Gesture Recognition for navigating web pages

Francesco Calì
Politecnico di Torino
s263433@studenti.polito.it

Daniele Pace
Politecnico di Torino
s265389@studenti.polito.it

Pietro Pingitore
Politecnico di Torino
s263063@studenti.polito.it

Abstract

Negli ultimi anni l'attenzione verso le interfacce umane è aumentata esponenzialmente vista l'immediatezza del mezzo. Il modello da noi proposto ha l'obiettivo di riconoscere i gesti delle mani dell'utente, catturati attraverso una webcam, per rendere possibile la navigazione all'interno di pagine web. Il riconoscimento di oggetti in real-time è notoriamente un task complesso, tra i possibili approcci abbiamo deciso di utilizzare la object detection per riconoscere i vari gesti. Per raggiungere risultati accettabili in termini di velocità di risposta e accuratezza è stato necessario trovare un compromesso tra la complessità del modello e la rapidità di predizione di quest'ultimo; dopo un'attenta valutazione sulle possibili architetture abbiamo deciso di utilizzare una SSD mobilenet che si è rivelata essere la migliore per i nostri obiettivi. Il dataset è stato interamente costruito da noi integrando immagini catturate da webcam con immagini generate sinteticamente tramite render fotorealistici di scene 3D.

1. Introduction

Il riconoscimento di gesti è un campo di ricerca in notevole fase di sviluppo. La Kinect è stato il primo dispositivo a rendere accessibile questo tipo di tecnologia a tutti. Nel caso del dispositivo Microsoft l'utilizzo era prettamente videoludico e quindi rivolto ad una relativa parte di popolazione. Il machine learning ha ampliato gli ambiti di applicazione del riconoscimento degli oggetti ed in particolare dei gesti, rendendoli parte della nostra quotidianità. La navigazione di pagine web è un aspetto fondamentale della vita di ognuno ed è strettamente vincolata alle periferiche (es. mouse, tastiera, joypad, ecc.) che rendono complesso e scomodo l'utilizzo da lunghe distanze. La nostra proposta ha l'obiettivo di rendere possibile la navigazione senza l'utilizzo delle periferiche citate prima ma semplicemente effettuando dei gesti che fungono da trigger per determinati eventi. In questo modo l'interazione diventa molto più intuitiva e permette una navigazione più facile anche in condizioni meno canoniche, ad esempio quando non stiamo utilizzando il computer da una scrivania o in generale quando non possiamo utilizzare un mouse. L'applicazione da noi proposta in particolare utilizza la gesture recognition per navigare l'indice di una qualsiasi pagina Wikipedia. Quando clicchiamo su una voce dell'indice veniamo immediatamente portati nel punto in cui quel paragrafo è presente nella pagina; se successivamente però vogliamo riutilizzare l'indice per raggiungere un'altra parte del documento dobbiamo necessariamente risalire in cima per poter visualizzare nuovamente l'indice e cliccare sulla voce desiderata. Con la nostra applicazione, invece, la tabella di indice rimarrà sempre presente nella pagina e sarà possibile utilizzarla con pochi gesti delle mani. I gesti da noi scelti permetteranno di:

- nascondere o visualizzare l'intero indice;
- scorrere le voci dell'indice;
- selezionare la voce scelta.

L'approccio da noi scelto è quello della object recognition: abbiamo creato un dataset etichettato, composto da foto scattate da webcam e render fotorealistici, con il quale è stata allena una rete SSD mobilenet. L'interazione con le pagine web è stata gestita tramite Selenium.

