Screenshot to Text

Progetto di Machine Learning Anno Accademico 2024-2025

Matteo Galletta, Marco Gionfriddo, Kevin Speranza

Indice

1 Problema **1**

2 Soluzione proposta **2**
2.1 Fase 1: Suddivisione in caratteri 2
2.2 Fase 2: Classificazione del carattere 4
2.2.1 Utilizzo del Bounding Box globale 4
2.2.2 La necessità di euristiche 5
2.2.3 L'applicazione dell'euristica 6
2.2.4 Spazi e classi aggiuntive 6

3 Dataset **7**
3.1 Dataset dei simboli singoli 7
3.1.1 Permutazioni di margini 8
3.1.2 Struttura del dataset 9
3.2 Dataset degli screenshot 10

4 Modello di Classificazione **11**

5 Metodi **12**

6 Valutazione **13**

7 Demo **14**

8 Codice **16**

9 Conclusione **17**

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.

Innanzitutto, l'immagine viene convertita in scala di grigi, così da operare su un unico canale. Successivamente, l'immagine viene normalizzata nell'intervallo 0-255, aumentando così il contrasto tra le aree chiare e scure, migliorando 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. 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. Infine, l'immagine viene

convertita in formato binario: attraverso un'operazione di thresholding, i pixel vengono trasformati in nero (0) o bianco (255).

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. 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 dedicata.

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.

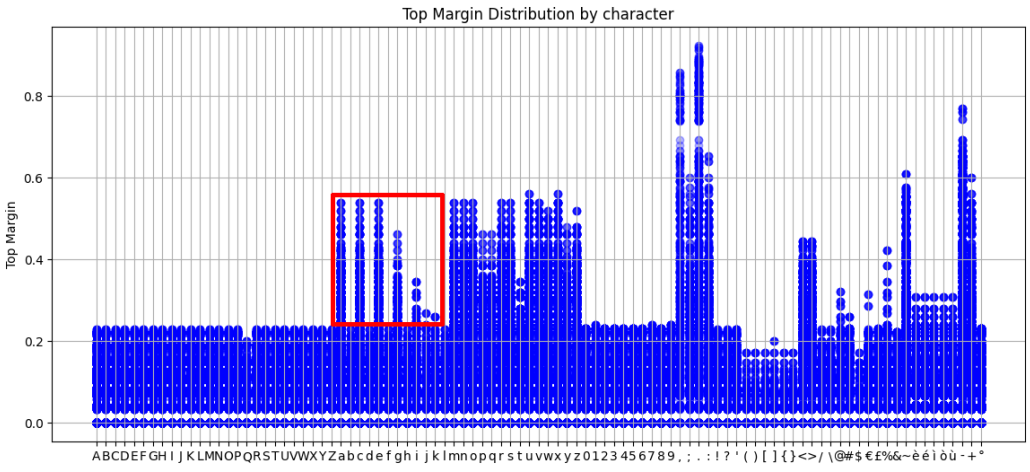

Figura 2.3: Distribuzione dei caratteri rispetto al margine superiore. Il rettangolo rosso evidenzia i caratteri confondibili identificabili con affidabilità.

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.

2.2.4 Spazi e classi aggiuntive

La fase 1, utilizzando le componenti connesse, non è in grado di classificare direttamente gli spazi. Per poterli individuare, è necessario utilizzare delle euristiche. Si considera lo spazio tra i vari caratteri e sulla base di questo si decide se aggiungere uno spazio o meno. In particolare, viene inserito uno spazio se la distanza tra due caratteri è superiore al 60% della differenza tra la distanza massima e la minima tra i caratteri. Inoltre, le virgolette, essendo composte da due componenti connesse, vengono classificate artificialmente andando a unire una sequenza di due apostrofi consecutivi.

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.

I due dataset creati sono i seguenti:

- **Dataset dei simboli singoli**: contiene immagini di singoli simboli, che consente di testare esclusivamente la fase di classificazione del singolo carattere.
- **Dataset degli screenshot**: 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.

3.1 Dataset dei simboli singoli

Questo dataset è utilizzato per addestrare il modello di classificazione del singolo carattere. Viene inoltre utilizzato per estrarre delle metriche di valutazione della fase di classificazione, in modo da poter valutare l'accuratezza del modello.

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.

Sono stati presi in considerazione 58 font differenti, ognuno con caratteristiche diverse (ad esempio, grassetto, sottile, ecc.).

