UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

CASO DI STUDIO

SISTEMI AD AGENTI

2022-2023

HUMAN TV INTERACTIONS

Analisi di Clip Video di Film/Serie TV per il Riconoscimento di Interazioni Umane

Gruppo di Lavoro

- Michele Astarita 744455 m.astarita@studenti.uniba.it
- Dario Liantonio 704535 d.liantonio3@studenti.uniba.it

GitHub Repository

https://github.com/michele-astarita99/Sistemi-ad-Agenti-22-23

Sommario

Introduzione

In questo caso di studio affrontiamo il problema del riconoscimento delle interazioni tra due persone, in scenari realistici, per scopi di ricerca. Il nostro obbiettivo sarà quello di realizzare un'intelligenza artificiale in grado di riconoscere tali iterazioni. In particolare, ci concentriamo su quattro interazioni: strette di mano, battere il cinque, abbracci e baci.

Le interazioni tra due persone possono anche essere utilizzate direttamente o come elemento costitutivo per creare sistemi complessi in applicazioni secondarie.

Il codice che andremo a sviluppare implementa un flusso di lavoro completo per l'elaborazione dei video, l'estrazione dei frame, la previsione delle classi e la valutazione della confidenza delle previsioni. Ogni passaggio che abbiamo eseguito verrà descritto in modo chiaro e approfondito, per consentire una comprensione dettagliata del funzionamento.

Dataset

Per sviluppare il nostro caso di studio abbiamo utilizzato un Dataset chiamato '**TV Human Interaction Dataset**', messo a disposizione dall'Università di Oxford.

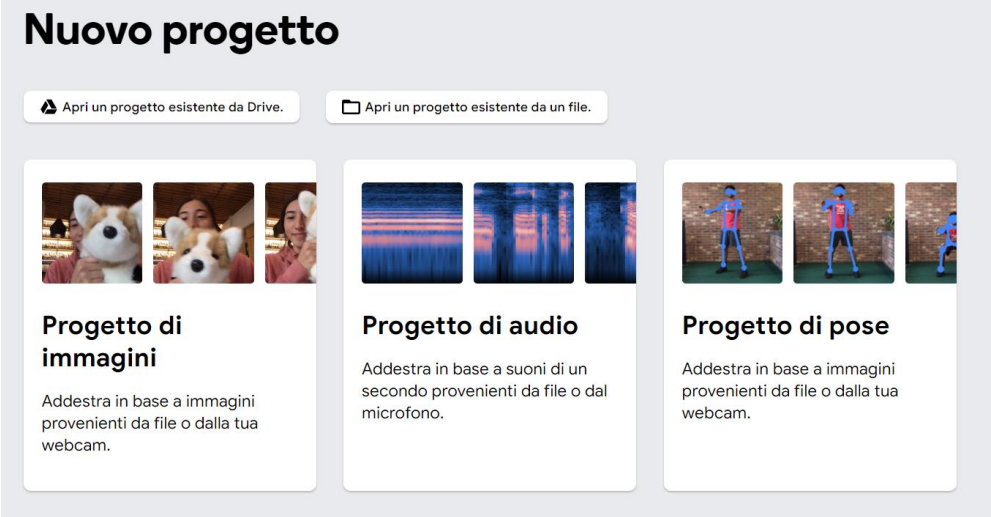
Link: [TV Human Interaction Dataset (ox.ac.uk)]

Il Dataset è composto da 300 clip, collezionate da oltre 200 serie TV diverse, suddivise in cinque sottocategorie: hand shake, high five, hug, kiss and negative. Quest'ultima contiene clip in cui non sono presenti nessuna delle interazioni precedentemente citate.

Addestramento

Per addestrare il nostro modello di riconoscimento abbiamo utilizzato **Teachable Machine**, un'applicazione web sviluppata da Google che ti permette di addestrare modelli di intelligenza artificiale senza dover scrivere codice. Utilizza un approccio di apprendimento automatico noto come "apprendimento transfer" per creare modelli che possono riconoscere immagini, suoni o posizioni delle mani.

Per addestrare un modello con Teachable Machine, inizi selezionando il **tipo di input** che desideri utilizzare: immagini, audio o pose. Puoi quindi acquisire o caricare un set di dati di addestramento, che consiste in esempi di input etichettati correttamente.



