Studenti:

Cassano Lorenzo (matricola 718331)

l.cassano25@studenti.uniba.it

D'Abramo Jacopo (matricola 716484)

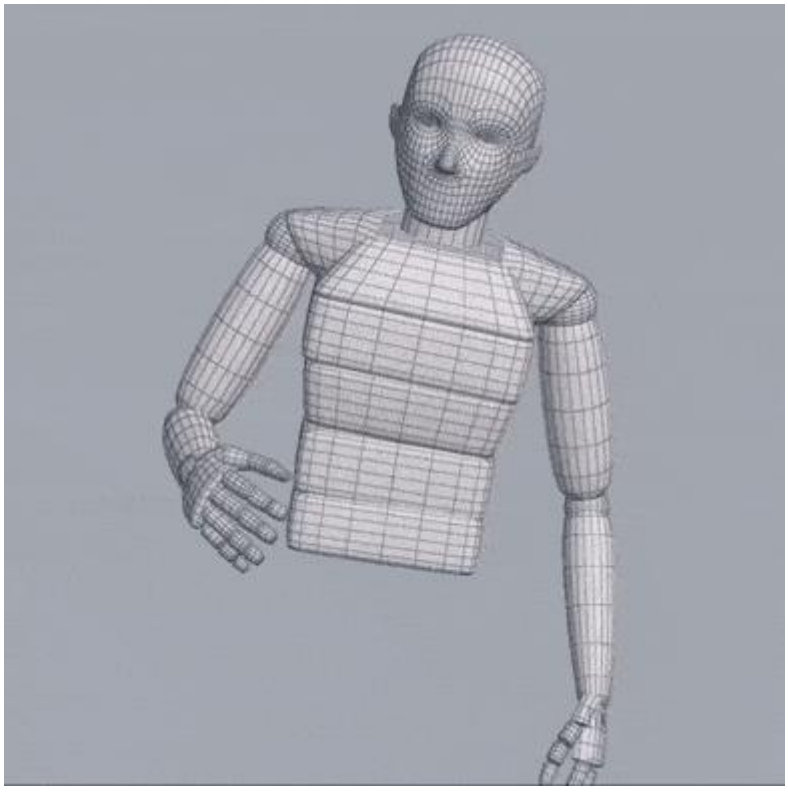
j.dabramo@studenti.uniba.it

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

# DOCUMENTAZIONE CASO DI STUDIO – CORSO DI SISTEMI AD AGENTI

## A.A. 2021-2022

## Speech-Controlled Body Animations



# INDICE

# 1 LA MISSION

Questo progetto è stato realizzato sulla base della collaborazione tra il dipartimento di informatica dell'università degli studi di Bari "Aldo Moro" e l'azienda Quest-it. L'obiettivo principale è quello di creare un avatar in grado di gesticolare sulla base di una traccia audio fornita in input. Le applicazioni di tale progetto sono molteplici dal mondo dei videogiochi a modelli per il miglioramento della user-experience.

Il lavoro consiste nell'ampliamento di un progetto già esistente: "SpeechDrivesTemplates" (https://github.com/ShenhanQian/SpeechDrivesTemplates), il quale effettua predizioni dei gesti in base ad un audio fornito in input, su uno spazio bidimensionale. Il nostro obiettivo è stato quello di espandere tali predizioni su uno spazio tridimensionale.

Andando nello specifico le modifiche riguardano le seguenti sezioni:

- **Preprocessing3D**: è stato creato un preprocessing per poter creare un dataset basato su tre dimensioni
- **Training**: è stata modificata la precedente fase di training per addestrare il modello nell'effettuare predizioni su uno spazio tridimensionale.

# 2 STRUMENTI ADOTTATI

Il linguaggio utilizzato per sviluppare il progetto è stato Python, data la grande mole di librerie ad agevolezza d'uso delle stesse, in particolare di quelle funzionali ai nostri obiettivi. Le principali librerie da noi utilizzate sono state:

- **Pandas** per la gestione del dataset ed il relativo preprocessing
- **NumPy** per la gestione dei file npy e npz
- **sklearn** e **pytorch** per definire il modello ed effettuare predizioni
- **mediapipe** e **OpenCv** per poter ottenere i keypoints e analizzare le immagini
- **yacs** per definire e gestire le configurazioni di sistema
- **FFmpeg** per la generazione dei video
- **Os** per la manipolazione dei percorsi, file e cartelle

Inoltre, nella fase iniziale del lavoro è stato utilizzato il software Bandicut (https://www.bandicam.com/bandicut-video-cutter/) per poter dividere i video iniziali in segmenti da 30 secondi l'uno.

