American Sign Language

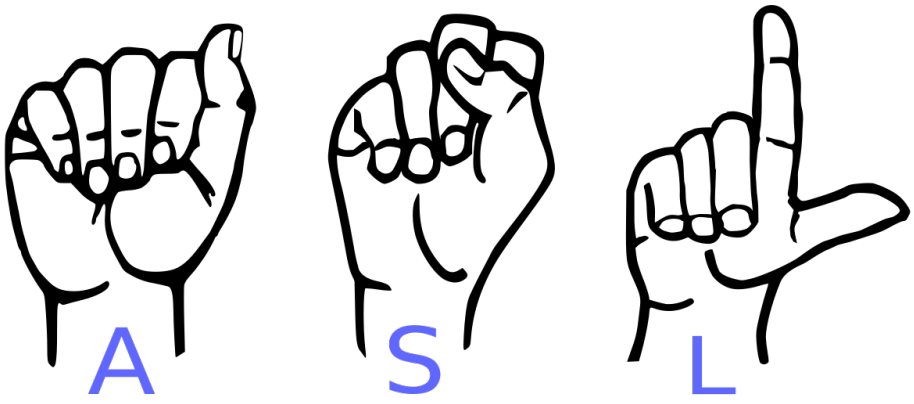
Sign Detection & Recognition

Arianna Soriani

February 12, 2024

Contents

0.1 Introduzione



0.1.1 Introduzione al Database

Il **Linguaggio dei segni** è una delle modalità di comunicazione più usata per l'interazione con persone con difficoltà a parlare o a sentire. La maggioranza delle persone, però, non è in grado di comprenderne il significato. Questo linguaggio consiste nell'assegnazione ad ogni lettera dell'alfabero ad una specifica posizione della mano. Di conseguenza, per ogni diverso alfabeto, si vengono a delineare diversi linguaggi dei segni.
Quello che viene analizzato nello specifico in questo progetto è **American Sign Language**(ASL).

0.1.2 Obiettivi

Al fine di facilitare la comunicazione tra persone e abbattere il limite della conoscenza di questo specifico linguaggio, l'obiettivo principale del progetto è quello di realizzare un sistema automatico in grado di riconoscere immagini relative al linguaggio dei segni e di tradurne il significato.
Lo scopo di questa ricerca è quindi quello di allenare un modello in grado di rilevare e riconoscere gesti e segni delle mani e di tradurli in lettere dell'alfabeto americano.

0.2 Preprocessing

Ogni progetto di Computer Vision vede una prima fase fondamentale di **preprocessing** dei dati.
L'importanza di questa fase è dovuta alla necessità di istanziare una serie di funzioni atte alla preparazione dei dati in funzione dell'algoritmo che verrà utilizzato. Ogni algoritmo di detection e recognition necessita di uno specifico formato dei dati.
A tale scopo, dunque, sono state definite alcune funzioni utili tra cui:

- *parse_txt_annot*: una funzione per convertire file txt in liste
- *create_paths_list*: una funzione per creare liste di file paths
- *creating_files*: una funzione per creare file nel formato dizionario

Ma prima di descriverne più dettagliatamente l'utilizzo, è importante capire il significato dei file txt.

Il processo di **Object Detection** restituisce una *bounding box*, l'etichetta della classe e spesso una probabilità (o score) per ogni oggetto di interesse presente nell'immagine di input.
Per ogni file (immagine) presente rispettivamente nella directory di train, validation e test si trova un corrispondente file testuale contenente le coordinate di una o più bounding box. Più precisamente, ad ogni immagine corrisponde un file con (necessariamente) lo stesso nome, ma diversa estensione (.txt). Il file testuale è suddiviso in righe e ad ogni righe corrispondono 4 valori separati da uno spazio.

- *bbox_cat*: è il primo valore (intero) e indica la classe di appartenenza dell'oggetto individuato, definito da un'opportuna funzione di mapping;
- *x*: indica l'ascissa relativa alle coordinate del centro di posizionamento della bounding box (può essere espressa in forma relativa o assoluta);
- *y*: similmente a prima, è l'ordinata relativa al centro di posizionamento della bounding box (anch'essa, relativa o assoluta);
- *width*: indica la lunghezza orizzontale relativa alla bounding box in oggetto;
- *height*: indica, invece, l'altezza della bounding box

Questo file, quindi, contiene i dati per delineare i contorni di rilevamento degli oggetti nell'immagine e la loro categoria, appartenente ad un insieme definito.

