Advanced Machine Learning - Assignment 3

Luca Gandolfi, matricola 807485

Novembre 2019

1 Introduzione

L'assignment consiste nel classificare le immagini presenti nel MNIST dataset, utilizzando una Convolutional Neural Network (CNN) con un vincolo di massimo 7.500 parametri utilizzabili nel modello. In questo report verranno discusse le scelte progettuali della rete e successivamente verranno presentati i risultati ottenuti, sfruttandoli per trarre delle conclusioni.

2 Analisi preliminari

Il dataset MNIST è composto da *train set* e *test set*, i quali contengono immagini di numeri scritti a mano. I numeri sono singole cifre e variano tra 0 e 9. Il train set, in particolare, contiene immagini di 60.000 numeri, mentre il test set contiene 10.000 esempi.

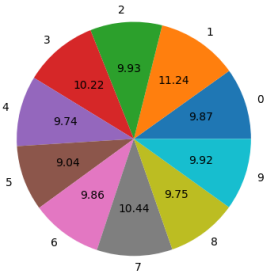

Figura 1: Distribuzione cifre nel train set.

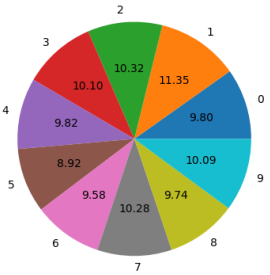

Figura 2: Distribuzione cifre nel test set.

Come è possibile osservare in Figura 1 e Figura 2, la distribuzione dei valori è abbastanza omogenea nei due dataset, non sono presenti classi con una minoranza importante che possono complicare l'apprendimento delle relative cifre.

Nella seguente immagine, invece, è possibile osservare come sono fatte le immagini: avremo bisogno di un solo canale poiché sono in bianco e nero.

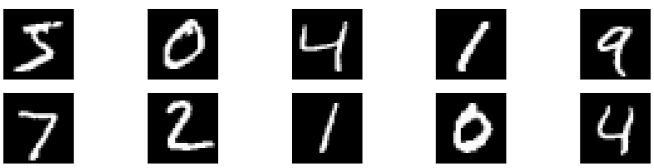

Figura 3: Prime 5 cifre nel train set (sopra) e prime 5 cifre nel test set (sotto).

3 Progettazione e design della rete

3.1 Preprocessing

Come prima operazione i dataset di train e di test sono stati riscalati in valori compresi tra 0 e 1 e dopo di ciò è stato necessario applicare un re-shape alle immagini. In particolare, è stato indicato al modello che verranno rappresentate come 28x28x1, dove 28x28 è la dimensione originale, mentre 1 indica il canale per il colore: dato che le immagini sono in bianco e nero è richiesto solo un canale.

Successivamente le etichette sono state processate e binarizzate. In particolare, è stata applicata la funzione *to_categorical* di *keras.utils*, che ha permesso di ottenere 10 nuovi attributi al posto del singolo attributo numerico. I nuovi 10 attributi sono nel formato *one-hot* e presentano un 1 soltanto nell'attributo riguardante la cifra corretta, negli altri attributi vi è uno zero.

Infine, il dataset di train è stato suddiviso in train e validation set, utilizzando la funzione *train_test_split* di *sklearn*. La suddivisione ha visto l'assegnamento del 20% dei dati al validation set, ottenendo quindi un train di 48.000 samples e un validation di 12.000 samples. Il validation verrà poi utilizzato per fare il tuning dei parametri durante la fase di training e il test set verrà utilizzato solo per misurare le performance del modello.

3.2 Stack convoluzionali

Una CNN è suddivisa in due macro blocchi, un blocco di *convoluzione* seguito da un blocco *fully-connected*. In questo paragrafo viene descritto il blocco di convoluzione.

