POLITECNICO DI TORINO

a.a. 2019/2020

LAUREA MAGISTRALE INGEGNERIA INFORMATICA
(COMPUTER ENGINEERING)

CAMERA CALCULATOR

Image Processing and Computer Vision

STUDENTI

Paolo Gallo (s257428)
Ilaria Gioda (s257312)

INTRODUZIONE ALL'APPLICAZIONE

L'obiettivo del progetto *Camera Calculator* è stato la realizzazione di un software di "calcolatrice visiva" che, a partire da un'immagine, un flusso video pre-registrato o catturato in tempo reale (e.g. da una webcam) in cui viene inquadrato un foglio di carta, fosse in grado di rilevare cifre e simboli rappresentanti un'espressione aritmetica contenente gli operatori elementari (+, -, *, /), risolverla e mostrarne il risultato all'utente in forma grafica.
Il sistema da implementare ha quindi previsto una parte di image processing, nella quale viene identificato il momento in cui l'utente ha finito di scrivere e vengono localizzati ed isolati i simboli dell'espressione, seguita poi da un classificatore (rete neurale) in grado di riconoscere i numeri e gli operatori ed infine la calcolatrice vera e propria che effettua il parsing dell'espressione e ne calcola il risultato.

STRUTTURA E FILE DEL PROGETTO

- **`camera-calculator.py`**: script di lancio dell'applicazione che definisce un'interfaccia da linea di comando per poter specificare alcuni parametri di esecuzione, utilizzati poi per avviare `cameraprocessor.py`
- **`cameraprocessor.py`**: modulo in grado di processare un input multimediale (immagine o video) e, tramite tecniche di Computer Vision, individuare ed isolare i simboli dell'espressione matematica scritti sul foglio
- **`calculator.py`**: modulo che implementa la calcolatrice vera e propria, effettuando il parsing di una sequenza di simboli e restituendo il risultato delle operazioni matematiche
- **`utils.py`**: definisce alcune funzioni di supporto usate dagli altri file del progetto
- **`multimedia.py`**: contiene il codice delle classi `InputMedia` e `MediaPlayer`, utili a generalizzare ed unificare la gestione di immagini, file video e webcam (sia in input che in output) da parte dell'applicazione, con una particolare attenzione alle performance del playback video
- **`neuralnetwork.py`**: classificatore basato su rete neurale, utilizzato per predire i simboli aritmetici (cifre e operatori) a partire da immagini ritagliate dal frame originale
- **`net\`**
  - **`neuralnetwork_training.py/.ipynb`**: script utilizzato per eseguire il training della rete neurale, riportato per completezza anche se non necessario per l'esecuzione dell'applicazione
  - **`NN.pth`**: file contenente la rete neurale prodotta in fase di training
- **`images\`**
  - contiene immagini utilizzate per il testing dell'applicazione
- **`videos\`**
  - contiene file video (registrati da smartphone o webcam) usati per testare l'applicazione in diversi scenari (scrittura frettolosa, presenza di oggetti nell'inquadratura, movimento della mano…)

UTILIZZO DELL'APPLICAZIONE

Camera Calculator, nella sua forma attuale, è un'applicazione avviabile tramite linea di comando lanciando lo script `camera-calculator.py` con gli opportuni parametri, come indicato di seguito:

```
camera-calculator.py [-h] -t {image,video,webcam} -p PATH
```

Parametri:

- `-h/--help`: mostra un messaggio di guida all'utilizzo della linea di comando
- `-t/--type {image,video,webcam}`: permette di selezionare la tipologia di sorgente multimediale su cui eseguire il riconoscimento di espressioni aritmetiche
- `-p/--path PATH`: consente di specificare il path della sorgente multimediale che si vuole utilizzare, nel caso di webcam è necessario specificare l'indice intero della camera secondo la numerazione di OpenCV

Nel caso si scelga come input un'immagine, il processamento della stessa avviene all'istante e l'applicazione mostra subito il risultato sullo schermo; nel caso di file video o di uso di una webcam, il software lavora invece in tempo reale, aspettando che l'utente abbia finito di scrivere prima di avviare il riconoscimento dei simboli e di produrre un risultato.

Al termine dell'esecuzione, per uscire dall'applicazione è sufficiente premere il tasto ESC oppure chiudere la finestra principale facendo clic sulla relativa X.

RETE NEURALE

RETE UTILIZZATA

Avendo la necessità di riconoscere i simboli che compongono l'espressione matematica scritta dall'utente, si è deciso di fare uso di una rete neurale. La scelta è ricaduta su AlexNet, una rete neurale convoluzionale per la classificazione di immagini opportunamente modificata (adattando l'ultimo layer al numero di etichette di classe desiderato) e allenata.

