Screenshot to Text

Progetto di Machine Learning Anno Accademico 2024-2025

Matteo Galletta, Marco Gionfriddo, Kevin Speranza

Indice

Capitolo 1

Problema

Da anni ormai si tenta di risolvere il problema del riconoscimento di testi contenuti in immagine. Il problema è noto come Optical Character Recognition (OCR) e consiste nel riconoscere i caratteri di un testo contenuto in un'immagine. Il problema è complesso e presenta una serie di insidie che non sono di immediata risoluzione. Nonostante ciò, allo stato dell'arte esistono diversi algoritmi che consentono di ottenere risultati soddisfacenti. Quello che viene presentato in questo documento è un modello che mira a semplificare il problema a una sottoclasse di immagini, avendo il vantaggio di ottenere un algoritmo più leggero ed efficiente, a discapito della sua versatilità.

Capita spesso che le immagini da cui è utile estrarre il testo siano screenshot. L'algoritmo presentato si occupa di estrarre il testo contenuto in uno screenshot, indipendentemente dal font e dai colori utilizzati. In realtà, viene inizialmente affrontato il problema assumendo che lo screenshot comprenda una sola parola. Il problema viene ulteriormente semplificato ai font in stampatello e agli alfabeti italiano e latino esteso (punteggiatura compresa), escludendo il corsivo e altri alfabeti. Tramite l'uso di euristiche, si può estendere facilmente l'implementazione comprendendo frasi (purché non siano divise su più righe).

Figura 1.1: Screenshot di esempio

Capitolo 2

Soluzione proposta

La soluzione proposta è suddivisa in due fasi principali:

- **Fase 1**: Suddivisione in caratteri
- **Fase 2**: Classificazione del carattere

Per la prima fase vengono utilizzati algoritmi di image processing per suddividere la parola in caratteri. Per la seconda fase viene utilizzato un modello di deep learning per classificare i singoli caratteri.

mettere schema pipeline?

2.1 Fase 1: Suddivisione in caratteri

Prima di procedere alla suddivisione dell'immagine in singoli caratteri, vengono eseguite alcune operazioni di pre-processing fondamentali.

Innanzitutto, l'immagine viene convertita in scala di grigi, così da ridurre la complessità dell'elaborazione e operare su un unico canale di intensità luminosa. Successivamente, l'immagine in scala di grigi viene normalizzata nell'intervallo 0-255, con l'obiettivo di aumentare il contrasto tra le aree chiare e scure, migliorando così la visibilità dei dettagli.

Nel passaggio successivo viene calcolata l'intensità media dei pixel, utile per stimare la luminosità complessiva dell'immagine. Se tale valore supera una soglia prefissata, si assume che l'immagine abbia uno sfondo chiaro; in tal caso, viene applicata un'inversione dei colori, trasformando i pixel chiari in scuri e viceversa, per facilitare l'analisi visiva.

Infine, l'immagine può essere convertita in formato binario: attraverso una sogliatura, i pixel vengono trasformati in nero (0) o bianco (255). Questo rende l'immagine più adatta per successive operazioni di segmentazione o riconoscimento dei caratteri. Questa procedura è particolarmente utile perché la rete neurale utilizzata è stata addestrata su immagini con testo bianco su sfondo nero; l'euristica basata sull'intensità media permette quindi di invertire automaticamente i colori, se necessario, per uniformare l'input al formato atteso dalla rete.

Figura 2.1: Immagine dopo il Preprocessing.

Il primo approccio utilizzato per la suddivisione in caratteri è stato quello di considerare le proiezioni verticali dell'immagine. Come prima cosa si individuano le colonne in cui è presente almeno un pixel bianco. L'euristica quindi considera due colonne consecutive come appartenenti allo stesso carattere se presentano entrambe almeno un pixel bianco. Nonostante questo approccio possa sembrare ragionevole, gli esperimenti effettuati mostrano non essere efficace per immagini a bassa risoluzione. Infatti, in questo caso, i caratteri tendono a sovrapporsi e le colonne consecutive presentano pixel bianchi in comune. Per questo motivo, si è deciso di utilizzare un approccio alternativo.