2. Data

Le classi scelte per il nostro progetto includono cinque differenti gesti, poi ampliate per permettere al modello di discernere fra quelli più simili. Le classi che definiamo "positive", ossia quelle che il modello deve correttamente riconoscere in funzione della nostra applicazione:

- pugno
- pollice in su
- pollice in giù
- due dita
- mano aperta

Dato che le ultime due classi sono facilmente confondibili con gesti simili, sono state inserite classi "negative", ossia

non utili direttamente ai fini dell'applicazione, ma necessarie affinchè il modello possa distinguere gesti simili, composte da

- tre dita
- quattro dita
- un dito
- elle
- corna

Non avendo trovato dataset pubblici contenenti i gesti da noi scelti, il dataset è stato assemblato interamente da noi. Delle circa 5000 immagini che compongono il dataset, inizialmente grande la metà, poi ampliato per includere anche le classi "negative", solo 78 sono state prese da un dataset già esistente [1], dall'università di Singapore, per il gesto "Pugno".

Figure 1. Sample dal dataset.

Circa metà delle 500 immagini di ogni classe è stata raccolta da webcam, cercando di riprodurre le diverse illuminazioni e scale che si potrebbero verificare in una situazione di uso reale, raccogliendo fotogrammi da diverse ambientazioni e punti di vista.

Per accelerare la raccolta, l'altra metà delle immagini, per ogni classe, è stata prodotta sinteticamente usando Blender come piattaforma per la modellazione 3D. Sono stati generati automaticamente gesti della mano fotorealistici che, anche qui, posavano su diversi sfondi, con diverse scale, rotazioni ed illuminazioni; tramite uno script sono state renderizzate le mani con una rotazione e scala ogni volta diversa, stabilita in maniera randomica, ripetendo il processo per ogni ambiente (i.e. illuminazione e sfondo diversi).

Tutte le immagini sono quindi state raccolte sulla piattaforma IBM Annotations, e lì annotate a mano, generando un bounding box con relativa classe per ogni immagine. Il formato scelto per incorporare le informazioni relative ai bounding box associati ad ogni immagine è stato Pascal VOC, generando un file XML per ogni immagine, contenente informazioni sul nome, la dimensione, la posizione del bounding box ,con quattro dati sulle x e le y massime e minime, e il nome della classe. Le immagini sono state raccolte lasciandole divise, con una cartella per ogni classe contenente le immagini originali più il file XML associato.

Figure 2. Immagini sintetiche in ambienti differenti.

Al dataset principale, usato interamente per il training, sono state aggiunte 40 immagini per ogni classe, usate per il testing, prese da ambienti differenti da quelli usati per il training per massimizzare l'attendibilità delle valutazioni sulla capacità del modello di generalizzare.

Le immagini sono state successivamente raccolte in un'unica cartella, con i corrispettivi xml file, per procedere con la generazione dei file in estensione .tfrecord, codifica necessaria per procedere alla fase di training con le API di Tensorflow per la object detection. È stato realizzato uno script per generare automaticamente un unico file .tfrecord: una prima funzione legge i file XML e organizza tutte le informazioni riguardo etichette e bounding box in un file csv; utilizzando le funzioni messe a disposizione da Tensorflow, le immagini sono scritte e salvate con i rispettivi dati sui bounding box, codificate come sequenza di bytes.

3. Metodi

La ricerca da noi effettuata ci ha portato alla conclusione di utilizzare una rete SSD mobilenet FPNLite 320x320. Dovendo applicare l'algoritmo di object detection all'interno di una sequenza video in real time, abbiamo ambito ad un compromesso fra velocità e precisione; il modello di rete da noi scelto ha soddisfatto pienamente le nostre ambizioni. Prima di scegliere la rete conclusiva abbiamo esplorato altre tipologie di reti che, però, non hanno soddisfatto i nostri parametri di velocità, in particolare sono state oggetto di studio le SSD ResNet e la SSD mobilenet FPNLite 640x640 che risultavano leggermente più precise di quella da noi scelta ma più lente.

Abbiamo utilizzato il modello pre-trained di SSD [2] mobilenet [3] (fig. 3) FPNLite 320x320 di Tensorflow.

Per gestire il modello abbiamo utilizzato l'API di tensorflow per l'object detection.