Una volta caricati i dati di addestramento, Teachable Machine utilizza **tecniche di apprendimento automatico** per creare un modello che possa fare previsioni su nuovi dati. Durante il processo di addestramento, il modello cerca di trovare pattern o caratteristiche comuni nei dati di addestramento per distinguere le diverse classi di input.

Dopo l'addestramento, puoi **testare** il tuo modello utilizzando nuovi dati che non sono stati utilizzati durante l'addestramento. Teachable Machine mostrerà le previsioni del modello e ti permetterà di valutare le sue prestazioni.

Una volta soddisfatto delle prestazioni del tuo modello, puoi **esportarlo** in diversi formati, come un file di **TensorFlow.js**, un file di modello **TensorFlow Lite** o un codice incorporabile nel tuo progetto. In questo modo, puoi utilizzare il tuo modello addestrato in un'ampia varietà di applicazioni, come app mobili, siti web o progetti di intelligenza artificiale.

È importante notare che Teachable Machine è uno strumento che **semplifica** il processo di addestramento dei modelli di intelligenza artificiale, ma la sua precisione dipende dalla **qualità dei dati di addestramento** e dalla **complessità del problema che si sta affrontando**. Se si ha un grande set di dati di addestramento ben etichettati e un problema relativamente semplice, potresti ottenere risultati molto accurati. Tuttavia, se si dovesse avere un set di dati più limitato o un problema complesso, potrebbero essere necessarie ulteriori ottimizzazioni o l'utilizzo di strumenti più avanzati.

Nel nostro caso, avendo a disposizione un dataset di clip, abbiamo prima creato uno script in linguaggio Python che permettesse di dividere ogni clip in immagini frames. In questa parte di codice abbiamo utilizzato le librerie **os** e **cv2**.

```python
def convert_to_frames(video_path, output_dir):
    # Ottieni il nome del file senza l'estensione
    file_name = os.path.splitext(os.path.basename(video_path))[0]

    # Apri il video
    video = cv2.VideoCapture(video_path)
    success, frame = video.read()
    count = 0

    # Leggi e salva tutti i frame del video
    while success:
        # Crea il nome del file per il frame corrente
        frame_file_name = f"{file_name}_{count}.jpg"

        # Crea il percorso di output per il frame corrente
        frame_path = os.path.join(output_dir, frame_file_name)

        # Salva il frame come immagine
        cv2.imwrite(frame_path, frame)

        # Leggi il frame successivo
        success, frame = video.read()
        count += 10

    # Rilascia la risorsa del video
    video.release()
```

Successivamente abbiamo creato le nostre classi di riferimento (Hand Shake, High five, Hug, Kiss e Negative) all'interno di Teachable Machine. Dopo di che abbiamo impostato i parametri di addestramento:

- **Periodo**: abbiamo scelto di inserire come numero di periodi 200, ricordiamo che un periodo indica che ogni singolo campione nel set di dati di addestramento è stato inserito attraverso il modello
- **dimensione del batch**: abbiamo lasciato quella di default, ovvero 16. Un batch indica un insieme di campioni utilizzati in un'iterazione di addestramento.
- **Tasso di apprendimento**: impostato a 0,001, in modo da renderlo il più specifico possibile.

Abbiamo avviato l'addestramento, ed una volta terminato, abbiamo esportato il modello in formato **Tensorflow** e convertito in **Keras** con estensione ".h5".

Sviluppo applicazione

Abbiamo sviluppato il codice seguendo le seguenti fasi.

1. Importazione delle librerie:

Abbiamo importato le seguenti librerie:

- La libreria `**keras.models**` viene importata per caricare il modello preaddestrato.
- Le librerie `**PIL.Image**` e `**PIL.ImageOps**` vengono importate per elaborare le immagini.
- La libreria `**numpy**` viene importata per lavorare con array multidimensionali.
- La libreria `**cv2**` (**OpenCV**) viene importata per la manipolazione dei video e delle immagini.

Per migliorare la leggibilità delle stampe, il codice utilizza `**np.set_printoptions(suppress=True)**` per disabilitare la notazione scientifica.

