UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Informatica

Progetto Computer Vision

Sviluppo di una CRNN per OCR

Docente Francesco Isgrò

Anno Accademico 2021/2022

Studenti
Turco Mario N97/343
Longobardi Francesco N97/344
Rauso Giuseppe. N97/357

Contents

Chapter 1

Metodo

Le reti neurali convoluzionali profonde (DCNN) sono utilizzate in moltissimi task nel campo della computer vision grazie alla loro natura. In questo progetto il problema trattato è quello di riconoscere una sequenza di caratteri, e quindi delle parole. A differenza di un task di object recognition, riconoscere diversi caratteri in una sequenza richiede una serie di label in output invece di uno solo. Inoltre, la lunghezza di queste parole può essere variabile, di conseguenza anche l'output è variabile. Le reti convoluzionali classiche non possono essere applicate direttamente per risolvere questo problema, siccome operano su input e output di dimensioni fissate. Per questo motivo, nel paper di riferimento [1], gli autori hanno costruito un sistema composto da una rete convoluzionale per estrarre le feature significative dalle immagini in input, e da una rete ricorrente (LSTM bidirezionali) per riconoscere le lettere. Le reti ricorrenti lavorano con sequenze, caratterizzate da step temporali. Per questo motivo c'è bisogno di uno step intermedio tra le due reti per convertire le immagini in una sequenza di feature per la rete ricorrente. L'intera struttura prende il nome di **Convolutional Recurrent Neural Network** (**CRNN**).

1.1 Descrizione del metodo

L'architettura proposta in [1] ha 3 componenti: i livelli convoluzionali, i livelli ricorrenti e il livello di trascrizione. I livelli convoluzionali estraggono una sequenza di feature da ogni immagine in input. Successivamente, la rete ricorrente è utilizzata per fare prediction su ogni frame della sequenza di feature (prodotta dalla rete convoluzionale). Il livello di trascrizione invece è introdotto per tradurre le prediction per-frame in una sequenza di label. La CRNN, anche se costituita da tre diverse architetture, può essere addestrata utilizzando una sola funzione di loss.

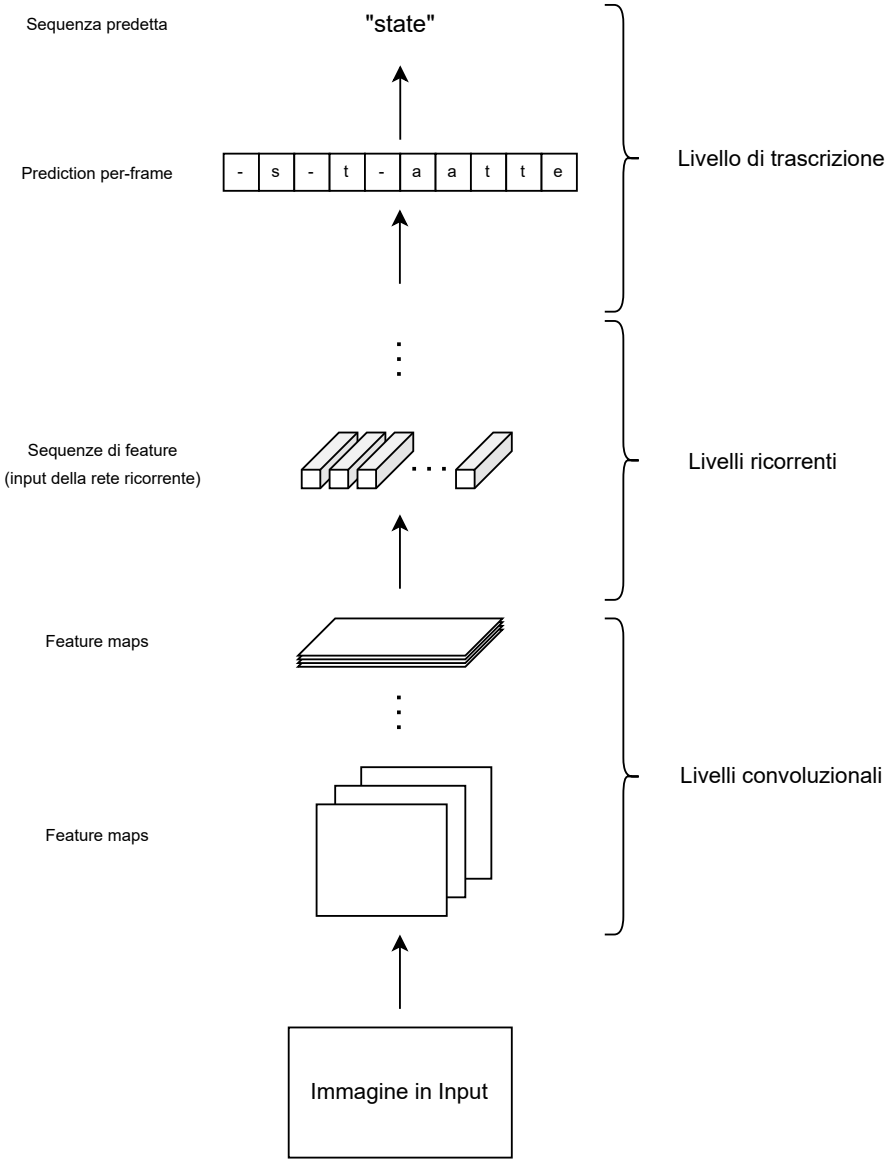


Figure 1.1: Architettura

1.1.1 Estrazione delle feature

Siccome i livelli convoluzionali, il max-pooling e le funzioni di attivazione element-wise operano localmente sulle regioni, sono *translation invariant*. Quindi, ogni colonna delle feature maps corrisponde ad una regione rettangolare dell'immagine originale (*receptive field*), e queste regioni sono nello stesso ordine rispetto alle colonne corrispondenti nelle feature maps, da sinistra verso destra. Ogni colonna delle feature maps quindi rappresenta un descrittore per la corrispondente regione rettangolare dell'immagine originale, e la sequenza di colonne rappresenta la sequenza di feature in input alla rete ricorrente.