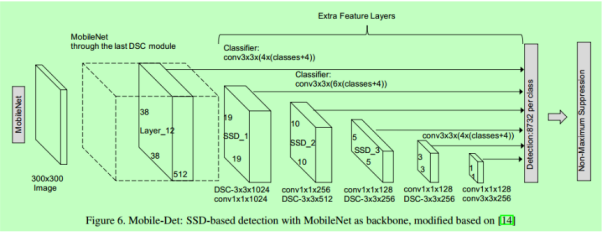

Figure 6. Mobile-Det: SSD-based detection with MobileNet as backbone, modified based on [14]

Figure 3. Schema della rete SSD.

La rete prende in input immagini di dimensione fissa 320x320 pixel. Provando l'immagine a 640x640 pixel il valore temporale risultava circa doppio e non consono all'utilizzo in real-time.

Riguardo la parte di feature extraction la funzione di attivazione da noi utilizzata è la Relu 6, che utilizza come limite massimo il valore "6", in modo tale da perdere solo un numero limitato di bits e quindi aumentare la robustezza del nostro modello vista la precisione relativamente bassa; abbiamo utilizzato la regolarizzazione di Tichonov (o L2 regularization) che penalizza il quadrato dei pesi, e fa in modo che i pesi non siano mai nulli. L'inizializzazione dei pesi è randomica con media pari a 0 e deviazione standard a 0.01.

Date le capacità di hardware limitate ed il gran numero di dati abbiamo deciso di utilizzare dei mini-batch: la grandezza di questi è stata limitata principalmente dalla RAM delle GPU a disposizione. Inizialmente settata a valori eccessivamente bassi che aumentavano esageratamente la fluttuazione della loss di train, abbiamo deciso di utilizzare dei mini-batch di 16 immagini. La fluttuazione è comunque rimasta ma ridotta, il decadimento e l'apprendimento, invece, sono migliorati notevolmente.

Per la definizione dei box abbiamo mantenuto gli iperparametri della feature extraction integrando con l'utilizzo di anchor boxes di proporzioni definite in base alla tipologia dei gesti da riconoscere: per mantenere alte prestazioni in termini di rapporto accuratezza velocità, sono stati definiti due rapporti, 1 (quadrato) e 2/3 (rettangolare verticale). Le prestazioni di similarità sono state calcolate tramite l'IoU.

Le loss utilizzate sono due: per la localizzazione la weighted smooth L1 che lavora molto bene nella box regression, mentre per la classificazione la weighted sigmoid focal che ha buone prestazioni per i task di object detection.

In fase di training abbiamo utilizzato la data augmentation in modo che il modello fosse più robusto a determinati cambiamenti, in particolare lucentezza e contrasto, dato che sono caratteristiche molto variabili in base alla webcam ed all'illuminazione della stanza.

Riguardo gli iperparametri dell'optimizer abbiamo scelto di utilizzare un learning rate di warm up basso del valore di 0.0022 per i primi 1000 steps in modo da diminuire la loss velocemente nella parte iniziale per poi aumentarlo fino a 0.005. La scelta del valore di learning rate è stata relativamente lunga ed in concomitanza con il valore di batch size.

La parte più sfidante nell'allenamento del modello è scaturita dalle prime prove dello stesso nelle quali abbiamo notato che aveva bassi valori di precision. Abbiamo risolto questo problema aumentando le classi del modello da 5 a 10. Questo aumento notevole di dati ci ha portato ad aumentare la size del batch ed a diminuire il learning rate scelto precedentemente.

Riguardo la valutazione delle performance abbiamo utilizzato delle immagini di webcam esterne al dataset di training variando la posizione delle mani e l'illuminazione delle immagini in modo da avere dei test più veritieri possibile. Abbiamo confrontato le loss di localizzazione e classificazione ed in più abbiamo creato la matrice di confusione che ci ha permesso di valutare la classificazione e la capacità della rete di discernere le varie classi. Tramite i valori di mAP (mean Average Precision) e AR (Average Recall) abbiamo valutato le prestazioni del nostro modello.

