emphasis may produce different features.

The log-mel spectrogram basically is a logaritmic spectrogram adapted to the Mel-scale: this scale was creted by Stanley Smith Stevens, John Volkman and Edwin Newman during the year 1937 and it is a non-linear scale that represents better the frequencies that we use more while we speak [1]. Having said these, it can be imagined that where there is a pitch shifting (for example when two different voices are used), there is also a very different log-mel spectrogram even if two or more people pronounce the same sentences; this of course can't happen if the features are extracted from written words as *WordToGesture*. But also *WordToGesture* has some important drawbacks: with written words we loose the voice articulation as well as expressiveness which are very important for the co-speech gestures generation, and the synchronization with the gestures (also words can be synchronized but not as much as a speech audio sampled at 44100 or 16000 samples per second).
Some previous researches in linguistics and psychology (de Meijer (1989), Feyereisen (1991)) suggests that gestural kinematics and rhythm are associated with prosody while gestural form is more strongly tied to the semantic meaning, which can be learned by text (Mcneill (1994)). So, prosody and motion are known to correspond to emphasis and were individually found to correlate with emotional state. Moreover, these studies also established strong links between prosody and the timing of gestures. Some aspects of motion that correlate with emotion, such as directness and velocity (de Meijer (1989)), are better described as dynamics or kinematics than form. Finally, beat gestures are "formless" (Mcneill (1994)) and are differentiated only by their kinematic style, so it is necessary to use prosody to learn them. For all these reasons, it was preferred the usage of audio features

[1] For more info press here

instead of text ones, even if without the latter the shape of gestures cannot be completely learnt.
The exploitation of a voice conversion method made possible to use the *AudioToGesture* project even with different voices: in particular, by using a many-to-one Voice Conversion method as the one of (Sun *et al.* (2016)), there can be obtained speech audios as they came from the same person. This approach for audio conversion is ideal since it does not require parallel data (i.e. source and target syncronized audios) for training, data that cannot be available in the TED talk videos. This method produces a little bit of noise during the conversion phase, but it is able to generate audios that are syncronized with the source, that preserve the articulation and also some of the expressiveness that we had before, and that have the same tonality (i.e same pitch since we have always the same voice as output) for all the audios. These were the most important reasons which pushed the authors of *WordToGesture* to rely on written words instead of audios features. Researchers that worked on this project used a synthesized speech audio only for the tests made at the end with the aim of obtaining some comparisons, relying on the Text To Speech (TTS) Google API which is able to generate a very good audio from text, but, as it is possible to imagine, the audio cannot be perfectly syncronized with the poses.
In the next sections there will be explained more in details *AudioToGesture*, *WordToGesture*, some other different end-to-end methods and the state of the art for voice conversions.

2.3.3.1 Audio to Gesture

This project, that was originally called *speech2gesture*, it was the main one exploited for this thesis. The code is available online on a Github repository while some of the data used for testing the method is available on Cloud services, but, unfortunately the code for creating the given datasets was not available. The Gans used for this method were very similar to the ones used by Isola *et al.* (2016). A main difference here consisted on the type of data used as input on G and D, indeed the loss used is described by 2.5 were $\mathbf{m}$ is a vector which represent the difference of 2D poses p between frames taken in some temporal extent T of a video (in the project T=64 frames taken in videos running at 15 frames per second), so $\mathbf{m} = [p_2 - p_1, ..., p_T - p_T - 1]$, while $\mathbf{s}$ is the input speech audio taken with $f = 16000 samples/s$ in the same video with the same temporal extent. Finally, here the translation architecture learn a mapping from audio to poses (instead of image-to-image) using an UNet (Ronneberger *et al.* (2015)).

$$L_{GAN}(D,G) = E_{\mathbf{m}}[log(D(\mathbf{m})] + E_{\mathbf{s}}[log(1 - D(G(\mathbf{s})))] \tag{2.5}$$

The UNet is basically a Convolutional Autoencoder which downsamples the

input through a series of convolutions, and then it upsamples the features (again with convolutions) adding also the ones that were taken from the same level of input, giving as result a network which is shown in 2.3. This architecture proved to give better results than a common Convolutional Autoencoder, it requires less data to be trained and provides better outputs since these are reconstructed using also the features of the ground truth.
So, we have the UNet final architecture shown in 2.2, and an objective to be minimized shown in 2.6, where L_1, as before but without the input x on G because we don't use cGan, is defined as 2.7.

$$O^* = arg\,\underset{G}{min}\,\underset{D}{max}\ , L_{GAN}(D,G) + \lambda L_{L1}(G) \tag{2.6}$$

$$\lambda L_{L1}(G) = E_{\mathbf{p},\mathbf{s}}[\|\mathbf{p} - G(\mathbf{s})\|_1] \tag{2.7}$$

Notice that the input audios were represented by a log-Mel spectrogram (Medium.com (2020)), which more suited for representing human voice than a common spectrogram, and for training the Neural Network, while poses are represented as a set of x, y coordinates extracted with *model_23* of OpenPose (Cao *et al.* (2019a)), for more information see 3.3.2.

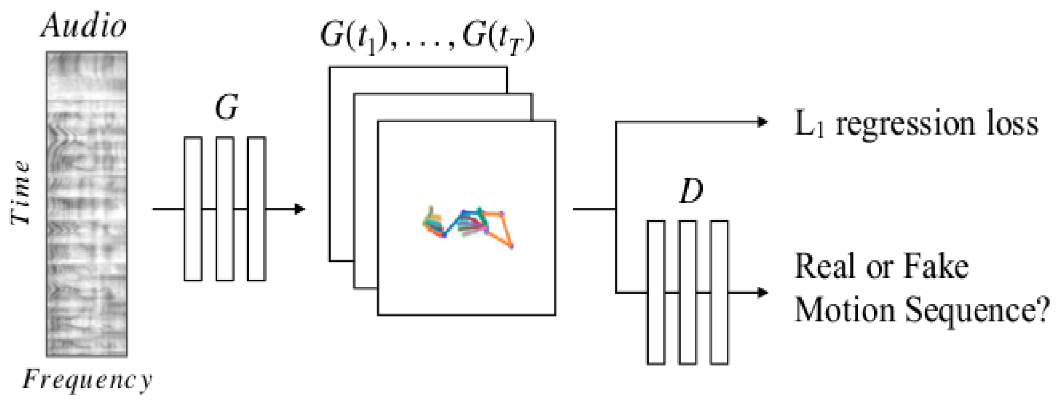


Figure 2.2: Audio To Gesture architecture. A convolutional audio encoder downsamples the 2D spectrogram and transforms it to a 1D signal. The translation model, G, then predicts a corresponding temporal stack of 2D poses. L1 regression to the ground truth poses provides a training signal, while an adversarial discriminator, D, ensures that the predicted motion is both temporally coherent and in the style of the speaker.

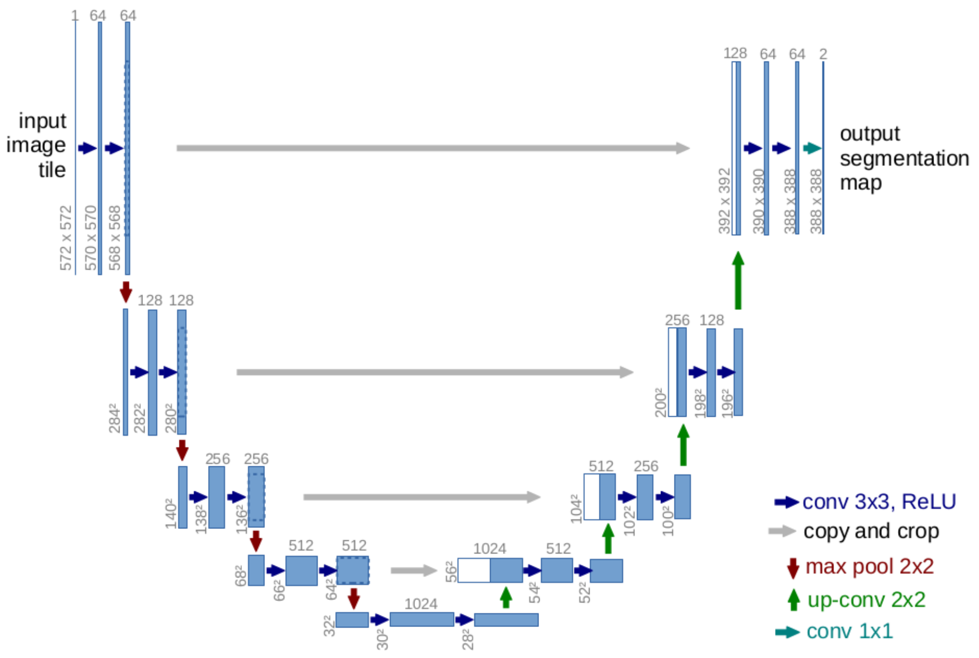


Figure 2.3: UNet architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

This technique gave very good results in learning individual styles of gestures, but required a lot of data (more or less 100 of hours of video for each person) to provide them. Since all the videos for a given person were taken in more or less the same conditions, the normalization of poses it was computed by subtracting the neck keypoint to all the other keypoints, by then subtracting for each keypoint the average coordinate of that in all the videos for the given person, and finally by dividing by the standard deviation of all the coordinates of the same keypoint. So, for each keypoint, after subtracting x, y coordinates of the neck, we have $((\frac{x-\mu_x}{\sigma_x}), (\frac{y-\mu_y}{\sigma_y}))$. This method can't work if the NN is trained by using data of different people or if the data is taken in very different condition (e.g. if the view of the speaker changes a lot in the videos or if in some videos a person is still while in some others is seated ecc..), so it is very important to choose correctly all the data used: this can be a very important limit because it is not simple to get hundred of hours of good videos for a unique speaker. The keypoints extracted from poses were 49: 40 for the hands (5 for each finger), 8 for arms (central

point of the hand, wrist, elbow, shoulder) and one for the neck, and for the tests there were used both objective and subjective evaluations. Before the start of the training, it is necessary to create the dataset as it is explained on the repository and download some files that were already provided.
The proposed method for loading the data into the network is not suited for *Google Colaboratory* since it requires the load of million of single *.txt* files, but all the details about this and how the technique was adapted for the thesis there will be explained better in the next chapters.
For the objective evaluation it was checked the value of the *Loss* defined before compared with some other different methods, and the PCK (Percent of Correct Keypoints) index (Yang & Ramanan (2013)) which is a widely accepted method for pose detection and which classifies a keypoint as correct if it falls inside a range $\alpha\ max(h, w)$ (where h is the height and w is the width of a predefined bounding box) of the same ground truth keypoint: in the specific case $\alpha = 0.2$ while the bounding box is represented by an ellipse formed by an axis parallel to x and another parallel to y of the images; the first represents the maximum variation of width of that keypoint in the ground truth video sample (which is taken at 15 fps and formed by 64 frames), while the second axis represents the maximum variation in height of the same keypoint in the same ground truth sample. PCK is not the best method in this case since it is very sensitive to big variations of keypoints, fact that can often happen since GANs are able to generate very different movements with respect to ground truth (also in humans it is common to use different gestures to represent the same words). Also the *Loss* defined before is not a perfect evaluation method: the $L1$ regression part can be very fine in the case that predicted movements are unnatural but have keypoints that fall near the ground truth ones. Anyway, those indexes were proven to be higher by using the proposed method than with other approaches for gesture generation. For the subjective evaluation, there were shown two different versions of some videos, one that is the ground truth and the other which is a synthesized one using the ground truth face and the predicted motions, to the Amazon Workers Turk. The workers were asked to recognize the one that is fake and the one that is truth. Overall, this method provided good results, so this is was used as inspiration to the evaluation conducted for the thesis.

2.3.3.2 Word to Gesture

Also this project is available online and it can be found as Co-Speech_Gesture_Generation. For some aspects, like the fact that encoders and decoders were used and the fact that it is based on raw-data, it is similar to *speech2gesture*, but for many others they have some clear differences. At first, the data used for training the network was completely different: as it was explained before, here the dataset is composed

by videos of different people talking in TED talks, so also the normalization of the poses must be different. In fact, since here all the videos were taken using different view-angles, different cameras, different resolution and frame rates, and since the main speakers were all different, it didn't make sense to compute an average and a standard deviation of the keypoints. The normalization was instead computed by subtracting to all the keypoints of a given frame the x, y coordinates of the neck keypoint, and then by computing the distance between left and right shoulders: each keypoint (x, y coordinates) is divided by this factor if the shoulder length seams to have reasonable value with respect to the height of the neck, otherwise the keypoints is divided by a factor that can reasonably represents what can be a shoulder length with respect to the neck height detected.
The keypoints were extracted using a more recent model of OpenPose, *body_25*, which gives not only the coordinates of the keypoints as model_23, but also the confidence value with which they were taken. From the keypoints extracted, only 8 were used: the neck, the head, and then the wrist, the elbow and the shoulder for both arms.
Before going inside the Neural Network, the poses needed to be hardly processed in order to take only good data for the training, validation and test sets: there were taken only the keypoints of the main speaker (the confidence values were exploited to find it), only parts of videos (clips) which have visible keypoints for at least 60% of the time, only frontal view was used, in case of jittering and missing joints (may happen for problems of detection) the keypoints were filled with reasonable values computed by regression, the motion were smoothed using filters and the clips were the main speaker is still were deleted. This time also the code for creating the Dataset was fortunately available on Github and it revealed to be a good starting point for creating the dataset of this thesis
The words were taken from the transcripts that come with the videos: in fact there were taken only the ones which have English subtitles.
This time, differently from *speech2gesture*, gestures were represented as 10-dimensional vectors where each dimension represent a component extracted thanks to the Principal Component Analysis (PCA) (Sehgal *et al.* (2014)) technique: with these 10 components the 94.8% of the variance in the training dataset is represented.
Also words were represented as multi-dimensional vectors, in the specific case they were represented as one-hot vectors that describe the word index in a dictionary, and then they were compacted to 300-dimensional vectors using a model for word embedding, which made possible to represent in a similar way words that have similar meanings.
This kind of representations made the learning process less "end-to-end" with respect to *speech2gesture*, in fact only a limited number of gestures and words of the original dataset can be used for the training purpose. A bidirectional Recurrent Neural Network (RNN) was used in this case for the model for capturing the

speech context, while for the decoder it was used an RNN with pre- and post-linear layers. So, practically the model is composed by an encoder-decoder system where each of them has a two-layered Gated Recurrent Units (GRUs) (Cho *et al.* (2014)) with 200 units, and it is represented in 2.4.

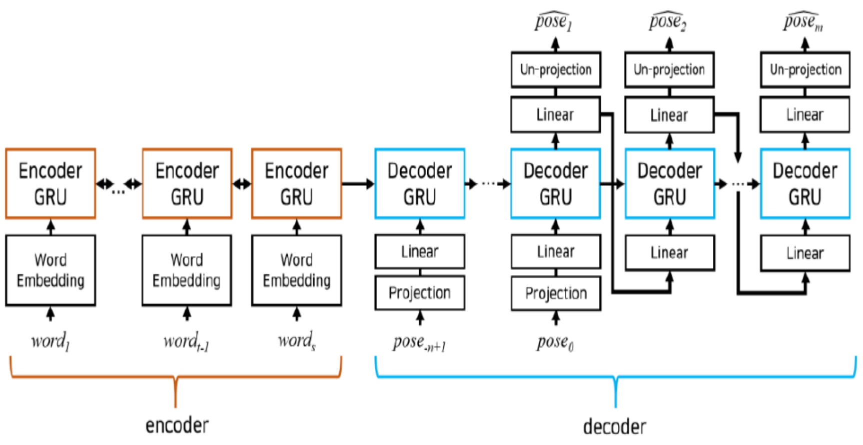


Figure 2.4: Word to Gesture architecture. The encoder GRU interprets s speech words, and the decoder GRU generates m human poses of gestures. The decoder GRU inputs n previous poses to make the series of poses continuous. The soft attention mechanism is used but not depicted here coherent and in the style of the speaker.

Only a user study was conducted to test the method: relying again on the evaluation of Amazon Turk Workers, the results related to this approach were compared with the ones obtained with other gestures generation approaches, such as Ground Truth gestures, Nearest Neighbour, Random gestures and Manually Crafted gestures.
All the movements were reproduced by the Nao robot (Softbank (2018)) by mapping keypoints from 2D to 3D and some videos representing all the methods were taken relying on Google Text-to-Speech for the speech of the robot, so that everything has been shown to the Workers.
These ones have to complete a survey at the end, and three different indices were measured: anthropomorphism, likeability and speech-gesture correlation. The Ground Truth was the one which gave the best values of the three indices, the Random was the worst while no statistically significant differences were found with the proposed method, Nearest Neighbour and Manual Crafted gestures. In the open questions, the workers said that sometimes the movements seemed jerky and fast, while some other times the robot were not moving in a predictable hu-

man fashion even if it seemed not stiff and moving freely as humans do when they talk.
It is important to notice that, for mapping the poses from 2D to 3D, a new Neural Network was created since the other existing studies for 3D pose estimation found to be not successful in the TED dataset, owing to environment mismatches. Considering that all the poses extracted for the dataset were facing near front, the problem was easier than the classical pose estimation problem. The network consists of a cascade of three fully connected layers with 30, 20, and 7 nodes with batch normalization, while the data used to train the network was coming from CMU Panoptic Dataset (Joo *et al.* (2015a)) which provides highly accurate 3D poses of social activities including many co-speech gestures. The code of the project is publicly available on a Github repository.

2.3.3.3 Speech Gesture Generation using Trimodal Context

This project is the earliest end-to-end method created and it is an evolution of *WordToGesture*: here there used not only the written words as a speech context for generating gestures, but there were exploited also the audios and the Speaker-IDs of the people(Yoon *et al.* (2020)). Unfortunately the paper of this project was available only after few months the actual project was started, so it was not exploited for the purpose neither explored in details even if the code is currently available online on a Github repository.
The overall network consists of three encoders (one for each speech modality) and a decoder for gesture generation; the architecture is shown in 2.5. As it can be noticed in the image, the Generator create poses frame-by-frame from an input sequence of features containing the speech context. Differently from *WordToGesture* and *AudioToGesture*, here the problem of the synchronization was specifically taken into account by synchronizing the different modalities such that they share the same time steps, and by configuring the Generator to use parts of speech text/speech audio of the current time step and the near ones instead of using the whole speech context.
For the speech text, it is assumed known the exact utterance time of words, so there were inserted padding tokens ($\diamond$) to make a padded word sequence to be with the same length as gestures. All words were then transformed as *WordTo-Gesture* in 300 dimensions using a word embedding technique, and finally encoded by a Temporal Convolutional Network (TCN) (Bai *et al.* (2018)) to make a 32-D feature vector.
For the speech audio, a raw audio waveform goes inside a 1D CNN to generate the 32-D feature vector.
Finally, the speaker IDs were represented as one-hot vectors where only the element of a chosen speaker is not zero. A CNN then maps the speaker IDs to a

style embedding space with 8 dimensions.
For the Generator, a bidirectional Gated Recurrent Unit (GRU) network was used. The encoded features of speech text, audio, and speaker ID are concatenated to form a big feature vector for each time instant, so the Generator takes the feature vector f_i as input and generates the next pose d_{i+1} iteratively.
In this project, gestures are represented as a sequence of poses composed by 10 joints and described by directional vectors that represents the relative position of a child joint with respect to a parent one (i.e. we have 9 directional vectors for each pose).
To push the gestures to be more realistic, an Adversarial Discriminator was used, obtaining finally a complete GAN model.
The architecture was tested objectively by using a new index called FGD, and subjectively by a human study. FGD is an index which applies the concept of FID metric (Heusel *et al.* (2017)) to the gesture generation problem: instead of computing the Frechet distance between the distributions of the features of real and generated images, it does the same with features of gestures.
The proposed method was compared with *WordToGesture* and *AudioToGesture*, obtaining better results in both the evaluations. Indeed, even if *AudioToGesture* showed better motion than *WordToGesture* because of the usage of an Adversarial Discriminator, this motions seemed repetitive since they were learned by a single modality. In particular, *AudioToGesture* showed a lot of beat gestures but not many others of different kind. *WordToGesture* instead showed to produce different gestures for different words, but the motion seemed to be sometimes slow and with discontinuities between ground truth and generated poses. Even if the results were better with FGD metric showing also large and dynamic gestures and even if the results were better subjectively, with the last kind of evaluation method it was not possible to find significant differences with *AudioToGesture* concerning the human-likeness of motion and speech–gesture match questions.