Per l'esecuzione del modello è preferibile utilizzare le risorse messe a disposizioni da google Colab (https://colab.research.google.com/?utm_source=scs-index), dati i lunghi tempi di esecuzioni e il grande dispenso di memoria RAM utilizzata durante i diversi run.

# 3 GERARCHIA DIRECTORY

Il progetto è diviso in due sottocartelle:

- **Preprocessing3D**: nel quale si trovano tutti i file per la costruzione del dataset in 3D
- **SpeechDrivesTemplates**: nel quale si trova il progetto con le relative modifiche al modello

La parte principale del progetto si trova nella cartella SpeechDrivesTemplates3D nel quale vi si trova la rete neurale per poter addestrare il modello.

Questa directory è organizzata nel seguente modo:

- **configs**: in questa cartella si trovano i file di configurazione, più precisamente:
  - *default*.py file attraverso il quale è possibile cambiare i parametri di default; tra i parametri più importanti abbiamo: il numero delle epoche, il learning rate, il salvataggio degli output su file npz, la dimensione del batch e se effettuare la fase di validazione.
  - voice2pose_s2g.yaml
  - voice2pose_sdt_bp.yaml: unico file di configurazione usato per trasformare le predizioni da 2D a 3D
  - *voice2pose_sdt_vae.yaml*
  - *pose2pose.yaml*
- **core:**
  - **datasets**: sottodirectory nella quale si trovano i file per la gestione del dataset su cui addestrare il modello
  - **networks:** sottodirectory nella quale si trovano i file per la gestione della rete neurale, più precisamente per la predizione dei keypoints e delle pose
  - **pipelines:** sottodirectory che si occupa della gestione delle reti neurali durante la fase di training
  - **utils:** sottodirectory che si occupa di effettuare ulteriori fasi di preprocessing durante la fase di addestramento. Essa si occupa anche di gestire il disegno delle predizioni su uno spazio 2D, in un futuro lavoro sarebbe meglio proiettare le coordinate su uno spazio 3D per avere una panoramica migliore dei risultati.
- **dataset:** cartella che deve essere create per inserire il dataset sul quale addestrare il modello, in particolare questa directory deve essere organizzata nel seguente modo:
  - **speakers:**
    - *oliver*
    - *xing*
    - *…*
- **output:** cartella che deve essere creata per salvare l'output
- *main.py*

# 4 PREPROCESSING E CREAZIONE DEL DATASET

## 4.1 INTRODUZIONE AL PREPROCESSING

La fase del preprocessing è stata di fondamentale importanza per la corretta esecuzione del modello e l'ottenimento di buoni risultati. In particolare, è necessario definire un dataset i cui video inquadrino uno speaker con una buona precisione, ovvero le principali parti del corpo: mani, viso, spalle, busto e bacino devono essere ben inquadrate. Per questo task abbiamo deciso di utilizzare dei video di delle lezioni di economia tenute da un professore del politecnico di Milano.

Tali video sono disponibili sui seguenti link:

- Parte 1 (https://www.youtube.com/watch?v=dHPZVh3NWbA)
- Parte 2 (https://www.youtube.com/watch?v=z-W4_ABRLAE)

Se si vuole eseguire il modello con un custom dataset, come precedentemente accennato, è necessario scegliere i video con particolare accuratezza. Il modello deve essere in grado di poter sempre riconoscere lo

speaker e registrare i suoi movimenti. Di seguito si riportano degli esempi di frames corretti, estratti dal dataset, utilizzato nel nostro esperimento, da cui **mediapipe** riesce a ottenere i keypoints.

Come è possibile notare i contorni del viso, del busto e delle mani sono ben definiti ed è possibile notare con chiarezza la posa dello speaker in questione.

In linea generale in questa fase è necessario andare a suddividere il video in una serie di frames da cui ottenere i keypoints e successivamente salvare tali dati in dei file npz per poter essere utilizzati in un secondo momento dal modello.

Per una corretta esecuzione del preproccessing è necessario eseguire i file nell'ordine in cui si trovano all'interno della cartella Preprocessing3D.