Per l'utilizzo del modello all'interno delle pagine web abbiamo utilizzato Selenium e preso come esempio il sito di Wikipedia. Indipendentemente dalla pagina di Wikipedia abbiamo fatto in modo che l'indice si ritrovi fisso all'interno della pagina e che sia navigabile attraverso i gesti delle mani da noi scelti.

Table 1. Scelta degli iperparametri.

Parametri	$Valori$
Learning Rate	0.005
Batch Size	16
Step	40,000
Classi	10

4. Esperimenti

I principali esperimenti effettuati riguardano i valori degli iperparametri come il learning rate, la batch size e il numero di step per l'allenamento della rete.

Nelle prime prove effettuate abbiamo utilizzato un learning rate piuttosto alto, dell'ordine di 10-2, ottenendo risultati accettabili ma non particolarmente buoni. La loss è scesa piuttosto velocemente e si è assestata a circa 0.3 senza riuscire più a migliorare, questo è dovuto probabilmente al fatto che un SGD con un learning rate così alto non permetteva alla funzione di costo di abbassarsi ulteriormente; questa si stabilizzava su valori che andavano da 0.3 a 0.35. Ulteriori prove sono state fatte con un learning rate con un ordine di grandezza più basso, 10-3, con risultati migliori e quindi una loss leggermente migliore. Per riuscire a migliorare ulteriormente il modello si è provato a diminuire ulteri-

ormente il learning rate, 10-5, il che ci avrebbe permesso di avere un andamento molto più lento ma la maggiore probabilità di raggiungere il minimo della funzione di costo. Tuttavia i risultati non sono stati quelli attesi: la loss non si è mai stabilizzata ma dopo un certo numero di step ha fluttuato tra 1 e 1.6 pe, la cui causa probabilmente è la presenza di un plateau per quei valori della funzione. La configurazione migliore che abbiamo trovato per quanto riguarda il learning rate è 0.005.

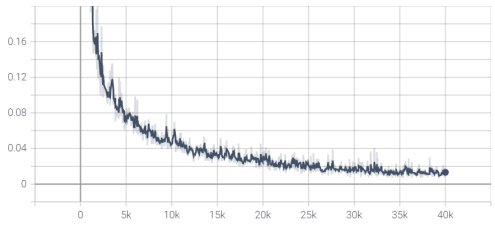

Figure 4. Classification Loss Modello Finale.

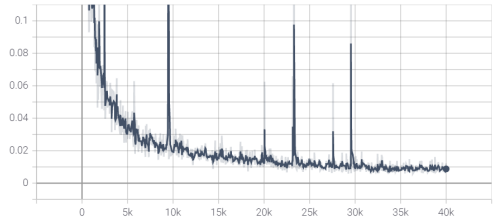

Figure 5. Localization Loss Modello Finale.

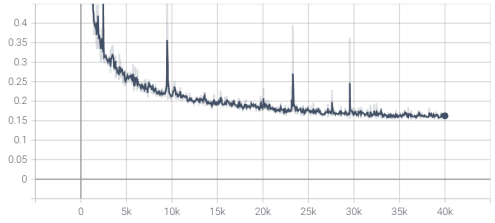

Figure 6. Loss Totale Modello Finale.

Contestualmente alla variazione del learning rate, abbiamo provato a variare il numero di step dell'allenamento. Partendo da una base di 7000, abbiamo notato che la funzione di loss non si stabilizzava del tutto e che quindi era possibile migliorare ulteriormente aumentando il numero di iterazioni. Utilizzando un learning rate molto basso abbiamo deciso di utilizzare molti step, con la speranza che la loss riuscisse a scendere il più possibile, ma, come detto prima, questo non è successo portandoci alla conclusione che un numero di step elevato risulti utile solo con una scelta di learning rate adeguato, e che dunque la durata dell'allenamento non necessariamente porti a risultati migliori. A tal proposito, durante i nostri esperimenti abbiamo notato che un buon compromesso per quanto riguarda il numero di step è di circa 20000, poiché è più o meno a quel punto che normalmente la loss si stabilizza. Nonostante ciò, le performance migliori sono state ottenute utilizzando 40000 step; si può notare infatti che, anche se di poco, la funzione di costo continua a scendere fino a 35000 step circa.