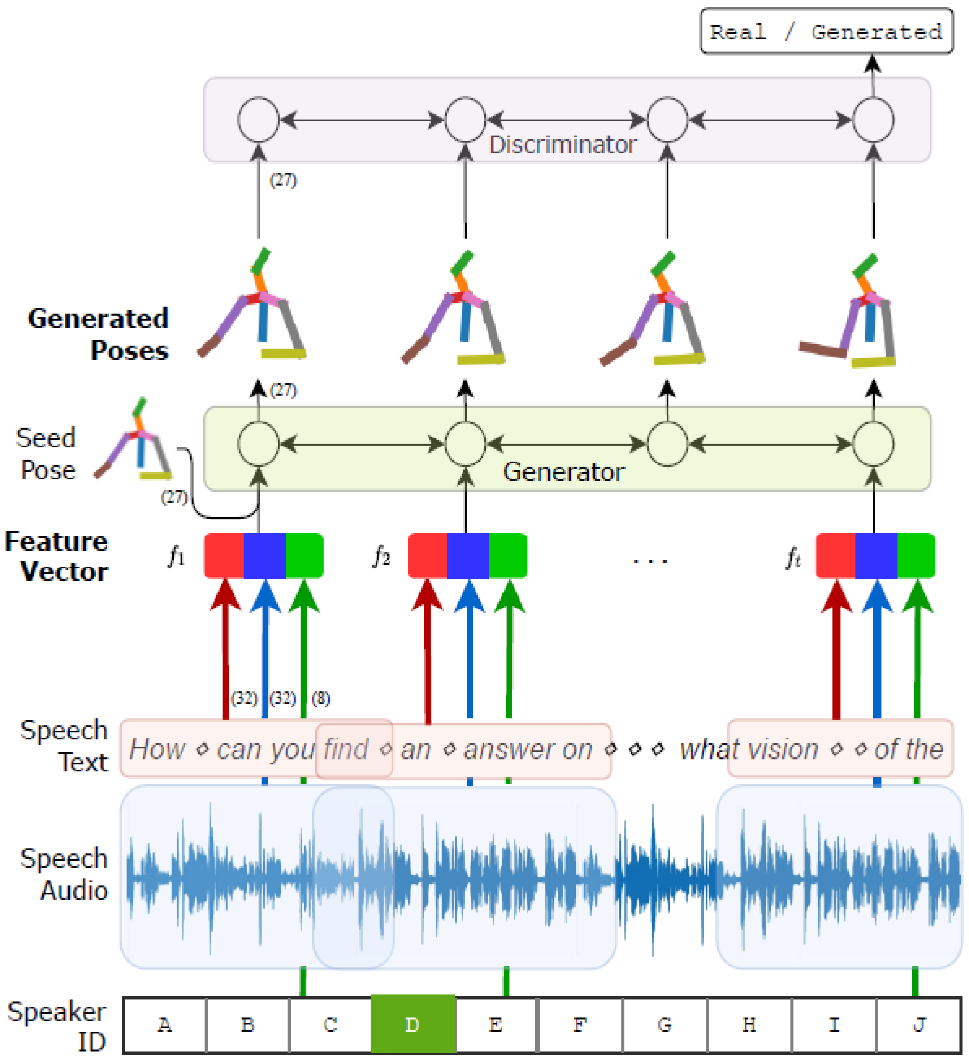


Figure 2.5: The architecture of the proposed Trimodal gesture generation model. The generator generates a sequence of human poses from a sequence of context feature vectors that contain the encoded features of speech text, speech audio, and speaker identity (ID). The features of text, audio, and speaker ID are depicted as red, blue, and green arrows, respectively. The seed poses are also used to ensure continuity between consecutive syntheses. The discriminator is a binary classifier that distinguishes between real human gestures and generated gestures. The number in parentheses indicates the data dimension. The poses are in 27 dimensions since there are nine directional vectors in 3D coordinates

2.4 Voice Conversion

On the basis of the proposed analysis, and for all the reasons given before, I've decided to use the *AudioToGesture* approach as a starting point for my Thesis work. However, since in my case the dataset was composed by different persons with different voices, I have decided to adopt a Voice Conversion technique. In fact, it was thought that if *AudioToGesture* works well when associating features coming from spectral evenlops of the voice of a person with his non verbal movements, this technique can work fine even with different people if they use exactly the same voice. The results of the proposed approach, both without and with VC, will be shown and discussed in chapters 4 and 5.
There are many different kinds of VC depending on what we want to achieve, indeed we may want a one-to-one correspondence between voices, many-to-one, many-to-many or many-to-one. In this thesis it was exploited a technique based on Deep Learning for many-to-one VC which uses a Neural Netowork based on Phonetic PosteriorGrams (PPGs) to bridge between speakers (Sun *et al.* (2016)). A PPG is a time-versus-class matrix representing the posterior probabilities of each phonetic class for each specific time frame of one utterance (Hazen *et al.* (2009)). A phonetic class may refer to senones, words and phones: in this paper senones were used (an example can be found in 2.6). Senones don't depend only from the sounds produced in a speech but also by the context: for example "a" with left phone "b" and right phone "d" in the word "bad" sounds a bit different than the same phone "a" with left phone "b" and right phone "n" in word "ban". The context depends itself by the past and future contexts (left and right contexts) of a word. It is important to notice that senones are speaker indipendent because they rely only on the language used for a speech and by its typical "sounds" used for speaking the words. This is why there is a believe that PPGs obtained by the Speaker Indipendent-Automatic Speech Recognition (SI-ASR) system can be a good element for mapping the voices; the objective and subjective results shown that this fact revealed to be true.
The model is shown in 2.6 and, as it can be noticed, it is composed by many parts.
Firstly, the SI-ASR system (that is a DNN created using Povey *et al.* (2011) and Garofolo *et al.* (1992)) learns a PPGs representation of the input speech from the MFCCs (Mel-frequency cepstral coefficients) features, where MFCC is the short-term power spectrum of a sound based on a linear cosine transform of a log power spectrum on a nonlinear mel-scale of frequency, coming from a multi-speaker ASR corpus (TIMIT dataset, Garofolo *et al.* (1992)).
Then, exploiting a dataset containing the audios of the desired target speaker (in this case the target comes from the ARCTIC dataset, Kominek & Black (2004)) by extracting the PPGs features of him/her with the SI-ASR model, and

the MCEPs features (MCEPs are the Mel Log Spectral Approximation (MLSA) parameters which approximate Mel- Frequency Cepstral Coefficients (MFCCs)), these two are mapped together using a DBLSTM model (as Sun *et al.* (2015) but this time trained without parallel data), that is shown in 2.7 and which gives as output the new Converted MCEPs.
Finally the trained DBLSTM is used to extract MCEPs features from the PPGs of the source speaker (extracted by the trained SI-ASR); by taking these features as long as the Log F0 (foundamental frequency or pitch of a voice) translated to the pith of the target, and AP (Aperiodic component, a component generated by the modulation of the air flow and which is responsible of the generation of fricative or plosive sounds, but it also present in the voiced sounds as well), the new syntetized speech is created using a vocoder. In the implementation found on Github, the training sounds as well as the ones produced by an inference, lasts exactly 4 seconds; this limit was overcome in the thesis as it will be shown in the next chapter.

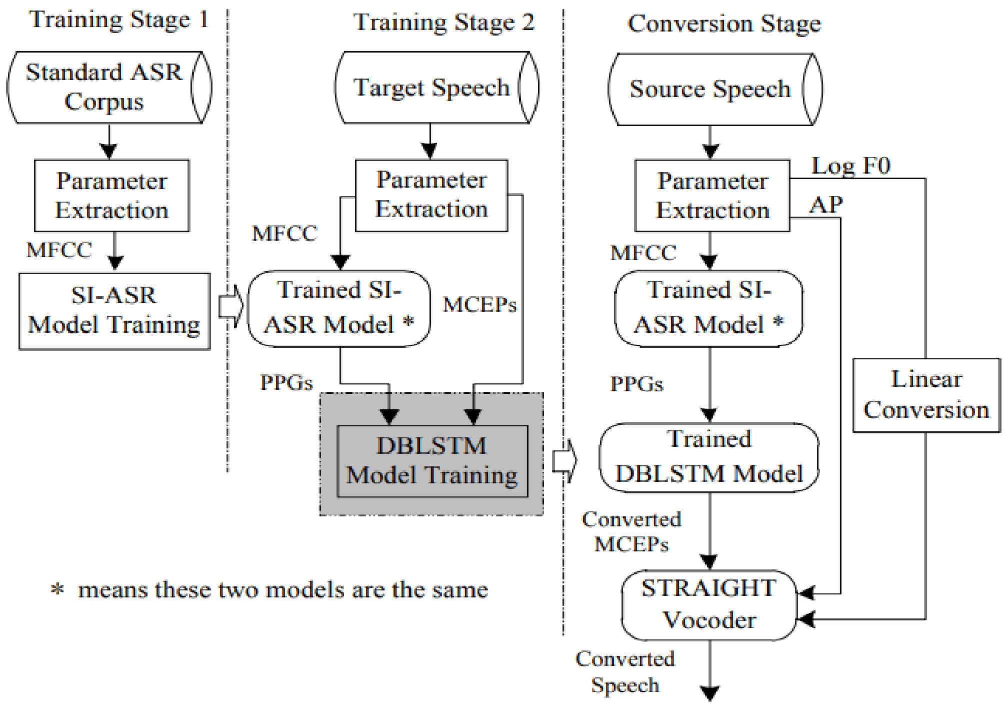


Figure 2.6: Schematic diagram of VC with PPGs. SI stands for speaker-independent. Target speech and source speech do not have any overlapped portion.

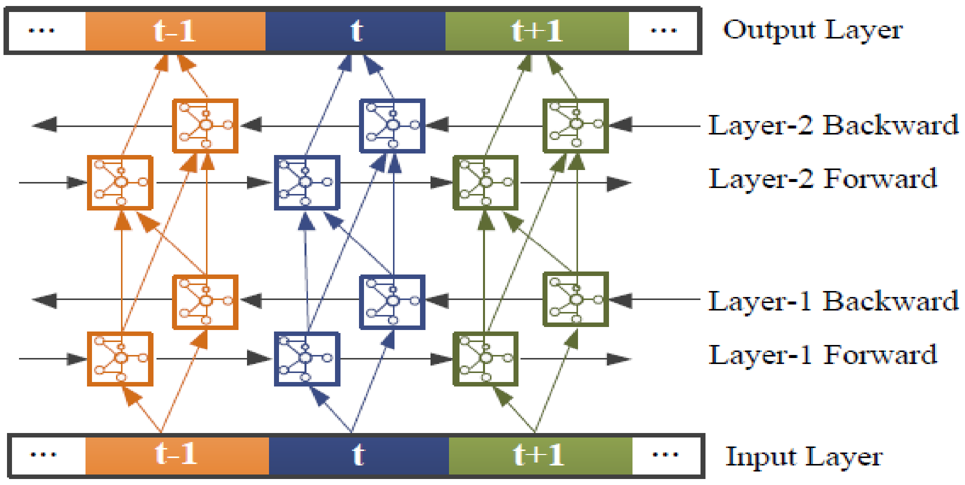


Figure 2.7: Architecture of DBLSTM. The DBLSTM network architecture including memory blocks and recurrent connections makes it possible to store information over a longer period of time and to learn the optimal amount of context information

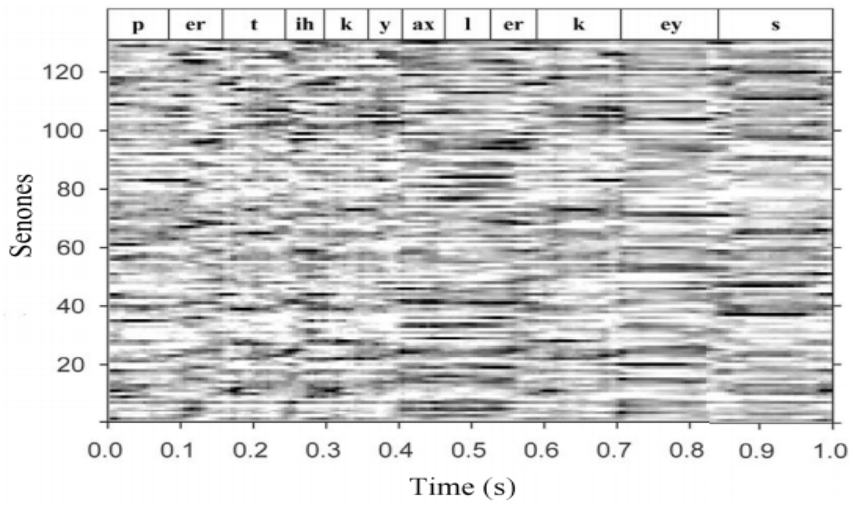


Figure 2.8: PPG representation of the spoken phrase “particular case”. The horizontal axis represents time in seconds and the vertical one contain indices of phonetic classes. The number of senones is 131. Darker shade implies a higher posterior probability

Chapter 3

Methodology

3.1 Summary

In this chapter, there will be shown everything about the methodology applied for this thesis, the code implemented and the computational problems addressed for reaching the defined goals.
Before the start, it is essential to say that the whole project has been tested on Ubuntu 18, which is the Operating System used by *Google Colaboratory*, Python 3.6, and then Python 2.7, the latter only used for the script for mapping the gestures to the Pepper Robot. The project *speech2gesture* was the only one also tested with Ubuntu 20 and CUDA 11, so for the others there is no proof that they can also work with other versions of CUDA, Python and Ubuntu. There is a belief that it is at least necessary to maintain Python 3.6 since some packages are not compatible with other Python versions.
All the Python libraries that are needed to run the project are written inside a *requirements.txt* file, which can be passed as an argument to the command *pip* for installing them; the list of the required packages is shown in the section 3.11.
The whole project can be briefly described by the following steps which are also shown in the schematic diagram 3.1:

1. Creation of a personalized Youtube Playlist for the TED videos which will be used as Dataset.
2. Download of videos and extraction of the Dataset with poses
    - Download of all the videos
    - Poses extraction using Openpose
    - Scene extraction using PySceneDetect
    - Scene filtering and poses selection

- Creation of a Dataset with training,validation and test sets
- Adaptation of the Dataset for *speech2gesture*
  - Downscaling of the fps of clips to 15
  - Mapping of the poses to *speech2gesture* configuration
  - Preprocessing of the motions
  - Creation of a csv file for describing data

3. Training of the model for many-to-one voice conversion
4. Extraction and conversion of the audio files
5. Training of the *speech2gesture* model
6. Inference of *speech2gesture* for generating poses from audio files
7. 2D to 3D pose mapping
8. Mapping of poses to Pepper Joints configuration and computation of the angles between joints

Let's now see in detail every single part shown above and how it was implemented.

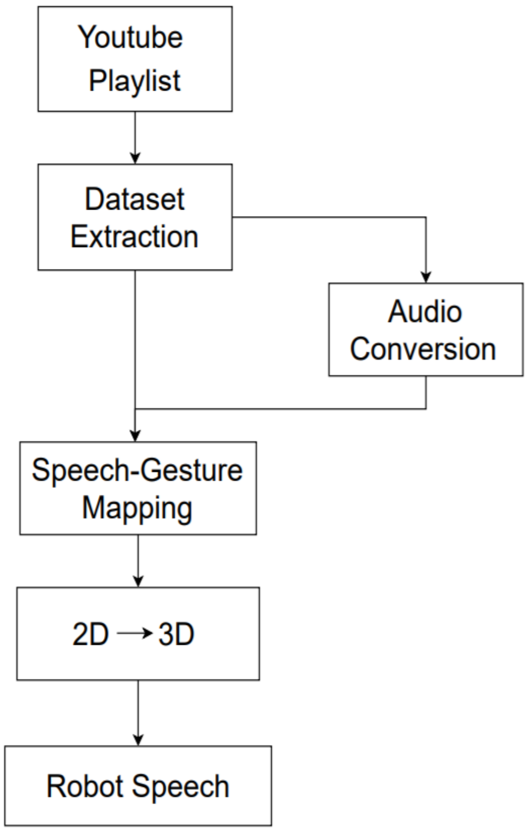


Figure 3.1: Schematic diagram of the thesis workflow. A Personalized Youtube Playlist is created, then the videos are downloaded so that the poses and audios can are extracted. After the audio conversion, the model is trained so that it can be inferred by passing an audio thanks to which it can generate poses. The motion is then mapped in 3D and finally reproduced by the Robot as well as the spoken words.

3.2 Youtube Playlist

The Youtube Playlist contains all the data that has to be extracted and then used to train the *speech2gesture* model. Since in this thesis the goal is to learn co-speech gestures from a culture, it is important to use a playlist containing videos of people belonging to the same culture. Even if it can be found on Youtube a Channel called *Josh Talks* (a frame of a Josh Talks video is shown in the figure 3.2) which contains videos of only people coming from India talking about their stories exactly as it happens inside the *Ted Talks* Channel (except for the fact that here the speakers come also from other countries), it cannot be

completely exploited since the people speak using different languages. For this reason, it was necessary to create a personalized Youtube Playlist.
But, considering that *Josh Talks* also contains a lot of useful videos, some of them were exploited to create a specific Playlist for the Indian culture.
To reach the goal, there were mainly exploited the two above mentioned Youtube Channels and rarely some unofficial channels containing videos of the same kind, and there were taken only the videos with people talking using only the English language. By seeing the videos, it was noticed that in some of them the main speaker used different languages during the speech, so the task of choosing appropriate data became more challenging. To overcome this problem, all the videos were checked for a few seconds in some random points and, if there was found that in all the points the main speaker was always speaking English, the videos were considered valid for the Playlist. Of course there was not a proof for that, but there is a belief that on average the data is as expected.
Also the impossibility of knowing in advance the culture of the main speakers is a problem for creating the Playlist, so there were considered all the speakers inside the *Josh Talks* as belonging to the Indian culture while for the *TED Talks* and the other minor channels, there were checked the information used as description of the videos and eventually, if nothing helpful were found, the origin and where the person lived for most of the time.
By using this approach, 501 videos with a variable duration from few minutes to an hour were chosen, and from these, only 439 were taken and used for the creation of the dataset. Indeed, sometimes problems happened during the download phase and, as will be explained in the next section, there were taken only the ones with at least HD 1280x720 resolution to better detect the poses.

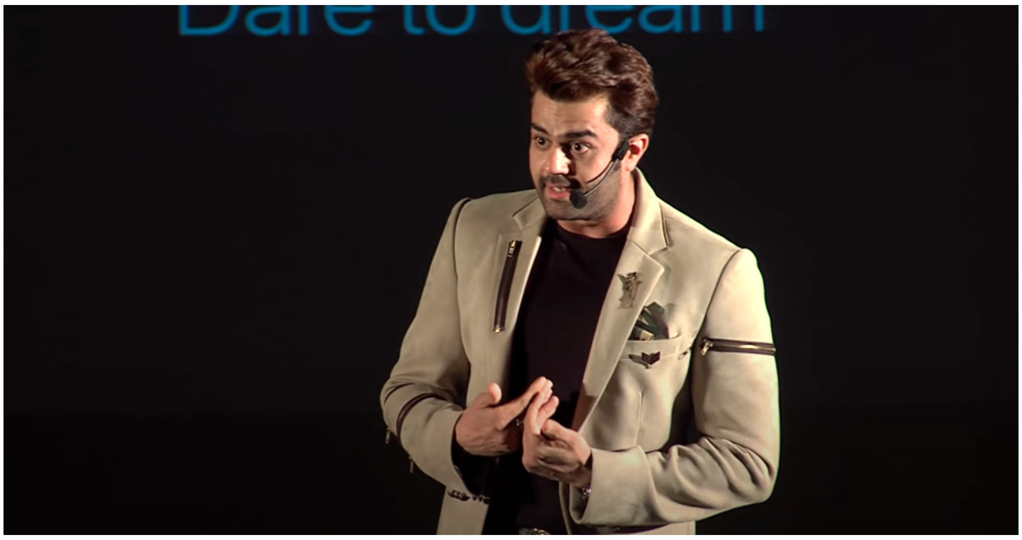


Figure 3.2: A frame of a Josh Talk video showing a person coming from India telling stories. From this frame it is possible to notice that the speaker is clearly visible so that it is easy to extract poses with Openpose.

3.3 Dataset Extraction

In this section there will be explained in detail all the steps that were necessary for preparing the training data for the *speech2gesture* model. To write all the scripts, it was taken as a reference point the code of *youtube-gesture-dataset* used for creating the dataset of the project *WordToGesture*. It should be noticed that the implementation of the latter shows that the model was not trained using the dataset coming from *youtube-gesture-dataset* but it was instead used another dataset which gave better results.
All the parts that are shown in the next have a common configuration script called *config.py* which has inside some useful constants shared among them. A schematic diagram for poses extraction is shown in figure 3.3.

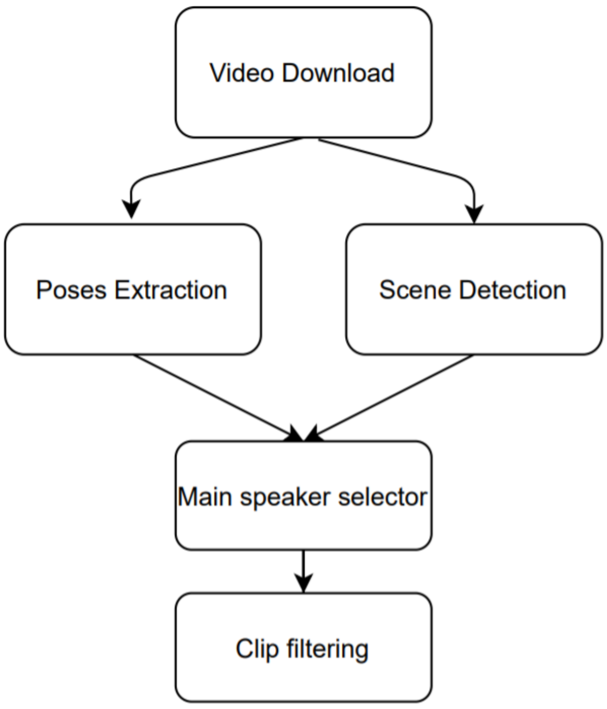


Figure 3.3: Diagram showing how the dataset is extracted. The videos are firstly downloaded, then the scenes and the poses are extracted from them. By using these data, the main poses of the main speaker are chosen from clips and finally these ones are filtered such that only the good ones are taken

3.3.1 Download of the videos

This operation was executed by a Python script called *download_video.py*. To run this one, it is necessary to be registered as Google Developer (Google (2021a)) so that it can be possible to get a Developer Key and use the Youtube Data API (Google (2021b)) with the Python package *google-api-python-client*.
The code contains 3 simple functions called *fetch_video_ids*, *video_filter*, *download* and a *main* function which executes them and also prints the video list returned by *fetch_video_ids* inside a file called *video_ids.txt*. *fetch_video_ids* basically takes all the video IDs inside a Playlist and returns a list with all of them. For doing this operation, the IDs of a specific Playlist and the Channel that contains it must be passed as arguments. The *download* function instead takes all the IDs returned by *fetch_video_ids* and downloads all the videos that can be downloaded. Before this operation, it takes all the info of the videos using the Youtube APIs, it prints them inside a JSON file (we have one JSON for each video), it calls the function

video_filter, which filter and takes the videos that have some specifications, it downloads the videos and finally writes inside a log file some info the videos that were downloaded and the ones that were skipped. By filtering the videos with *video_filter* and by using the function *YoutubeDL* of the Python library *youtube_dl*, only 439 were downloaded from the 501 available inside the Playlist. The function *video_filter* takes as argument the info of a video and it returns a positive boolean variable only if the video has a duration which is less than 1 hour (longer videos can require a lot of time to be processed by the other scripts), minimum resolution of 1280x720 (lower resolution can make the detection of poses more challenging from the Openpose point of view) and if it is saved on Youtube as .mp4 file (it is necessary to have videos of the same kind for processing them).
Even if a video is positively filtered, *youtube_dl* sometimes fails to download it for many reasons: before processing the next video ID, it tries for maximum 3 times to download and, if it cannot do it, the video is considered skipped. If the Playlist is not completely processed, the script does some checks to skip the videos that are already downloaded, and gives the possibility to insert a video ID of the list returned by *fetch_video_ids* to start the downloads from that point.