## 4.2 SUDDIVISIONE DEL PREPROCESSING

La fase del preprocessing può essere suddivisa in 4 sotto parti:

1. Manipolazione video e generazione frames
2. Generazione dei keypoints
3. Produzione dei file npz, delle tracce audio e del csv
4. Generazione del mean e del std

## 4.3 MANIPOLAZIONE VIDEO E GENERAZIONE DEI FRAMES

La prima parte del preprocessing consiste nel preparare i video per la fase di generazione dei keypoints, ovvero cambiare gli FPS per i singoli video e generare i frames.

Prima di poter iniziare ad eseguire i file del preprocessing bisogna suddividere i video scelti in dei video più piccoli di uguale dimensione, nel nostro caso i singoli video hanno una durata di 30 secondi. Per questo task è stato utilizzato il software bandicut. Successivamente si può iniziare con l'esecuzione del file '1_1_change_fps_py' il quale converte ogni video in 15 fps, tale passo risulta utile per una migliore

estrazione dei frames. Si consiglia di non eccedere con la durata dei singoli video in quanto ffmpeg ha una soglia massima di durata.

Dopo avere convertito i video, il passo successivo consiste nell'estrazione dei frames dai singoli video, tale passo si ritrova nell'esecuzione del secondo file '1_2_video2frames.py'. Nell'esecuzione di tale file bisogna rispettare i path riportati nel codice, ovvero all'interno del DATASET_PATH devono essere presenti le cartelle 'videos' e 'frames' che conterranno rispettivamente l'insieme dei video in 15 fps e tutti i frame generati per ogni video.

## 4.4 GENERAZIONE DEI KEYPOINTS

La fase della generazione dei keypoints è la parte più importante della creazione del dataset perché l'addestramento del modello si baserà quasi esclusivamente sui keypoints ottenuti dallo speaker che si sta utilizzando.

Questa è stata la principale parte del preprocessing cambiata, perché rispetto alla vecchia fase di creazione del dataset in questa vi è il bisogno di stimare tre coordinate sul frame che si sta analizzando: x la larghezza, y l'altezza e z la profondità.

Altro fattore molto importante che è stato preso in considerazione è stato l'utilizzo della libreria **mediapipe** di python per stimare le pose rispetto all'utilizzo di **OpenPose** che veniva utilizzato nella vecchia fase di preprocessing per stimare i punti.

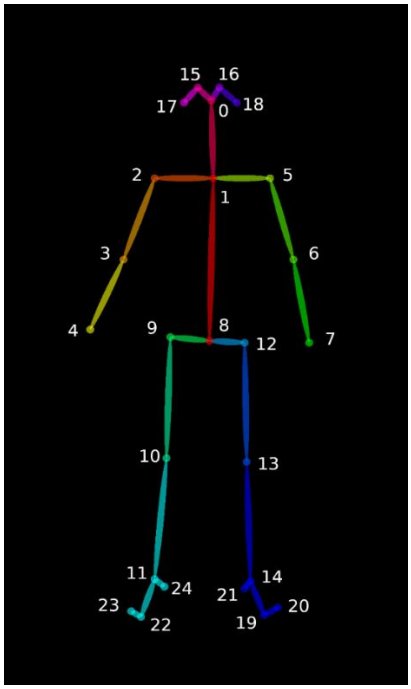
La scelta di utilizzare mediapipe è stata effettuata per due ragioni:

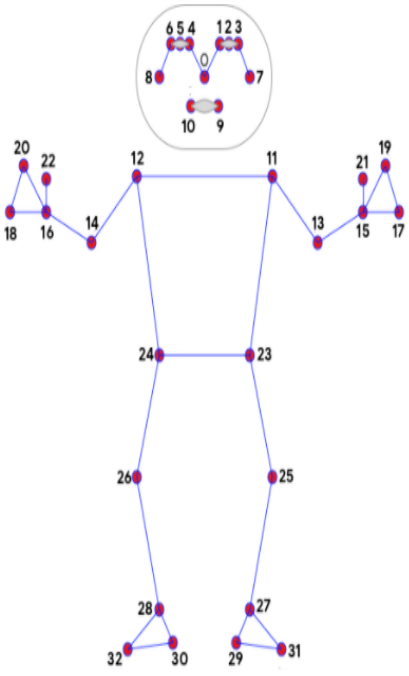
1. una maggiore velocità in termini di esecuzione nello stimare i punti rispetto ad **OpenPose**
2. una minore difficoltà nello stimare la terza coordinata

Questo cambio di strumento che abbiamo utilizzato ha fatto sì che la costruzione dei keypoints fra la vecchia fase di creazione del dataset e la nuova abbia delle notevoli differenze.