DATASET

Il dataset utilizzato per il training della rete neurale è stato creato a partire da quello disponibile al seguente link:

https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset

Tale dataset risulta suddiviso in due sottoinsiemi di immagini, *train* e *eval*, usati rispettivamente come training set e validation set; inoltre, solo 15 classi di simboli del dataset originale sono state mantenute e opportunamente rinominate ('*0*', '*1*', '*2*', '*3*', '*4*', '*5*', '*6*', '*7*', '*8*', '*9*', '*div*', '*equal*', '*minus*', '*mul*', '*plus*'). Molte delle immagini di tale dataset sono state poi rimosse (in quanto di scarsa qualità e leggibilità) e rimpiazzate da altrettante, contenenti simboli scritti a mano e create apposta per l'occasione.

Per questioni implementative, si è scelto di scartare anche la classe "*decimal val*": inizialmente il separatore decimale era incluso nel progetto come sedicesima classe, ma al

momento dell'esecuzione dell'applicazione sono emerse diverse problematiche dovute alla rimozione dei punti di rumore. Avendo scelto di identificare (e rimuovere) i punti di rumore tramite una strategia basata sulla dimensione dei relativi rettangoli, si ricadeva nella situazione in cui, in base al valore di dimensione minima impostato per considerare "valido" un simbolo, molti punti di rumore non venivano scartati e venivano erroneamente riconosciuti come appartenenti alla classe "*decimal val*" o, dall'altra parte, si arrivava ad identificare come outliers anche veri separatori decimali, portando quindi all'eliminazione di simboli "corretti" senza effettuarne il riconoscimento con la rete. Essendo troppo variabile da caso a caso e non potendo imporre troppe condizioni sulla dimensione degli outliers, si è scelto quindi di scartare questa classe di simboli problematica.

TRAINING DELLA RETE

La rete AlexNet accetta al suo ingresso solamente immagini quadrate, per cui l'input (già convertito in RGB) viene trasformato mediante la funzione `get_squared_img()`, dopodichè l'immagine viene salvata in una classe `Symbols(VisionDataset)` appositamente ideata. Questa classe mantiene al suo interno un array composto da coppie (PIL image, indice dell'etichetta di classe) e, quando richiesto, restituisce tali coppie trasformando opportunamente l'immagine mediante una funzione `transform()`. Infine, scelta particolare è stata quella di aggiungere una tecnica di regolarizzazione, chiamata *data augmentation*, effettuando una modifica al dataset a runtime: ad ogni epoca il dataloader applica all'immagine di input una rotazione casuale di un angolo nel range (-15°, +15°) in modo tale da introdurre più variazioni dei dati di training.

FUNZIONAMENTO DELL'APPLICAZIONE

INPUT / OUTPUT DEI DATI MULTIMEDIALI

In fase di ideazione del progetto, si è deciso di sviluppare l'applicativo in maniera tale che potesse lavorare indifferentemente con immagini, video oppure con la cattura in tempo reale mediante webcam. Per ottenere questo risultato in modo conveniente, è stata sviluppata una classe `InputMedia` contenente dei metodi di apertura, chiusura, lettura dati e controllo dello stato di una risorsa multimediale indipendentemente dalla sua natura, così da snellire il resto del codice nascondendo questi dettagli al suo interno.
Un'ulteriore difficoltà riscontrata lavorando su flussi video di cui effettuare il playback in tempo reale è stata la scarsa performance derivante dalla gestione dei video da parte di OpenCV, la quale non permetteva all'applicazione di raggiungere un playback fluido a 30 fotogrammi/secondo. Dopo alcune ricerche, è emerso che questa situazione fosse dovuta all'utilizzo di un singolo thread per l'esecuzione delle operazioni di I/O, oltre che al processamento dei fotogrammi; è stata quindi sviluppata una classe `MediaPlayer` per gestire la visualizzazione delle immagini processate che sfruttasse, per il playback video, un thread secondario dedicato al prelevamento dei frame da una coda (riempita dal thread principale man mano che questi vengono processati) e alla loro visualizzazione ad intervalli regolari tramite la funzione `cv.imshow()`. Questa soluzione permette all'applicazione di mantenere un playback fluido a 30FPS (o più) in modo semplice ed efficace.