Il secondo approccio utilizzato è quello di considerare le componenti connesse bianche dell'immagine. Questo metodo è più efficace, consentendo di individuare più facilmente caratteri diversi, anche se parzialmente sovrapposti. L'algoritmo non consente di individuare correttamente i caratteri non contigui, come nel caso di 'i' e 'j', che presentano un punto sopra il corpo del carattere. Un'ulteriore euristica risolve il problema in maniera efficace, andando a unire due componenti connesse se parzialmente sovrapposte orizzontalmente. Nello specifico, si prende in considerazione la componente connessa più piccola in larghezza e la si confronta con la parte in sovrapposizione con l'altra componente connessa. Se più del 30% (valore verificato sperimentalmente) della larghezza è sovrapposto, allora le componenti vengono unite. In questo modo, si riesce a ottenere un'immagine in cui ogni carattere è rappresentato da una singola componente connessa.

Una volta individuati i bounding box dei caratteri, si procede anche a calcolare un bounding box globale che racchiude tutti i caratteri. L'utilità di questo bounding box viene mostrata nella fase di classificazione.

Figura 2.2: Individuazione Bounding Box

2.2 Fase 2: Classificazione del carattere

Per la classificazione del carattere viene utilizzato un modello di deep learning. La rete è una Convolutional Neural Network: è composta da 2 layer convoluzionali, ognuno dei quali è seguito da un layer di pooling e da una funzione di attivazione ReLU. Dopo i layer convoluzionali, la rete è composta da tre layer fully connected, che producono l'output finale. Per evitare l'overfitting riscontrato sperimentalmente durante l'addestramento, viene utilizzato il dropout tra i layer fully connected. Il modello è stato addestrato su un dataset di immagini di caratteri e simboli in stampatello, con una risoluzione di 28x28 pixel. È quindi necessario ridimensionare i caratteri estratti dalla fase 1 prima di applicare l'inferenza. Il dataset viene approfondito nella sezione **??**.

Fornendo al modello esclusivamente il carattere ridimensionato, di quest'ultimo verrebbero ignorate la dimensione e la posizione all'interno della parola. Questa semplificazione causerebbe problemi nella classificazione della punteggiatura e di caratteri *confondibili*.

Senza informazioni sulla posizione, il modello non sarebbe in grado di distinguere tra ‘,‘ e ‘’‘. Inoltre, non sarebbe in grado di distinguere tra maiuscole e minuscole *confondibili*.

Per carattere *confondibile* si intende una lettera in cui la rappresentazione in stampatello minuscolo coincide con quella in stampatello maiuscolo, se ridimensionata. Ad esempio, ‘C‘ e ‘c‘ sono caratteri confondibili, così come ‘S‘ e ‘s‘, mentre ‘A‘ e ‘a‘ non lo sono. L'insieme dei caratteri confondibili maiuscoli (CI) è definito come segue:

$$\text{CI} = \{C, J, K, O, P, S, U, V, W, X, Z\}$$

Ovviamente la controparte minuscola contiene gli stessi caratteri.

2.2.1 Utilizzo del Bounding Box globale

È possibile utilizzare il bounding box globale per fornire al modello informazioni sulla posizione e la dimensione del carattere all'interno della parola. Partendo dal bounding box del carattere e da quello globale, è possibile estrarre il margine superiore e inferiore del carattere rispetto al bounding box globale. Una volta normalizzati rispetto all'altezza del bounding box globale, il margine superiore e inferiore del carattere possono essere utilizzati come due parametri aggiuntivi per il modello.

Con questo accorgimento, il modello è adesso in grado distinguere tra ',' e ''. Inoltre, nel caso della parola 'Bob', è in grado di classificare correttamente la 'o'. Questo è possibile in quanto il margine superiore della 'o' è solo presente nel caso in cui il carattere sia maiuscolo.

2.2.2 La necessità di euristiche

Nonostante quest'ultimo approccio possa sembrare efficace, non è sempre in grado di distinguere tra maiuscole e minuscole. Mostriamo il motivo attraverso un esempio e lo formalizziamo successivamente. Consideriamo due parole d'esempio:

- 'Cocco': la prima lettera non ha margine superiore e inferiore, e deve essere classificata come maiuscola.
- 'cocco': la prima lettera non ha margine superiore e inferiore, e deve essere classificata come minuscola.

Il modello non è quindi in grado di classificare correttamente i caratteri confondibili quando hanno la stessa altezza del bounding box globale. È necessario utilizzare un'euristica che, confrontando l'altezza dei vari caratteri, sia in grado di 'correggere' la forma maiuscola o minuscola del carattere.

Guardando la distribuzione dei caratteri rispetto al loro margine superiore, è possibile notare quando è possibile classificare con certezza i caratteri confondibili.