La prima grande differenza può essere osservata nei due scheletri.

### 4.4.1 KEYPOINTS SCHELETRO BASE



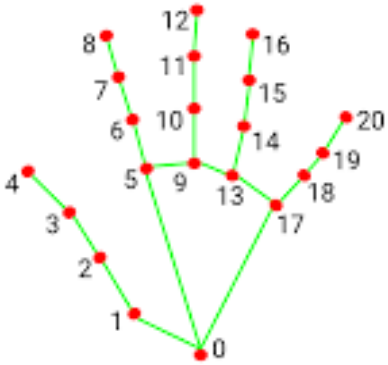



Le relative immagini rappresentano lo scheletro di **Open Pose** (immagine a sinistra) e lo scheletro di **mediapipe** (immagine a destra). Una prima grande differenza che si può osservare sono i punti stimati, in **OpenPose** vengono stimati 24 punti mentre in **mediapipe** ne vengono stimati 33.

Altro fattore importante sono le stime delle pose per i punti più particolare dello scheletro come il viso e le mani, anche nella stima di questi punti particolari **OpenPose** e **mediapipe** si differenziano fra di loro.

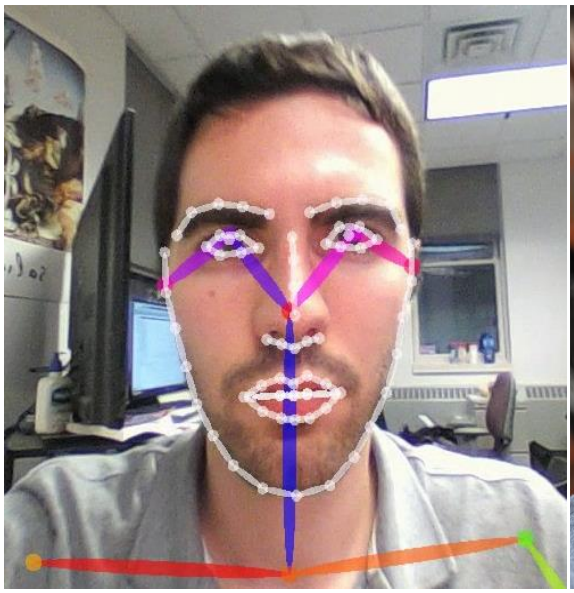
### 4.4.2 KEYPOINTS PER LE MANI

OpenPose

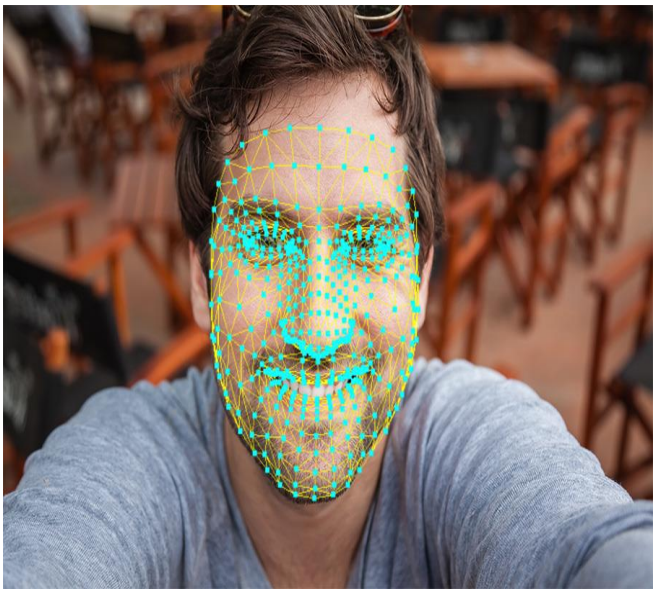



Mediapipe

### 4.4.3 KEYPOINTS PER LA FACCIA



OpenPose



Mediapipe

## 4.5 ANALISI DEI KEYPOINTS

Essendo che OpenPose funziona in maniera diversa da mediapipe, si sono dovute effettuare delle operazioni di analisi e di riadattamento per la creazione del dataset.

Il primo elemento da prendere in considerazione è che mediapipe stima molti più punti di OpenPose, più precisamente OpenPose in totale stima 137 punti mentre mediapipe 543 punti; quindi, per poter effettuare una buona creazione del dataset, si avrà bisogno di frame di alta qualità come già indicato nel paragrafo: “Introduzione al preprocessing”.