0.2.1 Parse_txt_annotation

- La funzione legge una singola immagine e il relativo file txt associato e traduce le diverse coordinate in tre array. In particolare:
  - dall'immagine: estrae le dimensioni, altezza e larghezza
  - dal file testuale: i 4 valori, per ogni riga, espresse in forma relativa

  Successivamente, calcola le coordinate assolute delle bounding box relativamente ai 4 angoli che delineano il rettangolo di rilevazione dell'oggetto nell'immagine (xmin,ymin,xmax,ymax).
  La funzione ritorna, perciò, 3 liste:
  - img_path: indirizzo dell'immagine
  - classes: lista contenente le rispettive classi rilevate all'interno del file
  - boxes: i 4 valori assoluti di tutte le bounding boxes calcolate

0.2.2 Creating_files

- La funzione consente la creazione di 3 RaggedTensor (image_paths, classes, bbox).
  Un *RaggedTensor* è un tensore con una o più dimensioni ragged (irregolari); dimensioni che possono assumere lunghezze diverse.
  Più in generale, invece, un *Tensor* è un array multidimensionale con un tipo uniforme. Come caratteristica fondamentale hanno l'immutabilità: non puoi mai aggiornare il contenuto di un tensore, ma crearne solo uno nuovo.
  Per ottenere questo, la funzione, utilizzando due liste, inizialmente create e contenenti tutti i percorsi relativi alle corrispondenti immagini e file testuali, per ciascuna immagine parsa il file txt associato da cui separa i valori di bounding box da quelli della classe.
  La funzione restituisce dunque le informazioni associate ad ogni singola immagine e relative al: percorso, classe di appartenenza e bounding box.

0.2.3 Creazione datasets

A questo punto, è possibile procedere alla definizione dei datasets in formato compatibile con l'algoritmo di Detection che verrà successivamente utilizzato.

- **Img_preprocessing**

```python
def img_preprocessing(img_path):
    img = tf.io.read_file(img_path)
    img = tf.image.decode_jpeg(img, channels = 3)
    img = tf.cast(img, tf.float32)
    return img
```

- *tf.io.read_file(img_path)*: utilizza TensorFlow per leggere il contenuto del file immagine specificato dal percorso img_path e lo memorizza nella variabile img. Questo metodo restituisce i byte del file.
- *tf.image.decode_jpeg(img, channels=3)*: decodifica l'immagine JPEG rappresentata dai byte ottenuti dalla lettura del file. Il parametro channels=3 specifica che l'immagine verrà decodificata con 3 canali di colore (RGB). Il risultato è un tensore che rappresenta l'immagine con i suoi pixel e i relativi canali di colore.
- *tf.cast(img, tf.float32)*: trasforma il tipo di dati dei pixel dell'immagine da interi (che di solito sono rappresentati da valori compresi tra 0 e 255) a float. Questa conversione è comune in molte operazioni di deep learning, poiché spesso si desidera eseguire calcoli con i pixel dell'immagine e i tipi di dati float sono più adatti per tali operazioni.

- **Resizing**

```
resizing = keras_cv.layers.JitteredResize(
    target_size=(416, 416),
    scale_factor=(0.75, 1.3),
    bounding_box_format="xyxy",
)
```

- La funzione *JitteredResize* esegue una ridimensionamento e jittering (sfocatura) casuale delle immagini in modo da aumentare la variabilità dei dati durante l'addestramento. Questo è utile per migliorare la capacità di generalizzazione del modello. I parametri principali della classe sono:
  - *target_size*: specifica la dimensione target per l'immagine ridimensionata. In questo caso, le immagini vengono ridimensionate a una dimensione di 416x416 pixel.
  - *scale_factor*: specifica il fattore di scala per il jittering casuale. Il jittering casuale viene applicato prima del ridimensionamento dell'immagine. In questo caso, il fattore di scala è impostato su (0.75,1.3), il che significa che le immagini vengono ridimensionate con un fattore di scala casuale compreso tra 0.75 e 1.3 lungo entrambi gli assi.
  - *bounding_box_format*: specifica il formato delle coordinate dei bounding box. In questo caso, è impostato su "xyxy", che indica che le coordinate dei bounding box sono rappresentate come coppie di coordinate (x_min, y_min, x_max, y_max).

  In pratica, quando si utilizza questa funzione, le immagini di input vengono sottoposte a un jittering casuale seguito da un ridimensionamento al formato specificato. Questa tecnica può essere utile per

migliorare la robustezza del modello durante l'addestramento, poiché esposta a una maggiore variabilità nei dati di input.

- **Load dataset**

```python
# loading dataset
def load_ds(img_paths, classes, bbox):
    img = img_preprocessing(img_paths)

    bounding_boxes = {
        "classes": tf.cast(classes, dtype=tf.float32),
        "boxes": bbox }

    return {"images": img, "bounding_boxes": bounding_boxes}
```

  - La funzione *Load_ds* è progettata per caricare un dataset di immagini, con associate le relative classi e bounding boxes
    - *img_preprocessing(img_paths)*: restituisce un tensore contenente l'immagine pre-elaborata.
    - *bounding_boxes = {"classes": tf.cast(classes, dtype=tf.float32), "boxes": bbox}*: costruisce un dizionario (bounding_boxes) che contiene le classi delle immagini e le relative bounding boxes. Le classi vengono convertite in float32 utilizzando tf.cast. 'classes' è un tensore delle classi corrispondenti all'immagine e 'bbox' è un tensore delle relative bounding boxes.