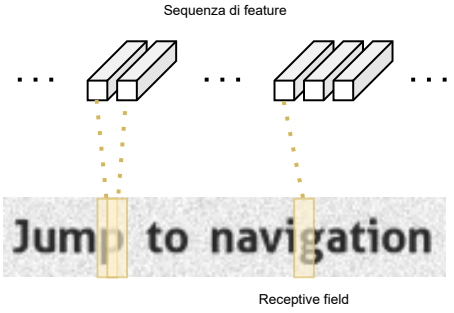


Figure 1.2: Estrazione delle sequenze di features

1.1.2 Labeling delle sequenze

I livelli ricorrenti predicono una distribuzione di label y_t per ogni frame x_t nella sequenza di feature $x = x_1, \ldots, x_T$. Una rete ricorrente riesce a catturare informazioni di contesto all'interno di una sequenza. Questo aiuta particolarmente nel caso di riconoscimento di sequenze caratteri all'interno di immagini rispetto a processare i singoli caratteri (utilizzando ad esempio solo reti convoluzionali). Ad esempio, dei caratteri molto larghi potrebbero richiedere diversi frame successivi per essere riconosciuti completamente. O, ancora, è più facile riconoscere "il" confrontando le diverse altezze (perché l'immagine viene processata come sequenza). Inoltre, una RNN può propagare all'indietro le derivate della funzione di loss rispetto agli input, *i.e.* i livelli convoluzionali, permettendo di addestrare congiuntamente le due architetture. Infine, con questa tipologia di rete è possibile operare su sequenze di lunghezza arbitraria, andando dall'inizio alla fine.
Nei livelli ricorrenti, le derivate sono propagate anche tra gli step temporali, cioé tra le connessioni dei nodi nascosti nel livello ricorrente (**Back-Propagation Through Time**). Le derivate vengono propagate dai livelli ricorrenti a quelli convoluzionali passando per uno step intermedio che inverte l'operazione di convertire le feature maps in sequenze di feature (**Map-To-Sequence**).

1.1.3 Transcription

Matematicamente, questo step consiste nel trovare la sequenza di label con la più alta probabilità condizionata rispetto alle predizioni per-frame. Si adotta la probabilità condizionata definita nel layer **Connectionist Temporal Classification**[2] dove la probabilità è definita per la sequenza di label l condizionata dalle predizioni per-frame $y = y_1, \ldots, y_T$ e ignora la posizione dove ogni label l è localizzato. Si usa poi la *log-likelihood* negativa di questa probabilità per addestrare la rete

$$O = -\sum_{I_i, l_i \in X} \log p(l_i | y_i)$$

dove I_i è l'immagine di training e l_i la sequenza di label ground truth.

1.2 Connectionist Temporal Classification

La rete restituisce in output dei punteggi per carattere per ogni elemento della sequenza, che formano una matrice. Con questa matrice si vuole:

1. calcolare la funzione di loss per addestrare la rete
2. decodificare la matrice per ottenere il testo contenuto nell'immagine

Entrambi i compiti sono svolti mediante l'operazione di **CTC**.
Un'alternativa a questa operazione potrebbe essere quella di specificare per ogni posizione orizzontale dell'immagine il carattere corrispondente. Quindi, per ogni slice orizzontale dell'immagine si specifica il carattere presente. Ci sono però diversi problemi legati ad un metodo di questo tipo:

- annotare un intero dataset in questo modo può essere un processo lungo e dispendioso, perché vanno considerate tutte le slice orizzontali di ogni immagine
- l'output sarà composto da punteggi per carattere, quindi serve processare ulteriormente i risultati per ottenere il testo. Inoltre, un carattere può occupare più posizioni orizzontali dell'immagine (ad esempio una "o" molto larga). Ad esempio, un possibile label per la parola "to" potrebbe essere "ttooo". A questo punto vanno rimossi i duplicati. Se la parola iniziale fosse "too" allora il procedimento sarebbe più complicato.

La CTC invece richiede direttamente il testo presente nell'immagine (nel caso di "to" il label è direttamente "to"), e quindi si ignorano la posizione e la larghezza dei caratteri.

1.2.1 Encoding del testo

Il problema dei duplicati viene risolto aggiungendo un carattere speciale, chiamato blank, e identificato con "-". È possibile aggiungere un numero arbitrario di blanks, che saranno poi rimossi in fase di decoding.

