Classificazione Linguaggio dei Segni Americano (ASL) –
Daniela Nardone

RELAZIONE

Introduzione:

Il seguente progetto ha come obiettivo quello di sviluppare un sistema di riconoscimento del linguaggio dei segni americano (ASL) utilizzando tecniche di Deep Learning.

Il linguaggio dei segni è una forma di comunicazione utilizzata da persone con disabilità uditive che consiste in gesti fatti con le mani per rappresentare lettere, parole o frasi.

Il riconoscimento automatico dei segni attraverso immagini può avere un impatto significativo sull'inclusione sociale poiché faciliterebbe la comunicazione tra persone sorde ed udenti.

Il sistema che propongo utilizza due reti neurali convoluzionali (CNN), una creata ex-novo ed una creata sulla base di un'architettura VGG-19 preaddestrata, per riconoscere i segni dell'alfabeto ASL dalle immagini delle mani facendo utilizzo della tecnologia *Mediapipe* che estrae i punti di riferimento chiave dalle mani nelle immagini.

Processi preliminari:

(Valido per entrambe le reti)

Come prima cosa, sono state importate alcune librerie essenziali per l'elaborazione delle immagini e per la costruzione ed addestramento del modello. Tra queste, *OpenCV (cv2)* è stata utilizzata per l'elaborazione delle immagini, mentre *MediaPipe (mediapipe)* ha fornito strumenti avanzati per il rilevamento e la tracciatura dei punti di riferimento delle mani. Le librerie *matplotlib* e *IPython.display* sono state utilizzate per visualizzare grafici e immagini all'interno di un ambiente di notebook. Inoltre, sono state importate librerie come *numpy*, *pandas*, e *os* per manipolare dati e gestire i file. Per la costruzione del modello di Deep Learning, sono state utilizzate le librerie *keras* e *tensorflow*.

Successivamente, sono state definite delle funzioni per rendere più chiaro e lineare il codice. Sono state implementate:

- la funzione *extract_hand_landmarks(image_path)* che utilizza Mediapipe per identificare e disegnare i punti di riferimento chiave delle mani.
- la funzione *plot_confusion_matrix(y, y_pred)* che genera e visualizza una matrice di confusione.
- la funzione *preprocess_image(image_path)*.
- la funzione *testing(test_image_path)* che fa un test su un'immagine nuova.
- la funzione *build_model()* che si occupa della creazione di una rete neurale a partire da una rete preaddestrata. (Valido solo per la rete costruita a partire da VGG-19)

Funzioni:

(Valido per entrambe le reti)

- Funzione *extract_hand_landmarks(image_path)*:

  Inizialmente, la funzione legge l'immagine per poi convertirla in RGB al fine di facilitare il lavoro di *Mediapipe*.

  Viene poi utilizzato la funzione *mp.Hands()* di *Mediapipe* che permette di rilevare i punti di riferimento delle mani.

  Infine, se vengono trovati dei punti di riferimento, vengono disegnate le connessioni tra di essi sull'immagine e viene restituita l'immagine annotata, altrimenti viene restituito *None*.

- Funzione *plot_confusion_matrix(y, y_pred)*:

  Come prima cosa, viene definita la matrice di confusione tramite la funzione *confusion_matrix(y, y_pred)*.

  Successivamente, viene impostata una figura per visualizzare la matrice con colori nella sfumatura del viola e annotazioni, bianche o nere a seconda del contrasto con gli altri colori, per rappresentare meglio i risultati.

- Funzione *preprocess_image(image_path)*:

Questa funzione utilizza *extract_hand_landmarks(image_path)* per ottenere l'immagine con i relativi punti di riferimento identificati da *Mediapipe*.

Successivamente, se l'immagine è stata processata correttamente, viene ridimensionata a 128x128 pixel e normalizzata (con valori tra 0 e 1), altrimenti, se non vengono rilevate mani nell'immagine, restituisce *None*.

- Funzione *testing(test_image_path)*:

  Come prima cosa, questa funzione prende un'immagine dal percorso e la processa tramite la funzione *preprocess_image(image_path)*.