L'iperparametro che ha influito molto sulle prestazioni del modello è la batch size. Per motivi di memoria, non è stato possibile andare oltre una dimensione di 16. Come prevedibile, utilizzando valori crescenti di batch size abbiamo ottenuto risultati sempre migliori e la sensazione è che avendo potuto aumentare ulteriormente la dimensione saremmo riusciti a migliorare ulteriormente il nostro modello.

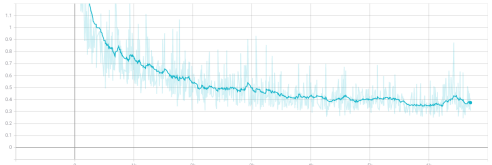

Figure 7. Classification Loss Learning Rate 0.04.

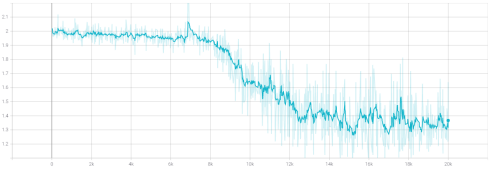

Figure 8. Classification Loss Learning Rate 0.00005.

Già dai primi modelli che abbiamo allenato, le prestazioni sembravano piuttosto accettabili ma abbiamo avuto dei problemi di falsi positivi: ad esempio, mostrando alla webcam solo 3 dita della mano, il gesto riconosciuto era quello della mano aperta. Calcolando la confusion matrix è stato confermato questo comportamento anomalo. Il problema era probabilmente dovuto al fatto che il modello non imparasse delle feature abbastanza complesse da riuscire a riconoscere le differenze tra i 2 gesti. Per ovviare a questo problema, abbiamo deciso di aumentare il numero di classi che il nostro modello era il grado di riconoscere portandole a 10, inserendo quindi anche altri gesti che avrebbero potuto creare dei problemi di falsi positivi e allenando la rete anche su questi. Successivamente, vedendo che il problema comunque persisteva, abbiamo deciso di aumentare il numero di immagini di training e questo ha permesso di raggiungere delle ottime prestazioni. Una soluzione alternativa sarebbe potuta essere cambiare modello e utilizzarne uno più complesso come la SSDResNet (RetinaNet): questo avrebbe por-

tato probabilmente a una maggiore precisione nella classificazione ma avrebbe anche aumentato il tempo di risposta del modello, ed essendo il real-time una delle componenti fondamentali della nostra applicazione abbiamo ritenuto inappropriata questa soluzione.

Table 2. Test performance.

Class. Loss	0.51
Loc. Loss	0.15
mAP	0.55
AR	0.63

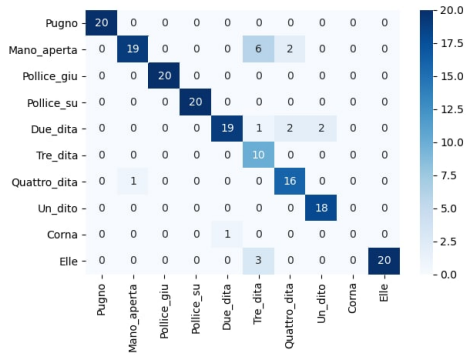

Figure 9. Confusion Matrix del modello iniziale, con falsi positivi per le tre dita. Sull'asse delle Y troviamo la classi predette, sull'asse X le classi originali, con valori espressi in termini assoluti.