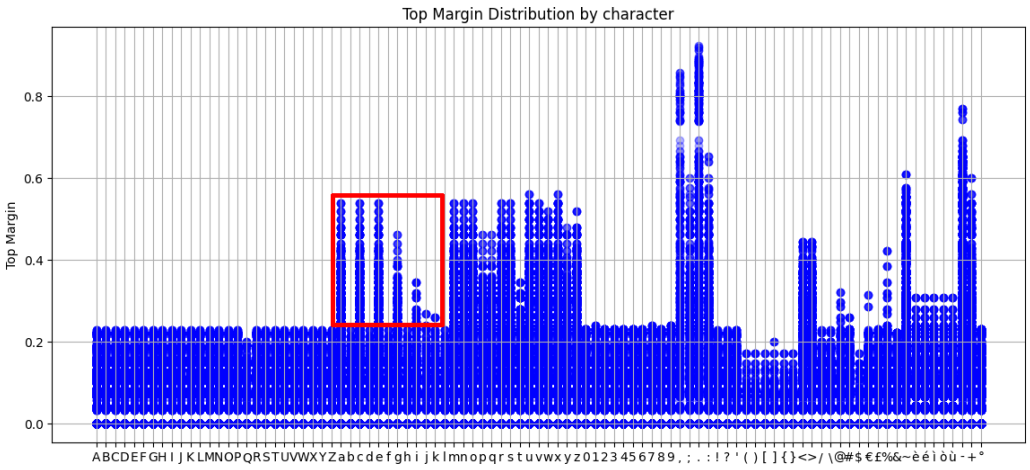

Figura 2.3: Distribuzione dei caratteri rispetto al margine superiore. Il rettangolo rosso evidenzia i caratteri confondibili identificabili con affidabilità.

Rimane comunque il fatto che parole come ‘COCCO’ e ‘cocco’ rimangono indistinguibili anche all’occhio umano, se non affiancate da altre parole che possano disambiguare.

2.2.3 L’applicazione dell’euristica

Data una determinata immagine, per carattere *affidabile* intendiamo un carattere non confondibile oppure un carattere confondibile di forma minuscola con margine superiore significativo. Un carattere è quindi affidabile quando la sua interpretazione agli occhi del modello è priva di ambiguità.

L’euristica per la correzione di maiuscole e minuscole può essere applicata solo quando è presente un carattere confondibile la cui altezza coincide con quella del bounding box globale. In questo caso, l’euristica confronta l’altezza del carattere con quella degli altri caratteri affidabili della parola. L’unico caso in cui l’altezza di un carattere confondibile coincide con quella del bounding box globale è quando non sono presenti caratteri che aumentano l’altezza del bounding box globale. Tra questi sono inclusi tutti i caratteri maiuscoli e qualche carattere minuscolo, come ‘b‘, ‘d‘ e ‘h‘. Purtroppo però nella pratica le cose non funzionano. Questo avviene in quanto in ogni font i caratteri minuscoli hanno una proporzione di altezza diversa rispetto ai caratteri maiuscoli. Per questo motivo, l’euristica viene applicata sempre, se possibile, ovvero se è presente almeno un carattere affidabile.

Capitolo 3

Dataset

Essendo il problema dell'OCR uno dei più studiati in ambito di Computer Vision, esistono diversi dataset pubblici che possono essere utilizzati per addestrare e testare i modelli. Tuttavia, la maggior parte di questi dataset sono stati creati per risolvere problemi generali e non sono specifici per il riconoscimento di testi contenuti in screenshot. Per questo motivo, è stato necessario creare una coppia di dataset ad hoc per il problema in questione.

In particolare, essendo l'algoritmo diviso in due fasi, avere due dataset distinti consente di poter valutare in modo individuale ognuna delle due fasi, consentendo di valutare l'accuratezza del modello in modo più preciso. Il primo dataset è composto da immagini di singoli simboli, che consente di testare esclusivamente la fase di classificazione del singolo carattere. Il secondo dataset invece contiene immagini di screenshot contenenti una sequenza di simboli casuali. Questo consente di testare la pipeline nella sua interezza, comprendendo sia la suddivisione in caratteri che la classificazione del singolo carattere.

I caratteri considerati nel dataset comprendono quelli dell'alfabeto latino in stampatello maiuscolo e minuscolo, le cifre da 0 a 9 e i seguenti simboli di punteggiatura:

, ; . : ! ? ’ ( ) [ ] { } < > / \ @ # $ € £ % & ˜ à è é ì ò ù - + °

In totale, il dataset contiene 96 classi.

3.1 Sintetizzazione del dataset

Capitolo 4

Metodi

Capitolo 5

Valutazione

Capitolo 6

Esperimenti

Capitolo 7

Demo

Capitolo 8

Codice

Capitolo 9

Conclusione