Si aggiunge un blank tra le lettere duplicate (come in "t**oo**"). La parola "too" quindi può essere codificata come "–ttttoo-o", o "-t-o-o-" (non "too" perché non ci sono blanks, quindi non si possono inserire duplicati), e così via, mentre la parola "to" può essere codificata come "–ttttooo", o "t-o-" e così via.
La rete neurale impara a produrre un output **codificato** come appena descritto.[2]

1.2.2 Calcolo della loss

La rete produrrà delle probabilità per ogni carattere (compreso il blank) per ogni time-step (slice orizzontale)[1]. Sia L l'insieme dei caratteri nel dataset e $L' = L \cup \{" - "\}$. Sia $B : L'^T \rightarrow L^{\leq T}$ una funzione di mapping dai label codificati (come descritto nel precedente paragrafo) ai label originali, dove $L^{\leq T}$ è l'insieme dei possibili label di lunghezza minore o uguale a T sull'insieme di caratteri L. Quindi, ad esempio, B("-hhh–el-l-o—o") e B("–hh-e-l-ll-oo–") restituiscono "hello". Lo score di un allineamento, o **path**, è definito come

$$c(a) = \sum_{\pi:B(\pi)=a} \prod_{t=1}^{T} p(y_t = a_t)$$

dove T è il numero di step temporali (o lunghezza della sequenza in output dalla rete ricorrente), a è il label, y_t è l'ouput per lo step temporale t e a_t è il carattere t-esimo del path. Questa quantità si può esprimere anche in termini di probabilità condizionata:

$$p(a|y) = \sum_{\pi:B(\pi)=a} p(\pi|y)$$

con $p(\pi|y) = \prod_{t=1}^{T} p(y_t = a_t)$.
L'obiettivo è quindi quello di addestrare la rete neurale in modo tale da avere in output delle probabilità alte in corrispondenza delle classificazioni corrette, cioé massimizzare il prodotto delle probabilità di classificazioni corrette. L'output scelto sarà quindi quello formato dal carattere più probabile π_t per ogni time-step t, mappato tramite la funzione B

$$l^* \approx B(\arg\max_\pi p(\pi|y))$$

La funzione di loss sarà quindi la somma delle probabilità in logaritmo cambiata di segno, come presentata nel paragrafo 1.1.3 .
Calcolare direttamente la sequenza più probabile è computazionalmente intrattabile, per questo si usa l'algoritmo **forward-backward** descritto in [2].
L'approccio appena descritto prende il nome di **lexicon-free transcription**, contrapposto ad un altro approccio che fa uso di un lexicon[1].

[1] Di conseguenza, la somma delle probabilità per ogni time-step è 1

Chapter 2

Implementazione

In questo capitolo vengono introdotte le caratteristiche implementative del progetto, illustrando due diversi modelli di diversa complessità. Entrambi i modelli sono divisi in una parte dedita alla **estrazione di features** ed una sezione incaricata al **riconoscimento**, implementati rispettivamente con una serie di livelli convoluzionali e una coppia di LSTM bidirezionali ognuna formate da due LSTM dette *backward* e *forward*. Il motivo per cui le reti da noi utilizzate sono strutturate in questo modo, è dato dal fatto che le Long Short Term Memory classiche utilizzano solo informazioni temporali passate mentre, in questo contesto, anche informazioni nell'altro verso sono utili e complementari, come specificato nella sezione 2.2 di [1].