2. Caricamento del Modello:

Per caricare il modello preaddestrato dal file "**keras_model.h5**" creato con Teachable Machine, utilizziamo la funzione `**load_model**` del modulo `**keras.models**`. In questo modo viene creato un oggetto `model` che rappresenta il modello preaddestrato.

3. Caricamento delle Etichette:

Il codice apre il file "**labels.txt**" in modalità lettura e legge le etichette delle classi. Le etichette vengono memorizzate nella lista `**class_names**`.

4. Definizione del Percorso del Video:

Andiamo a specificare il percorso del video da analizzare tramite la variabile `**video_path**`.

5. Apertura del Video:

Attraverso la funzione `**cv2.VideoCapture**` viene aperto il video specificato nel percorso `**video_path**` e viene creato un oggetto `video` per rappresentare il video aperto.

7. Inizializzazione delle Variabili:

Viene inizializzata la lista `**total_predictions**` per contenere tutte le previsioni.

```
# Caricamento del modello addestrato tramite TM
model = load_model("keras_model.h5", compile=False)

# Caricamento delle etichetta da file labels.txt
class_names = open("labels.txt", "r").readlines()

# Sorgente video da analizzare e predirre
video_path = "Test_esterni/pexels-fauxels-3044910-3840x2160-24fps.mp4"

#Apertura video tramite OpenCV
video = cv2.VideoCapture(video_path)

# Inizializzazione dell'array delle predizioni
total_predictions = []
```

8. Elaborazione dei Frame del Video:

Con un ciclo while viene letto ed elaborato ogni frame del video. La condizione `**ret**` viene utilizzata per verificare se il frame è stato letto correttamente o se si è raggiunta la fine del video. Per ogni frame, vengono eseguiti i seguenti passaggi:

- **Conversione del Frame in un'Immagine PIL**: Il frame viene convertito nel formato immagine PIL utilizzando `**cv2.cvtColor**` per convertire lo spazio dei colori da BGR a RGB.
- **Ridimensionamento dell'Immagine**: L'immagine viene ridimensionata al formato desiderato utilizzando `**ImageOps.fit**` e la tecnica di ridimensionamento LANCZOS. L'obiettivo è ottenere un'immagine quadrata con dimensioni 224x224 pixel, che è il formato di input richiesto dal modello.
- **Normalizzazione dell'Immagine**: L'immagine viene convertita in un array numpy utilizzando `**np.asarray**`. L'array dell'immagine viene normalizzato dividendo per 127.5 e sottraendo 1 per ottenere valori compresi tra -1 e 1. Questa normalizzazione garantisce che l'input sia coerente con l'intervallo di valori su cui è stato addestrato il modello.
- **Espansione delle Dimensioni dell'Immagine**: L'immagine normalizzata viene espansa in una dimensione aggiuntiva utilizzando `**np.expand_dims**`. Questa espansione è necessaria per adattare l'immagine alla forma di input del modello, che richiede un array tridimensionale.
- **Previsione delle Probabilità di Classe**: Utilizzando il modello preaddestrato, il codice prevede le probabilità di classe per il frame utilizzando `**model.predict**`. La funzione di previsione restituisce un array di probabilità che rappresenta la distribuzione delle classi. La previsione ottenuta viene aggiunta alla lista `total_predictions`.

```python
# Lettura ed elaborazione delle clip del video in input da analizzare
while True:
    ret, frame = video.read()
    if not ret:
        break

    # Il video viene convertito in immagini (frame)
    image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))

    # Pre-elabora il frame in esame
    size = (224, 224)
    image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)
    image_array = np.asarray(image)
    normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1

    # Adatta la forma di input del modello per renderlo compatibile per l'analisi
    data = np.expand_dims(normalized_image_array, axis=0)
```

9. Selezione delle Migliori Previsioni:

Dopo l'elaborazione di tutti i frame, il codice decide se prendere in considerazione tutti i frame o solo una selezione di essi. Se il video contiene meno di 10 frame, vengono considerate tutte le previsioni come le migliori, altrimenti, viene effettuata una selezione dei 10 frame con la maggiore confidenza. Viene utilizzata la funzione `**np.argsort**` per ottenere gli indici dei frame ordinati in base alla confidenza delle previsioni. Vengono selezionati i frame corrispondenti agli ultimi 10 indici, che corrispondono alle previsioni con la maggiore confidenza. Le previsioni corrispondenti ai frame selezionati vengono memorizzate nella lista `**top_predictions**`.