3.3.2 Poses extraction

To achieve the goal of pose extraction, Openpose (Cao *et al.* (2019b)) was used. It was written a very simple script whose name is *run_openpose* that it is composed by 3 functions and by the *main* one. The *main* manages the directories for the JSON files generated by Openpose (each one containing a single pose) and the Pickle ones created by the script (this will be explained in details), then it iteratively gets the video IDs from the path where we previously downloaded the .mp4 files thanks to *get_vid_from_filename*, which returns a video ID from a path, and then it launches a command for running Openpose with the chosen setup:

- Default accuracy
- Model *Body_25* for extracting 25 keypoints of the body comprising the head, the eyes, the neck, the hip, the legs, the foots and the arms (so the face and the hands keypoints were not extracted)
- The image files showing the detected poses that can be generated by Openpose are not saved
- The video for checking in real-time the poses extracted from the videos is not displayed
- Detection of a maximum of 3 people for each frame

It was decided this setup because:

- Saving JPEG images is not important for the goal and it can slow down the process
- The display of poses in real-time can again slow down the process and it is useless if *Google Colaboratory* or an external Server are used (as it was done in this thesis) since they both can't show the videos
- Hands and face poses can't be mapped on Pepper robot because it hasn't enough Joints for the purpose
- It was chosen to maintain the default accuracy because it is the best trade off among speed of poses extraction, accuracy and business of GPU memory occupation (more accuracy requires more graphic memory and in *Google Colaboratory* we can't choose GPUs that have a lot of memory available).
- Detect a maximum of 3 people can be useful since in some frames many people are shown and we want to choose for as much time as possible the main speaker among them (if there are more than 3 people, Openpose choose the three that are detected better)

An example of Openpose detection inside a frame is shown in image 3.4. It is important to notice that the Pepper robot it is allowed to close and open the hands but not to control each Joint of each finger directly, so the problem of hands management was not addressed in this work.
The poses extracted by Openpose, as well as the confidence values of the key-points, are saved in JSON files so that in the end we have a JSON file for each frame of the video. To group all these files into a single .pickle file, it was installed the Pickle python module and it was called the function *save_skeleton_to_pickle* that basically groups all JSON files for each video by relying on the function *read_skeleton_json* for reading them. After this process we will have a single a pickle file for each video downloaded. To run this project it is important to have Openpose installed: to do it in *Google Colaboratory* it was written a little script while for executing it on an external Server, a Docker Image which have inside Openpose was used (available in Openpose Docker Image). For testing the project it was created a little dataset whose poses were extracted on *Google Colaboratory*, while for extracting the whole dataset it was used the external Server which is much faster than *Google Colaboratory*. For more info heck the section 3.10.

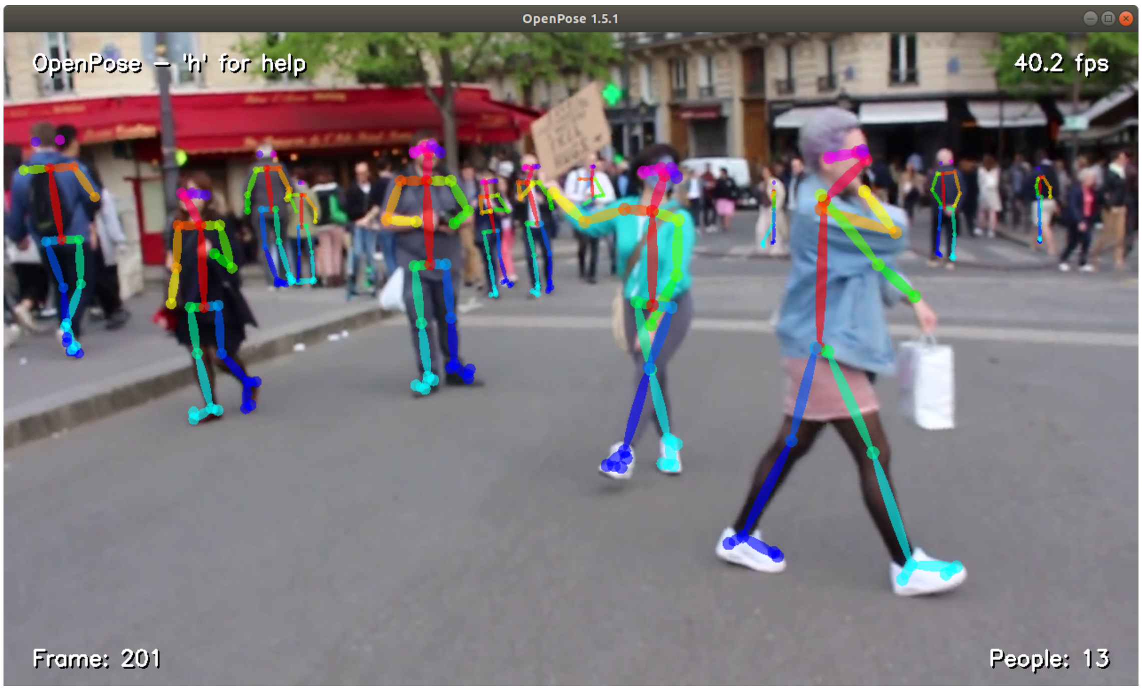


Figure 3.4: Example of Openpose detection inside a frame

3.3.3 Scene extraction

Also this script is very simple and its name is *run_scenedetect.py*. It has only one function called *run_pyscenedetect* and the *main* one that only checks if a video was already processed: if yes it jumps to the next video, if no it runs *run_pyscenedetect* for the selected video. PyScenDetect is a powerful Python library based on computer vision algorithms for detecting scene changes in videos and automatically splitting them into clips: by doing these operations we can check if the length of a scene is large and good enough (from the poses point of view) to create a training / validation or test set sample for *speech2gesture*. *run_scenedetect.py* only launches a command for starting to detect scenes with PySceneDetect (that has to be installed in advance as shown in the 3.11) with the options that we choose: in the specific case, it was used the *detect-content* option that compares each frame by sequentially looking for changes in content and which is useful for detecting fast cuts between video scenes although slower to process, and *list-scenes* option for printing the scene list and outputting to a CSV file on a path defined by us (totally 439 files were obtained since there were downloaded 439 videos). All the other parameters were not set and default values were used.

3.3.4 Scene Filtering and poses selection

This is one of the main parts in the process of dataset creation, it is implemented inside the *run_clip_filtering.py* script which relies on three python modules called *main_speaker_selector.py*, *clip_filter.py* and *data_utils.py*, and has inside two functions, called *run_filtering* and *read_sceneinfo*, as well as the *main* one. Before explaining *run_clip_filtering.py*, let's have a look to the contents of the three modules and *run_clip_filtering.py* in the next sections.

3.3.4.1 data_utils.py

As the name suggests, this module contains some useful classes and functions for loading and managing poses and videos:

- class *VideoWrapper*: given the path of a video, the class extracts from the video some useful information thanks to *OpenCV* (a module that is necessary as shown in 3.11). More specifically, it extracts the frame rate, the height of the frames, and the total number of frames inside the video. It has also some useful functions for computing the amount of seconds passed from the start to a specific point of the video (*frame2second*) given the frame rate and the number of frames (it is used to compute the total length of a video in seconds), for doing the opposite (*second2frame*, for setting the current frame(*set_current_frame*), and to get the video reader (*set_current_frame*) with OpenCV Video Reader; this was also done in the initialization phase of the class.

- function *read_video*: giving a path and the name of a video, it takes that video in the path and initializes the OpenCV Video Reader.

- functions *load_clip_data* and *load_clip_filtering_aux_info*: given the name of a video, the first reads the JSON file related to the clips while the second does the same with clips info (these files are generated by *run_clip_filtering.py* as it will be explained in the next).

- class *SkeletonWrapper*: it simply loads, given a path and the name of a video, the associated pickle file which contains all the poses that we extracted previously with Openpose; if it doesn't exists, *SkeletonWrapper* creates it from JSON files generated by Openpose. It has also two functions inside: *read_skeleton_json*, that is used by the class to read JSON files in the case the pickle file is not found, and *get* which returns the poses from a start frame to an end frame (parameters passed by us).

- function *draw_skeleton_on_image*: given a pose, an image and a thickness value, it draws the pose into the image using lines with the given thickness, finally returns the result of this operation.
- function *is_list_empty*: given a Python list, it simply returns true if it is empty.
- function *get_skeleton_from_frame*: since we can detect maximum 3 people with Openpose, we expect to have a maximum of three poses for each video frame inside the pickle file containing all the poses of a video. To take the data of all the people for a given frame, we choose the frame (an index where 0 indicates the start frame) and then we go to the *pose_keypoints_2d* element: here we have as many poses as people were detected in the frame. We can take the one that we want by specifying another index (from 0 to 2 in this case).
  What we pass to this function is the latter index so it returns the 2D keypoints.

3.3.4.2 main_speaker_selector.py

This module contains a single class called *MainSpeakerSelector* which is used to filter the poses inside a clip video such that only the keypoints related to the main speaker are extracted.
Passing the keypoints related to a clip and calling the method *get*, the class calls the function *find_main_speaker_skeletons* that firstly checks for the first frame on which at least one person is detected. If more than a person is detected, it is chosen the person (the keypoints are taken thanks to the *get_skeleton_from_frame* function defined in *data_utils.py*) whose average of confidence values of detected keypoints is higher than the others, with the belief that the keypoints of the main speaker are the ones detected better: indeed, we expect that the main speaker's body is much bigger in the frame with respect to other people because in a TED talk most of the time the camera will focus on him; a bigger body in Openpose will be simpler to detect with respect to smaller ones so the confidence values related to the acquisition of the keypoints will be higher.
Only the 9 keypoints of interest were checked: head, neck, right shoulder, right elbow, right wrist, left shoulder, left elbow, left wrist, hip. The order is the same used by Openpose and the other keypoints of the model *body_25* were not considered since they cannot mapped to the Pepper Joints.
After the main speaker is chosen and after him keypoints are taken again with *get_skeleton_from_frame*, *MainSpeakerSelector* iteratively calls, for all the remaining frames, its method called *get_closest_skeleton*: given a frame and the pose of the main speaker taken before, this function checks in that frame the pose of the

person (by exploiting again *find_main_speaker_skeletons*) which is closer to the pose of the main speaker by computing the absolute distance among each keypoint; if this distance is higher than a predefined threshold (in this case defined as maximum between neck height * 3 and shoulder length * 2), the tracking is considered broken and a empty list is returned, otherwise the keypoints of the tracked person are returned and so taken by *MainSpeakerSelector*.
We expect that, in a clip detected by PySceneDetect, the position and the size of the body of the main speaker will not change a lot in the frames: if a point of view changes so much (so if the camera change or if it fastly moves for focusing on something else), PySceneDetect will classify this as a different clip, this is why the predefined threshold makes sense.
Finally, the keypoints of the first frame (that represents the speaker whose keypoints have the highest confidence) and all the tracked ones are grouped together and returned by the *MainSpeakerSelector* class. If we instead set Openpose to detect only one person for each frame, the existence of *MainSpeakerSelector* doesn't make any sense.

3.3.4.3 clip_filter.py

This is the last module and it is exploited for filtering the video clips. It has inside a class called *ClipFilter* which has many methods: it has one for each filter applied on the clip, one for calling all of them and another one for returning the results (*get_filter_variable*). Given the video ID, the start frame and the end frame of a clip, the poses of that clip and finally the poses of the main speaker extracted by *MainSpeakerSelector*, *ClipFilter* firstly initializes some variables by exploiting the passed values and then, by calling its method *is_correct_clip*, it starts the clip filtering process. The outputs are three and are returned by the function *get_filter_variable*: a list of boolean values containing the results of the filters (if it is 1 then the clip is considered valid for that filter), a list of messages indicating why the clip is not considered valid, and a list of info about back poses, missing joints, looking sideways, small body and still body (explained better in the next lines).
Totally seven filters were applied and only the 9 keypoints of interest were considered (same keypoints of the previous chapter):

- *is_skeleton_back*: takes a ratio as argument and then checks the position of the shoulders of the main speaker for each frame of the clip: if the left shoulder detected by Openpose is on the right, then the body is considered turned back and that frame is not considered valid. The total number of invalid frames is saved in the list of info and the function returns True if the amount of incorrect frames divided by the total number of frames in the clip is higher than the ratio passed as argument.

- *is_skeleton_sideways*: even this function takes as argument a ratio and checks for each frame of the clip if the main speaker is sideways with respect the camera: if the x coordinate of the head is greater than the x coordinate of left shoulder or lower than the x coordinate of the right shoulder (or opposite depending on the fact that the main speaker can be turned back), then the frame is considered incorrect. The total number of incorrect frames is saved in the info list and the ratio passed to the function must be lower than the ratio of the amount of incorrect frames divided by the total number of frames in the clip for the function to return True.

- *is_skeleton_missing*: again a ratio is passed and for each frame of the clip the function checks if any of the 9 joints of the main main speaker is missing: if yes, that frame is not considered valid and again the ratio between total invalid frames and total frames in the clip must be higher than the ratio passed for the function to return True. The number of incorrect frame is saved in the list of info.

- *is_skeleton_small*: as before, a ratio is passed and a check for all the frames of a clip is done: if the distance (in pixels) between the two shoulders of the main speaker is less than a given threshold (100 is the value passed by default by *config.py*), that frame is not considered valid. Again the info is saved and True is returned if the given ratio is lower than the total number of invalid frames divided by the total number of frames in the clip.

- *is_too_short*: a very simple filter that checks if a clip contains at least the number of frames that we expect: since the average fps of a video is 25 fps and since *speech2gesture* requires samples that are long 4 seconds, it was set a minimum of 25 * 4 frames; this condition does not always guarantee that clips are long enough for the model but it is a good starting point. It returns True if this condition is not verified.

- *is_picture*: it samples a clip 5 times extracting 5 frames and then, by using OpenCV and by going in the frames sampled inside the video, the function computes the average distance between two consecutive frames by checking the pixels values. If the distance is below a given threshold (here is 3000000), the function return True. The distance is saved in the info list.

- *is_many_people*: it checks for each frame of the clip how many people are present. If the average is more than 5, the function returns True. For the setup that we choose in Openpose, this condition can't happen and the maximum of people detected in a frame is 3. It can be useful if the Openpose setup changes.

is_correct_clip calls sequentially all this filters by passing the ratio value to the functions that require it: it was chose to pass 0.3 to *is_skeleton_back* and 0.5 to *is_skeleton_missing*, *is_skeleton_small* and *is_skeleton_sideways* following the tuning values set in the Github project. The function set also the messages (like "too Short" is a clip is too short) that are generated in case a filter returns True. Finally, if all filters return False, *is_correct_clip* returns True and a message "Pass" is created.

3.3.4.4 run_clip_filtering.py

Now that we have described all the modules, it is much simpler to talk about the behaviour of this script: basically the *main* function iteratively loads, for each video ID, the pickle data by calling *SkeletonWrapper*, the video reader by calling *VideoWrapper*, and the clips data by searching the JSON file generated by PySceneDetect through the function *read_sceneinfo*. If a specific video ID was not processed before, *run_filtering* is called, otherwise we pass to the next video ID.
read_sceneinfo not only loads the data of the clips, but it also splits all the clips that lasts more than 10 seconds in smaller ones: this operation is executed because it is easier to preprocess the motion and to convert the audios. Indeed, as we will see next, preprocessing motions that lasts too long is not very useful with the proposed technique, and converting long audios can be very challenging or even unfeasible for the *many_to_one* Audio Conversion model. 10 seconds has been chosen as a good trade off between the speed of the audio conversion and the effectiveness of motion preprocessing. Moreover, since the *speech2gesture* model will shrink these clips in samples of 64 frames taken at 15 fps, we will not lose so much data: to give an example, if we would choose to have a duration of 5 seconds per clip, in that case we will take 3 samples by shifting 5 frames, operation that is done in the project (0 to 64 frames i.e. 0 to 4.3 seconds, 5 to 69 or 0.3 to 4.6 seconds, 10 to 74 or 0.6 to 4.9 seconds), and we will delete 0.1 seconds of data. The less we split clips, less data we will loose. The function returns the new modified list containing the information of the clips with maximum duration of 10 seconds (it is indicated the starting frame of each one). These ones are then saved in a JSON file.
Using the *VideoWrapper*, the *SkeletonWrapper* and the scene info, for each clip the function *run_filtering* takes the poses with the method *get* of *SkeletonWrapper*, then it extracts the poses of the main speaker by calling *MainSpeakerSelector*, and finally it process the clip by initializing *ClipFilter*, calling its method *is_correct_clip* and returning the result by calling *get_filter_variable*.
The results (messages, info and boolean values of each filter) are put together inside a Python Dictionary representing all the info of that clip (start frame,

end frame, a boolean variable that is True if it passed all the filters, i.e. the value returned by *is_correct_clip*, the results for each filter and the messages returned), while some info of the same clip (start frame, end frame and result of *is_correct_clip*) are also put together with the poses of the main speaker in another Python Dictionary (if *is_correct_clip* is False than the poses of the main speaker are not taken). For each video we will have these two Dictionaries containing the above info for each clip.
The dictionaries are then returned by the function *run_filtering*, and finally saved by the *main* as JSON files: the first will represent auxiliary info of a video ID, while the second will represent the poses of the main speaker in that video (so if it is not detected or some clips didn't pass the filter, for those clips we will have no poses of the main speaker).

3.3.5 Creation of the Dataset

Now that we have extracted all the poses of the main speaker, it is possible to create the Dataset composed by training, validation and test sets. To do so, it was written a script called *make_ted_dataset.py.* This script also calls the functions of a module whose name is *writecsv.py* and that is used for adapting the Dataset such that it can be used by *speech2gesture* model; *writecsv.py* will be explained in details in the next chapter as shown in the *Summary* section.
make_ted_dataset.py contains only one function, whose name is *make_ted_gesture_dataset*, that is called in the *main. make_ted_gesture_dataset* manages the paths for loading and putting the data, then, for each video in the Dataset, the JSON file generated by *read_sceneinfo* is loaded. Thanks to the info inside, it is possible to load all the data that we saved before in the two main Dictionaries: all the clips with inside the poses of the main speaker are used to create the three sets, and by taking 80% of them for the training set, 10% for the validation set and 10% for the test set, the Dataset that contain the poses is created. Note that by shrinking clips such that they lasts at least 10 seconds we could have a much more uniform Dataset: indeed, if we have for example 10 clips where one lasts 100 seconds while the other 9 last 10 seconds, and we then randomly sample 80% of them such that they can be used for the training set, it is more likely to find inside this set the one that lasts 100 seconds, having in this way 170 seconds for the training set, and only 10 seconds of data for the other two sets: practically, this means that the training set will be constituted by 89% of data, instead of 80%.
From the two Dictionaries the data is extracted and then, for each clip, a Dictionary containing the following info is saved:

- A string indicating if it is a training, validation or test set.

- The name of the video from which it was extracted.
- The number indicating the start frame.
- The number indicating the end frame.
- The poses of the main speaker.
- The times (expressed in seconds) of all the video frames on which the poses were extracted.
- The culture of the main speaker.
- An unique Id (integer number) for that clip.

The clips are saved as pickle files and their name was set such that they can be easily found by the other parts of the project: for instance, it is formed by the type of the Dataset, the ID, the start and the end frames, and by name of the video from which they were extracted.
There were crated more than 7000 clips for the training set and more than 700 for validation and test sets. A schematic diagram showing how the Dataset is created and adapted can be seen in 3.5.

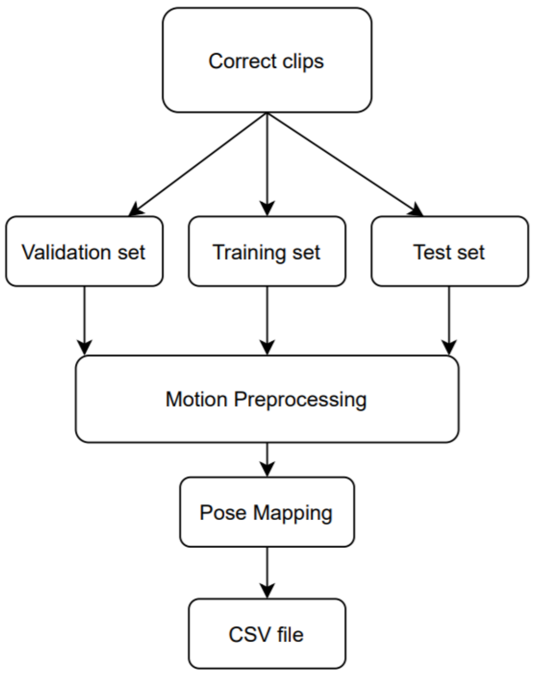


Figure 3.5: Diagram showing how the Dataset is crated. From the correct clips are extracted 80% of training data, 10% of validation data and 10% of test data. From these data, the poses are preprocessed then mapped such that they have the same configuration used by *speech2gesture*. At the end a csv file containing info on the data saved is created. The process is still not finished since there must be loaded also the audio files to have the final Dataset.

3.3.6 Adaptation of the Dataset

As said before, this part is entirely processed inside the *writecsv.py* module which is executed inside the *main* function of the *make_ted_dataset.py* script. It contains three functions and in order to preprocess the poses it is exploited another module called *motion_preprocessor.py*. Let's have a look on it before showing the *writecsv.py* one.

3.3.6.1 motion_preprocessor.py

This code is very important since it is exploited to preprocess all the poses before using them as a Dataset for *speech2gesture*: if the data is not good enough it can be expected to have a model that does not work as we may want.

motion_preprocessor.py has a class called *MotionPreprocessor* with inside many methods for processing the motions. Firstly, when it is initialized, it takes a set of poses and put them inside a Numpy array; then by calling its method *get* the preprocessing starts and all the methods are called one after the other: if the poses can't be processed by at least one filter, an empty array and a message indicating the error encountered are returned otherwise the processed poses and a message containing the word "PASS" is returned. Let's see all the methods in details:

- *has_missing_frames*: it checks how many frames in the clip are empty and without any pose: if more than 10% are empty, a True value is returned as well as a printed message indicating how many frames are missing, otherwise the function returns False. Notice that clips can't be longer than 10 seconds so poses must be detected for at least 9 seconds.

- *fill_missing_joints*: if the method described above returned False, this function fills all the missing joint by using the one-dimensional linear interpolation. If more than half x or y coordinates are missing for a joint in video clip then an empty set of poses is returned, otherwise it is returned the set of interpolated poses.

- *is_static*: this function is called only if the joints were correctly filled. It has two other functions inside: *joint_angle* that computes all the angles between two joints in a frame and returns the angles in degrees, *get_joint_variance* that extract the poses in all the frames of a clip, it calls *joint_angle* and finally computes the circular variance (a variance used for circular data, i.e. angles) of all the angles by calling *circvar* function of the *scipy* library (see 3.11). *get_joint_variance* is called by passing firstly the keypoints of the right and then the ones of the left arm, indeed we want to measure the variance of the left and right arm which are the parts of the body that move more when a person is talking.
  If the variance of both the arms is lower than 150 degrees the movement is considered too static and the function returns True after printing a message; if the variance is equal or greater than 150 degrees then the function returns False and prints another message indicating that the movement is not too static. If the movement is not static, the method *get* jumps to the next check otherwise, as it was told before, it returns an error message and empty poses.

- *has_jumping_joint*: this method is used to detect if there are joints that are moving too fast. To do this, this function checks the joint that varies more along the x coordinate from the start to the end of the clip; if any of the 9 joints of interest (the ones that are mapped to the Pepper robot) varies more than half of this distance between any two consecutive frames, *has_jumping_joint* returns a error message and a True value, otherwise False is returned as well as a positive message.