RILEVAMENTO DELLA MANO (funzione: `detect_hand`)

Nel caso in cui l'opzione di lancio dell'applicazione scelta risulti '*video*' oppure '*webcam*', la prima operazione che viene effettuata dal programma è il rilevamento della mano all'interno del frame: tale elemento viene usato dal programma come indicatore per decidere quando iniziare l'identificazione dell'espressione matematica.
La presenza/assenza della mano all'interno del frame viene rilevata mediante la funzione `detect_hand()`: operando in HSV, essa si occupa di individuare in ciascun frame gli oggetti che presentano un colore simile a quello della pelle umana attraverso l'utilizzo della funzione di sogliatura `cv.inRange()` e, successivamente, effettuare le seguenti operazioni:

- applicazione degli operatori morfologici di chiusura (prima) e di apertura (poi) per migliorare l'edge detection degli oggetti color pelle
- individuazione dei loro contorni mediante la funzione `cv.findContours()`
- visualizzazione (all'utente) di un unico rettangolo contenente tutti gli oggetti color pelle identificati nel frame, attraverso la funzione `cv.rectangle()`

INDIVIDUAZIONE DEI SIMBOLI (funzione: `detect_symbols`)

RILEVAZIONE DEI CONTORNI

Il primo step necessario ad isolare ciascuno dei simboli matematici nel frame è la rilevazione dei loro contorni: questo viene ottenuto applicando all'immagine (convertita in grayscale e modificata tramite un filtro 9x9 di sfocatura gaussiana) una sogliatura di tipo `cv.adaptiveThreshold()`, utilizzando poi l'algoritmo di Canny per la rilevazione dei bordi ed, infine, individuando i contorni dei caratteri tramite la funzione `cv.findContours()`. Infine, prima di passare allo step successivo, viene estratto il bounding box rettangolare di ciascun contorno grazie alla funzione `cv.boundingRect()`.

POST-PROCESSING DEI RETTANGOLI

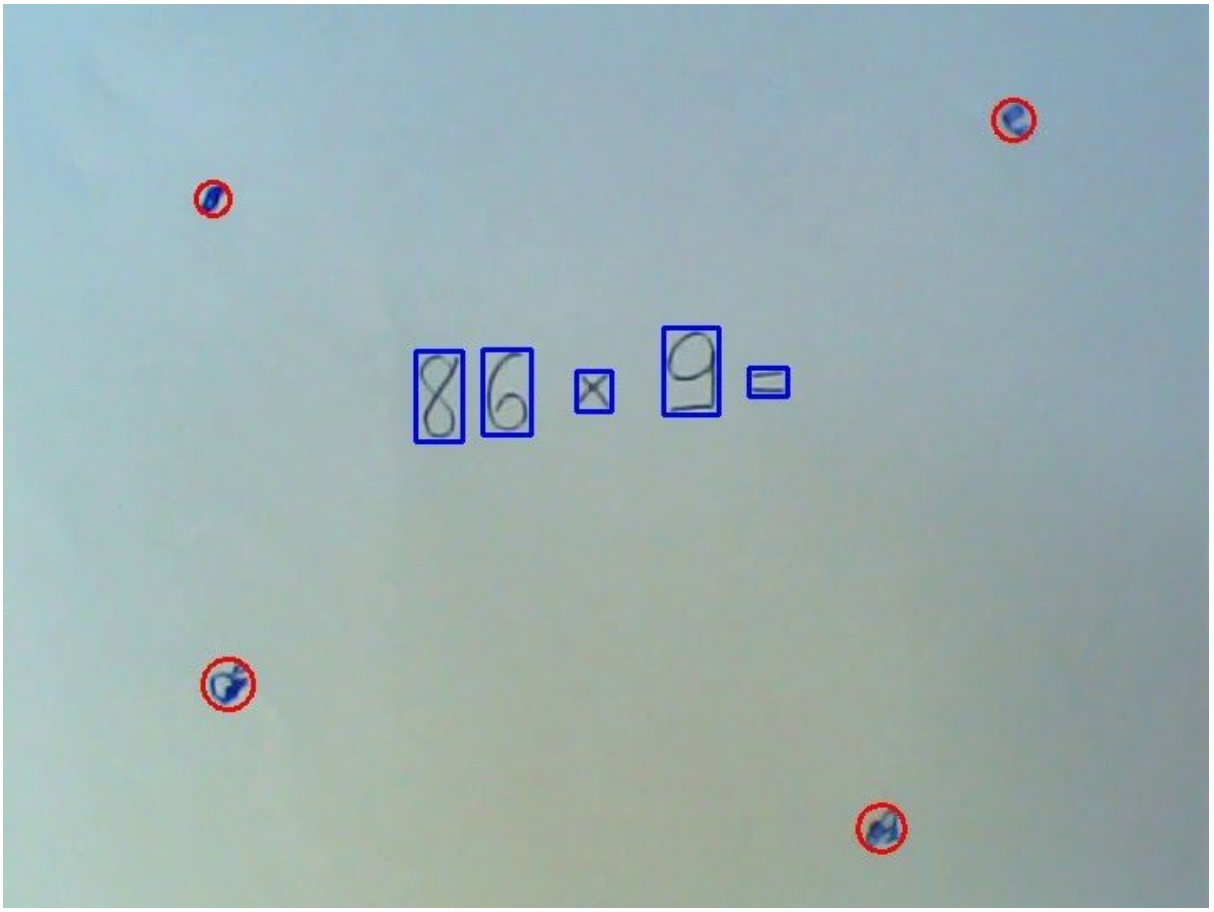
Durante lo sviluppo dell'applicazione si è notato come, occasionalmente, i contorni rilevati al passo precedente risultassero erroneamente disgiunti, dando quindi origine a più rettangoli contenenti ciascuno solo una parte dell'effettivo simbolo. Per ovviare a questo problema si è sviluppata la funzione `try_blend()`: essa è in grado di effettuare la fusione di due rettangoli nel caso in cui questi risultino sufficientemente sovrapposti oppure allineati verticalmente e molto vicini tra loro (caso piuttosto frequente per il simbolo di '=') e viene applicata a tutte le possibili coppie di rettangoli, iterando fino alla convergenza (ovvero quando non si trova più nessuna coppia di rettangoli da unire). Seguendo questo procedimento si è in grado di risolvere il problema descritto in precedenza, anche se ne emerge un altro: in alcuni casi due simboli distinti potrebbero venir scritti molto vicini tra loro (o addirittura parzialmente sovrapposti) e, seguendo questo approccio, verrebbero riconosciuti come uno solo, portando ad un'erronea interpretazione dell'espressione da parte del programma.