Per creare le clips sul dataset vi è anche il bisogno di stimare i punti di ogni frame di ogni video per poi poterli collegare alla traccia audio attraverso il quale sarà possibile poi effettuare la fase di training.

I problemi di questa fase erano:

- OpenPose riusciva quasi sempre a stimare i punti anche quando i frame non avevano una buona qualità (dato anche dal fatto che aveva meno punti da stimare)
- Mediapipe riusciva ad effettuare la stima solamente quando la qualità dei frame era molta elevata
- Vi era il bisogno di avere dei piccoli video di alta qualità e con una buona inquadratura dello speaker
- Non avevamo tanti dati con questa elevata qualità

Durante l’esecuzione dello script di preprocessing.py il quale si occupa di generare i keypoints per ogni frame, mediapipe non riesce a stimare i keypoints per ogni frame per le ragioni precedentemente elencate.

Per risolvere questo problema bisogna eseguire lo script fixing.py il quale effettua una fase di analisi ed aggiustamento; più precisamente nel nostro caso essendo che il video principale è suddiviso in video da 30 secondi lo script analizza per quali video si sono prodotti più di 400 frame (dato che può essere cambiato in base alla dimensione dei vari video) e di questi video controlla se ci sono un intervallo di frames minore di 10 per i quali mediapipe non è riuscito ad ottenere i keypoints. Per questi video i keypoints verranno stimati in questa maniera:

- Se l’intervallo non stimato va dal frame 0 fino al massimo del frame 10, verrà preso il primo keypoints stimato e verrà utilizzato per gli altri frame
- Se l’intervallo va dal frame 458 fino all’ultimo frame 468 (scelto questo frame perché multiplo di 64 visto che i file npy vengono caricati 64 alla volta), verrà preso l’ultimo keypoints stimato e verrà utilizzato per gli altri frame
- Se l’intervallo non stimato va da un frame x ad un frame y, verrà preso l’ultimo keypoints stimato prima del frame x e il primo keypoint stimato dopo il frame y, verrà calcolata una media dei keypoints e verrà utilizzato questo keypoints per i frames non stimati.

Questa procedura nel nostro caso porta a schernire di molto i frames sulla quale vengono calcolati i keypoints e le stime vengono calcolate su un intervallo di frame minori di un secondo.

Questa operazione nel nostro caso è stata obbligatoria e in parte “forzata” data dai pochi video di buona qualità che avevamo a disposizione, in futuro l’operazione migliore sarebbe andare a registrare un video di buona qualità inquadrando in maniera precisa lo speaker in modo tale da poter usare lo script preprocessing.py (se il video è di buona qualità tutti i frames saranno perfetti per far si che mediapipe riesca a stimare i keypoints) e snellire di molto ( o eliminare del tutto) il lavoro svolto dallo script fixing.py.

## 4.6 PRODUZIONE DEI FILE NPZ, DELLE TRACCE AUDIO E DEL CSV

In questa terza fase i keypoints prodotti al punto precedente vengono archiviati a gruppi all’intero di file NPZ. In particolare, è stato scelto di mantenere la stessa dimensione adottata dai precedenti autori in SpeechDrievesTemplates, ovvero 64 frames per ogni NPZ.

Questa terza fase si attua tramite l'esecuzione di '3_1_generate_clips.py' il quale richiede che i file npy vengano posti sotto il percorso /tmp/raw_pose_2d. I file NPZ conterranno quindi 64 vettori quadridimensionali e i percorsi ai rispettivi 64 frames. Inoltre, in tale fase vengono estratti gli audio per ogni video per cui sono stati prodotti gli NPZ, e, come ultima operazione lo script definisce un file .csv in cui sono racchiuse tutte le informazioni, necessarie in fase di training, per reperire gli npz.

### 4.7 SPLIT TRAIN TEST E VALIDATION

Il csv prodotto al punto precedente viene manipolato da '3_2_split_train_val_test.py' per la suddivisione del dataset rispettivamente in train test e validation. In particolare, è stato deciso di mantenere come percentuale di esempi di train l'80% mentre il rimanente 20% viene utilizzato per il test e il validation.