- *smooth_motion*: now that we have reached this point, we have only to smooth the movements between frames: since Openpose cannot detect perfectly all keypoints, sometimes fast joint transitions between frames may happen. To address this problem, *smooth_motion* applies a Savitzky–Golay filter on the poses. This filter was applied also on the original code available on Github and it was adopted in this implementation as well.
  Similarly to the function *is_skeleton_sideways* of the class *ClipFilter*, also here it was addressed the problem of checking if the main speaker is sideways or not: if the speaker is sideways for at least one frame in the clip then an error message is printed and again an empty set of poses is returned. This operation was not done by a specific method but it was applied inside the *get* method as the last preprocessing operation. From now the poses of the clip are ready to be used for the training of *speech2gesture*

3.3.6.2 writecsv.py

This is the last script used for preparing the poses such that can be used for the network. After this, as we will see later, the poses are put together with the converted audios and to create the final Dataset.
writecsv.py is called in the *main* function of *make_ted_dataset.py* after the method *make_ted_gesture_dataset* and, as we said before, it is composed by 3 functions:

- *take_correct_keypoints*: given a pose and the number of keypoints to be mapped (for the moment this function works only with the first 9 keypoints since all the others were not useful for the purpose of this thesis), this function maps the keypoints from the *body_25* Openpose model to the *model_23* model used for *speech2gesture*.
  More specifically, inside *model_23* the keypoints are represented as two separated lists, one for the x coordinates and the other for the y ones while the confidence values are not returned. In *body_25* instead each keypoint is a dictionary element with inside x,y and confidence values.
  The 9 keypoints used to train *speech2gesture* are ordered as follows:

  0. neck
  1. right shoulder
  2. right elbow

3. right wrist
4. left shoulder
5. left elbow
6. left wrist
7. head
8. hip

The first 9 keypoints extracted by *body_25* model of Openpose are ordered as follows:

0. head
1. neck
2. right shoulder
3. right elbow
4. right wrist
5. left shoulder
6. left elbow
7. left wrist
8. hip

The function returns three different lists containing x,y and confidence values of the keypoints ordered as *model_23*.

3.3.6.3 write_final_skeletons

Given an objective frame rate, the data of a clip (the one that of the Dictionary created by *make_ted_dataset.py*), the number of keypoints and the path to save the new data, this function takes the keypoints by downsampling the frame rate to 15 and then it converts them by calling *take_correct_keypoints*. To take the keypoints at the correct frame rate, it was checked the frame rate of the the video from which the clip was extracted: if it is lower than 15 then that clip was not considered (the upsampling is avoided since it can creates poses that are very different from the ground truth), if it 15 then the poses are taken from all the frames and finally if it is higher than 15, the poses are taken by sampling them from frames that were chosen by considering the conversion factor. If for example we have a source fps equal to 30, it is necessary to take one pose each 2 frames, if instead is 15 then every 25 frames we must take 5 poses that came from 2 consecutive frames and 10 poses that came from two frames separated by another one;

in this way we will sample $10 + 5 = 15$ poses in $10 * 2 + 5 * 1 = 25$ frames. Since in the last case the frames can be chosen in many different ways, the sampling indexes are given by a distribution that allow us this conversion: for instance, in case we want to pass from 25 fps to 15, we expect to sample 33% of the time two consecutive frames and 66% of the time two frames that are separated by another one.
Finally, the keypoints extracted in this way and converted by *take_correct_keypoints* are saved in a pickle file (one for each clip) and the new times that indicates the exact position in the video from which the keypoints are extracted were returned in a list format as well as the name of the new saved files.

3.3.6.4 writecsv

This function whose name it is the same as the module, calls iteratively *write_final_skeletons* for all the clips; for each file saved, it writes some lines in a csv file indicating for each cell the following info:

1. video ID from which it was extracted
2. type of data (training/validation/test)
3. ID of the clip
4. number of frame
5. pose time in that frame
6. culture
7. path of the clip
8. fps

This format is very similar to the one used in the original *speech2gesture* project and it is useful for loading the data into the Neural Network and for preparing the last Dataset file containing all the samples with poses and converted audios (more details will be given in the next). Notice that in this file, for each clip there are as many rows as the number of frames taken at 15 fps; this file is finally saved with the name *frames_df.csv.*

3.4 Training of the model for many-to-one Voice Conversion

During this step the two training stages of the many-to-one Voice Conversion model are processed (see the figure 2.6). Since the model was available on Github and since it was not changed anything in the code for this phase except for the paths of the data inside the file *default.yaml*, details about the scripts will not be given. The only problem that was encountered following the instructions of the Github project is that the files of the TIMIT Dataset had extension .WAV and .PHN instead of .phn and .wav, so they were all renamed. Since the project was thought to run on a configuration with 4 Gpus and since for on *Google Colaboratory* only one was available, it was also necessary to change the parameter num_gpu in the *default.yaml* file for speeding up the training process.
Notice that the project is not written using *Tensorflow* but instead using *Tensorpack*: this makes a lot of difference since *Tensorpack* is no more supported and runs only on Cuda 9.0. This Cuda Version, at the time that the training was started, could be installed only on the *Google Colaboratory* Gpus.
For doing this, two *Google Colaboratory* notebooks were created so that one was used for training the first Network while the other was created tot train the second one.
As described in 3.10, *Google Colaboratory* has a lot of limits and in this case they made the difference: since a lot of data had to be loaded iteratively for training the Networks and since *Google Colaboratory* saturates the flow of this data, the training process was terribly slow. For this reason, the two networks were trained until they reached an acceptable results and they were capable to extract decent converted audios. Some users show that slightly better results can be obtained, reaching lower loss values, higher accuracy and obtaining audios that have less noise.
Practically, the Network 1 was trained for 20 epochs (100 steps each) with a batch_size=32 and it reached an accuracy of 78% with an average loss of 0.68 while Network 2 was trained for 261 epochs reaching a loss of 0.0049.
The training process of Network 2 was slower with respect to Network 1 so the data was directly loaded inside the *Google Colaboratory* machine to speed-up the process. For both the Networks there were saved all the checkpoint files required to load them or to restart the training from the point that was left, some log files that can be opened by Tensorboard (one for each time we ended the training) and some JSON files containing the stats (average loss, accuracy, ecc.).

3.5 Audio Extraction and Conversion

This is the final process for creating the final Dataset (the final steps are shown in the diagram 3.6) that will be used for *speech2gesture*. To reach this goal, a script called *extract_data_for_training.py* was created taking as example the original one created in the original project of *speech2gesture* which is also publicly available on Github.

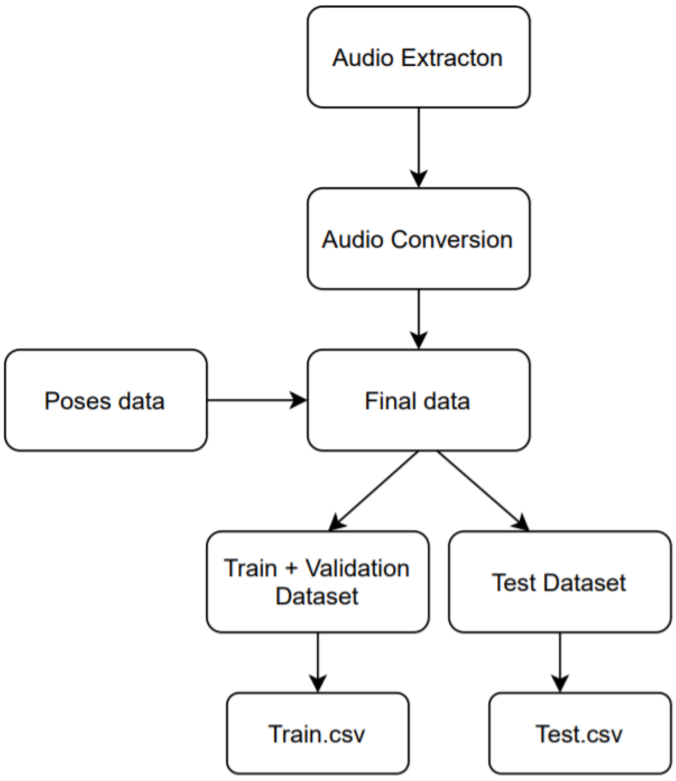


Figure 3.6: Diagram showing how the final Datasets are created. Using ffmpeg the audios are extracted from videos, then by using the trained many-to-one voice conversion all the audios are converted. By adding also the poses extracted in the previous steps, all the samples from the training, validation and test sets are created. The training data and the validation data is put together inside the training Dataset whose info is inside the train.csv file while the test samples are put together inside the test Dataset and a test.csv file containing its info is created.

To iteratively convert all the audios, this time some changes were applied to the model of many-to-one VC. The main limit that was overcome is the one related to the length of the audios used in the inference process. To solve this problem, these changes were made in the original code:

- In the file *models.py* containing the model of the first and second Neural Networks, it was crated another model for the second Neural Network called *Net2_1*: it is exactly the same as the original model, but here for the initialization of the class it is required the duration of the audio to be converted. This custom duration is exploited to compute the time-steps necessary to convert the audio following the formula 3.1.

  $$floor(duration \, * \, sampling\_rate) \, / \, hop\_length \, + \, 1 \tag{3.1}$$

  In this formula *floor* is the operator that returns the greatest integer value less or equal to its argument and the *hop_length* represents how many samples we advance before passing to the next window.
  To better understand this value, it is necessary to know how the Short Time Fourier Transform (STFT) is applied to the audio that has to be converted (it is necessary to extract features like MFCCs or the log-mel spectrogram): a little part of the signal composed by the first *n_fft* samples is taken (512 samples is the value used here as it is suggested for speech audio), then a custom window, for instance the default *raised cosine window* (i.e. *hann*), with a given window length, for instance 400 samples, is applied to the chosen *n_fft* samples; finally we do a translation of *hop_length* samples (for instance 80), we take again another *n_fft* samples of the signal and we repeat the process till the end.
  To match the length of the audio, the samples taken at the end are padded with zeros. A smaller *hop_length* implies more frame oversampling (many successive frames including the same samples) which gives better results but it is computationally slower, a *hop_length* greater than *n_fft* implies that some samples are not considered each time a filter is applied, a window length smaller than *n_fft* implies more temporal resolution of the STFT (i.e. the ability to discriminate impulses that are closely spaced in time) at the expense of frequency resolution (i.e. the ability to discriminate pure tones that are closely spaced in frequency, a window length greater than *n_fft* implies that the edges of the filter will not be used for filtering the signal.
  These values were not changed for the thesis.
  Since sometimes numerical approximations of the computers creates some problems during the computation of 3.1, it was necessary to approximate the argument of *floor* to the first eight decimal numbers.

Net2_1 is used only in the conversion phase since for the training even audio samples with the same length are fine, while a model for the first Network, i.e *Net1* was not necessary since it does not depend from the length of the audios.

- The file *data_load.py*, that is necessary to load the data to the two Neural Networks, was slightly modified indeed it was changed the function *get_mfccs_and_spectrogram* such that it is does not cut the audios in the conversion phase: by doing this, when it calls the function *_get_mfcc_and_spec* for extracting the MFCC features and the normalized amplitudes of the STFT and log-mel spectrograms, the whole audio is passed as a Python list.
  For completeness, there will also shown in the next the classes that were changed to iteratively loads the data for the two networks. Since these does not work, the *get_mfccs_and_spectrogram* function was called everytime an audio has to be processed by *Net2*.
  - The class *DataFlow*, that is inherited by the classes *Net1DataFlow* and *Net2DataFlow* used for loading the data inside the Networks, was modified such that it can take as argument a boolean value called *rand* (indicating whether or not the Networks must load the data randomly), the duration of the audios, the path and the size of the batch. Default values *rand* and duration were passed to make it compatible also with the training phases described before. These values were set as attributes as well as an index *i* with a 0 value (used to iteratively load .wav files and their duration in *Net2DataFlow*).
  - While *Net1DataFlow* remained unchanged because in its training phase *Net1* can also load the data randomly while in the conversion phase is used to converts the same data as *Net2*, it was necessary to change *Net2DataFlow* such that it does not load the data randomly in the conversion phase (in the original project it can be done since only one file at time is converted). To do this, a condition was used: if rand is equal to True (as in the training phase), the data can be loaded randomly otherwise it is iteratively loaded the file at index *i* (*i* is incremented after the load) and the function *get_mfccs_and_spectrogram* is yield such that it can be called everytime the data is available (this happens also in the training phase).
- The file *convert.py* was rewritten such that it converts iteratively all the audios that we want without cutting them: now it is no more a script with the *main* function but it is instead a python module whose function

audio_conversion takes as arguments the input and the output paths, and then calls all the functions needed to process the audios.
This script firstly loads the latest Checkpoint files for loading the two Networks, then it calls the function *do_convert* passing to it the path indicating the location of the files to be converted (input path) and the location on which we desire to put the converted files (output path).
The function *do_convert* gets all the files that are inside the input path and then it iteratively loads them using the *wave* library (if it is not already installed, it may be installed using the command *pip install wave*), extracts the name and the duration, and now it starts the conversion:

- It initializes *Net2_1* by passing the duration
- It initializes the *Tensorpack Predictor* by specifying the model *Net2_1*
- It extracts the audio features by calling *get_mfccs_and_spectrogram*
- It converts the features by passing them to the *Tensorpack Predictor* (Phonetic Posteriorgrams (PPGs), converted audio and original audio spectrograms are returned by it)
- It converts the audios from frequency domain to time domain by exploiting the *convert* function
- It finally saves the converted audio in the output path

convert.py is thought to work even if exceptions occurs or some audios were already converted.

Now it's time to prepare the final Dataset file for *speech2gesture*. This is done by the script *extract_data_for_training.py* that has inside a *main* function as well as *create_dataset*, *save_video_samples* and *convert*. The latter is only a utility created in case we want to change the audio paths inside the *frames_df.csv*.
save_video_samples is the first function called by the *main* and it is used to save the audio samples from the video files. To reach the goal, it extracts all the unique IDs, as well as the start and the end frames, of all the clips inside the *frames_df.csv* file, then it calls a function available on the original project of *speech2gesture*: this function whose name is *save_audio_sample_from_video* simply extracts the audio file from a video and saves it; the name of the video from which extract the audio, the start and the end frames and the output paths are required by this function. To do so, the function simply launches a command for running *ffmpeg* (see 3.11) with the following options:

- -i for the input (path of the video)

- -ss that is a command which when used as an output option as in this case (before an output path), decodes but discards input until the timestamps reach position (start frame is passed)
- -to to stop writing the output or reading the input at position (end frame).
- -ac to set the number of audio channels (2 in this case)
- -ar to set the audio sampling frequency (44100 in this case)
- -vn to skip the inclusion of the video in the output path that is passed (so only audio file is saved)
- -y to overwrite the output files without asking
- -loglevel to set the level of the logs ("warning" was set)

For more information about *ffmpeg* check the Documentation.
After this operation, the *main* calls *audio_conversion* to convert all the audio files that were extracted.
Finally, by calling the *create_dataset* function, the dataset is created. To do it, this function exploits the *frames_df.csv* file to do, for each clip ID, the following operations:

- Extraction of all the x and y coordinates from all the poses of the clip by loading the pickle file in the path indicated in *frames_df.csv*
- Loading of original and converted audios with a sampling rate equal to 16000 by using the *librosa* library (see 3.11). Only original audios are available at a sampling rate of 44100 but loading them using this library means that the Dataset will become much heavier (and it already is since it occupies near 57 GB of memory).
- Splitting all the data by creating data samples of 64 frames (sampled at 15 fps): this means taking exactly 64 consecutive poses since all the poses were extracted at 15 fps. In the audio case, it is chosen to take all the samples between the starting frame of poses and the end one (these information are again inside *frames_df.csv*). The data are acquired by taking overlapped samples: we acquire the data of poses and audio by shifting the clips by 5 frames each time; this means that between 2 consecutive frames there are 60 overlapping frames which are exactly four seconds at 15 fps.
  This fact is very important since training the model with overlapping frames indirectly push the model to learn the past and future contexts of the samples: by doing this, the Network can learn to map the poses that are asyncronous with respect to the speech by learning what is the correlation of all

the overlapping samples till to a maximum of 4 seconds.
It is worth to notice that in the project *Speech Gesture Generation using Trimodal Context* the syncronization between speech features with gestures is not learned by the Network because there is a belief that these features are always syncronous with respect the co-speech gestures. This fact is not always true as it is explained in Butterworth & Hadar (1989), this is why in *speech2gesture* this element is learned directly by the model.

- Creating for each training sample a dictionary containing:
  - the start frame
  - the end frame
  - the type of data (training/validation (dev in the code) /test)
  - the culture (in the specific case it is always Indian)
  - the video from which the sample was extracted
  - the path to the original audio
  - the path of the converted audio
  - the index indicating the sample position in the Dataset

  This dictionary is returned by *create_dataset* function.

- Saving a Dataset containing training and validation data, and a Dataset for the test. Saving two different Datasets allows to test the Network also in *Google Colaboratory* since the test set is much smaller than the others and so avoids to face the *Google Colaboratory* limits (see 3.10). This time they were saved as hickle files: hickles are similar to pickle files so they can be managed very easily, but they are instead coded using the HDF5 format. This makes the data faster to be processed and also it bypass the limit of memory of the pickle files: experimentally there was found that these can't be larger than $\sim 2GB$.
  There was also tested the possibility of managing hdf5 files instead of hickle ones by exploiting the h5py Python library, but it was much more complex and the performances were nearly the same, so the hickle format was chosen.

create_dataset was called two times, one for the training and validation sets, and one for the test set. The dictionaries returned are finally saved as *train.csv* (*test.csv*) files by the *main* function.
At the end of the process, turn out to be saved 10572 test samples, 10722 validation samples and 95679 training samples: the amount of data results similar to the amount used for training the original project of *speech2gesture* on a specific speaker. As it will be shown in the 4 chapter, this amount of data revealed to enough for correctly training the Network.

3.6 Training of the speech2gesture model

This is maybe the most important phase to see if the proposed approach whether makes sense or not. To reach this goal of course it was necessary to modify some parts of the original code. Preliminarily all the code was firstly adapted to Python 3 and *Tensorflow2* for running it with the Server available into the University; the version used in Google Colaboratory for the inference instead it was not adapted to *Tensorflow2* since it must work together with the model for many-to-one VC: the latter works only with *Tensorflow1* as explained in the section 3.10. Before talking about the changes made, it is necessary to show what is the structure of the *speech2gesture* project:

- it is divided into three main folders: *audio_to_multiple_pose_gan* that contains the main code and the models used for the DNN, *common* that contains some constants and useful functions necessary for the model to run, *data* that contains the code for preparing the data samples and the *train.csv* file. The operations made by the functions inside the *data* folder are the same explained in the last chapter except for the facts that all the samples weren't saved in hickle files, they did not contain both the original and converted audio, and the problem of Voice Conversion was not addressed.

- the *common* folder contains a lot of modules exploited by the model inside *audio_to_multiple_pose_gan*:

  - *utils.py* that contains some functions for getting the paths of the data
  - *pose_plot_lib.py* for plotting the poses and creating videos showing the movements
  - *pose_logic_lib.py* for managing the keypoints (normalization, denormalization, translation ecc.)
  - *mel_features.py* which was not used and contains some functions for managing the features of the mel spectrogram (they were instead managed in the file *static_model_factory.py* one step before getting used as input inside the Network)
  - *evaluation.py* that contains the implementation of the PCK index
  - *const.py* that contains some useful constants used by the model
  - *audio_repr* only used for loading audio files
  - *audio_lib.py* that has some functions for saving audios from videos, save an audio sample ecc. By seeing the code, it can be thought that this module contains some functions for creating the Dataset but in my project only the function *save_audio_sample_from_video* was used

- *audio_to_multiple_pose_gan* instead contains:
  - *train.py* that only launches the training and it contains the main function of the project
  - *tf_layers.py* that contains some functions that implements some specific operations used by the layers of the model (e.g. 2D and 1D Convolutions, Normalization ecc.) and some functions for processing the data like *to_motion_delta* that computes the difference between poses before using them as training samples (for more details, see the chapter 2). It also has a function for implementing the L1 Loss and L2 Loss (only L1 was used).
  - *static_model_factory.py* contains the models used for the Generator, the model used for the Discriminator and a function ofr extracting log-mel features given an audio. Even if there is only one model for the Generator, two were implemented: they are both exactly the same U-Net CNN models described in the chapter 2), but the main difference is that one of them have some max-pooling layers. The one with pooling layers is used for the no-GANs model, maybe because these layers can optimize the time for the training process by extracting features that are sharper. Even if this is not explained well in the project, I used the model without pooling layers for the GANs while I used the model with pooling layers for the no-GANs model, exactly as it was done in the original project. Moreover, the name of these models suggest which one to use during the training, indeed one is called *audio_to_pose_gans* while the other *audio_to_pose*
  - *model.py* is the core of the project: it initializes and trains the chosen model.
  - *config.py* contains all the configurations used. Some are necessary to launch the project while some others have already some default values.
  - *dataset.py* manages the data data inside the Dataset, it creates the batches for the training and it preprocess the data for the the training.
  - *predict_audio.py* is a script used for the inference of the Network by giving to it an arbitraty audio file as input
  - *predict_to_videos.py* is a script used for the inference but here there were used samples coming from the Dataset

Now let's see which of these files were changed and how, by also explaining what was instead used in the original project.
Staring form the *const.py* file, there were defined 4 Python lists containing the new keypoints order and it was changed the shape of the poses:

- BASE_KEYPOINT for the neck keypoint (0), same as the original project
- RIGHT_BODY_KEYPOINTS for the keypoints of the right part of the body (1,2,3); in the original project there was also the keypoint of the hand but as we said previously, the hands keypoints are not considered
- LEFT_BODY_KEYPOINTS for the keypoints of the left part of the body (4,5,6); also here the left hand was not considered
- CENTRAL for the central keypoints: 7 for the head and 8 for the hip. This list is new since the head and the hip were not considered in the original project
- POSE_SAMPLE_SHAPE now is (64,18) instead of (64,96) since we have 18 coordinates for each pose
- AUDIO_SHAPE remained unchanged and it loads 67267 samples which correspond to little ca. 4 seconds of audio sampled at 16000 samples/sec but there is a big difference from my project: in the original project indeed all the audio samples used for training already had this shape, meaning either that audios have been previously process or that they were extracted in a different way from the videos by taking for each sample exactly 67267 samples. In my case this was not done, instead there were extracted the audios corresponding to the 64 poses taken in the video, so they always have different number of samples: this can make a big difference in the training process since to be used for that purpose, my audio samples need to be preprocessed in advance; if they have more samples then the last samples are deleted, if they have less samples then 0 values are added to have always 67267 samples. This may cause slight syncronization problems since the Network learns also to map poses that have not a ground truth audio, even if this happens for very short time instants. It is expected that in average the result may be fine but a test for checking if there is a difference among these kind of samples and the *speech2gesture* ones was not executed.

In *pose_logic_lib.py* many changes were applied since it is the module that manages the poses:

- the normalization function called *normalize_relative_keypoints* was completely changed and it was instead used the one proposed in the chapter 2: taking as input a set of 64 poses, the neck coordinates and a resize factor (this parameter was not used since this factor is computed inside the function), the function computes for each pose the distance between neck and head keypoints and between the two shoulders; if their ratio is higher than 0.6 (meaning that neck is big or that the speaker is little bit sideways) then the resize factor is defined as $neck/0.6$ (something similar to what can be the shoulder lenght), otherwise it is equal to the shoulder length. Each coordinate is divided by the resize factor and finally the normalized poses, as well as the resize factors used to normalize them (one for each pose), are returned by the function. The resize factor it is needed because if we test the Network using for example the validation or the test sets, then we may also want to denormalize the poses at the end of the inference to check the results.