RIMOZIONE DEI PUNTI DI RUMORE

La procedura descritta finora, nella pratica, dà spesso origine a falsi positivi dovuti ad ombre, oggetti estranei oppure segni di scrittura involontaria presenti nell'inquadratura. Per evitare questi problemi, è stato definito nella funzione `clear_outliers()` un algoritmo per la rimozione dei cosiddetti "punti di rumore". In particolare, vengono immediatamente scartati tutti i rettangoli considerati troppo piccoli per essere significativi (soglia definita a 15px per lato), dopodiché viene applicato un algoritmo ispirato al clustering DBSCAN per eliminare tutti i rettangoli troppo lontani o isolati dagli altri (la distanza per definire un oggetto 'troppo lontano' è calcolata sulla base dell'altezza media dei simboli rilevati). Questa procedura si è dimostrata piuttosto efficace nella rimozione dei punti di rumore nei casi più comuni.

ISOLAMENTO DEI SIMBOLI RILEVATI

Come ultima operazione, i rettangoli ottenuti dal processing precedente vengono ordinati da sinistra a destra in base alla loro posizione nell'inquadratura e vengono usate le loro coordinate per ritagliare dall'immagine originale la relativa porzione di interesse, contenente il simbolo individuato.



L'immagine mostra in blu i rettangoli contenenti i simboli dell'espressione matematica, individuati ed isolati mediante la procedura `detect_symbols()`, ed evidenzia con dei cerchietti rossi gli elementi identificati come punti di rumore (e quindi ignorati) dalla funzione `clear_outliers()`.

RICONOSCIMENTO DEI SIMBOLI (funzione: `predict_symbols`)

Le immagini così ottenute sono passate una ad una alla funzione `predict_symbol()` del file `neuralnetwork.py`: questa funzione effettua su di esse alcune operazioni di pre-processing (thresholding e operazioni morfologiche) per poi predirne l'etichetta di classe, ovvero il tipo di simbolo contenuto al loro interno, attraverso l'utilizzo di una rete neurale.
Prima di tutto viene creato un oggetto `net`: al suo interno viene costruito un nuovo modello di rete neurale, `NET`, immediatamente caricato con i pesi della rete pre-allenata salvata nel file system (`NN.pth`) e configurato con i parametri necessari.
Il pre-processing prevede l'applicazione dei seguenti step a ciascuna delle immagini di input:

- l'immagine è convertita in grayscale e le viene applicato un filtro di blur gaussiano di dimensione 15x15
- il simbolo viene separato dallo sfondo mediante una funzione di sogliatura "adattiva", la cui soglia varia in base al vicinato del pixel, `cv.adaptiveThresholding()`
- viene creata una nuova immagine quadrata aggiungendo lungo tutti i bordi un padding di 8 px in modo tale da renderla più simile alle immagini di training della rete
- vengono infine applicati gli operatori morfologici di chiusura (prima) e di apertura (poi) rispettivamente per rimuovere eventuali punti di rumore ancora presenti nell'immagine e per riparare / riempire eventuali gap creatisi con la prima delle due operazioni

Dopodichè, l'immagine viene convertita nuovamente in RGB e trasformata mediante una funzione di `transform()` in modo tale da poter essere data in input alla rete. Come ultima operazione, l'immagine viene data in pasto alla rete neurale e viene finalmente generata la predizione del simbolo contenuto in essa.

PARSING DELL'ESPRESSIONE MATEMATICA E CALCOLO DEL RISULTATO (funzione: `calculator.compute`)

Una volta ottenuto un array di caratteri, rappresentanti ciascuno un simbolo dell'espressione aritmetica (cifra o operatore), questo viene passato alla `calculator.compute()` per il calcolo della soluzione. Questa funzione innanzitutto verifica che l'espressione sia terminata da un '=' (necessario sia come controllo di correttezza sia per la visualizzazione del risultato, come descritto nel paragrafo successivo), dopodichè si occupa di ricostruire i valori numerici effettuando il parsing dei caratteri nell'array. Infine, essa procede ad applicare uno ad uno gli operatori aritmetici sulle coppie di valori adiacenti, preoccupandosi di mantenere il corretto ordine di precedenza (prima moltiplicazioni e divisioni, poi somme e sottrazioni), e restituisce al chiamante un risultato numerico oppure un messaggio di errore, nel caso in cui qualcosa sia andato storto.

STAMPA A VIDEO DEL RISULTATO (funzione: `write_result`)

Il passaggio finale consiste nel mostrare all'utente il risultato ottenuto dalle operazioni precedenti. Tale valore viene scritto all'interno del frame mediante la funzione `cv.putText()`, più precisamente accanto al segno di '=' dell'espressione aritmetica, immediatamente sotto o eventualmente nell'angolo in alto a destra in base alla dimensione del risultato (calcolata mediante la funzione `cv.getTextSize()`) e allo spazio residuo attorno al simbolo '='.

STATO DELL'APPLICAZIONE (funzione: `write_status`)

Durante l'intera esecuzione dell'applicazione, in alto a sinistra nella finestra principale viene mostrato lo stato corrente del programma, determinato in base ai valori ritornati dalle funzioni descritte in precedenza. L'applicazione può trovarsi in uno dei seguenti stati:

- **WAITING**: in modalità '*video*' o '*webcam*', sono stati rilevati oggetti color pelle all'interno del frame; il numero tra parentesi rappresenta quanti fotogrammi devono passare senza rilevazioni da parte della funzione `detect_hand()` prima di procedere con l'identificazione dei simboli matematici (nello specifico, questo valore è calcolato per corrispondere a 0.5s)
- **ERROR**: è stato riscontrato un errore nella predizione di un simbolo da parte della rete neurale oppure nel calcolo del risultato dell'espressione aritmetica durante l'esecuzione di `calculator.compute()`
- **SUCCESS**: l'identificazione dei simboli e la risoluzione dell'espressione sono andate a buon fine ed il risultato è mostrato correttamente a schermo

POSSIBILI SVILUPPI FUTURI

Nonostante l'applicazione sviluppata rispecchi in maniera piuttosto soddisfacente l'idea originale ed i requisiti definiti all'inizio del progetto, ci sarebbe ancora un ampio margine di miglioramento per lo sviluppo delle sue funzionalità.
In particolare, molte delle limitazioni dell'attuale versione di *Camera Calculator* derivano dalla scelta di un dataset ridotto e di qualità non eccellente con cui è stato effettuato il training della rete neurale: l'utilizzo di un dataset migliore e più vasto permetterebbe infatti di migliorare ulteriormente la precisione dell'identificazione di cifre e operatori e consentirebbe inoltre di ampliare l'insieme dei simboli matematici supportati, ad esempio introducendo le parentesi e le relative regole di precedenza.
Infine, un ulteriore miglioramento potrebbe consistere nella risoluzione dei problemi descritti riguardanti il separatore decimale: ideando una logica più complessa (che tenga conto della vicinanza con altri simboli, allineamento e dimensione) si potrebbero distinguere tali simboli dai normali punti di rumore, permettendo quindi all'applicazione di gestire non solo espressioni contenenti numeri interi, ma anche numeri decimali.