### 4.8 CREAZIONE MEAN E STD

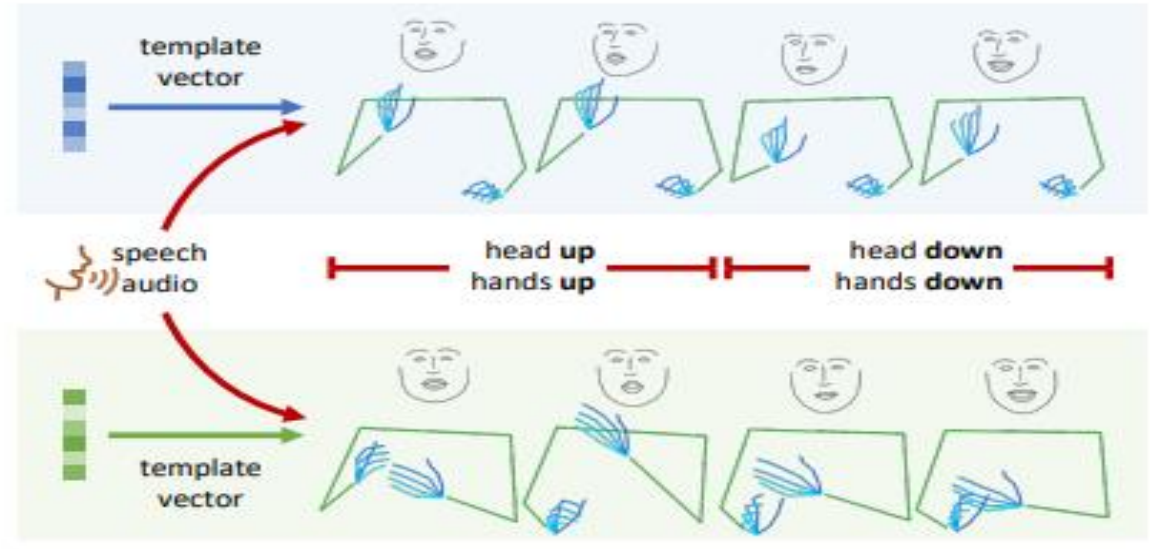
L'ultima fase relativa alla creazione del dataset riguarda il calcolo del mean e del std, cioè per ogni keypoints viene calcolata la media e la standard confidence che devono essere successivamente inseriti nello speaker_stats.py.

## 5 SPEECH DRIVES TEMPLATES

### 5.1 INTRODUZIONE

L'obiettivo di questo progetto è quello di creare un modello addestrato su un dataset di video, capace di riprodurre una sequenza di gesti dato in input una traccia audio.

Un primo problema che si pone è la possibilità di avere diverse sequenze di "**Gestures**" presa una stessa traccia audio in input, ovviamente la differenza tra queste "**Gestures**" cambia in base a differenti fattori: emozioni, movimento precedente e abitudini.



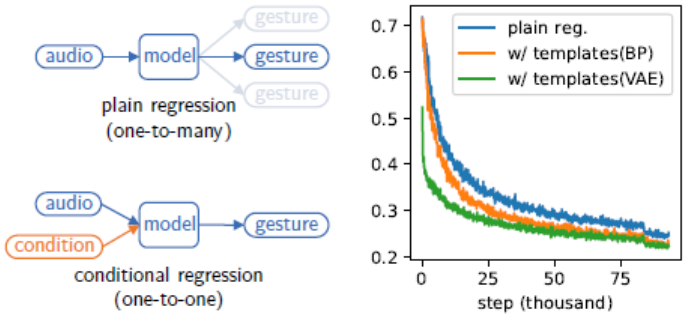

*Preso in input lo stesso audio, ci possono essere in output diverse sequenze di gesti*

Per risolvere questo problema visto che l'obiettivo non era quello di avere diverse sequenze di gesti ma un'unica sequenza di gesti, è stato introdotto il ***condition vector.***

Il ***condition vector*** fornisce al modello le informazioni mancanti in modo tale da avere in output una sola sequenza di gesti.

Concretamente il ***condition vector*** viene inizializzato a zero e viene poi riempito con i valori della ***regression loss***, esso viene creato per ogni 4 secondi di video, che corrispondono a 60 frame (visto che il video è in 15 fps), la quale più o meno corrisponde alla grandezza dei nostri file npz.

Per la sincronizzazione viene utilizzata la ***regression loss*** perché è l'unico metodo di apprendimento supervisionato affidabile per avere una buona qualità di match tra **Speech** e **Gestures**.




(a) Plain regression versus conditional regression.

(b) Regression loss curves on the training set.