- the function *preprocess_to_relative* used to subtract the neck keypoint from poses (so it can be the reference point) was adapted to work with the new number of keypoints (9) and also to return the coordinates of the neck subtracted since it will be necessary to functions like *normalize_relative_keypoints*.
- of course also the function *de_normalize_relative_keypoints* used to denormalize the keypoints was changed, for instance now it simply multiply the normalized poses by the resize factor used to normalize them. This is why now it takes as argument also the resize factor instead of taking only the pose.
- *decode_pose_normalized_keypoints* is a function that, given a pose, reshapes the vector such that it can be used by *de_normalize_relative_keypoints*, then it call it, reshapes again the vector as it was before and finally translates the denormalized keypoints if necessary (it is necessary only if we desire to show side by side the generated pose and the real one). Also this function was changed such that it reshapes vectors that have inside the coordinates of the 9 keypoints.

All the other functions remained unchanged since they were not necessary or they did not depend from the number of keypoints

Also *pose_plot_lib.py* was adapted to the specific case but since it was exploited only after the inference, it will be explained better in the next section.
Regarding the model itself, i.e. the Generator and the Discriminator (used only in the GANs case) implemented inside the module *static_model_factory.py*, only the sizes of the output vector was changed since the Generator receives as input the audio features of training samples (not dependent from the number of keypoints in a pose) while the Discriminator receives as input poses that were previously adapted in the processing phase (explained in the next lines). The output of the Generator, as well as the one of the Discriminator, is a vector containing 64 poses

of the 9 keypoints each, so its dimensions needed to be adapted for the specific case.
Let's now talk about the most important files, i.e. *model.py* and *dataset.py*. In *model.py* the initialization of the models remained nearly unchanged except for the fact that poses were taken by different functions: *_get_training_keypoints* creates an array by using the constants defined in *consts.py* (BASE_KEYPOINT, RIGHT_BODY_KEYPOINTS, LEFT_BODY_KEYPOINTS,CENTRAL) for indicating what keypoints must be taken from the samples and in which order, while *keypoints_to_train* which is defined in *tf_layers.py* extract the keypoints selected by *_get_training_keypoints* and then creates a vector with all of them, so it was necessary to change the dimension of the latter.
During the training phase, it is loaded the Dataset file containing the all the training and the validation samples , and the *train.csv* file that indicates the position and the type of these samples in the Dataset as well as some other information as described at the end of the last section. Many of these information were not necessary this time but they were put to be sure that errors are not returned by the program by making this file very similar to the one of the original project.
From this csv file, 512 random validation samples were chosen for validating the Network then eventually checkpoints files are loaded to start the training from the point it was left, and finally the training loop is started.

- The first operation executed is the test of the Network by using the samples chosen before. The Loss, as well as the PCK, are computed with this inference process by calling the function *predict_df* that was rewritten for my project: this time it takes as argument also the entire Dataset, it loads the audio converted and not converted from the validation samples and then calls iteratively the function *predict_row* used to predict a single test sample, and finally it returns all the predicted and real keypoints (that are 512 x 64 x 2 x 9 where 512 is the number of test samples, 64 is the number of poses in each sample, 2 is the number of coordinates, i.e. x,y, and 9 are the keypoints), the losses of the test samples (512), all the converted and original audios (512) and all the resize factors computed.
  The function *predict_row* instead extract and preprocess the data of a sample from the Dataset by using all the functions needed inside *dataset.py* and then it uses them as inference for the Network. At the end of the inference, each single pose is denormalized by exploiting the resize factors computed by calling the function *_post_process_output*: this is the main change with respect the function of the original *speech2gesture*. Finally it returns all the denormalized keypoints (for ground truth and predicted poses), the losses and the resize factors computed to the function *predict_df*.

_post_process_output only calls the function *decode_pose_normalized_keypoints* described above for denormalizing the poses.
In all this process the shifting of the keypoints is not executed since all the plots and videos were created and saved externally by using the data returned. This avoided problems with *Google Colaboratory* that gets stuck when it plots the data.

- Then the PCK is computed: while the functions for computing it remained unchanged, it was passed a different number of keypoints this time.
- All the data returned by *predict_df* are saved in an npz files called *real_and_predicted_keypoints.npz* every 15 epochs inside folders that have as name the number of epoch from which they were saved. As said before, this data is used to make the plots and the videos of the data to check subjectively how the Network performs.
- Now the train starts: for each step of the epoch it is generated a sample with the same dimensions of the batch size (32) containing itself as many training samples extracted from the Dataset file as the number of the batch size. The samples are chosen in a random way and after they are loaded by the generator inside *dataset.py* then are used to train the Generator and the Discriminator in case GANs are used. Moreover, the Discriminator for each step uses also the fake poses coming from the Generator.

Finally, let's how the *dataset.py* module works and the changes made:

- In the case are used the validation or test samples, the first function called is the function *get_processor* that only loads in memory the functions *decode_pose_normalized_keypoints* used to denormalize the data, and *audio_pose_mel_spect* used to extract the data. In this way functions are initialized, loaded and ready for getting the data such can be processed. In this function no changes were made

  audio_pose_mel_spect is used to take and preprocess all the data from the dataset: this was completely changed since this time it must loads the data of the Dataset file by finding their positions inside the *train.csv* file. The positions (indexes) are extracted by *predict_df* in the case of validation / test data, otherwise are extracted by other functions inside the module.
  The audio of each sample is processed such that it match the size defined in *consts.py*, i.e. AUDIO_SHAPE: if the audio has less samples than AUDIO_SHAPE then zeros are added, if it has more samples than the last samples are removed. This was the best choice since by trying to remove or add samples using interpolating methods much more noise is added. If this

noise is added to samples that are already noizy because of the audio conversion process, then the samples become useless for the training purpose. The poses instead were firstly subject to the neck extraction process of *preprocess_to_relative* then to the normalization process of *normalize_relative_keypoints* defined above. At the end, the function returns the new data as well as the resize factors computed by *normalize_relative_keypoints*.

- In case training samples are used, it is necessary to create a generator that randomly takes as many samples as the number of batch size from the *train.csv* file and passes them to the function *audio_pose_mel_spect* for extracting the data from the Dataset file. The training generator is initialized by the function *load_train* inside *dataset.py* that exploits the *GeneratorEnquer* method of Keras and some other methods inside the module. For instance, the generator is set by calling the function *load_set* which extracts only the type of data (train/validation) that we want from *train.csv* and then calls *set_generator*. The latter creates batches iteratively by calling the method *generate_batch* and passes to the next one by using the *next* method implemented by the Keras decorator threadsafe_generator. Finally, *generate_batch* calls as many times as it is the size of the batch the function *audio_pose_mel_spect* to process each sample. All these methods were modified such that they also take as argument the Dataset file since it is required by *audio_pose_mel_spect*.

All the other functions remained unchanged and the training process was repeated for 300 epochs, 300 steps each, reaching a total number of 9000 iterations to complete the training process as it was done in the original *speech2gesture* project. The training was repeated 3 times as described in the 4 chapter.
The training can be launched by specifying if gans are used (1 yes and 0 no) with command –gans, the arbitraty name of the test with the command –name, the name of the model that we want to train with the command –arch_g or -ag, the output path with the command –output_path and finally the checkpoint file only if we want to restart the training from a specific point with the command –checkpoint.

3.7 Inference of the speech2gesture model

The inference of the model was done in two different ways and for each method a script was created.
The first methods is the testing inference where the data used comes directly from the test set while the second method is the audio inference where an audio of arbitrary length and without any ground truth data is provided.

For the first method, it was created the script *test_s2g.py* that has only a main function and requires some configuration info to be used: the model that is used (audio_to_pose or audio_to_pose_gans) by using the command -ag, a flag for telling if gans are used (0 no, 1 yes) with the command -gans, the output path of the test with the command –test_path, and finally the checkpoint file that is required since this time we are using a trained Network that can be loaded with the command –checkpoint. The main function only loads the Dataset file containing the test set, then it loads the Network and finally calls the method *test* defined inside the *models.py* module. *test* is a very simple function, indeed it takes 512 random test samples and it computes the mean and the standard deviation of the loss extracted thanks to *predict_df*, it does the same with the PCK using the function *compute_pck* defined inside *evaluation.py*, and finally also with the Jerk by using the function *compute_jerk* defined by me inside *models.py* (for more details about this index, check the chapter 4). All the operations are repeated by a personalized test sample that contains audios of some people mixed with poses of other people as explained again in 4): to mix them, the function *predict_df* was modified such that by taking a flag called *random_prediction* as argument, it can choose whether or not mixing the poses and the audios of the test samples.
Finally all the results are saved inside a hickle file.
For the second method, i.e. the one that does the inference using an audio of arbitrary length, the script *predict_audio.py* was created. Also this script contains a main function and to work it requires the path of the audio to be converted. To generalize this function, it was written such that it can do the inference of many arbitrary audios so in the path of original audios can be more than one.
It also requires the path for saving the converted audios such that both these paths can be passed to the method *audio_conversion* for converting all the audios. Both the original and the predicted audios are reshaped by adding zeros such that they can match a set of poses multiple of 64: by doing in this way, the model is inferred multiple times and each time it predicts 64 poses. The model is firstly loaded by using the checkpoint file, then it is called the function *predict_audio* defined inside *model.py*: this one passes to the model the padded converted audios and then denormalize the returned poses: this time the resize factor of each pose is not known in advance so a unique value is chosen in a way that the poses can all result clearly visible in the plots. *predict_audio.py* finally saves the predicted poses as well as the padded original and converted audios.

At the end of the process, it was created a script called *video_test.py* that plots all the results: the plots and the videos are created by using the same functions inside *speech2gesture*. Of course, the number and the type of the poses plotted were changed to match the specific project. Since the audios were padded were zeros, the poses generated at the end are still (the model can't extract any feature from the audio) and the video is stopped when some zeros are detected. If the data contains also the ground truth poses (i.e. the one saved during the training epochs), then *video_test.py* prints side by side poses as it is shown in 3.7, instead if there is no ground truth (as in the data saved by *predict_audio.py*) then only the predicted poses are shown as in 3.8.

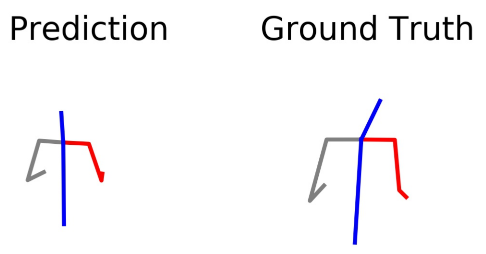


Figure 3.7: Plot of side by side poses obtained during the training of the model

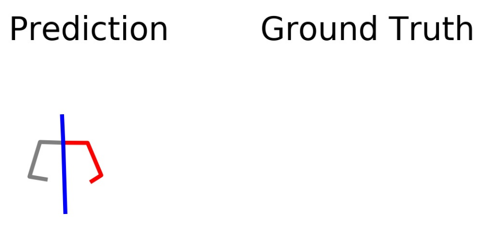


Figure 3.8: Plot of the poses obtained during the inference

3.8 2D to 3D pose mapping

Now it is time to map the poses from 2D to 3D. For this purpose, it was trained and inferred the project available online on Github called 2d_to_3d_human_pose_converter; in the main page of this project there are written some instruction to run the training of the model and what is necessary to do. As in this written in this page, the first step is to prepare the CMU Panoptic Dataset (Joo *et al.* (2015b)). To reach this goal, there must be used the toolbox PanopticStudio again available online on Github. Since the *2d_to_3d_human_pose_converter* model requires a pickle file for the training and since the pickle files can occupy no more than 2 GB, the dataset was filtered such that only the most important parts are taken:

- Range of Motion: contains data about range of motion of diverse subjects
- Haggling: contains data about a game where two sellers promote their own competitive products and a buyer selects one between them

- Dance: contains data about performances by professional dancers

Filtering the CMU Panoptic Dataset means changing the .sh files of the *PanopticStudio Toolbox* such that only the poses of these kind are downloaded, for instance only
hdPose3d_stage1_coco19 tar files of each video that need to be extracted before creating the pickle file. All this step was implemented in *Google Colaboratory.*
What must be noticed is that in the original *2d_to_3d_human_pose_converter* project, these files had different names so it was necessary to change a little bit of code. To finally create the dataset it must be used the script *generate_dataset.py* which loads all the needed
hdPose3d_stage1_coco19 poses inside a folder, it augments the data by rotating the 3D keypoints and by adding some noise, it normalizes them in the same way proposed for the *speech2gesture* method and it finally creates the pickle file.
The training process remained unchanged with respect to the original project, except for the fact that are loaded 2 more keypoints for the head and for the heap; the keypoints order finally is:

0. Neck
1. Nose (center of the head)
2. BodyCenter (center of hips)
3. Left Shoulder
4. Left Elbow
5. Left Wrist
9. Right Shoulder
10. Right Elbow
11. Right Wrist

Notice that there are some keypoints in (6,7,8) that are not needed: this is because the model used for this detection is different from the one used by Openpose and so also the order of poses is different.
The training process starts by launching the script *train.py*: the model of
2d_to_3d_human_pose_converter is defined in a class called *Net* which is explained in the chapter 2, it is defined a function for the evaluation called *validate* and finally the training function *train.* The latter initializes the model, initializes the data loader defined in the module *data_loader.py* which loads the 3D keypoints

written above and defines the input and the output of the model, and finally it trains the Network for 10 epochs where for each epoch there are as many steps as it is needed to load all the data with a batch size of 32. The Loss computed is the mean squared error (squared L2 norm) between the z coordinate generated by the Network from 2D ones and the ground truth z (this ones represents the coordinate missing to have 3D poses). At the end of the training (which is very fast in this case) a checkpoint file called *trained_net.pt* is generated.

The inference was done inside the script *test.py*. Here is loaded the npz file outputted by *predict_audio.py* containing the predicted poses, it is defined the output path, it loaded the trained network by loading the *trained_net.pt*, the keypoints of the *speech2gesture* model are mapped to the ones defined above thanks to a function called *s2g_to_panoptic*, they are loaded inside the Network and finally the predicted z is put together with the 2D coordinates to save finally all the 3D poses.
The result of this process is shown in figure 3.9 thanks to some functions that were already implemented in the original project; the saved poses are the ones that finally will be mapped to the Pepper robot.
The *test.py* script is optimized such that it can iteratively convert an entire set of movements: if we have for example 512 movements to convert (for example the ones obtained by inferring the test set into *speech2gesture*), it can extract all the 3D poses from them, saving everything to a unique npz file. The whole project differently from the many-to-one VC and *speech2gesture* was implemented using the Torch module (see the section 3.11) and it can be run both in *Google Colaboratory* as well as locally since it does not have any restriction given from the Cuda libraries.

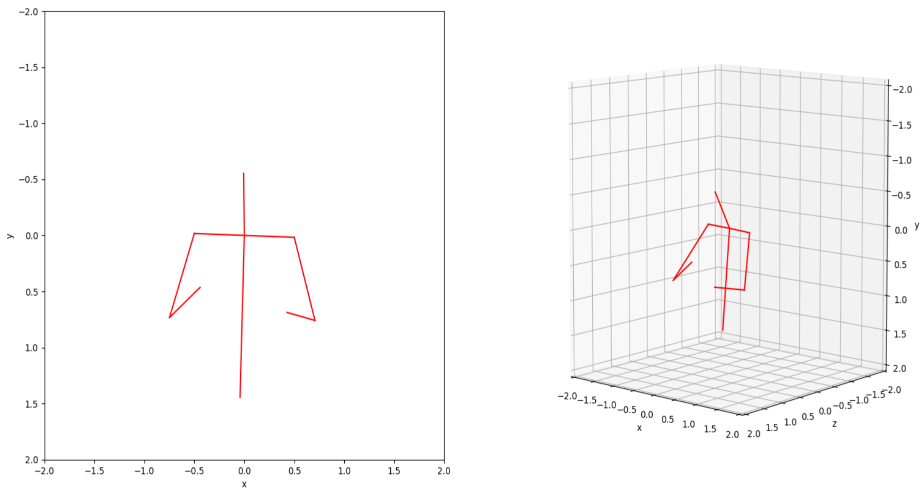


Figure 3.9: Plot of a 3D pose (at the right) obtained starting from 2D one (at the left)

3.9 Mapping to Pepper Joints configuration

Finally we arrived to the final part of this project, the mapping of the 3D poses to the Pepper robot. This is implemented inside the script *test_pepper.py* written using Python 2.7 since it is required by the *naoqi* libraries to work. This one have a *main2* function for mapping the poses, a *random_movement* function for generating random movements for Pepper (see the chapter 4, a function for creating a SSH client (needed only for putting recorded audios inside the memory of Pepper, operation that is not needed in the official evaluations) and some functions for computing the 3D angles: *rotmat2euler* and *rotation_matrix_from_vectors*. The main initializes the a qi session (necessary to use the methods for controlling the robot) by connecting to the IP of the robot (it must be inside the same network) and then sequentially calls the *main2* and *random_movement* functions which both require the session created to work.
The script *main2* initializes all the methods needed for controlling the robot, for instance his TTS (text to speech), the audio player (used only in unofficial tests), the motion service (see Softbank (2018) for more info) and some other methods

necessary for the robot to be ready. Then, this function extracts all the poses from the file saved by *test.py* of the project *2d_to_3d_human_pose_converter* and creates all the vectors needed: we have have one vector for the neck, one for the hip, 2 for the links that start from the neck and end to the shoulder (one for each side of the body), 2 for the links that start from the shoulder and go to the elbow and 2 for the links that start from the elbow and go to the wrist. From this set of vectors are computed all the possible angles of joints by using firstly the function *rotation_matrix_from_vectors* for returning the rotation matrix between each couple of vectors, and then the function *rotmat2euler* for returning the 3 Euler angles from the rotation matrices computed.
From all these angles are extracted for instance:

- Head Pitch
- Head Yaw
- Hip Roll
- Hip Pitch
- Left Shoulder Roll
- Left Shoulder Pitch
- Left Elbow Roll
- Left Elbow Yaw
- Right Shoulder Roll
- Right Shoulder Pitch
- Right Elbow Roll
- Right Elbow Yaw

These angles refer to the reference system of the robot (see Softbank (2018)) which is different from the reference system of the poses extracted from *2d_to_3d_human_pose_converter*, so it was taken into account.
All these angles are extracted by taking poses at 3 frames per second which means taking one pose every 5 ones since the poses of the model are taken at 15 fps. For every frame, the angles described above are extracted and a time vector is created to tell the robot at what time execute each angle.
Since the people talking in TED Talks are in average much taller than the robot

and since many times they are on a stage so they usually look down it was necessary to scale the Head Pitch angle by multiplying it by 0.3 (this value was chosen by me after some subjective evaluations).
Finally, two thread are started, one for the poses and one for the speech. To making smooth the first movement, the robot was programmed such that it goes to the starting position in a second before starting the set of poses. After a second also the thread of the speech starts and it was always used the Animated Speech method of the *naoqi* libraries. The function *main2* was also optimized in a way that it can run many movements iteratively one after the other after taking a little pause, this was used to make the robot speak and move for 4 consecutively sentences as described in chapter 4.

Let's now talk about the last function, i.e. *random_movement*: also this function initializes the same methods required by the robot plus the animated speech used in the next, and for each sentence that the robot must say, it generates some random movements defined as a set random angles. The type of angles computed are the same as the ones used by *main2* as well as the fps at which they are generated. At the start, the robot takes one second to go to a starting position that is similar to the ones of *main2*, then it starts to move by going to the goal poses and speak using the Animated Speech. All the angles were limited such that the generated poses are not so much different from the human ones and all the tuning values were chosen by subjectively check the quality of the generated movements.
random_movement has also a commented part that is the one used for the 4: in this part it only calls the animated speech method by passing to it all the sentences that it must say.

3.10 Computational Issues

In this section there will be explained all the computational issued addressed for this thesis. Since there were used many different models of Neural Networks created by different researchers with different libraries, it is easy to think that many compatibility problems may occur, as well as computational problems since large models like *speech2gesture* or the model of many-to-one Voice Conversion require a lot of resources. Indeed, it was discovered that running these ones relying only on the CPU is unfeasible so it is necessary to use something that is more powerful like GPUs. Using a GPU requires the installation of Nvidia drivers as well as CUDA and cuDNN Nvidia libraries necessary for some frameworks like *Tensorflow* (Abadi *et al.* (2015)) to be exploited for the creation and data management of DNNs.

Unfortunately, both these models were created using old libraries, moreover the model of many-to-one VC was created using Tensorpack (Wu *et al.* (2016)) that is an optimized library based on Tensorflow capable of running models 1.2 to 1.5 times faster as it is explained in the Github repository. Many of these frameworks loose their compatibility with CUDA $>= 10$ that are the only versions available for modern GPUs. For the *Tensorflow 2* there is an utility for migrating the old code to newer versions: this one is called tf_upgrade_v2 and automatically converts most of the *Tensorflow* code. This script was necessary for converting the *speech2gesture* model without putting so much effort even if some functions required to be adjusted manually or to be imported from the Tensorflow addons library. For *Tensorpack tf_upgrade_v2* was not available because the libraries itself are tought to be used only with *Tensorflow 1* and CUDA $<= 10$: for this reason and since there was not available a computer with powerful GPUs and CUDA 9, *Google Colaboratory* was exploited.
Google Colaboratory is a Google product that allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.
The latter has a lots of limits so it was used only for testing the whole setup (except the part used to control the robot Pepper) with a little Dataset composed by 49 videos, for training the model for many-to-one VC and exploiting it for converting all the audios, testing the trained model of *speech2gesture*.
The main *Google Colaboratory* limits that I had to deal with were:

- Limited amount of RAM memory ($\sim 13GB$)
- Limited amount of Disk memory ($\sim 40GB$ free if GPU is used, otherwise $\sim 70GB$)
- Slow input/output operations even if data is loaded inside the Server
- Limited amount of files that can be stored on Google Drive
- Limited access (12 hours a day if free account is used otherwise 24)
- Can't be established a TCP/IP connection
- Not always available
- Impossibility to install Python
- Impossibility to choose the GPU in the free version

There are also many other limits but ones these were the main and they were addressed as follows:

- Openpose was used only for preparing a small Dataset: even if it slow because of the bottleneck in the input/output operations and even if it cannot be displayed, it is possible to use it in case all the JSON files are deleted from *Google Drive* when a video finishes to be processed (only the pickle file that groups the poses has to be maintained), or in case all the JSON files are saved inside the Server memory and then exported. In the first case we have to delete them since only 400000 files can be saved inside *Google Drive*: 400000 seems to be a difficult limit to be exceeded but it must be considered that Openpose outputs a JSON file for each frame of the video processed.
- *speech2gesture* was modified such that it uses only one Dataset file instead of loading each time the poses: loading million of files is not possible because of the limit of Input/Output operations and because of the total number of data that can be stored on *Google Drive*.
- Also by taking this choice, it was possible to run *speech2gesture* only with small Datasets because of the limit on RAM and Disk memories: for this reason it was used only for the inference and the test phases while for the training phase it was used a Server available in University with Ubuntu 20 and CUDA 11.
- Since only $\sim$ 200 audios can be converted in one step because after that the RAM memory gets full, many accounts and other versions of the *extract_data_for_training.py* script were created: each version converts the audios following different orders so it is possible to convert more than $\sim$ 200 each time by using different accounts. Notice that this is possible because *Google Colaboratory* allows to access the same *Google Drive* memory at the same time by using different accounts; with more accounts it is possible to overcome the limit of 12 hours of usage.
- The part of *speech2gesture* that saves plots and videos was removed and all the data that can be plotted is exported and plotted offline. Moreover, *speech2gesture* that was originally written using Python 2.7 was converted to a version compatible with Python 3. By doing this, it is possible to run the project both on *Google Colaboratory* and the Server of University.
- Many modules work only with Python 3.6: at the moment *Google Colaboratory* works with Python 3.7 but fortunately there is still Python 3.6

available. To use it, I run the scripts and *pip* commands by writing something like "!python3.6 -m ...".