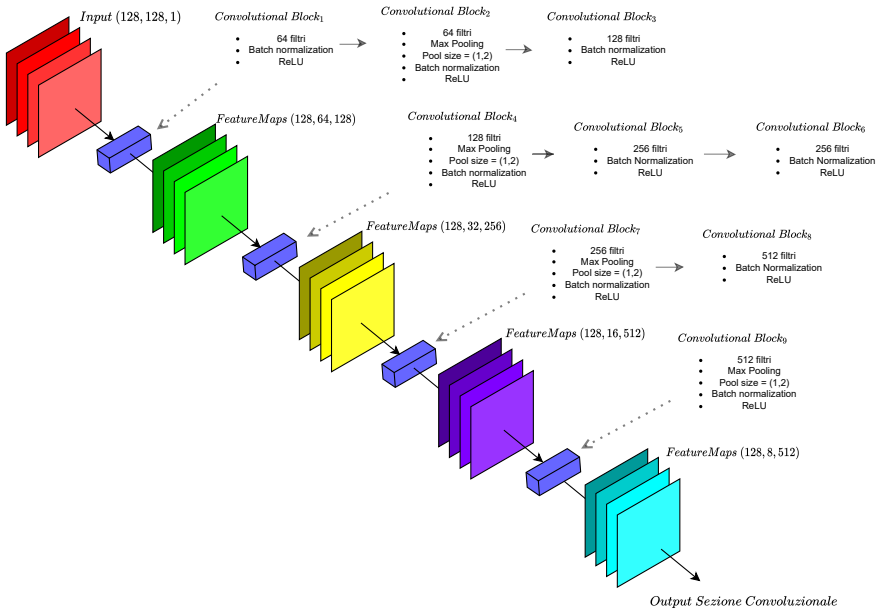
2.1 Struttura delle reti

2.1.1 Prima architettura

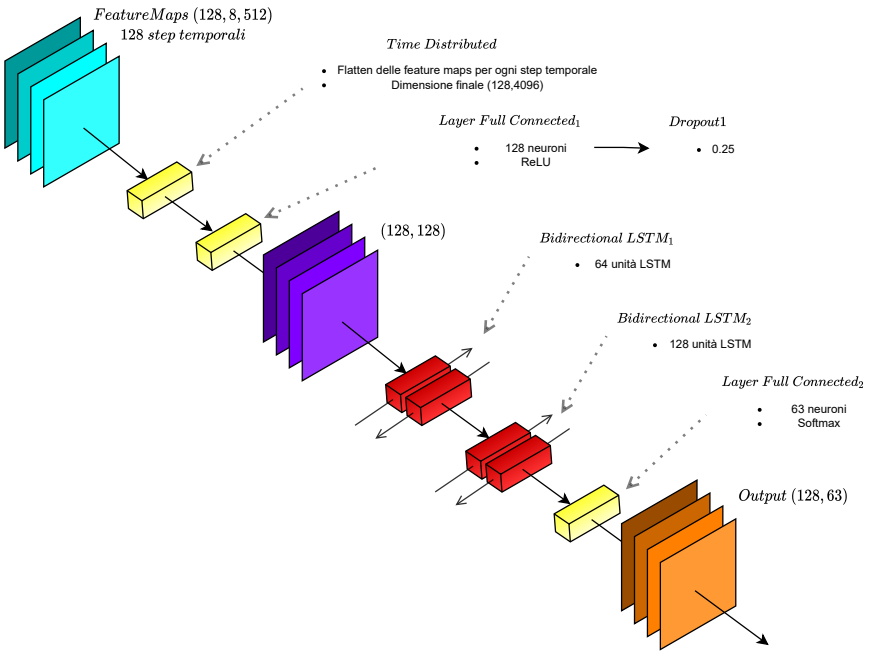
La prima rete che presentiamo segue la linea generale illustrata nel paper di riferimento, infatti è possibile notare

Sezione convoluzionale

come la prima parte della rete sia costituita da una Deep Convolutional Neural Network composta da nove livelli convoluzionali con un numero crescente di filtri e batch normalization. Quattro livelli fanno uso di max pooling con una finestra di pooling di dimensioni $(1, 2)$. In seguito l'output di questa sezione convoluzionale viene passato attraverso un layer wrapper, chiamato *TimeDistributed*, che applica un'operazione di *Flatten()* indipendente per ogni step temporale del batch. La successiva sezione è data da un layer full-connected, il cui output viene passato alla rete ricorrente. È stato anche introdotto un layer di *Dropout*. L'output finale della rete è dato da un ultimo livello full-connected a cui viene applicata un'operazione di softmax. La rete in totale fa uso di 6,185,599 parametri, di cui 4,352 non apprendibili.


Input (128, 128, 1)
Convolutional Block1
64 filtri
Batch normalization
ReLU
Convolutional Block2
64 filtri
Max Pooling
Pool size = (1,2)
Batch normalization
ReLU
Convolutional Block3
128 filtri
Batch normalization
ReLU
FeatureMaps (128, 64, 128)
Convolutional Block4
128 filtri
Max Pooling
Pool size = (1,2)
Batch normalization
ReLU
Convolutional Block5
256 filtri
Batch Normalization
ReLU
Convolutional Block6
256 filtri
Batch Normalization
ReLU
FeatureMaps (128, 32, 256)
Convolutional Block7
256 filtri
Max Pooling
Pool size = (1,2)
Batch normalization
ReLU
Convolutional Block8
512 filtri
Batch Normalization
ReLU
FeatureMaps (128, 16, 512)
Convolutional Block9
512 filtri
Max Pooling
Pool size = (1,2)
Batch normalization
ReLU
FeatureMaps (128, 8, 512)
Output Sezione Convoluzionale