    La funzione restituisce un dizionario contenente l'immagine pre-elaborata (img) e le informazioni sulle bounding boxes (bounding_boxes). Questo dizionario sarà utilizzato per creare un batch di dati durante l'addestramento o la valutazione del modello.

- **Dataset loaders**

```python
train_loader = tf.data.Dataset.from_tensor_slices((train_img_paths, train_classes, train_bboxes))
train_dataset = (train_loader
                 .map(load_ds, num_parallel_calls = AUTO)
                 .shuffle(BATCH_SIZE*4)
                 .ragged_batch(BATCH_SIZE, drop_remainder = True)
                 .map(resizing, num_parallel_calls = AUTO)
                 .map(dict_to_tuple, num_parallel_calls = AUTO)
                 .prefetch(AUTO))
```

  - *train_loader*: è un oggetto (di tipo 'tf.data.Dataset') che viene creato utilizzando il metodo *from_tensor_slices*, prendendo in input una tupla di tensori ('train_img_paths', 'train_classes', 'train_bboxes'). Più

precisamente, viene creato un dataset dove ogni elemento corrisponde ad una tupla contenente un elemento da ciascun tensore.

– *train_dataset*: è il dataset di addestramento completo, ottenuto a seguito di un pre-processing su *train_loader*.
Ad esso, infatti, vengono applicate una serie di trasformazioni utilizzando specifici metodi della libreria TensorFlow: **map, shuffle, ragged_batch, e prefetch**.
In particolare:

  * *map(load_ds, num_parallel_calls = AUTO)*: applica la funzione 'load_ds' ad ogni elemento del dataset in parallelo.
    'num_parallel_calls = AUTO' indica a TensorFlow di utilizzare un numero di thread appropriato sulla base della capacità della CPU.
  * *shuffle(BATCH_SIZE*4)*: effettua uno shuffle (mescolamento) sugli elementi del dataset utilizzando una dimensione di buffer di dimensione BATCH_SIZE*4. Questa operazione è indispensabile al fine di evitare che l'algoritmo di addestramento impari correttamente sui dati.
  * *ragged_batch(BATCH_SIZE, drop_remainder = True)*: raggruppa gli elementi del dataset in batch di dimensione BATCH_SIZE, scartando eventuali elementi rimanenti se non sono sufficienti per formare un batch completo. L'opzione 'ragged_batch' permette di gestire batch di dimensioni diverse.
  * *map(resizing, num_parallel_calls = AUTO)*: applica la funzione 'resizing' ad ogni elemento del dataset in parallelo.
  * *map(dict_to_tuple, num_parallel_calls = AUTO)*: applica la funzione 'dict_to_tuple' ad ogni elemento del dataset in parallelo.
  * *prefetch(AUTO)*: carica in memoria i dati in anticipo in modo asincrono per migliorare le prestazioni dell'addestramento. Lavorare su dati presenti sulla cache offre vantaggi sui tempi di esecuzione nettamente rilevanti.

0.3 Yolo v8 Model

0.3.1 Algoritmo

YOLO, acronimo di "You Only Look Once", è un popolare framework per l'*Object Detection* in immagini e video. È noto per la sua capacità di effettuare rilevamento di oggetti multipli in tempo reale con elevata accuratezza.
L'approccio di YOLO si differenzia da molti altri metodi di object detection in quanto considera il problema come un'unica regressione diretta. Piuttosto che dividere l'immagine in varie regioni di interesse e applicare il classificatore separatamente a ciascuna regione (come nel caso di metodi basati su regione, come R-CNN), YOLO suddivide l'immagine in una griglia e predice le bounding boxes e le probabilità di classe per ciascuna cella della griglia contemporaneamente. Viene definita anche *rete single-stage*.

Le caratteristiche principali di YOLO sono:

- **Efficienza**: YOLO è noto per la sua efficienza e la sua capacità di effettuare il rilevamento di oggetti in tempo reale. Il suo approccio di "You Only Look Once" consente una *rapida* inferenza su singole immagini o flussi video.
- **Simultaneità**: YOLO effettua la predizione delle bounding boxes e delle probabilità di classe in un'unica passata attraverso la rete neurale. Questo lo rende *più veloce* rispetto ai metodi tradizionali che richiedono più passaggi per l'individuazione degli oggetti.
- **Buona generalizzazione**: YOLO generalizza bene su una vasta gamma di classi di oggetti e condizioni di illuminazione, grazie alla sua capacità di apprendere pattern spaziali e contestuali nell'immagine.
- **Open Source**: YOLO è un framework open source ed è disponibile in diverse versioni, ciascuna con miglioramenti e ottimizzazioni rispetto alla versione precedente. Nel progetto è stata utilizzata l'ultima versione, v8. Yolov8 è un algoritmo allo stato dell'arte.

YOLO è ampiamente utilizzato in applicazioni di sorveglianza, veicolari, di analisi video e in altri contesti in cui è necessario il rilevamento e la classificazione degli oggetti in tempo reale.