10. Calcolo della Media delle Previsioni:

Viene calcolata la media delle previsioni dei frame selezionati utilizzando `**np.mean**`. La media delle previsioni viene memorizzata nella variabile `**average_prediction**`.

```python
    # Effettua predizione e successivamente l'aggiunge alla lista
    prediction = model.predict(data)
    total_predictions.append(prediction)

# Se ci dovessere essere meno di 10 frame nel video, prende tutti i frame
if len(total_predictions) < 10:
    top_predictions = total_predictions
else:
    # Altrimenti trova gli indici dei primi 10 frame con la maggiore confidenza
    top_indices = np.argsort([p[0][np.argmax(p)] for p in total_predictions])[-10:]
    # Seleziona i frame corrispondenti agli indici delle prima 10 posizioni sopracitate
    top_predictions = [total_predictions[i] for i in top_indices]

# Calcola la media delle predizioni
average_prediction = np.mean(top_predictions, axis=0)
```

11. Determinazione della Classe Prevista e del Punteggio di Confidenza:

Utilizzando la media delle previsioni, il codice utilizza la funzione `**np.argmax**` per ottenere l'indice della classe prevista con il punteggio di confidenza più alto. La classe prevista viene ottenuta dalla lista `**class_names**` utilizzando l'indice calcolato. Il punteggio di confidenza viene ottenuto dalla previsione media utilizzando l'indice calcolato. La classe prevista e il punteggio di confidenza vengono memorizzati nelle variabili `**class_name**` e `**confidence_score**`, rispettivamente.

12. Stampa dei Risultati:

Alla fine del processo, il codice stampa a schermo la classe prevista e il punteggio di confidenza per il video analizzato. La classe prevista viene stampata rimuovendo i primi due caratteri (`**class_name[2:]**`) per eliminare eventuali spazi o caratteri di formattazione.

13. Rilascio delle Risorse:

Alla fine dell'analisi del video, il codice rilascia le risorse utilizzate. L'oggetto `video` viene rilasciato utilizzando `**video.release()**`. Eventuali finestre aperte vengono chiuse utilizzando `**cv2.destroyAllWindows()**`.

```
# Ottieni l'indice della classe prevista e il punteggio di confidenza dalla previsione media
index = np.argmax(average_prediction)
class_name = class_names[index]
confidence_score = average_prediction[0][index]

# Stampa a video la previsione complessiva e il punteggio di confidenza per il video
print("Classe (Media):", class_name[2:])
print("Punteggio di Confidenza (Media):", confidence_score)

# Rilascia l'oggetto di acquisizione video e chiude tutte le finestre aperte
video.release()
cv2.destroyAllWindows()
```

Conclusione

La documentazione progettuale fornita rappresenta un'importante risorsa per comprendere in modo completo e approfondito il processo di realizzazione del caso di studio e del codice per l'analisi dei video di interazioni umane. Ogni fase del flusso di lavoro è stata descritta in dettaglio, consentendo agli utenti di acquisire una conoscenza approfondita delle operazioni eseguite.

Questa documentazione offre una serie di vantaggi significativi. Innanzitutto, fornisce una base solida per gli utenti che desiderano comprendere il funzionamento del codice e del sistema nel suo complesso. Grazie alle spiegazioni dettagliate e accurate, gli utenti possono acquisire una visione chiara di come sono state implementate le diverse funzionalità e come queste si integrano per raggiungere gli obiettivi desiderati.

Inoltre, la documentazione offre agli utenti la flessibilità di apportare modifiche o estensioni al codice. Grazie alla comprensione approfondita delle operazioni e dei flussi di dati, gli utenti possono personalizzare il sistema per soddisfare le proprie esigenze specifiche. Ad esempio, possono aggiungere nuovi esempi di iterazioni umane per ampliare il raggio di analisi o adattare il codice a nuovi contesti di utilizzo.

La documentazione progettuale contribuisce quindi a garantire la sostenibilità e la manutenibilità del sistema nel lungo periodo. Gli utenti possono fare affidamento sulla documentazione come fonte di riferimento affidabile per comprendere il codice e implementare eventuali modifiche o estensioni in modo accurato e sicuro.