- Since it is not possible to use Python 2.7 and TCP/IP connection required for connecting the Pepper robot, the data needed is saved, downloaded and then used in scripts external to the server of *Google Colaboratory.*
- Even if it possible to choose the GPUs, the ones that are available are sufficient for running the models with the given configuration. In case of very long audios to be converted or in case a particular setup of Openpose is used (e.g. for detecting hands and face and for having more accuracy), this is no more true. Fortunately this does not happened since the audios were cut to have a have a maximum duration of 10 seconds and Openpose was used only for detecting the body.
- All the code was optimized such that it is possible to start each operation from the last thing that has to be processed: this is very important because *Google Colaboratory* can disconnect at any ti*extract_data_for_training.py*me if an account is used a lot and all the data processed can be completely lost. This is the main reason to why it was always preferred to save the data directly to *Google Drive* instead of saving it inside the server and then exporting it after a while.

Running all the project is really heavy from the computational point of view, especially if *Google Colaboratory* is used. It is recommended to use a very powerful machine that is compatible with CUDA 9.0 or to manually convert all the *Tensorpack* code of the many-to-one VC model such that can be used with newer CUDA versions.

3.11 Requirements

In the next is shown a list of all the requirements needed to run correctly the project. Notice that some components need to be installed with the version shown because it isn't tried to run the project with other versions. All these packages, except Python,Cuda (and eventually Cudnn) and Openpose, can be easily installed with *pip.* It must be noticed that some packages were already installed on *Google Colaboratory* so except for *speech2gesture* model, 2D to 3D poses conversion model, the scripts used to show the results of *speech2gesture* and the ones used to control the Pepper robot, there is not any proof that the other parts of the project can run out of *Google Colaboratory* with only these packages installed.

- Python 3.6 and Python 2.7 (needed only for controlling the Pepper robot), see Python
- Cuda 9.0 if the project runs on *Google Colaboratory*, otherwise Cuda >= 10.1
- Installation of Openpose and all its requirements (see Openpose)
- Tqdm
- Torch >= 1.1.0
- Pillow 6.1.0
- Google API
- Youtube Downloader
- Scene Detect
- Scipy 1.2.2
- Numpy 1.16.4
- Tensorpack 0.8.6
- Tensorflow-gpu 1.9.0
- Keras 2.2.4
- Librosa 0.6.2
- Numba 0.48
- Pydub
- Google Text To Speech
- Soundfile
- Hickle
- Pickle
- PyYAML
- Scikit Learn 0.20.3

- Resampy 0.2.1
- Pandas 0.24.2
- Matplotlib 2.2.4
- Tensorflow plot
- Tensorflow addons
- OpenCV
- Python-csv
- Ffmpeg
- NaoQi only for controlling the Pepper robot

To run the *speech2gesture* model and *Openpose* on an external server, it is also necessary to install nvidia-docker on this, create 2 containers from personalized images (Docker Openpose Image, while for tensorflow-gpu check the one that is compatible in Docker Tensorflow Image) each one with a shared folder between host and server, eventually install the other required packages to run *Openpose* or *speech2gesture* and finally control the containers through an SSH connection.

Chapter 4

Evaluation

In this chapter there will be shown all the methods that were applied to evaluate the proposed approach, as well as the objective and subjective results obtained. To do the evaluation, there were followed the suggested guidelines of the recent study of Wolfert *et al.* (2021), in particular:

- A user study was conducted for the subjective evaluation
- An objective evaluation was exploited by using standard metrics like Jerk, PCK, L1 regression
- More contrastive approaches were used in the user testing evaluation
- Direct measures on human-likeness (naturalness), fluency, appropriateness, timing, amount and coherence were evaluated
- Definition and design of the experimental setup and metrics for evaluating the co-speech gestures of a humanoid robot

4.1 Objective Evaluation

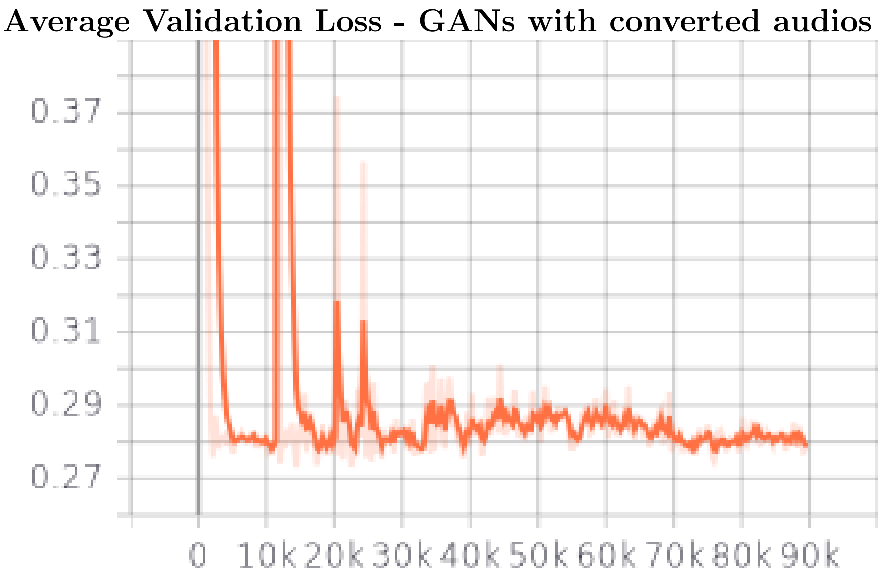
The objective evaluation was conducted by training and testing the *speech2gesture* model with and without using a Discriminator (i.e. whether or not GANs are used) and by using different types of data. For instance, the model was trained by using a GAN model for original audios, a GAN model for converted audios and finally a no-GAN model for converted audios.
Let's firstly see how the PCK index, i.e. the index which classifies a keypoint as correct if it falls inside a range $\alpha\ max(h, w)$ of the same ground truth keypoint (see 2), and how the *Loss*, which formula is defined in 2.6, perform in average during the training of the model.

As explained in chapter 2, for the PCK $\alpha = 0.2$ while the bounding box is represented by an ellipse formed by an axis parallel to x and another parallel to y of the images; the first represents the maximum variation of width of that keypoint in the ground truth video sample (which is taken at 15 fps and formed by 64 frames), while the second axis represents the maximum variation in height of the same keypoint in the same ground truth sample.
Regarding the *Loss* in equation 2.6, notice that in the no-GAN case it is represented by only the λL_{L1} component (2.7) since there is not a Discriminator; in the GAN cases λ was set to 1 as in the original *speech2gesture* project.
The models were trained in all cases for 300 epochs with 300 steps each, using a batch-size of 32 and by computing each time the average PCK and *Loss* on a set of 512 validation samples taken randomly from the Dataset. In the GANs case the Discriminator for each step learns two times by viewing one time the ground truth data (x 32 samples because of the batch size) and one time the fake data coming from the Generator (again x32 samples), while the Generator learns only one time by using the 32 data samples. In the no-GANs case the training of the Discriminator is simply skipped.
By inspecting the results shown below, clear differences can be noticed between the three cases:

- In the GANs model trained with converted audios (Image 4.1) the *Loss* converged very fast even if at the start the gradient seemed to diverge in some points: this behaviour can be justified by the fact that even if the Generator seems to learn something since the start, these poses appear to be not very similar to the ground truth from the Discriminator point of view. So the Loss goes up and down until the Generator leans how to create original data. The PCK index seems to slightly decrease a little bit but this does not necessary mean that the Network is performing worse: in fact, since the co-speech gestures are multimodal, the keypoints of these can fall distant from the ground truth keypoints without creating unnatural poses. With articulated and wide gestures the probability of having keypoints far away from the ground truth ones increase, that is a reason why PCK cannot be considered a good index for evaluating poses.

- In the GANs model trained with original audios (Image 4.2) it can be clearly noticed that the training is very unstable: the *Loss* continuously goes up and down showing that the gradient encounters a lot of difficulties before going to a point of convergence. This means that the Generator and the Discriminator can't find an equilibrium point and everytime the Generator seems to learn something, the Discriminator directly refuses its poses because they are too fake. This suggests that, by using original audios in a TED Dataset composed by a lot of different speakers, there can be too many audio features to be learnt by the Network clearly showing that either a very different model has to be used or an audio conversion must be executed in advance. As it will be shown in the next, the poses generated are very noisy even if in average they seem to fall often near the ground truth.

- The no-GANs model trained with converted audios (Image 4.3) performs as expected: the *Loss* always go down since there is not a Discriminator to recognize fake poses, and the PCK goes up since the Network guided by the *L1* Loss leans how to push poses near the ground truth. As we will see in the next, even if the Networks seems to work well theoretically, practically the movements are very stiff. Notice that the Average *Loss* is always lower than the other two methods, this because it is not considered the GANs term is not considered in the equation.



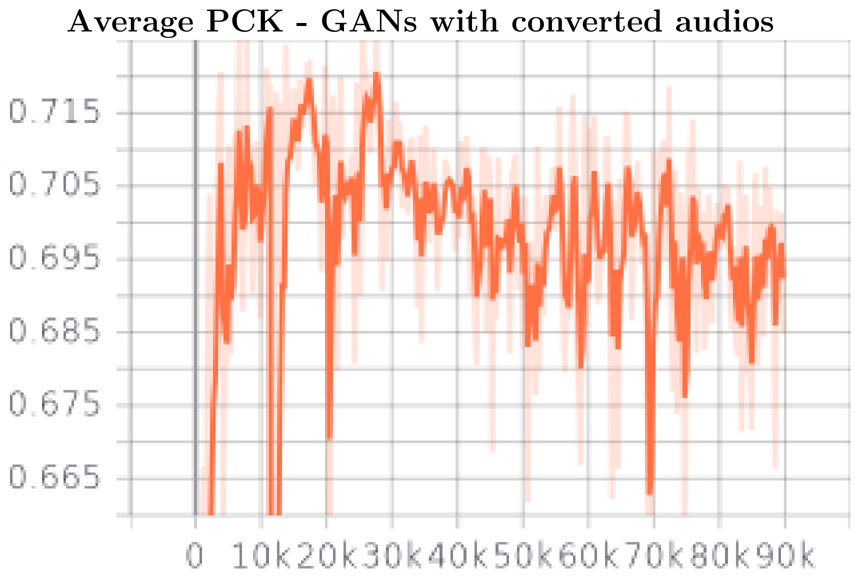
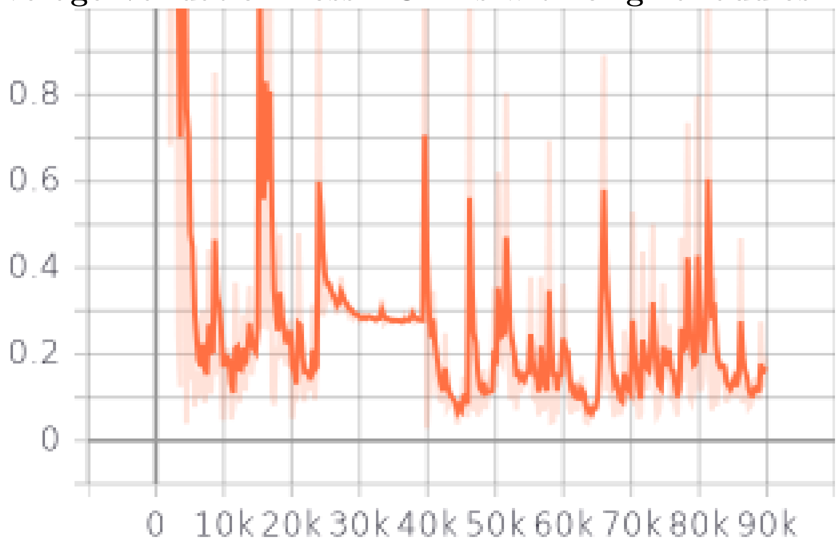


Figure 4.1: Average Validation Loss and PCK indexes evaluated for each training epoch in the GANs case with converted audios being used. On the x axis there are the step values (300 steps for each epoch) while on the y axis are shown the values of the indexes

Average Validation Loss - GANs with original audios



Average PCK - GANs with original audios

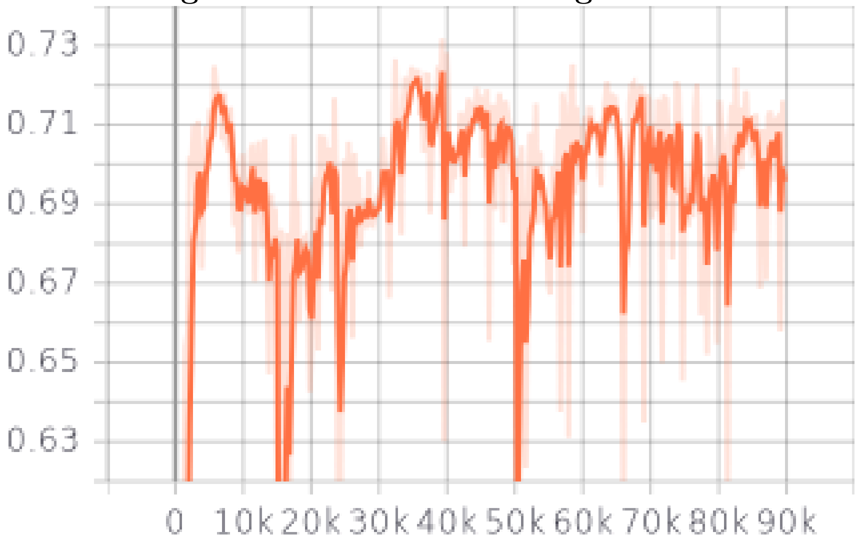
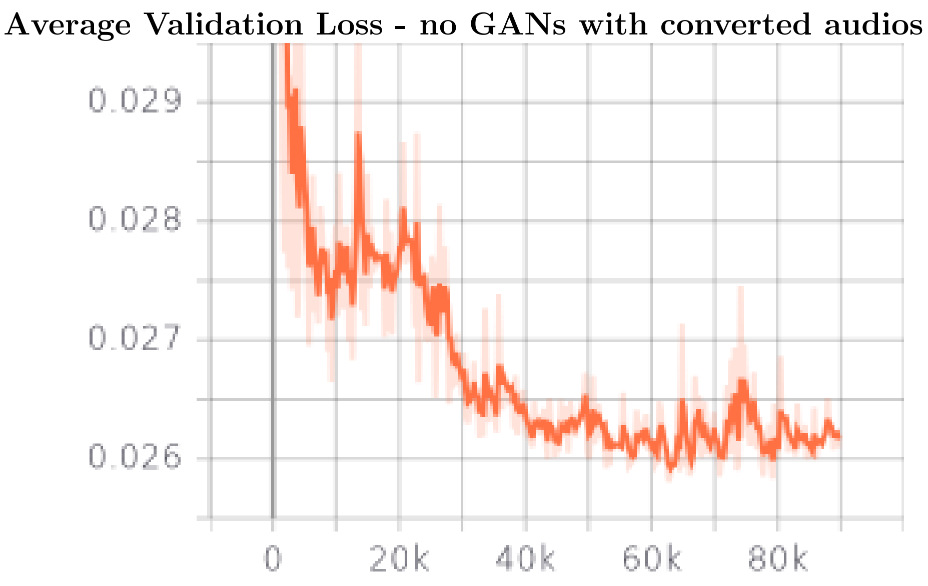


Figure 4.2: Average Validation Loss and PCK indexes evaluated for each training epoch in the GANs case with original audios being used



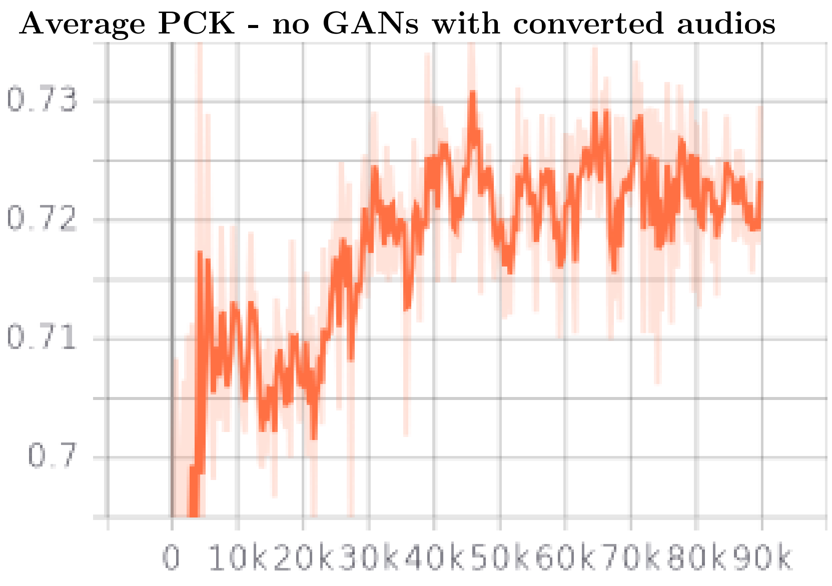


Figure 4.3: Average Validation Loss and PCK indexes evaluated for each training epoch in the no GANs case with converted audios.

Let's now move on and see how these three methods performed with the test set. Using this set it is possible to evaluate how good the Network is for generalizing its learning.
With respect to the original model of *speech2gesture*, here it was implemented another evaluating method called Jerk: this parameter (shown in 4.1) is widely used to compute the smoothness of movements and it is defined as the mean of the time derivatives of the norm of the accelerations. Since it requires three derivative to be computed, time T goes from the first to the fourth frame before the last frame, i.e. from 1 to 61 (64 - 3) in this case since each sample of the test set has 64 poses. Here the distance between 2 points is measured in pixels so the unit of measurement of the Jerk is $\frac{[pixels]x[frame]^3}{[seconds]^3}$.
For each test sample and for each method, it was computed the Average Jerk of the body, the Jerk of the two hands, the L1 Loss, the PCK, and finally it was computed the average of these indexes for all the test samples used (512).
Moreover, these indexes not only were computed for all the 3 methods using the test set, but also by considering test samples with audios desyncronized from poses for checking how much the poses generated by the model are coherent with the speech: to do this, I combined the audio of a sample used as test with the poses of another random test sample inside the test set, creating in this way 512 new test samples.
A set of Unpaired *T-tests* with 95% of significance ($alpha$=0.05) and two tails, were conducted to check whether or not some differences were significant. For using this kind of test, it was always assumed that the groups of samples are coming from a normal distribution, that they have the same variance (Unpaired), that are independent from each other and that the estimated value can be greater or less than a certain range of values (this is why it is called two tailed T-test).
T-test was also computed in the subjective evaluation, as it will be shown in the next section, and there were evaluated:

- If the Null Hypothesis H_0 of having relationship between two groups is accepted or rejected: if it is rejected then there can be a relationship between the two groups (i.e. they can came from the same population), if it is accepted then there can be a statistical difference with 95% of significance between the groups.
- The t-value which is a ratio between the difference between two groups and the difference within the groups. The larger the t-value, the more difference there is between groups.
- the p-value which is the probability that the difference between groups occurred by chance.
  The p-value range is from 0 to 1; in this case we must have $p{-}value <= 0.05$

(because of the 95% of significance) to have a statistical difference between means.

.

$$Jerk = \frac{1}{T}\sum_{t=1}^{T}\left\|\dot{accel_t}\right\| \tag{4.1}$$

To have comparable Loss values for the three different methods, in the Gans models it was not used the Discriminator for the test phase. As we can see from the table 4.1, even if it seems that there are not so much differences between the Loss and PCK parameters, this is instead not true for the Loss. Indeed, by computing the T-test between the different methods (table 4.3), it is possible to prove that there are always differences among these methods, except for the PCK parameter which seemed to be always similar and so to be not a good evaluating method.
What is interesting to notice is that GANs returned the highest average L1 loss: by analyzing the Jerk parameter, it is confirmed that having a higher L1 Loss does not necessary means that the Network performs worse, indeed in this Network the movements produced are much better than the other two since they are not too still (as in the no-GANs case), neither too shaky (as in the GANs with original audio case). This means that it is instead important to have poses that in average are close to the ground truth positions to obtain a lower L1 Loss. Since GANs produce much more natural movements, it may happen more often that the keypoints end in places far from the ground truth ones.
In the original project of *speech2gesture* the results obtained with PCK and L1 Loss were a lot more meaningful with respect the current project, but in that case there were used many more keypoints since there were detected also the hands and only the audio features coming from the same person. The latter can be an important difference with respect the proposed method, in fact even if the audios converted seem very similar among them, they are also more noisy and less human-like, so they can give a very different result if used for training.
For these reasons, this project was mainly based on the Jerk parameter for evaluating the three methods. As we can see from the images 4.4 and 4.5, even if it was confirmed by the T-test in the table 4.3 that the GANs method produces movements clearly different from the real ones, the latter performed way better than the other two methods. The case of the GANs trained with original audios showed to be the worse with very shaky movements, while the no-GANs method with converted audios shown to produce very stiff hand and body movements. For this reason, it was decided to use only the GAN method trained with converted audios among this three for the subjective evaluation.

Lastly, it must be noticed that all these methods performed nearly the same when the asyncronous test samples were used (see the tables 4.2 and 4.4): since also in this cases the poses were human-like and not completely random and since the L1 Loss and PCK parameters were nearly the same, this may means that none of these models is fine for producing movements syncronized with speech. But as said before, L1 Loss and PCK were not good parameters for the evaluation so to prove this fact objectively, other parameters must be used.