## 5.2 METODO

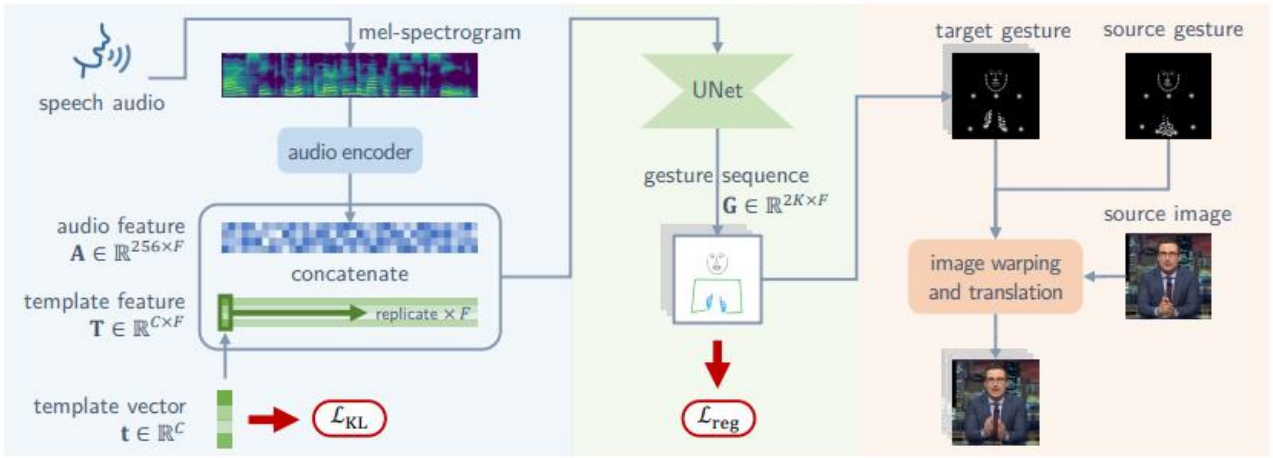
Formalmente, il metodo utilizzato prevede inizialmente la conversione dell'audio da forma d'onda a spettrogramma, il quale può essere inteso come un grafico in cui sugli assi vengono mappati tempo e frequenza. Successivamente, lo spettrogramma viene dato in input ad un audio encoder, il quale si occupa di estrarre il vettore dell'***audio feature*** definito in $A \in R^{256xF}$ dove F rappresenta il numero dei frames. Inoltre, viene fornito anche un ***template vector*** $t \in R^{C}$ il quale viene poi adattato per ogni frame ottenendo un ***template feature*** $T \in R^{CxF}$. Quindi l'input completo per il nostro modello è rappresentato dalla concatenazione dei due vettori A,T $[A|T] \in {}^{(256+C)xF}$.

La rete neurale che si occupa della generazione dei gesti è una **rete neurale convoluzionale** costituita da 7 strati di codifica e 6 strati di decodifica. Per la riduzione dell'errore sulla sequenza di gesti viene utilizzata la ***regression loss*** $L_1$:

$$L_{reg} = \frac{1}{F}\sum_{i=1}^{F}\left\|G^{(i)} - \hat{G}^{(i)}\right\|_1$$

Dove $G^{(i)}$ rappresenta il ground truth mentre $\hat{G}^{(i)}$ il vettore dei gesture predetto.

## 5.3 RETE NEURALE




La parte principale dell'apprendimento è svolto dalla rete neurale, la quale prende in input un ***audio feature*** A ∈ $R^{256xF}$ e un ***template vector*** T ∈ $R^{CxF}$ per poi riprodurre come output una sequenza di gesti G ∈ $R^{2KxF}$ dove F è il numero dei frame, C è la dimensione del template vector e K è il numero dei keypoints.

L'espansione del modello da 2D a 3D ha fatto sì che la sequenza di gesti G∉ $R^{2KxF}$ma G∈$^{3KxF}$.

L'apprendimento dei template vector avviene durante la fase di: **"Back propagation"**; più precisamente; il vettore viene inizializzato a zero e durate la fase di **back propagate** quest'ultimo viene aggiornato attraverso la ***regression loss.***

Per regolarizzare lo spazio vettoriale del ***template vector*** viene usata la KL ***divegence loss***.

$$\mathcal{L}_{\mathrm{KL}} = D_{\mathrm{KL}}\left(\mathcal{N}\left(\mu_{\mathbf{t}}, \sigma_{\mathbf{t}}^2\right) \| \mathcal{N}(0,1)\right)$$

Dove µt ∈ R C e $\sigma_t^2$∈ R sono la media e la varianza del ***template vector*** t.