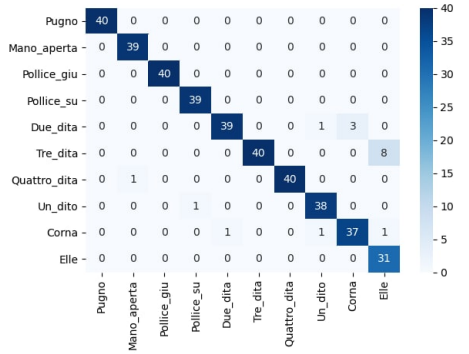

Figure 10. Confusion Matrix finale. Sull'asse delle Y troviamo la classi predette, sull'asse X le classi originali, con valori espressi in termini assoluti.

Un altro problema riscontrato è che il nostro modello risentiva in maniera abbastanza evidente delle condizioni di illuminazione con le quali si presentavano le immagini o i frame della webcam, dunque sotto certe condizioni, ad esempio un ambiente piuttosto buio o con una fonte di luce vicina all'utente, le predizioni erano molto meno accurate. Per risolvere il problema abbiamo deciso di effettuare dell'ulteriore data augmentation riguardante alcune caratteristiche dell'immagine, come la luminosità e il contrasto. Questo ci ha permesso di irrobustire la nostra applicazione e rendere meno sensibile il modello alle variazioni di luce; tuttavia il problema non è stato risolto del tutto e il motivo probabilmente è che le webcam integrate nella gran parte dei pc non hanno una risoluzione particolarmente alta e soprattutto hanno un range dinamico limitato. Per migliorare ulteriormente prestazioni si potrebbero dare al modello ulteriori train examples in condizioni di illuminazione più estreme.

Figure 11. Esempi test SSD. Per ogni coppia di immagini, sulla sinistra abbiamo la predizione della SSD, sulla destra il groud truth.

Un aspetto che ha richiesto particolare attenzione è stata la scelta di come gestire i trigger generati dalla detection di un gesto. Abbiamo notato che a volte, anche mantenendo lo stesso gesto della mano per più tempo, le detec-

tion non sono quelle attese. Se avessimo deciso di triggerare un'azione immediatamente, dopo la prima detection di un gesto, avremmo verosimilmente avuto dei problemi causati da errori nelle predizioni che, per quanto rari, sono comunque presenti. Per limitare questo problema abbiamo deciso innanzitutto di alzare la soglia di precisione, per cui una detection è accettata come buona a 0.8, e, successivamente, per rendere più robusta l'app, abbiamo impostato un controllo numerato sui frame predetti: l'azione è scatenata solo dopo un certo numero di detection uguali consecutive. Tale soglia è impostata diversamente per ogni gesto, tenendo conto della relativa azione e della precisione nel riconoscimento del gesto.

5. Conclusioni

In conclusione, possiamo dire di aver raggiunto i risultati che ci eravamo prefissati in partenza, cioè sfruttare la object detection per navigare all'interno di una pagina web. Un aspetto che ha preso molto tempo all'interno del progetto è stata la generazione del dataset. L'idea di realizzare immagini sintetiche con Blender ci ha fatto risparmiare molto tempo, tuttavia il vero collo di bottiglia è stato il processo di annotazione, che per quanto rapido e intuitivo, ha richiesto molto tempo viste le dimensioni del dataset. Un'idea per migliorare questo aspetto potrebbe essere utilizzare un'ulteriore rete che annoti per noi le nuove immagini, almeno per quanto riguarda le coordinate dei bounding box. Il nostro progetto può essere visto come un punto di partenza per un'applicazione più completa, che permetta la navigazione su altre pagine web e un'interazione diretta con il browser; l'aggiunta di ulteriori gesti risulta facile e intuitiva.

References

[1] https://www.ece.nus.edu.sg/stfpage/elepv/NUS-HandSet/.

[2] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, Stan Z. Li; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4203-4212

[3] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017