Perche CRAFT-pytorch?

Durante le prime settimane del Progetto abbiamo visitato varie repository su github contententi implementazioni di text detectors. Con questo non abbiamo avuto problemi di installazione di alcun genere, venivano forniti gia alcuni modelli trainati, fra cui quello multilingue, e hai dei buoni tempi e accuratezza in generale. Abbiamo iniziato a sviluppare il codice attorno a questa soluzione che, per come detto prima, ci ha permesso di andare avanti velocemente senza risolvere problem di alcuna sorta (quali training di modelli, difficolta’ di installazione su google colab e altri) per creare qualcosa di concreto.

Fasi tecniche dello sviluppo software?

Gli strumenti di sviluppo utilzzati sono stati:

-Google Colab come piattaforma di sviluppo

-CRAFT-pytorch

-Tessercat-OCR Engine (per Linux su google colabe e windows)

-Git per la gestione di una repository con il codice non eseguiibile su colab e per scaricare Il necessario dalle altre repository

-Moduli python: pytesseract (come wrapper di tesseract per poterlo utilizzare in python), opencv (come libreria grafica)

-Alcuni frammenti di codice da stackoverflow, controllati e capiti

-…

Il primo problema riguardava il fatto che le immagini fossero in bmp ma Il detector non le accettasse, sono state quindi convertite in jpg tramite software. Durante questa fase e’ stata anche applicato uno zoom che aggiunge una cornice bianca attorno ad ogni imagine in quanto in fase di prova il modello restituiva delle bounding box che a volte uscivano dai limiti dell’immagine in quanto angolati (e dovendo essere rettangoli) e creavano problemi, per come e’ scritto il codice, durante la fase di ritaglio delle bounding box. Una volta convertite, tutte le immagini sono state pre-processate attraverso 4 metodi diversi, per creare dei paragoni; lo scopo delle CNN e’ di poter utilizzare un preprocessing meno aggressive e funzionare bene quindi avremo metodi con un preprocessing molto aggressive e metodi meno aggressive con un confronto visivo dei risultati. Durante il preprocessing sono state utilizzte delle trasformazioni basilari delle immagini quali variazione del valore di luminosita/ contrasto e trasformazioni morfologiche. Abbiamo anche utilizzato una funzione trovata online per confront con una soluzione trovata online ma abbastanza votata. E’ stato sviluppato un software molto veloce che utilizza delle funzioni messe a disposizione da Opencv per analizzare in tempo reale i vari effetti della variazione di alcuni parametri sull’immagine velocizzando il processo di ricercar di valori che permettessero di migliorare un minimo tutte le immagini e ridurre le anomalie rilevabili dal detector (tipo le bolle che diventano o con la threshold). Usata una funzione trovata su github per applicare contrasto e luminosita come GIMP dato che in un primo momento era stato utilizzato quest’ultimo per trovare dei valori ma essendo troppo macchinoso nell’applicare le varie modifiche e’ stato sostituito dal mini software ad hoc. **<commenta vari metodi e varie osservazioni sulle pecche di ogni metodo magari>** La fase di preprocessing per il detector e poi per Tesseract risultano essere il punto critic di tutto il sistema in quanto le immagini delle sacche sono molto particolari con riflessi e particolari che creano confusion ad un computer. Si passa ora alla fase di detection del testo , le bounding box risultanti da questa fase verranno poi tagliate e riprocessate per essere date in pasto a tesseract per il riconoscimento finale del testo. Dopo aver clonato le repository, scaricato I modelli dalla pagina di github di CRAFT e installato I moduli richiesti proseguiamo con l’esecuzione del detector su ogni cartella preprocessata sia degli scarti che dei buoni sia con che senza il refiner (un modello aggiuntivo allenato e fornito con CRAFT che viene eseguito dopo la fase di detection con il modello vero e proprio per rifinire la posizione delle bounding box unendo per esempio quelle abbastanza vicine o creando un poligono con quelle molto spezzate). La proposta delle zone di interesse e’ molto veloce soprattutto nelle immagini in bianco e nero. A seguito della fase di detection del testo passimo ad estrarre le bounding box in preparazione del riconoscimento attraverso Tesseract. Sono stati proposti 2 metodi di cui il primo abbandonato e circa integrato all’interno del secondo piu’ avanti. Il primo metodo legge il file in cui sono state salvate le bounding box e ne crea una unica che le contenga tutte, la pecca di questo metodo e’ che la presenza di anomalie e’ difficile da garantire e unire tutte le bounding box indifferentemente da tutto porta a grossi errori visto che tesseract ha bisogno del testo in determinate condizioni per garantire una certa accuratezza. Si e’ avviato uno studio per calcolare la bounding box ad area minima e massima, ricercar di una threshold minima e scarto di bounding box troppo grandi o troppo piccolo ma per ora e’ stata lasciata stare e non e’ stata sviluppata. Il metodo 1 lavora male negli scarti dove la presenza di anomalie e’ alta (anche questa potrebbe essere utilizzato come metro per la rilevazione di anomalie, il numero elevato di bounding box, dato che spesso hanno il testo dell’etichetta dall’altra parte della sacca molto visibile e potrebbe creare problemi con la threshold **Meh da rivedere questa parte**). Il metodo 2 e’ stato quello portato avanti: viene ritagliata ogni bounding box, senza crearne una unica, a men oche non sia stato rilevato un poligono e in questo caso viene usata la soluzione al metodo 1 per creare un rettangolo, e non un poligono, che e’ possibile ritagliare attraverso Opencv (piu’ complicato ritagliare un poligono). Per raddrizzare ogni bounding box usiamo l’angolo di sfasamento dellla stessa e poi tagliamo. Le funzioni importanti per il codice di questo metodo sono:

-Incicciottisci (ampiamente usata nel codice per cercare di rendere il font regolare)

-Il preprocessing applicato ad ogni ritaglio dall’immagine; fatto da noi consiste in: una Maschera ottenuta con una threshold adattativa basata sulla media e su una costante, un tweaking di contrasto e luminosita’, una threshold del ritaglio attraverso l’algoritmo di Otsu, una and tra quest’ultima e la Maschera ricavata inizialmente, un inciccionimento dei puntini del font, un blur per pulire I contorni delle lettere e un’ultima threshold, a seguito del blur, per portare tutto in bianco e nero (visto che il blur potrebbe introdurre grigi). Viene fatto uno zoom per aggiungere area visto che Tesseract non accetta immagini con solo testo praticamente ma vuole un po’ di contorno. Salviamo sia l’immagine ottenuta che capovolta, visto che non possiamo conoscere se il testo e’ capovolto oppure no, a 300 DPI dato che Tesseract funziona meglio a questa risoluzione da varie ricerche.

Le immagini con il testo piccolo, trattasi di un’anomalia, non vengono riconosciuti ma si potrebbe: la fase di incicciottimento dei puntini del font per renderlo il piu’ possibile regolare deteriora le scritte piccolo che diventano illleggibili per tesseract. (dato che dovrebbero essere gli scarti questa cosa va bene!) I testi warpati producono, spesso, un poligono e non una bounding box quindi potremmo scartare direttamente I testi warpati. I font sbiaditi vengono quasi del tutto letti perfettamente quindi si deve pensare a qualche altra strategia per poter riuscire a rilevare l’anomalia in quell caso.