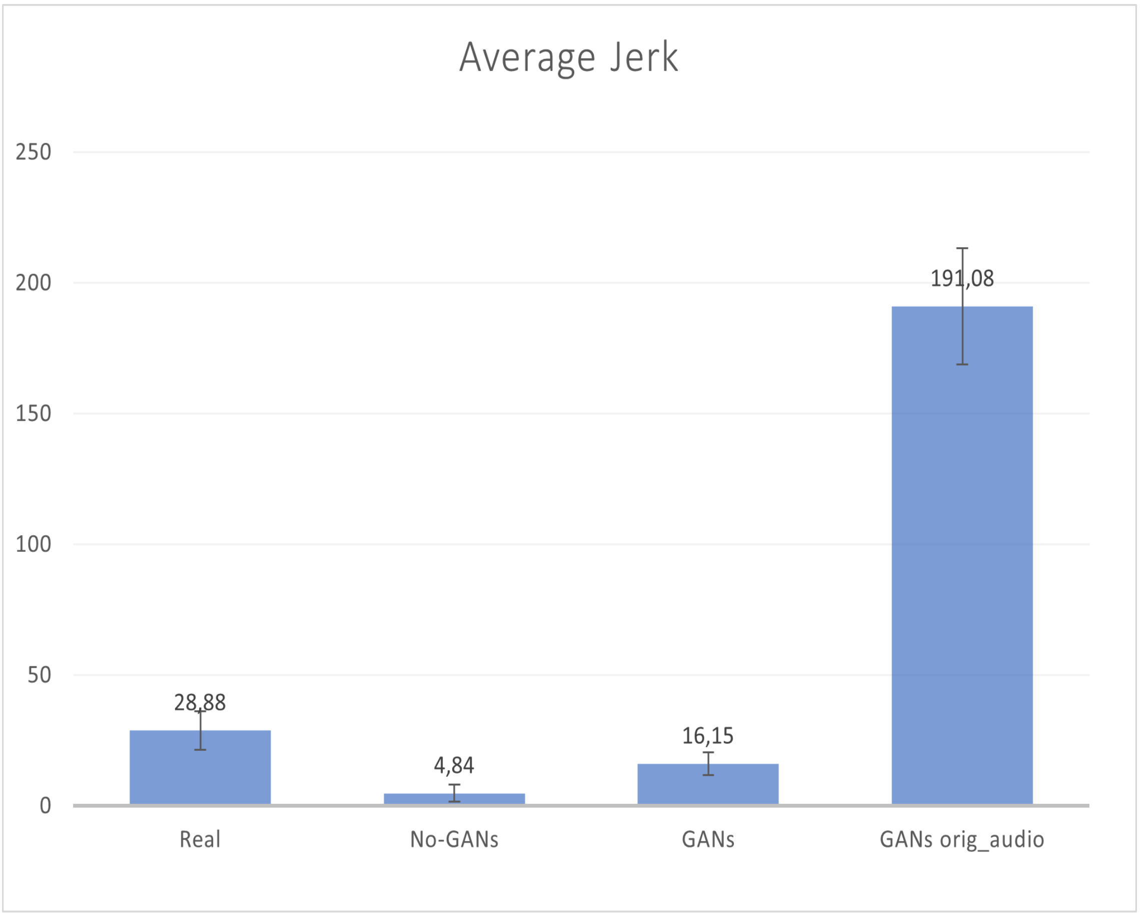


Figure 4.4: Histogram showing the Average Jerk values of the body computed using the 512 test samples by the GANs model trained with converted audios, the GANs model trained with original audios, and the no-GANs model trained with converted audios. Moreover, it is also shown the value of the Average Real Jerk (computed from the ground truth of test samples. The bars of the STD are also shown.)

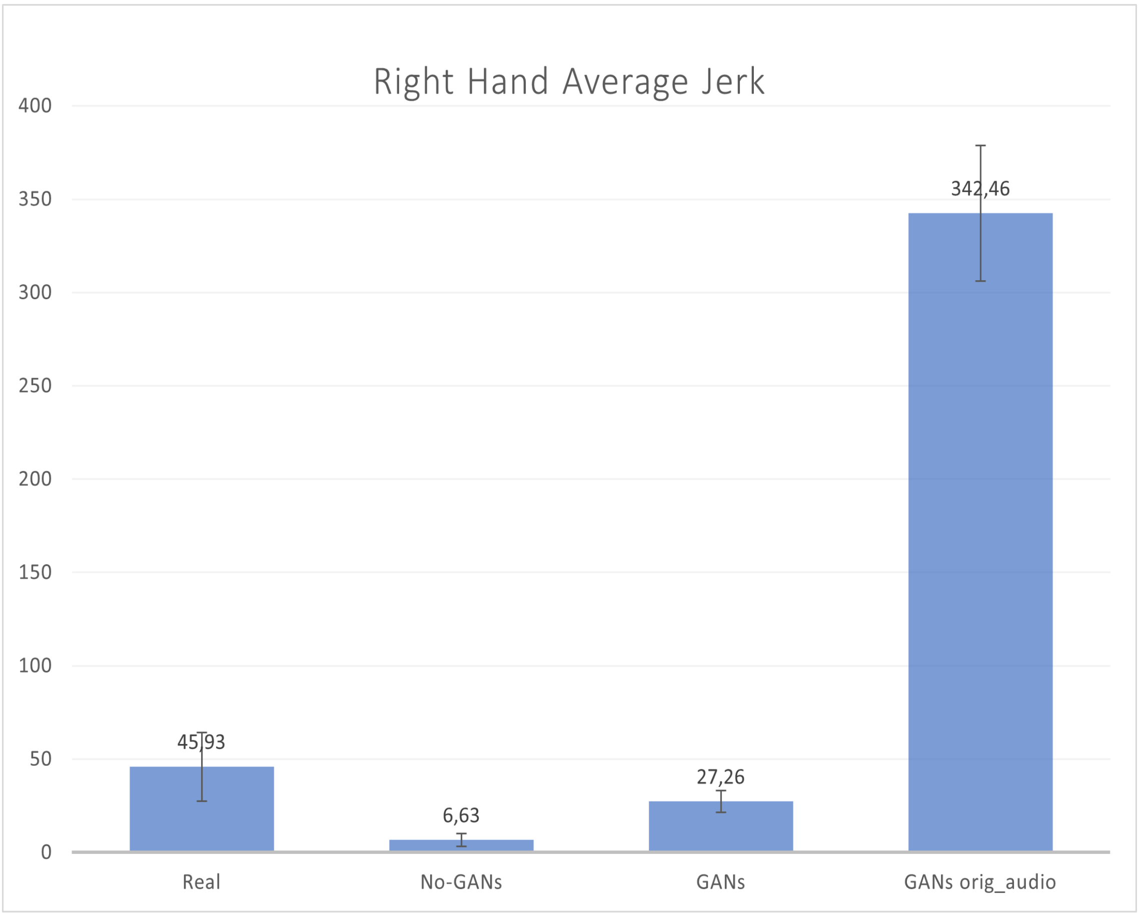


Figure 4.5: Histogram showing the Average Jerk values of the right hands computed using the 512 test samples by the GANs model trained with converted audios, the GANs model trained with original audios, and the no-GANs model trained with converted audios. Moreover, it is also shown the value of the Average Real Jerk (computed from the ground truth of test samples. The bars of the STD are also shown

	no_GANs	**GANs**	**Orig_Audio**
avg_loss	0,026±0,0055	0,03±0,007	0,028±0,006
avg_pck	0,74±0,145	0,7±0,145	0,72±0,15
avg_real_jerk	28,88±7,385	28,88±7,385	28,88±7,385
avg_pred_jerk	4,84±3,215	16,15±4,35	191,08±22,29
avg_real_rjerk	45,93±18,35	45,93±18,35	45,93±18,35
avg_pred_rjerk	6,63±3,525	27,26±5,77	342,46±36,21

Table 4.1: in this table are shown the Average values and the Standard Deviations of all the parameters computed using the 512 random test set samples on the three different models trained.
The models trained are: No GANs with converted audios, GANs with converted audios, GANs with original audio.
The parameters computed are: average Loss, average PCK, average Jerk of the body of the ground truth poses, average Jerk of the right hand of the ground predicted poses.

	no_GANs	**GANs**	**Orig_Audio**
avg_rand_loss	0,027±0,0055	0,031±0,0065	0,028±0,006
avg_rand_pck	0,73±0,145	0,69±0,145	0,73±0,15
avg_rand_real_jerk	29,14±7,43	29,63±7,295	30,16±8,075
avg_rand_pred_jerk	5,06±3,93	16,61±4,69	193,51±22,38
avg_rand_real_rjerk	44,26±15,05	46,43±16,275	46,2±17,16
avg_rand_pred_rjerk	6,9±4,28	27,86±6,21	346,63±34,385

Table 4.2: in this table are shown the Average values and the Standard Deviations of all the parameters computed using the 512 random test set samples which have inside the speech audio combined with random poses of other test samples on the three different models trained. The models trained as well as the parameters are the same of the table 4.1

	no-GANs, GANs	no-GANs, Orig_Audio	GANs, Orig_Audio
Loss	YES - t=4,74 - p<0.0001	YES - t=2,11 - p=0,04	YES - t=2,81 - p=0,01
Jerk	YES - t=23,63 - p<0,0001	YES - t=93,46 - p<0,0001	YES - t=87,05 - p<,0001
Rhand_Jerk	YES - t=34,48 - p<0,0001	YES - t=104,34 - p<0,0001	YES - t=97,17 - p<0,0001
PCK	NO - t=1,94 - p=0,05	NO - t=0,86 - p=0,39	NO - t=1,05 - p=0,29

Table 4.3: in this table are shown all the two-tailored and unpaired T-tests with 95% of significance between the three methods, i.e. no-GANs with converted audio, GANs with converted audios and GANs with original audios, evaluated with each of the four parameters, i.e. Loss, Jerk, Jerk of the Right hand, PCK, using the 512 test samples. For each value of the table there are shown the results of the T-test: YES means that the null hypothesis (H_0) of having a relationship between two group is accepted, i.e. data is not correlated, No means that H_0 is rejected i.e. data may be correlated, t is the T-value, p is the P-value.

	GANs, GANs_rand	no-GANs, no-GANs_rand	Orig_Audio, Orig_Audio_rand
Loss	NO - t=0,7 - p=0,45	NO - t=0,63 - p=0,53	NO - t=0,15 - p=0,88
Jerk	NO - t=0,81 - p=0,42	NO - t=0,50 - p=0,62	NO - t=0,87 - p=0,39
Rhand_Jerk	NO - t=0,80 - p=0,42	NO - t=0,56 - p=0,58	NO - t=0,9 - p=0,35
PCK	NO - t=0.85 - p=0,39	NO - t=0,49 - p=0,62	NO - t=0,53 - p=0,60

Table 4.4: in this table are shown all the two-tailored and unpaired T-tests with 95% of significance between each of the three methods, i.e. no-GANs with converted audio, GANs with converted audios and GANs with original audios, evaluated with each of the four parameters using one time the 512 test samples and the other time the 512 test set samples which have inside the speech audio combined with random poses of other test samples.

4.2 Subjective Evaluation

Let's now talk about the subjective evaluation. Even if this kind of evaluation is difficult to be reproduced, it shows results that are much more meaningful since the evaluation is directly conducted by the people. To evaluate this method there were exploited 41 people with Indian culture and 41 people with different cultures were recruited from the Amazon Mechanical Turk system. Each of them had to complete a survey (created on JotForm) on which there were proposed 3 different videos showing 3 different methods used and some questions. The methods proposed were:

- the GAN's method: to use it, it was inferred the *speech2gesture* model trained with converted audios. This model was also tried with human audios and produced better results than audios coming from text-to-speech methods (maybe because the human features are more similar to the ones used for the training, moreover the speech of the humans is more natural and articulated than conventional TTS), but since it was necessary to compare the result coming from here to the other methods, it was used the same TTS of the Pepper Robot as the one used for the other two methods. It was necessary to record the voice of the Pepper in advance and then use the recorded voice to infer the Network. The problem of this approach is that in any case it must be used the text-to-speech of Pepper and not the recorded audio to compare this method with the others, but by doing this, it is impossible to obtain a perfectly syncronized audios. So this fact has to be taken into account. Another fact that must be taken in consideration is that for the mapping the poses were downsampled to 3 fps because too fast movements can't be reproduced by the Robot. This can be seen as a tuning value, having many frames means that the Robot can't reach in time the defined positions, resulting in very still movements or in important delays since it can't process all of them, having too few frames means that the Robot has to interpolate too much between frames so the movements can be seen very different from the ones outputted by the model. Changing this value can change in a significant way the results.

- the Random method: for this method random gestures were generated by Pepper while it was talking. Instead of generating completely random gestures, there were instead generated gestures that stay in a range common to the human ones and their speed it was adjusted such that the movements seem smooth as the humans ones. In any case, they are completely asyncronous with the speech and not related to it. The motions starts with the arms flexed and with arms positioned near the chest, this position it can be seen as the average position that humans make while they talk.

- the Animated Speech method: this is the default method used by Pepper when it talks. It is a rule-base method so the movements are very similar to the human ones. Moreover, they seems to be also dependent from the speech context and articulation, but, as it is possible to imagine, they are not so many neither they are not always syncronous with the speech.

All these methods as mentioned before, were all tested using the same TTS of Pepper because different voices can have very different effects on people, and because the TTS of Pepper is the only one that can be used with the Animated Speech method.
There were recorded totally 6 videos: a video for each method (3) prepared for the Indian culture (Random, GANs, Animated Speech), and a video for each method prepared for all the other people (Random, GANs, Animated Speech). The difference consists on the sentences pronounced by Pepper, for instance it was chosen to use sentences more related to the Indian culture for the Indian people and more general sentences to the others: this because there is a belief that sentences which depend on culture can capture more the attention of the listener and they seem more natural to its point of view. Moreover, since the GANs model was trained with data of Indian people, there can be differences in the gestures that it can produce from sentences that depend on the Indian culture: if this is true, the gestures generated will be more natural and more related to the speech.

The Indian sentences were:

1. I know that traditional Indian men's clothes include dhoti and kurta, which are paired with a Topi, while Sarees and Salwar Kameez are two of the most famous traditional indian women's clothes.

2. Diwali is the festival of lights that celebrates new beginnings and the triumph of good over evil and light over darkness. I know that houses are decorated with candles and colourful lights.

3. Somebody told me that the music of India includes multiple varieties of classical music, folk music, filmi, Indian rock, and I know that the music of India is one of the oldest unspoken musical traditions in the world. Indian music is great!

4. I know that typical breakfast in India varies depending on region. It could comprise of parathas or chapattis and a vegetable dish eaten lightly, or it might include rotli, dosas or idlis, and different dips and chutneys, as well as spiced potatoes.

while the sentences used for all the other cultures were:

1. Playing an instrument is a great way to exercise your body and your mind. It is also a wonderful way to do something fun with other people! I think that, if you know how to play an instrument, you are very lucky and you should continue practicing it as much as you can.

2. The things we do every day sometimes may look all the same, but every day is different. I am really enthusiastic to start this day!

3. Somebody told me that the fundamentals of baseball involve throwing the ball, hitting the ball, and catching the ball. In other words, it is a game played with a bat, ball and glove.

4. I am told that the Golden Age of Hollywood started with the silent movie era. The first major silent movie was called the 'Birth of a Nation, and two of the most popular movies of the Golden Age of Hollywood are Gone with the Wind and Casablanca.

These sentences were not chosen randomly, in fact, the first sentences of both kinds are closely related to iconic gestures, the second are related to metaphoric ones, the third on the beat ones and the last to iconic gestures. Moreover, the Indian sentences were directly extracted from Caresses (Menicatti *et al.* (2018)) that is a system capable of generating cultural dependent sentences.

For each of the videos that the Workers had to see, they had also to reply to some questions closely related to some aspects of the gestures, for instance:

- Do the gestures interpret the verbal information correctly? - Related to the coherence with speech
- Did you consider the observed shapes shown at different speech segments appropriate? - Related with the appropriateness
- Was the movement fluid? - Related with the fluency
- Was the speed and the timing of the represented content appropriate? - Related with timing
- Was the number of movements sufficient? - Related to the amount of gesticulation
- How would you rate the overall robot's talking gestures? (unnatural - humanlike) - Related to the naturalness

The latter questions was considered the most important one since it depends from all the other questions and it is the one related to what people globally perceive from the robot; moreover, since the GANs method is trained using data of real people, it is expected to be more realistic and natural than at least the Random one. It may not be expected to be more realistic than the Animated Speech since the latter was created by human annotations, but it may be expected to

produce more varied movements since the gestures of Animated Speech are very limited.
To push the Workers to think more on the last question and to have a direct feedback to them, it was mandatory for them to reply to an open question related to the Naturalness, for instance: ”Why did you choose to give that vote to the last questions?”.
This question also suggest that all the others were not open questions but instead they where questions on which Workers had to reply with a vote from 0 to 10; by doing this, it is simpler to evaluate numerically the results. All were mandatory.
Finally, it was asked the age, the gender and the culture of the person before the start and it was necessary for them to complete the survey to get a code at the end: this code called also Survey code, was necessary to know whether or not they completed the test.
In any case, it was not possible to check if they saw all the videos and they did the experiment with commitment, but from the open questions it was possible to see who replied with coherence and who not: the ones that wrote random words or words not related to the survey were rejected.
To choose the workers adequately, it was filtered the country of origin and it was checked what was written in the ”culture” field of the survey. By doing this, 41 surveys were collected from each group and the average results of the people with Indian culture are shown in 4.6 while the results coming from people of all the other cultures are shown in 4.7. These results are also reported numerically in the tables 4.5 and 4.6.

The first thing that we can notice from these results is that the Animated Speech always performed better than the other two methods while no significant differences were found between Random and GANs on both the groups of people (see table 4.7 and 4.8). What is interesting is the fact that even if no significant differences were found between GANs and Animated Speech in the Indian test, they were instead found in the General test; moreover, the parameter that changes more in GANs is the Naturalness that is shown on 4.8. As it can be noticed, in this figure is shown that the Naturalness of the GANs method tested with Indian people changes in a significant way with respect to the GANs method tested with the people of all the other cultures. This fact is confirmed by the table 4.9 and suggests that the proposed GANs method is the only one that have a cultural dependence.
Another confirmation to this fact is that in table 4.7 it can be noticed that there is a significant difference between the Naturalness of the Random method and the one of the Animated Speech so even if people didn’t notice so much the difference of this parameter between GANs and Random, they still prefer the GANs one.
By analyzing the replies of Indian people to the open question proposed, some of

them said that in the GANs method there as a lack on movement, someone else said that gestures were not enough to represent the speech and finally someone noticed that gestures were not syncronized. These facts were also confirmed by the people of the other cultures, but these ones were more affected by the fact that the robot lacks in movements, suggesting that the in the Indian Culture people move less the arms when they talk. It must be noticed that many of the people which do not belong to the Indian Culture and participated to the General test were American so the test revealed to be more a comparison between American and Indian culture instead of being a test among Indian and all the other cultures.
It can be useful to notice also that someone that noticed a better voice in the Animated Speech suggesting that sometimes the voice is perceived differently if the robot uses different gestures. Finally, the lower standard deviation of the Animated Speech confirmed that people have less doubts on preferring this method.

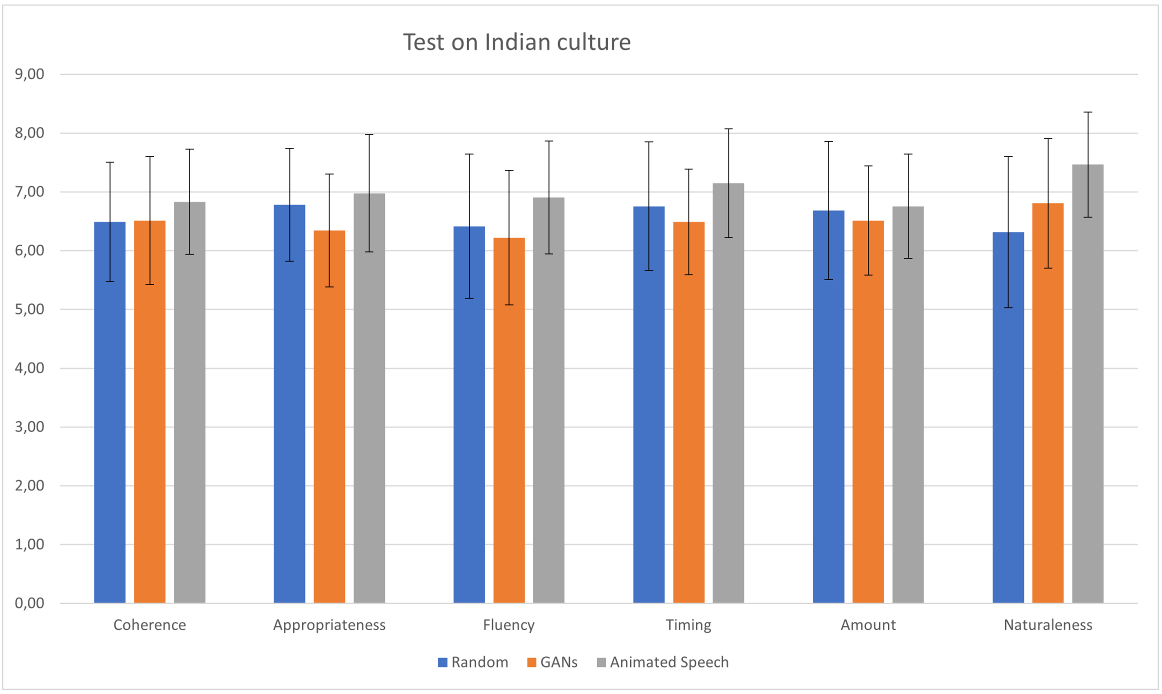


Figure 4.6: Histogram showing how the Animated Speech, the GANs and the Random methods were evaluated by the people of the Indian Culture). Also the STD bars are shown.