È stata generata una immagine per ogni simbolo, per ogni font. La dimensione considerata è di 28x28 pixel, in modo da essere compatibile con il modello di classificazione utilizzato nella fase 2.

Per la generazione di ogni immagine è stato utilizzato Pillow, una libreria Python per la manipolazione delle immagini. Ogni immagine è stata generata con uno sfondo nero e il simbolo in bianco (con scala di grigi per i bordi). Per come vengono estratti i caratteri dall'immagine da classificare, è necessario che il simbolo nel dataset non abbia margini e che il suo bounding box coincida con il quadro dell'immagine. Per garantire questo vincolo, l'immagine viene inizialmente generata in un'immagine di dimensioni più grandi, successivamente il simbolo viene centrato all'interno dell'immagine. L'immagine viene dunque ritagliata e ridimensionata in modo da ottenere un'immagine di dimensioni 28x28 pixel.

Figura 3.1: Esempio di simboli singoli del dataset.

3.1.1 Permutazioni di margini

Come menzionato precedentemente, il modello di classificazione utilizza, oltre al bounding box del simbolo, anche i margini superiore e inferiore del simbolo rispetto al bounding box globale dell'intera parola. Per questo motivo è necessario che il dataset contenga questa informazione per far sì che il modello possa apprendere le relazioni tra il simbolo e il bounding box globale.

L'approccio utilizzato è il seguente, per ogni font:

- Si scelgono dimensione di font e quadro dell'immagine di tali valori da consentire a qualsiasi simbolo stampato di entrare all'interno dell'immagine.
- Si genera un'immagine quadrata per ogni simbolo, con il simbolo centrato all'interno dell'immagine.
- Si calcolano, per ogni simbolo, i margini superiore e inferiore del simbolo rispetto al quadro dell'immagine.
- Si normalizzano i margini rispetto all'altezza del quadro dell'immagine, in modo da ottenere valori compresi tra 0 e 1.
- Vengono create delle permutazioni dei margini, in modo che nel dataset sia presente ogni possibile combinazione di margini per ogni simbolo.

Il dataset finale conterrà quindi un simbolo per ogni font, con le varie combinazioni di margini. Nel caso dei font considerati, il dataset contiene un totale di 179292 tuple.

È importante notare che a causa di questa ultima aggiunta il dataset dei simboli singoli non rimane perfettamente bilanciato: non tutte le classi (simboli) sono rappresentate dallo stesso numero di campioni. Ad esempio, le parentesi tonde (`(` e `)`) sono presenti con circa 600 campioni ciascuna, mentre il trattino (−) conta circa 5000 campioni.

Il dataset è stato suddiviso in due sottoinsiemi: train e test. La suddivisione è stata effettuata secondo una proporzione 75% per il training e 25% per il test, necessario per la valutazione delle prestazioni del modello.

3.1.2 Struttura del dataset

Il dataset si compone di una serie di cartelle, una per ogni font, contenenti le immagini dei simboli. Ogni cartella è denominata con il nome del font e contiene le immagini dei simboli in formato PNG. I nomi delle immagini contengono il nome del simbolo rappresentato. Sullo stesso livello della cartella del font, sono presenti due file CSV: uno per il training e uno per il test. Questi file contengono l'elenco dei campioni, con la seguente struttura:

(font, immagine, simbolo, margine_superiore (%), margine_inferiore (%))

Nel sorgente del progetto è presente un notebook che mostra come importare il dataset rapidamente in formato PyTorch.

3.2 Dataset degli screenshot

Il dataset degli screenshot è stato creato per testare l'intera pipeline, dalla suddivisione in caratteri alla classificazione del singolo carattere. Il suo unico scopo è quello di testare l'accuratezza della pipeline per poterne valutare le prestazioni.

Il dataset contiene immagini di screenshot contenenti una sequenza di 10 simboli casuali. In aggiunta ai simboli considerati nel dataset dei simboli singoli, sono stati aggiunti i caratteri "spazio" e "virgolette" (‘"‘), in modo da poter testare le euristiche relative al riconoscimento di questi caratteri.

Per la generazione di ogni immagine viene utilizzato un approccio simile a quello utilizzato per il dataset dei simboli singoli. Viene utilizzato Pillow per generare gli screenshot sintetici, scegliando un font casuale ed estraendo randomicamente 10 simboli da concatenare. Per aumentare la precisione delle metriche, vengono inoltre selezionati dei colori casuali per il testo e lo sfondo, in modo da simulare screenshot reali e verificare che le operazioni di pre-processing siano efficaci.