  Successivamente, utilizza il modello addestrato per fare la predizione.

  Infine, se viene rilevata una mano con relativi punti di riferimento, mostra l'immagine e stampa l'etichetta prevista.

  Altrimenti, se non viene rilevata nessuna mano, viene semplicemente mostrata l'immagine.

- Funzione *build_model()*:
  (Valido solo per la rete costruita a partire da VGG-19)

  Il primo passaggio è quello di caricare la rete VGG-19 come base del modello: tale caricamento comprende l'inclusione di parametri, quali *include_top=False*, *weights='imagenet'*, *input_shape=(128, 128, 3)*, che personalizzano la rete e la adattano allo specifico task.

  Il modello è personalizzato attraverso l'aggiunzione di strati alla rete di base:

  - Attraverso *base.output* si ottiene l'output degli strati convoluzionali dalla rete VGG-19.
  - L'output viene appiattito in un vettore monodimensionale da *Flatten().*
  - Vengono creati due set di livelli fully-connected, che comprendono *Dense()*, *Dropout()* e *BatchNormalization().*
  - Lo strato finale è un livello *Dense()* con 26 neuroni (uno per ogni lettera dell'ASL) con funzione di attivazione *softmax*, ideale per classificazioni multi-classe.

Successivamente, il modello viene fatto compilare con ottimizzatore *Adam*, funzione di perdita definita dalla *categorical_crossentropy* e metrica di valutazione *accuracy*.

Analisi del dataset:

(Valido per entrambe le reti)

A questo punto, da una directory specificata, vengono caricati i nomi delle cartelle che identificano anche i nomi delle classi. Ogni cartella rappresenta un segno specifico dell'ASL, a partire dalla lettera A alla lettera Z con l'aggiunta di una classe di immagini vuote (nothing), per un totale di 27 classi.

Dato che la cartella della classe 'nothing' contiene solamente immagini vuote, ho ritenuto superfluo fare il processamento dell'immagine e l'estrazione dei riferimenti chiave svolta dalla funzione *extract_hand_landmarks(image_path)* e pertanto ho deciso di eliminare tale classe dalla lista delle etichette: a questo punto, il numero di classi passa da 27 a 26.

Per verificare la correttezza del caricamento del dataset e per avere una panoramica generale dello stesso, sono state visualizzate 2 immagini per ogni classe con le relative etichette annotate al di sopra ed il conteggio delle immagini totale per ogni classe al di sotto.

Inoltre, è stato creato un grafico a barre per mostrare la distribuzione delle immagini per ciascuna classe, aiutando così ad identificare eventuali squilibri nel dataset: questa visualizzazione preliminare è stata cruciale per assicurarsi che il dataset fosse adeguato per l'addestramento del modello.

Preparazione dei dati:

(Valido per entrambe le reti)

Il passaggio successivo è stato quello di elaborare le immagini contenute nelle 26 cartelle: questo procedimento comprende l'estrazione dei riferimenti chiave delle mani attraverso l'utilizzo di *Mediapipe* e il ridimensionamento delle stesse in immagini 128x128 pixel. Queste immagini vengono poi trasformate in array e inserite nella lista *x_data[ ]*.

Un processo simile viene svolto per le etichette: difatti, estratte le etichette dai nomi delle cartelle, vengono poi aggiunte alla lista *y_data[ ]*.

Per verificare la correttezza dei procedimenti appena svolti, vengono stampati i rispettivi valori numerici delle variabili *x_data[ ]* e *y_data[ ]*.

Per avere ancora più sicurezza che tutto il procedimento sia stato svolto correttamente, viene poi mostrata un'immagine per classe con i relativi *landmarks* (ovvero i riferimenti chiave delle mani).

A questo punto, la lista *x_data[ ]* viene normalizzata, ovvero ogni valore viene diviso per 255 in modo tale da ottenere valori tra 0 e 1; tale meccanica è cruciale per la stabilità del processo di addestramento della rete neurale.

Per quanto riguarda la lista *y_data[ ]*, le etichette contenute in essa vengono convertite in formato numerico tramite l'utilizzo di *LabelEncoder* ed infine, attraverso la funzione *to_categorical*, vengono convertite in vettori *one_hot* che rendono adatti questi dati al formato necessario per una classificazione multi-classe.