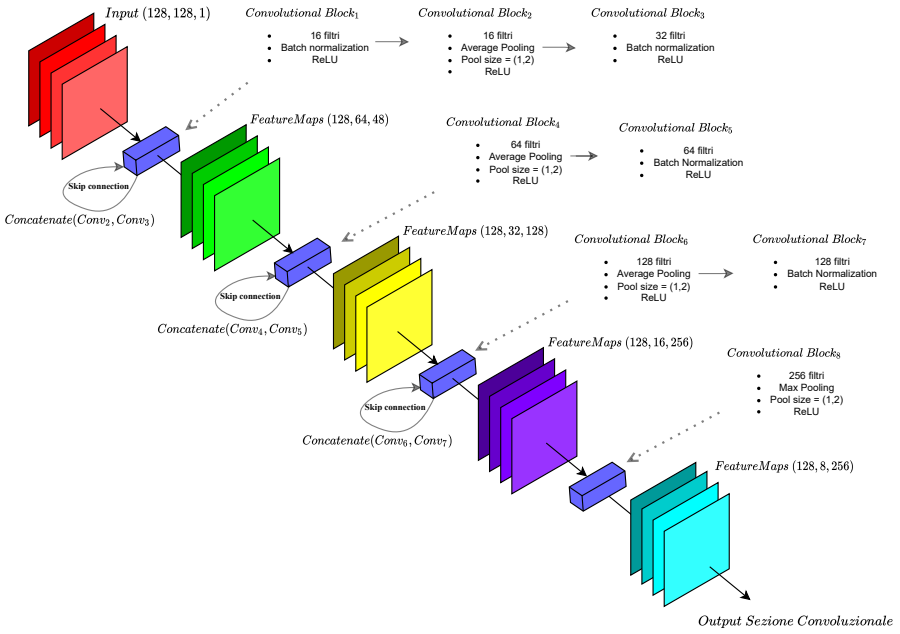
Sezione ricorrente



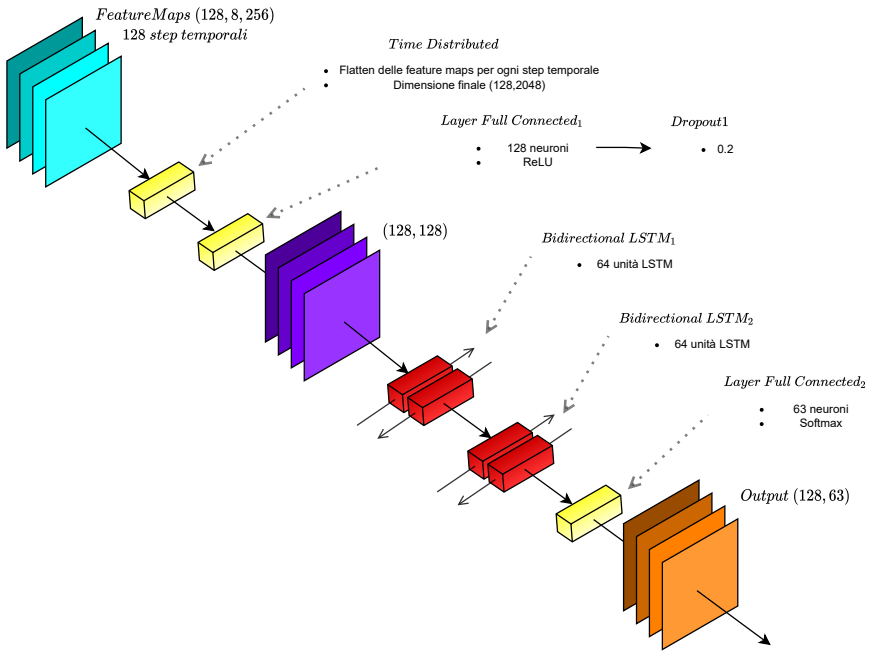
2.1.2 Seconda architettura

Il secondo modello implementato rappresenta un raffinamento del modello precedente, esso infatti risulta essere più leggero (∼4.95M di parametri in meno) pur mantenendo le stesse perfomance in generale. Anche questa rete presenta la stessa architettura di base, diversificandosi soltando nell'utilizzo di un layer convoluzionale in meno, di meno filtri e meno unità LSTM in generale e per la presenza di **skip connections** implementate tramite concatenazione tra feature maps apprese da diversi livelli convoluzionali. Questo approccio può infatti rendere il processo di apprendimento più efficiente [3]. Questo secondo approccio presenta 1,231,247 parametri, di cui 480 non apprendibili.

Sezione convoluzionale

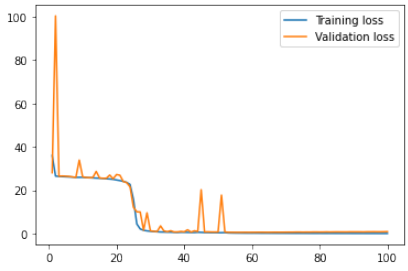


Sezione ricorrente

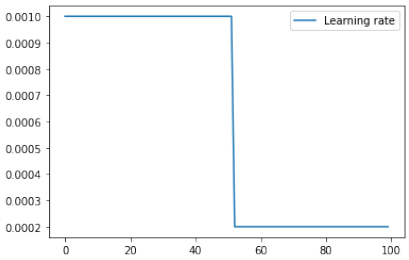


2.2 Training

Entrambe le reti sono state addestrate utilizzando **Adam** come ottimizzatore per 100 epoche. Durante i training sono stati utilizzate delle utility messe a disposizione da Keras sotto forma di callbacks, per fare uso di politiche di early stopping e di riduzione del learning rate nel momento in cui una metrica smette di migliorare (Plateau). Abbiamo utilizzato un approccio di tipo *mini-batch learning* modificando i pesi per ogni batch di 64 immagini.
Il 20% del dataset è stato utilizzato come validation set ed è stata utilizzata l'opzione *shuffle=True* nel metodo *fit* di keras per rimescolare casualmente il dataset dopo ogni epoca. Di seguito vengono presentati i grafici dell'andamento delle metriche durante il training della prima rete.



(a) Andamento dei valori delle loss di training e di validation per la prima rete



(b) Valori del learning rate utilizzati durante il training della prima rete

2.3 Dataset utilizzato

Il dataset utilizzato è stato generato utilizzando un generatore di testi casuali per text-recognition (`https://github.com/Belval/TextRecognitionDataGenerator`). In particolare, sono state generate 10000 immagini 32×128, convertite in scala di grigi, portate a 128×128 utilizzando interpolazione bilineare e trasposte per considerare le righe come step temporali.

Figure 2.6: Esempio di immagine generata

Chapter 3

Test e valutazione

Una delle parti fondamentali della progettazione e creazione di una rete neurale è la parte di valutazione e test della rete stessa in modo tale da avere una metrica che ci faccia capire quali scelte progettuali migliorino le performance della rete e quali le peggiorano.
Per i task di OCR spesso sono utilizzare metriche derivanti dalla **distanza di Levenshtein**. Essa è utilizzata per misurare quanto due stringhe siano differenti l'una dall'altra ed, in particolare, la distanza è rappresentate dal numero minimo di singoli caratteri da modificare (cancellando, modificando o inserendo un carattere) per passare dalla prima stringa alla seconda. [4]
Ad esempio, la distanza di Levenshtein tra le seguenti parole

- casa
- base

è di 2 poichè:

1. casa $\rightarrow$ ~~c~~**b**asa
2. basa $\rightarrow$ bas**e**~~a~~

3.1 Predizione di singole parole

3.1.1 Character Error Rate

La metrica che abbiamo utilizzato per valutare le predizioni di una singola parola è la Character Error Rate (CER), calcolata come segue:

$$CER_{\text{Normalizzata}} = \frac{S + D + I}{S + D + I + C}$$

Dove:

- **S** è il numero di caratteri da sostituire per passare dalla parola predetta al ground truth
- **D** è il numero di caratteri da cancellare
- **I** è il numero di caratteri da inserire
- **C** è il numero di caratteri corretti

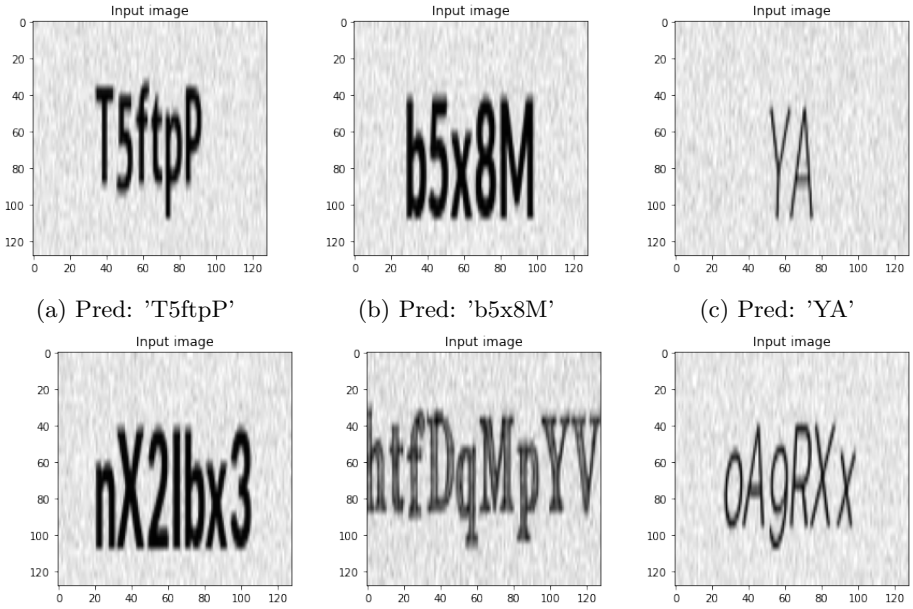
Il valore risultante dalla formula rappresenta la percentuale di caratteri della prima stringa che sono state predette in modo errato, quindi più basso è il valore e migliore è il risultato.
Utilizziamo la forma normalizzata rispetto a quella non normalizzata, in cui al denominatore abbiamo solo **N**, poichè se la rete genera un output più lungo del ground truth si avrebbe una percentuale di errore più alto del 100%
Per capire quanto effettivamente il nostro modello stesse lavorando bene abbiamo utilizzato dei valori di benchmark forniti dall'articolo *How good can it get?* [5] il quale fornisce una classifica di valori per gli algoritmi di OCR che sono riportati di seguito:

- Good = 98-99% accuracy (1-2% CER score)
- Average = 90-98% accuracy (2-10% CER score)
- Poor = meno di 90% accuracy (più di 10% CER score)

Nel nostro caso, la seconda rete presentata nel capitolo precedente, ha uno score dell' 1.373% sul training set e del 2.995% su di un test set generato contenente 5000 immagini. Risultati comparabili sono stati ottenuti valutando il primo modello descritto in precedenza, nello specifico è stato ottenuto uno score dell' 1.306% sul training set e del 3.006% sullo stesso test set.

3.1.2 Esempi di predizioni

Di seguito vengono riportati alcuni esempi di predizioni effettuati dalla rete su una singola parola:



(a) Pred: 'T5ftpP' (b) Pred: 'b5x8M' (c) Pred: 'YA'

(d) Pred: 'nX2Ibx3' (e) Pred: 'htfDqMpYV' (f) Pred: 'oA9RXx'

Figure 3.1: Aclune predizioni dal Test Set

3.2 Prediction di più parole separate da spazi

3.2.1 Word Error Rate

Per il task di predizione di una frase la metrica che abbiamo utilizato è la Word Error Rate (**WER**) ed è calcolata con la seguente formula:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

Dove, data la prima stringa e la seconda stringa (che rappresentano due frasi) abbiamo:

- S - il numero di parole da sostituire nella prima stringa
- D - il numero di parole da eliminare nella prima stringa
- I - il numero di parola da inserire nella prima stringa
- C - il numero di parole corrette
- N - il numero di parole della stringa ground truth ($N = S + D + C$)

La WER è una formula analoga alle CER con la differenza che, la CER opera a livello di carattere mentre la WER opera a livello di parola.
Nel nostro caso abbiamo calcolato questa metrica su un test set contenente 5000 immagini, tutte con 3 parole e di larghezza variabile ottenendo i seguenti valori:

- Per quanto riguarda la prima rete, sono stati rilevati i seguenti punteggi:
  - Primo approccio preprocessing[1]: 84.446%
  - Secondo approccio preprocessing: 79.440%
- Per quanto riguarda la seconda rete, sono stati rilevati i seguenti punteggi:
  - Primo approccio preprocessing: 85.920%
  - Secondo approccio preprocessing: 80.340%

Lo score WER è molto penalizzante, perchè, nel caso in cui una parola avesse anche una sola lettera diversa dal label, l'intera parola viene considerata errata e quindi da sostituire.