Capitolo 4

Modello di Classificazione

Per la classificazione dei caratteri estratti dalla fase 1, è stato sviluppato 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.

img schema + numero epoche

Capitolo 5

Metodi

Capitolo 6

Valutazione

Capitolo 7

Demo

Per illustrare il funzionamento dell'algoritmo insieme ai suoi principali passaggi, è stata realizzata una demo utilizzando la libreria Python `Gradio`, che consente la creazione di applicazioni web semplici ed intuitive per modelli di machine learning e intelligenza artificiale, in poche righe di codice.

L'interfaccia guida l'utente attraverso tre fasi fondamentali: caricamento dell'immagine, pre-processing e riconoscimento (OCR).

Nella prima fase, l'utente può caricare uno screenshot da elaborare, tramite upload oppure incollandolo direttamente dagli appunti.

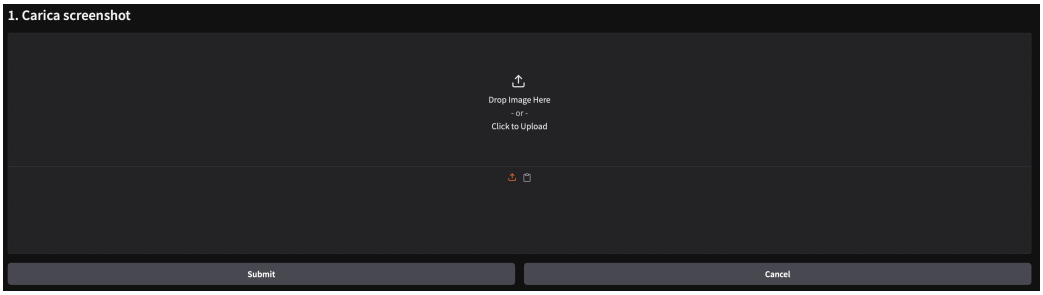

Figura 7.1: Interfaccia di caricamento dell'immagine nella demo.

Una volta cliccato il pulsante `Submit`, l'immagine viene passata alla funzione `process_image`, che la converte in un array NumPy e la fornisce come input a un oggetto `ImageToStringClassifier`. All'interno di quest'ultimo vengono eseguiti i passaggi di pre-processing e successivamente l'OCR.

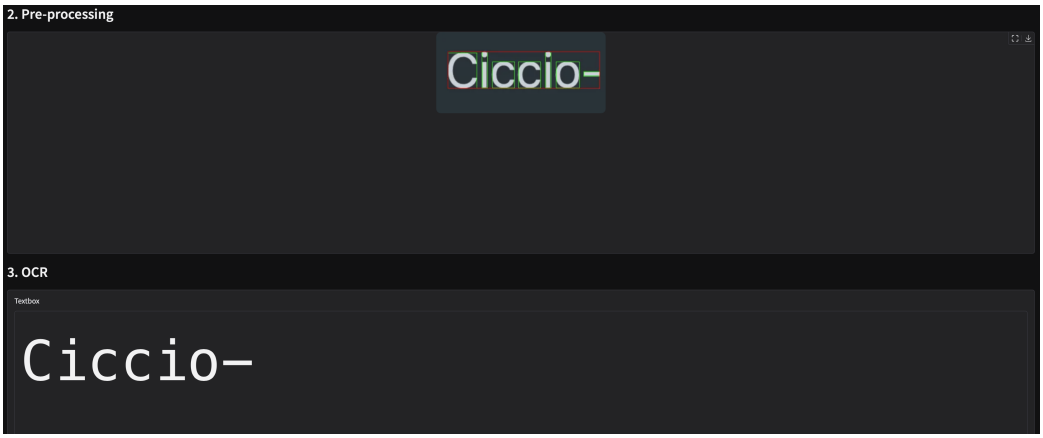

Figura 7.2: Output dell'immagine preprocessata e del testo riconosciuto.

In output, l'applicazione restituisce due risultati: l'immagine annotata con i bounding box rilevati durante il pre-processing (sovrapposti all'immagine originale) e il testo riconosciuto.

Capitolo 8

Codice

Capitolo 9

Conclusione