Dopo la preparazione, i dati vengono miscelati casualmente per garantire che il modello non apprenda schemi particolari basati sull'ordine dei dati e, di conseguenza, per garantire l'equità dell'addestramento e successivamente divisi, attraverso la funzione *train_test_split*, in training set (67% dei dati utilizzati) e test set (rimanente 33%); questa procedura permette di addestrare il modello su un set di dati e valutarlo su un altro set per misurare le sue prestazioni.

Costruzione ed addestramento del modello:

(Valido solo per la rete costruita ex-novo)

Il centro del progetto è stato la costruzione di una rete neurale convoluzionale (CNN) utilizzando *keras*.

Il modello è stato costruito come una sequenza (*Sequential()*) di livelli convoluzionali (*Conv2D()*) seguiti da livelli di pooling (*MaxPooling2D()*) e dropout (*Dropout()*) per ridurre l'overfitting.

Alla fine, i livelli convoluzionali vengono appiattiti (*Flatten()*) e connessi a un livello denso (*Dense()*) fully-connected.

La parte finale della rete è costituita da un livello con 26 unità, corrispondenti alle lettere dell'alfabeto ASL, e un'attivazione *softmax* per la classificazione multi-classe.

Il modello è stato compilato specificandone l'ottimizzatore *Adam*, la funzione di perdita *categorical_crossentropy*, appropriata per problemi di classificazione multi-classe, e la metrica di valutazione *accuracy*.

L'addestramento è stato effettuato per 8 epoche, utilizzando una porzione dei dati di training (nello specifico il 20%) come set di validazione per monitorare le prestazioni del modello.

Costruzione ed addestramento del modello:

(Valido per la rete costruita a partire da VGG-19)

Altrettanto importante è stata anche la parte di costruzione del modello a partire da una rete preaddestrata VGG-19.

In questa parte di codice, viene chiamata la funzione *build_model()* che procede con la costruzione di un modello di rete neurale convoluzionale basato sull'architettura VGG-19, con strati aggiuntivi specifici per il compito di classificazione richiesto: il modello utilizza tale rete come base per l'estrazione delle caratteristiche rilevanti dalle immagini in input.

Attraverso *model.summary* si può avere un riepilogo dettagliato dell'architettura del modello. Il riepilogo è risultato utile per comprendere la complessità del modello e verificarne la correttezza nella costruzione.

Infine, come anche in precedenza, l'addestramento viene effettuato per 8 epoche e utilizzando il 20% dei dati di training per il monitoraggio delle prestazioni del modello.

Valutazione delle prestazioni:

(Valido per entrambe le reti)

Dopo l'addestramento, vengono generati due grafici che mostrano rispettivamente l'accuratezza e la perdita del modello durante il processo di addestramento sia sul training set che sul set di validazione.

Il modello è stato testato sui dati di test per valutare le sue prestazioni: le predizioni del modello sono state confrontate con le etichette reali per calcolare l'accuratezza complessiva e sono stati generati report di classificazione per misurare *precision*, *recall* e *f1-score* per ogni classe.

Inoltre, viene visualizzata una matrice di confusione per analizzare quali classi vengono più spesso confuse tra loro.

Tutti questi strumenti di valutazione sono stati utili a comprendere meglio le performance del modello e quali aspetti potrebbero essere ulteriormente ottimizzati.

Testing:

(Valido per entrambe le reti)

Come ultimo passaggio, per verificare ulteriormente l'efficacia del modello, è stato eseguito un test utilizzando tre immagini esterne rappresentanti la lettera “A”, la lettera “V” ed un’immagine vuota.

La funzione di test ha preprocessato le immagini, estratto i punti di riferimento delle mani e utilizzato il modello per fare una predizione.

Il risultato ha mostrato la capacità del modello di generalizzare e riconoscere correttamente i segni dell'ASL anche su dati che non facevano parte del dataset di addestramento, indicando che il sistema ha un buon potenziale per essere utilizzato in applicazioni reali.