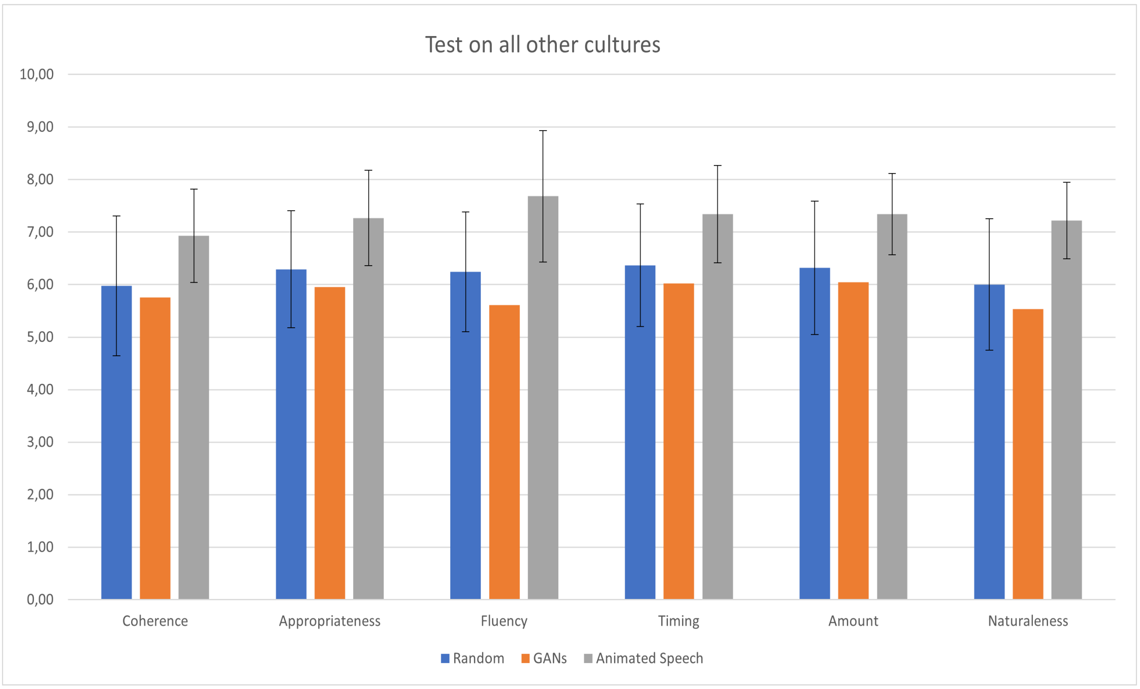


Figure 4.7: Histogram showing how the Animated Speech, the GANs and the Random methods were evaluated by the people of all the Cultures except for the Indian one). Also the STD bars are shown.

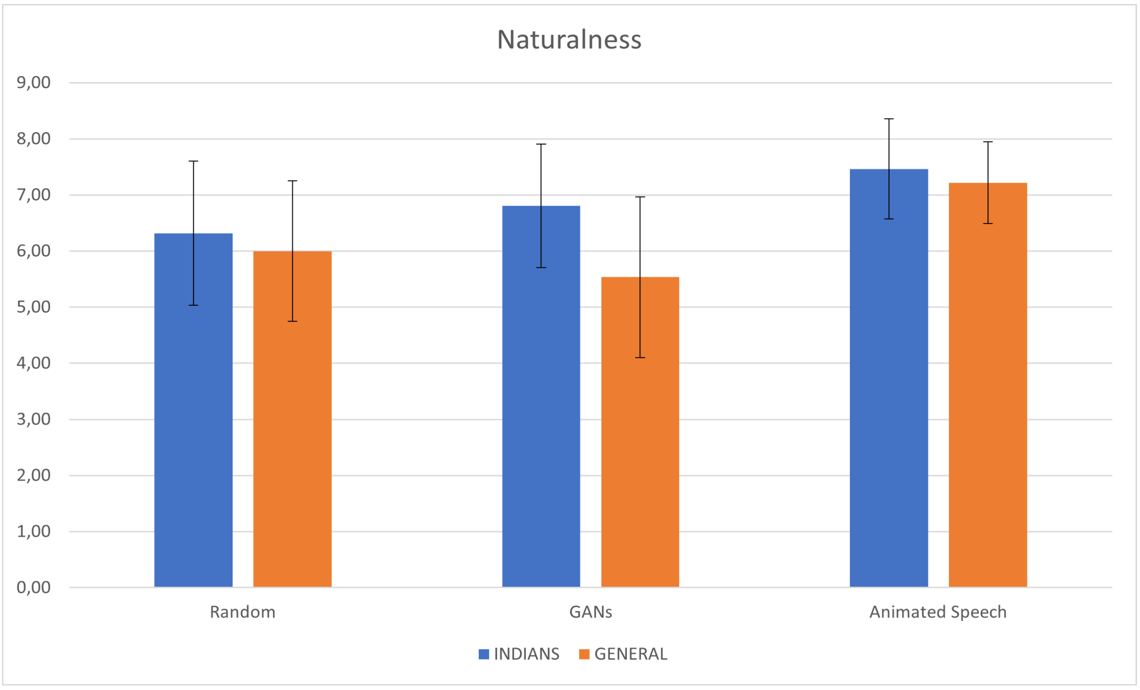


Figure 4.8: Histogram showing the evaluation differences on the Naturalness parameter by the people of Indian Culture and people of different cultures. Also the STD bars are shown.

INDIANS	**Mean**					
	Coherence	*Appropriateness*	*Fluency*	*Timing*	*Amount*	*Naturalness*
Random	6,49	6,78	6,41	6,76	6,68	6,32
GANs	6,51	6,34	6,22	6,49	6,51	6,80
Animated Speech	6,83	6,98	6,90	7,15	6,76	7,46

INDIANS	**STD**					
	Coherence	*Appropriateness*	*Fluency*	*Timing*	*Amount*	*Naturalness*
Random	2,03	1,92	2,46	2,19	2,35	2,57
GANs	2,18	1,92	2,29	1,79	1,86	2,20
Animated Speech	1,79	1,99	1,92	1,85	1,77	1,79

Table 4.5: in this table are shown all the Means and Standard Deviations of the indexes evaluated during the Subjective Indian test.

GENERAL	Mean					
	Coherence	*Appropriateness*	*Fluency*	*Timing*	*Amount*	*Naturalness*
Random	5,98	6,29	6,24	6,37	6,32	6,00
GANs	5,76	5,95	5,61	6,02	6,05	5,54
Animated Speech	6,93	7,27	7,68	7,34	7,34	7,22
GENERAL	**STD**					
	Coherence	*Appropriateness*	*Fluency*	*Timing*	*Amount*	*Naturalness*
Random	2,66	2,23	2,28	2,33	2,54	2,50
GANs	2,45	2,53	2,23	2,23	2,57	2,86
Animated Speech	1,78	1,82	2,50	1,85	1,54	1,46

Table 4.6: in this table are shown all the Means and Standard Deviations of the indexes evaluated during the Subjective test of all the cultures except the Indian one.

T-test INDIANS	Ind R, Ind G	Ind R, Ind A	Ind G, Ind A
Coherence	NO - t=-0,05 - p=0,96	NO - t=-0,81 - p=0,42	NO - t=-0,72 - p=0,47
Appropriateness	NO - t=1,12 - p=0,27	NO - t=-0,45 - p=0,65	NO - t=-1,47 - p=0,15
Fluency	NO - t=0,37 - p=0,71	NO - t=-1,00 - p=0,32	NO - t=-1,47 - p=0,15
Timing	NO - t=0,61 - p=0,55	NO - t=-0,88 - p=0,39	NO - t=-1,64 - p=0,11
Amount	NO - t=0,36 - p=0,72	NO - t=-0,16 - p=0,87	NO - t=0,61 - p=0,54
Naturalness	NO - t=-0,92 - p=0,36	YES - t=-2,34 - p=0,02	NO - t=-1,48 - p=0,14

Table 4.7: in this table are shown all the two-tailored and unpaired T-tests with 95% of significance among the three different methods used: Random gestures (R), Animated Speech (A), GANs (G).

T-test GEN	Gen R, Gen G	Gen R, Gen A	Gen G, Gen A
Coherence	NO - t=0,39 - p=0,70	NO - t=-1,90 - p=0,06	YES - t=-2,48 - p=0,02
Appropriateness	NO - t=0,65 - p=0,52	YES - t=-2,17 - p=0,03	YES - t=-2,71 - p=0,01
Fluency	NO - t=1,27 - p=0,21	YES - t=-2,72 - p=0,01	YES - t=-3,95 - p=0,0001
Timing	NO - t=0,68 - p=0,50	YES - t=-2,10 - p=0,04	YES - t=-2,91 - t=0,001
Amount	NO - t=0,48 - p=0,64	YES - t=-2,20 - p=0,03	YES - t=-2,20 - p=0,03
Naturalness	NO - t=0,78 - p=0,44	YES - t=-2,70 - p=0,01	YES - t=-3,35 - p=0,001

Table 4.8: in this table are shown all the two-tailored and unpaired T-tests with 95% of significance between each the three different methods used in the test that takes into account all the cultures except the Indian one (General test): Random gestures (R), Animated Speech (A), GANs (G).

T-test IND-GEN	**Ind R, Gen R**	**Ind G, Gen G**	**Ind A, Gen A**
Coherence	NO - t=0,98 - p=0,33	NO - t=1,48 - p=0,14	NO - t=0,25 - p=0,81
Appropriateness	NO - t=1,06 - p=0,29	NO - t=0,79 - p=0,43	NO - t=-0,70 - p=0,49
Fluency	NO - t=0,33 - p=0,75	NO - t=1,22 - p=0,23	NO - t=-1,58 - p=0,12
Timing	NO - t=0,78 - p=0,44	NO - t=1,03 - p=0,30	NO - t=-0,48 - p=0,63
Amount	NO - t=0,68 - p=0,50	NO - t=0,94 - p=0,35	NO - t=-1,60 - p=0,11
Naturalness	NO - t=0,57 - p=0,57	YES - t=2,25 - p=0,03	NO - t=0,68 - p=0,50

Table 4.9: in this table are shown all the two-tailored and unpaired T-tests with 95% of significance between each the three different methods used in the Indian and General test: Random gestures (R), Animated Speech (A), GANs (G). If there is a statistical difference, for example in the case of the GANs Naturalness parameter, there is also a difference between cultures in perceiving that method.

Chapter 5

Conclusions

This is the last chapter of this thesis and here there will be written a small summary of what is obtained, how it can be improved and what are the current limits. Starting from the results obtained, it should be noticed that even if some elements were proven, like the cultural dependence of gestures, the goodness of the GANs method over the no-GANs one in creating more natural gestures (even if they are as much as natural as the real ones), the very positive effect of the audio conversion for the training of the Network, ecc.; some others were not and performed worse than expected. One of these, maybe the most important one, is the non-significant difference among the votes of people evaluating generated gestures and random ones. This means that this method must be improved by a lot to have results similar or better to rule-based methods like the Animated Speech of the Pepper Robot. Another element that was not demonstrated is the significant difference of the PCK index among the different methods, indeed it seemed to be a completely useless index, and the importance of the L1 Loss. This suggests that other objective indexes must be used to evaluate this approach, moreover for the point of view of the culture whose difference was only proved by a subjective evaluation. But there are some considerations that have to be taken into account, let me show some.

Firstly, it is important to say that the GANs method was not tested in its best setup: in fact, the model of many-to-one Voice Conversion can be trained more to have less noise in the conversion stage, and getting audios with much less noise can make a lot of difference in the results. Maybe, by only doing this change, there can be obtained also more significant results with the PCK index, overall lower L1 Loss or also slightly higher Jerk by having smoother motions. Moreover, in the subjective evaluation, the GANs method was tested using a recorded audio of the voice of the Robot and this fact gives two main disadvantages: firstly the audio features are very different from the human speech, indeed sometimes the Robot mispronounces some words and its voice is less articulated than the human

one, secondly the recorded audio was not the one played during the speech and this creates asyncronous movements.
To solve the first problem there must be used a clear audio file with a human voice for the inference of the model, or as an alternative a better TTS system that avoids mispronounciatons or that articulates more the speech; to solve the second problem it is instead necessary to play the same audio used for the inference of the training.
By doing some other unofficial experiments which were not reported in this project, it was noticed that some audios related to TTS sometimes creates very unnatural movements in the GANs model and even more in the no-GANs one, and that the movements created by this method are slower than the movements created by using a human speech, suggesting that the solution proposed above can create better results.
Of course, there can't be expected results that are much better because in any case we are relying only on converted audio features so the context of the words is completely lost as well as the human expressiveness during the audio conversion stage. What is instead maintained is the articulation of the human voice which again helped to obtain better results in the unofficial results, and it was noticed that the element that the latter changes more is the rythm of the movements, confirming that the audio features affect more the *beat* gestures. To create more of the other kinds of gestures then some other approaches must be tried, one of them can be the *Multimodal* approach whose example is shown in the chapter 2. By taking into account the audio and the text features, much better results can be obtained.
Another element that can be improved is the syncronization between speech and co-speech gestures: with the proposed method the Network learns it without specific rules, so it is very important to put samples with audios and gestures perfectly syncronized. This may happen with the data created for the original project of *speech2gesture*, but instead it doesn't happen with the proposed method since all the audios have different number of samples and for training the model, there must be removed some of them or added some zeros such that they can have the same number of samples.
The problem of the *Multimodal* approach is that again the map between the speech and the gestures is learned by the Network and of course the result will depend from the Dataset used: even if the TED videos show people talking in a natural way, there must be considered that these ones are speaking in an unnatural environment, indeed they are talking in conferences and with unknown people, so there can be a lot of difference among this type of data and data acquired for example in some other natural contexts; moreover, the larger part of the gestures used in these setups are the *beat* ones while gestures of other kinds like the *iconic* that are used for illustrating proprieties of an element or *deictic*

for pointing objects, are much less used so the gestures of these types are very difficult to be learned by the Neural Network, even if text features are used.
An idea for getting around this problem is to create an hybrid approach between the end-to-end one and the rule-based one: by using rule-based gestures for some specific words or contexts, and end-to-end ones for all the others, it may be possible to create a much better setup for the co-speech gestures generation. Of course both these methods must depend from the culture to obtain better results, element that is very meaningful and whose importance was proven by a subjective analysis in this thesis.
Of course this element needs more studies to be proved: for example it can be meaningful to compare the differences between GANs methods trained with different cultures. Indeed, there is no proof, except for the Naturalness index, that the model recognizes differences among data of different cultures. So, many other indexes must be analyzed, in addition to the objective ones; indeed the cultural difference was not evaluated by any objective index.
This method has also many limits, one of them is the computational one. Preparing the data, training the models, mapping the poses in 3D and reproducing the poses on the Robot was a very long and challenging task, also from the point of view of finding a setup that is able to run all this processes at once. A purpose for the future work is to create a completely new model that maintains many of the concepts of this one but that instead does all the operations at the same time: for example, what will happens if Phonetic Posteriorgrams are directly used as training features of the Neural Network? What will happen if we add Multimodal features as suggested before? Is it possible to make the model compatible with all the future versions of the frameworks, drivers and libraries used in a very simple way? Is it possible to find a Dataset which is more controlled than the TED talks? How much difference there can be if all the Dataset is created in a controlled setup? There can be created a method fast enough to work online? These are some questions that must be addressed for future works, for the moment we can be satisfied with this approach since it is the first known approach for learning co-speech gestures by taking into account the Culture.
Another important actual limit in the subjective evaluation is that there isn't any control on the Workers that completed the survey, may the results be exactly the same if the tests are repeated or it is better to prefer a more controlled evaluation maybe also with less people? This is another question that must be addressed.
A physical limit is instead the one of the Robot: indeed it can't reproduce the movements smoother as they were computed so if we not choose a correct timing for poses they can results too slow, shaky or also have important delays between them, moreover, there must considered that when it is moving the localization of its joints may be worse with the passing of the time and the result can be very different from what we thought. These facts were noticed also by subjective eval-

uating the differences between the poses in the video and the poses reproduced by the Robot and this may heavily influenced the performance in the subjective evaluation. It is also to take into account that the 3D gestures were outputted by Neural Network and sometimes may be not be as expected: to limit this problem they can be again processed before being reproduced by the Robot. Having said that, this can of course be a good starting point for the future work and by applying some small as well as big changes it can be improved by a lot and there can be obtained very natural gestures. It must be taken into account that people may not like very natural gestures being reproduced from an agent, but this fact can be proved only by relying on some psychological studies.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. 3, 65

Alibali, M.W., Kita, S. & Young, A.J. (2000). Gesture and the process of speech production: We think, therefore we gesture. *Language and Cognitive Processes*, **15**, 593–613. 1

Archer, D. (1997). Unspoken diversity: Cultural differences in gestures. *Bibliovault OAI Repository, the University of Chicago Press*, **20**. 1

Bai, S., Kolter, J.Z. & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, **abs/1803.01271**. 17

Bisong, E. (2019). *Google Colaboratory*, 59–64. Apress, Berkeley, CA. 3

Bremner, P., Pipe, A., Melhuish, C., Fraser, M. & Subramanian, S. (2011). The effects of robot-performed co-verbal gesture on listener behaviour. In *2011 11th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS 2011*, IEEE-RAS International Conference on Humanoid Robots, 458–465. 1

Butterworth, B. & Hadar, U. (1989). Gesture, speech, and computational stages: a reply to mcneill. *Psychological review*, **96**, 168—174. 50

Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S. & Sheikh, Y.A. (2019a). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 12

Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S. & Sheikh, Y.A. (2019b). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 29

Chan, C., Ginosar, S., Zhou, T. & Efros, A.A. (2018). Everybody dance now. *CoRR*, **abs/1808.07371**. 8

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H. & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, **abs/1406.1078**. 16

de Meijer, M. (1989). The contribution of general features of body movement to the attribution of emotions. *Journal of Nonverbal Behavior*, **13**, 247–268. 10

Feyereisen, .d.L.J.D., P. (1991). Studies in emotion and social interaction. 10

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N. & Zue, V. (1992). Timit acoustic-phonetic continuous speech corpus. *Linguistic Data Consortium*. 20

Ginosar, S., Bar, A., Kohavi, G., Chan, C., Owens, A. & Malik, J. (2019). Learning individual styles of conversational gesture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2, 3, 9

Goodfellow, I. (2017). Nips 2016 tutorial: Generative adversarial networks. 2

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative adversarial networks. 2, 6, 7

Google (2021a). Google developers. **https://developers.google.com**. 28

Google (2021b). Youtube data api. **https://developers.google.com/youtube/v3**. 28

Hazen, T.J., Shen, W. & White, C. (2009). Query-by-example spoken term detection using phonetic posteriorgram templates. In *2009 IEEE Workshop on Automatic Speech Recognition Understanding*, 421–426. 20

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G. & Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, **abs/1706.08500**. 18

Huang, C. & Mutlu, B. (2014). Learning-based modeling of multimodal behaviors for humanlike robots. In *2014 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 57–64. 8

Isola, P., Zhu, J., Zhou, T. & Efros, A.A. (2016). Image-to-image translation with conditional adversarial networks. *CoRR*, **abs/1611.07004**. 7, 11

Joo, H., Liu, H., Tan, L., Gui, L., Nabbe, B., Matthews, I., Kanade, T., Nobuhara, S. & Sheikh, Y. (2015a). Panoptic studio: A massively multiview system for social motion capture. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 3334–3342. 17

Joo, H., Liu, H., Tan, L., Gui, L., Nabbe, B., Matthews, I., Kanade, T., Nobuhara, S. & Sheikh, Y. (2015b). Panoptic studio: A massively multiview system for social motion capture. In *The IEEE International Conference on Computer Vision (ICCV)*. 60

Kietzmann, J., Lee, L.W., McCarthy, I.P. & Kietzmann, T.C. (2020). Deepfakes: Trick or treat? *Business Horizons*, **63**, 135–146, aRTIFICIAL INTELLIGENCE AND MACHINE LEARNING. 2

Kipp, M. (2003). Gesture generation by imitation: from human behavior to computer character animation. 8

Kita, S. (2009). Cross-cultural variation of speech-accompanying gesture: A review. *Language and Cognitive Processes*, **24**, 145–167. 1

Kominek, J. & Black, A. (2004). The cmu arctic speech databases. *SSW5-2004*. 20

Krauss, R.M., Chen, Y. & Chawla, P. (1996). Nonverbal behavior and nonverbal communication: What do conversational hand gestures tell us? **28**, 389–450. 1

Le, Q.A., Hanoune, S. & Pelachaud, C. (2011). Design and implementation of an expressive gesture model for a humanoid robot. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, 134–140. 6

Levine, S., Krähenbühl, P., Thrun, S. & Koltun, V. (2010). Gesture controllers. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, Association for Computing Machinery, New York, NY, USA. 8

Mcneill, D. (1994). Hand and mind: What gestures reveal about thought. *Bibliovault OAI Repository, the University of Chicago Press*, **27**. 1, 10

Medium.com (2020). https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53. 12

Meena, R., Jokinen, K. & Wilcock, G. (2012). Integration of gestures and speech in human-robot interaction. In *2012 IEEE 3rd International Conference on Cognitive Infocommunications (CogInfoCom)*, 673–678. 6

Menicatti, R., Recchiuto, C.T., Bruno, B., Zaccaria, R., Khaliq, A.A., Köckemann, U., Pecora, F., Saffiotti, A., Bui, H.D., Chong, N.Y., Lim, Y., Pham, V.C., Tuyen, N.T.V., Melo, N., Lee, J., Busy, M., Lagrue, E., Montanier, J., Pandey, A.K. & Sgorbissa, A. (2018). Collaborative development within a social robotic, multi-disciplinary effort: the caresses case study. In *2018 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 117–124. 85

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlícek, P., Qian, Y., Schwarz, P., Silovský, J., Stemmer, G. & Veselý, K. (2011). The kaldi speech recognition toolkit. 20

Rodriguez, I., Martínez-Otzeta, J.M., Irigoien, I. & Lazkano, E. (2019). Spontaneous talking gestures using generative adversarial networks. *Robotics and Autonomous Systems*, **114**, 57–65. 8

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, **abs/1505.04597**. 11

Sehgal, S., Singh, H., Agarwal, M., Bhasker, V. & Shantanu (2014). Data analysis using principal component analysis. In *2014 International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom)*, 45–48. 15

Softbank, R. (2018). Naoqi api documentation. In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, vol. http://doc.aldebaran.com/2-5/home$_p$*epper.html*, $1--6$. 5, 6, 16, 63, 64

Sun, L., Kang, S., Li, K. & Meng, H. (2015). Voice conversion using deep bidirectional long short-term memory based recurrent neural networks. 21

Sun, L., Li, K., Wang, H., Kang, S. & Meng, H. (2016). Phonetic posteriorgrams for many-to-one voice conversion without parallel data training. In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, 1–6. 2, 11

Sun, L., Li, K., Wang, H., Kang, S. & Meng, H. (2016). Phonetic posteriorgrams for many-to-one voice conversion without parallel data training. In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, 1–6. 20

Wilson J.R., S.A.H.S.S.M.T.D.L., Lee N.Y. (2017). Hand gestures and verbal acknowledgments improve human-robot rapport. *Kheddar A. et al. (eds) Social Robotics*. 2

Wolfert, P., Robinson, N.L. & Belpaeme, T. (2021). A review of evaluation practices of gesture generation in embodied conversational agents. *CoRR*, **abs/2101.03769**. 3, 71

Wu, Y. *et al.* (2016). Tensorpack. `https://github.com/tensorpack/`. 66

Yang, Y. & Ramanan, D. (2013). Articulated human detection with flexible mixtures of parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**, 2878–2890. 14

Yoon, Y., Ko, W., Jang, M., Lee, J., Kim, J. & Lee, G. (2018). Robots learn social skills: End-to-end learning of co-speech gesture generation for humanoid robots. *CoRR*, **abs/1810.12541**. 3, 9

Yoon, Y., Cha, B., Lee, J.H., Jang, M., Lee, J., Kim, J. & Lee, G. (2020). Speech gesture generation from the trimodal context of text, audio, and speaker identity. *ACM Trans. Graph.*, **39**. 17