Infine, la ***funzione di loss*** è definita come segue:

$$\mathcal{L} = \lambda_{\mathrm{reg}}\mathcal{L}_{\mathrm{reg}} + \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}},$$

Dove $\lambda_{\mathrm{reg}}$ e $\lambda_{\mathrm{KL}}$ sono i pesi associati alla ***funzione di loss***.

## 5.4 EVALUATION

Nei nostri esperimenti abbiamo omesso la parte di **evaluation** in quanto richiedeva la modifica per il passaggio al 3D.

In linea generale, come esposto dai precedenti autori, come metodi di valutazione, la distanza $L_1$ e $L_2$ non vanno bene in quanto misurano la distanza tra le predizioni e il ground truth, ignorando la diversità dei gesti realizzabili per uno specifico audio. Per questo motivo come metriche di valutazione vengono scelti 2 indicatori che riescono a considerare rispettivamente la sincronizzazione e la naturalezza. Tali metriche sono:

- ***Lip-Sync:*** misura la distanza tra i keypoints delle labbra generati e il ground truth.

$$\mathcal{E}_{\text{lip}} = \frac{\frac{1}{F}\sum_{i=1}^{F} \|d^{(i)} - \hat{d}^{(i)}\|_2}{\max_{1 \leq n \leq F} \hat{d}^{(n)}}$$

- ***Frechet Template Distance***: misura la distanza tra i gesti predetti e quelli reali tra un gruppo di esempio piuttosto che su un singolo esempio.

$$\text{FTD} = |\mu_{\mathbf{t}} - \mu_{\hat{\mathbf{t}}}|^2 + \text{tr}\left(\Sigma_{\mathbf{t}} + \Sigma_{\hat{\mathbf{t}}} - 2\left(\Sigma_{\mathbf{t}}\Sigma_{\hat{\mathbf{t}}}\right)^{1/2}\right),$$

# 6 CONCLUSIONE E FUTURI LAVORI

Attraverso la challange offerta dal dipartimento di Informatica dell'università di Bari in collaborazione con Quest-it ed Exprivia, abbiamo avuto la possibilità di realizzare un caso di studio più complesso ma che allo stesso tempo può avere un riscontro applicativo nella vita reale.

Abbiamo avuto la possibilità di scegliere molti progetti ma la scelta è ricaduta su quest'ultimo per le varie applicazioni che esso può avere.

In particolare, questo caso di studio ci ha permesso di approfondire il campo della: "Computer vision" attraverso la creazione del dataset con mediapipe, più precisamente andare a toccare i keypoints che vengono riprodotti ed andare a capire la struttura di uno scheletro codificato attraverso un vettore.

Questo ci ha permesso di capire la forza di tale strumento ma allo stesso tempo ci ha anche posto davanti a problematiche importanti che abbiamo dovuto risolvere studiando in maniera più approfondita questa macroarea.

Oltre alla computer vision, questo caso di studio ci ha dato la possibilità di espandere la nostra conoscenza sul campo del deep learning visto che tutto il modello si basa su una rete neurale convoluzionale .

Per lo più, questa è stata anche la prima volta che analizziamo un progetto complesso nel campo del deep learning, ciò ci ha insegnato come approcciare a questi tipi di lavori.

L'estensione del modello da 2D a 3D può portare i risultati del modello in un dominio reale, visto che i risultati delle predizioni possono essere applicati ad un avatar o ad un robot facendo sì che possa notevolmente aumentare fattori di "User experience" nella comunicazione con una macchina.

Ovviamente la parte in 3D per poter essere portata in un dominio reale deve essere migliorata, partendo dalla creazione di un dataset molto più ampio di quello realizzato e cercando di usare dei buoni video in modo tale da aver una grande e buona qualità di dati e non trovarsi nelle situazioni descritte in precedenza.

L'estensione principale che deve essere fatta è quella di disegnare le predizioni su uno spazio tridimensionale in modo tale da avere un output più veritiero e per far sì che il modello possa essere effettivamente utilizzato.

Infine, si vuole far notare come questo studio sia estremamente importante perché i precedenti studi in questo campo venivano effettuati attraverso l'utilizzo di regole logiche, quindi la creazione di gesti da parte dei precedenti modelli non risultava naturale.