3.2.2 Prepocessing delle immagini

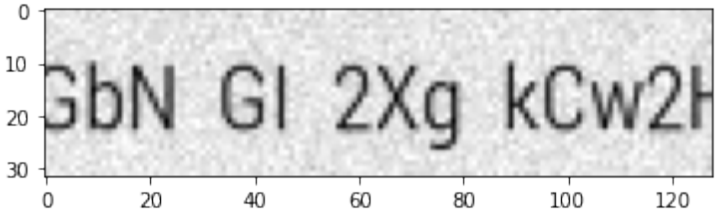
Essendo la rete addestrata a riconoscere immagini contenenti singole parole sono necessarie delle operazioni preliminari per riconoscere immagini contenenti intere frasi.
In particolare, è necessario estrarre dall'immagine le singole parole, predirre il loro label singolarmente ed infine mettere assieme tutte le parole predette per

[1] Spiegato nella pagina successiva

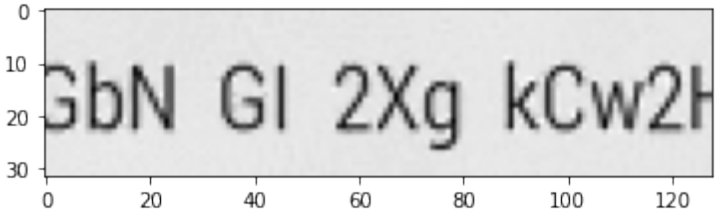
formare la frase originale.
Sono stati sviluppati due diversi approcci, che condividono gran parte della procedura, per la suddivisione della frase in diverse immagini contenenti le singole parole. Si differenziano per il posizionamento della singola parola nella rispettiva immagine. In particolare:

- Il **primo approccio** comprende i seguenti passaggi:
  1. conversione dell'immagine da RGB a Scala di Grigi
  2. rimozione del rumore salt and pepper
  3. dilation dell'immagine tramite kernel ellissoidale
  4. clipping dei grigi
  5. trovare i *contour* delle parole
  6. creazione di varie immagini della stessa dimensione di quella originale contenenti ognuna una parola della frase originale
  7. resize delle immagini a 128×128, posizionando l'$i - esima$ parola nella medesima posizione che aveva nell'immagine di partenza
- Il **secondo approccio** è del tutto analogo, variando semplicemente il passaggio finale:
  7. resize di ogni immagine a 128×128 facendo in modo che l'$i - esima$ parola appaia al centro dell'immagine e venga ingrandita il più possibile, calcolando i *bounding rectangles* per ogni insieme di *contour* rappresentativi di una parola e in seguito usando i punti calcolati per centrare la parola

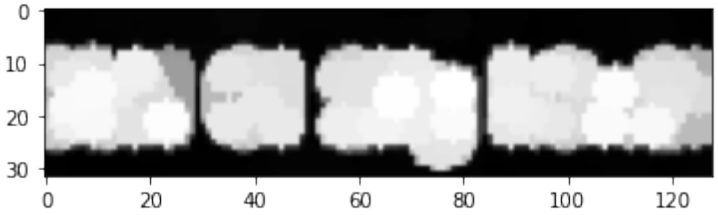
Di seguito è riportato un esempio grafico del prepocessing di una frase, illustrando le differenze fra i due metodi:



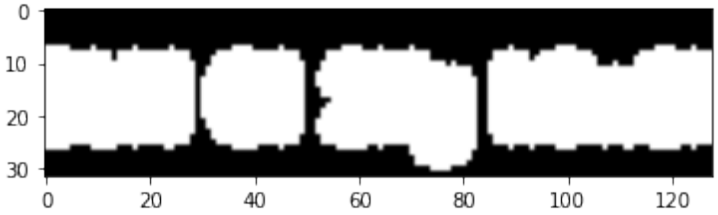
(a) Immagine originale



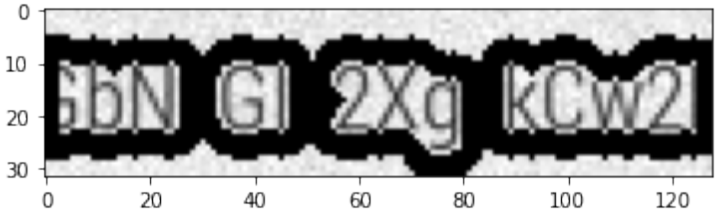
(b) Rimozione di rumore 'Salt and pepper'



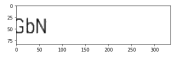
(c) Dilation dell'immagine



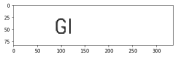
(d) Clipping dell'immagine



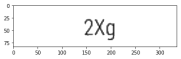
(e) In nero i contour rilevati delle parole rispetto all'immagine originale



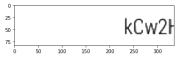
(f) Prima parola



(g) Seconda parola



(h) Terza parola



(i) Quarta parola

(j) Prima parola II metodo

(k) Seconda parola II metodo

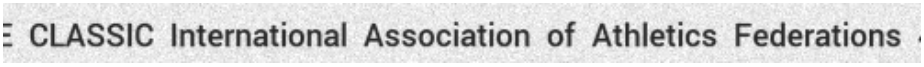
(l) Terza parola II metodo

(m) Quarta parola II metodo

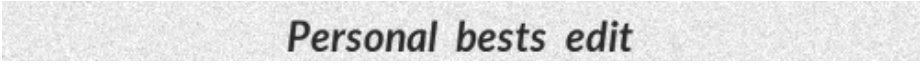
Figure 3.2: Esempio di preprocessing di una frase

3.2.3 Esempi di prediction

Di seguito vengono riportati alcuni esempi di predizioni effettuati dalla rete su frasi:



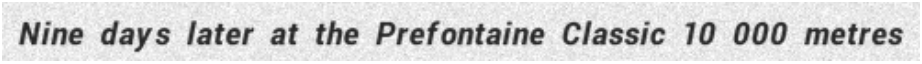
(a) Pred: '5','CLASSlC','International','Association','of','Athletics','Federations',"



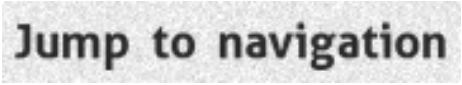
(b) Pred: 'PersonaI','bests','edit'

(c) Pred: '21','N0Vember','2004'

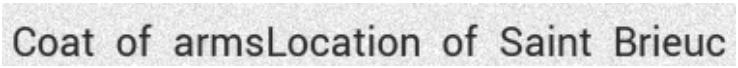
(d) Pred: 'areweII','InternationaI','Association','of','AthIetics','Fede'



(e) Pred: 'Nine','89YS','Iater','at','the','Prefontaine','CIassic','10','000','metres'



(f) Pred: 'Jump','to','navigation'



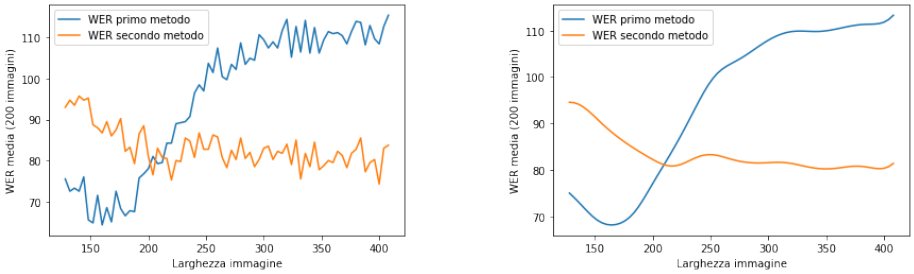
(g) Pred: 'Coat','of','armsLocation','of','Saint','Brieuc'

Figure 3.3: Alcune predizioni dal Test Set

3.2.4 Comparazione fra i due diversi metodi di preprocessing

In questa sezione presentiamo graficamente una comparazione fra i due metodi illustrati precedentemente, utilizzando la seconda rete.
Nei seguenti grafici si può osservare l'andamento grezzo e levigato[2] delle metriche di errore CER e WER al crescere della larghezza delle immagini. Per eseguire un confronto robusto, entrambi gli approcci sono stati sottoposti allo stesso insieme di 200 immagini, tutte composte da 2 parole, generate *ex novo* a step crescenti di larghezza.

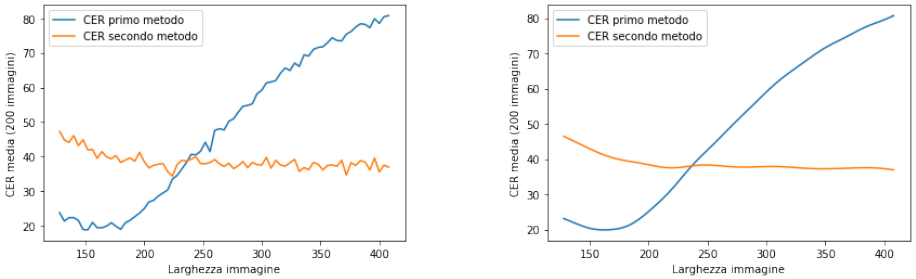
[2]La levigatura delle curve è stata ottenuta utilizzando la *kernel regression* implementata nel modulo python statsmodels.nonparametric.kernel_regression



(a) Versione non levigata (b) Curve levigate

Figure 3.4: WER calcolata utilizzando i due diversi approcci

Si può notare come, per immagini strette, il primo metodo sia molto superiore al secondo. Tuttavia l'errore tende a crescere molto velocemente all'aumentare della larghezza, mentre il secondo metodo garantisce, oltre una certa soglia, una curva di performance più stabile. È inoltre ragionevole supporre che il primo metodo sia destinato a peggiorare ulteriormente, qualora si incrementasse ancora la larghezza; mentre il secondo metodo sembra convergere verso un valore compreso fra 80 ed 85.



(a) Versione non levigata (b) Curve levigate

Figure 3.5: CER calcolata utilizzando i due diversi approcci

Le stesse identiche considerazioni possono essere applicate al caso del calcolo della CER, anche se è possibile notare come il punto di incrocio fra le due curve sia spostato leggermente più avanti rispetto alla metrica precedente e che il secondo approccio sembra convergere più velocemente in un certo intorno.

Nel complesso, le metriche ottenute nel caso di immagini composte da più parole non sono particolarmente brillanti e il motivo è, con grande probabilità, da ricercare nella ristretta dimensione delle immagini di ingresso che abbiamo adottato: 128x128. Tale decisione è stata presa a causa della limitata potenza di calcolo

a nostra disposizione, non sufficiente per addestrare modelli più complessi di quelli presentati in tempi ragionevoli per la consegna di questo elaborato.

Bibliography

[1] Baoguang Shi, Xiang Bai, Cong Yao, "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition.," *School of Electronic Information and Communications*, 2015.

[2] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks.,"

[3] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, "Densely Connected Convolutional Networks.," 2018.

[4] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals.," *Soviet Physics Doklady*, vol. 10, pp. 707–710, feb 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[5] R. Holley, "How good can it get? analysing and improving ocr accuracy in large scale historic newspaper digitisation programs," *D-Lib Magazine*, vol. 15, no. 3/4, 2009.