E' stato deciso di utilizzare 3 stack di filtri convoluzionali, che permettessero di estrarre determinate informazioni, dette features, e rimpicciolire la matrice iniziale, ovvero l'immagine. La struttura costruita è la seguente:

1. Il primo stack convoluzionale è composto da 4 elementi importanti:
    - Layer di convoluzione con 8 filtri di dimensione 3x3(x1), niente Zero Padding e Stride 1.
    - Funzione di attivazione ReLU.
    - Layer di Pooling, in particolare è stato utilizzato un MaxPooling 2x2 con Stride 2.
    - Layer di Normalizzazione. La libreria Keras fornisce soltanto una BatchNormalization.
2. Il secondo stack convoluzionale è molto più semplice:
    - Layer di convoluzione con 16 filtri di dimensione 3x3(x8), niente Zero Padding e Stride 1.
    - Funzione di attivazione ReLU.
3. Il terzo stack convoluzionale è una via di mezzo tra i precedenti:
    - Layer convoluzionale con 4 filtri di dimensione 3x3(x16), niente Zero Padding e Stride 1.
    - Funzione di attivazione ReLU.
    - Layer di MaxPooling 2x2 con Stride 2.

La ReLU è stata preferita ad altre Activation Functions perchè permette maggiore velocità per convergere, non essendo limitata superiormente ed essendo molto "ripida". La buona velocità di apprendimento della ReLU è ottenuta anche grazie alla capacità di non considerare neuroni con pesi troppo bassi.

La scelta di mantenere i filtri della stessa dimensione è stata presa ispirandosi al modello VGG-16. Altre dimensioni sono state sperimentate, come ad esempio partire con filtri 5x5, scendere a 3x3 nel secondo stack e a 2x2 nel terzo stack, tuttavia non sono state notate differenze di performance rilevanti statisticamente.

I layer di Pooling sono utili per ridurre la dimensione dell'immagine e quindi il numero di parametri. Generalmente si utilizza il MaxPooling.

Il BatchNormalization aiuta e velocizza l'apprendimento andando a normalizzare i dati dopo ogni batch, mantenendo la media vicina a 0 e la varianza vicina a 1.

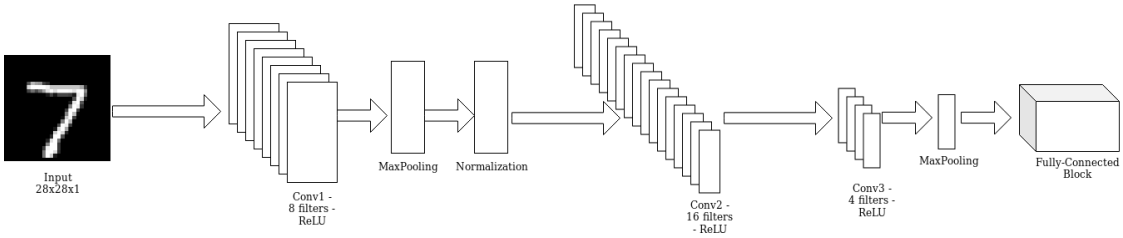

Figura 4: Schema del blocco convoluzionale.

In Figura 4 è possibile osservare una rappresentazione grafica degli stack di convoluzione. In particolare, le funzioni di attivazione sono state rappresentate internamente ai layer di convoluzione. La rappresentazione sempre più piccola sta ad indicare come l'immagine originale sia stata compressa estraendo le feature dopo ogni stack.

3.3 Stack fully-connected

Dopo il processo di convoluzione, è stato applicato prima di tutto un Dropout di 0.25. Il Dropout è una tecnica che serve principalmente per evitare overfitting, spegnendo e quindi settando a zero circa il (in questo caso) 25% dei neuroni del layer prima di esso. Successivamente, i dati in forma matriciale sono stati portati in forma vettoriale tramite la funzione *Flatten*, per permettere alle immagini di essere processate in una Fully-connected Neural Network e quindi per permettere al modello di apprendere come generalizzare e classificare nuovi input.

Il blocco fully-connected vero e proprio, è stato costruito con un hidden layer, seguito da un Dropout e quindi da un output layer. In particolare, l'hidden layer è composto da 64 neuroni, scelta presa principalmente per due motivi: il vincolo sui parametri e la dimensione dei vettori in input, ovvero 64. Anche in questo caso è stata utilizzata la ReLU come funzione di attivazione. Un semplice Dropout per spegnere circa il 10% dei neuroni è stato implementato dopo l'hidden layer per forzare la rete ad imparare anche utilizzando una ridotta capacità. Infine, l'output layer è stato realizzato con 10 neuroni (ovvero il numero di classi del dataset) e una *softmax* come funzione di attivazione, scelta classica per quanto riguarda task di apprendimento multi-classe.

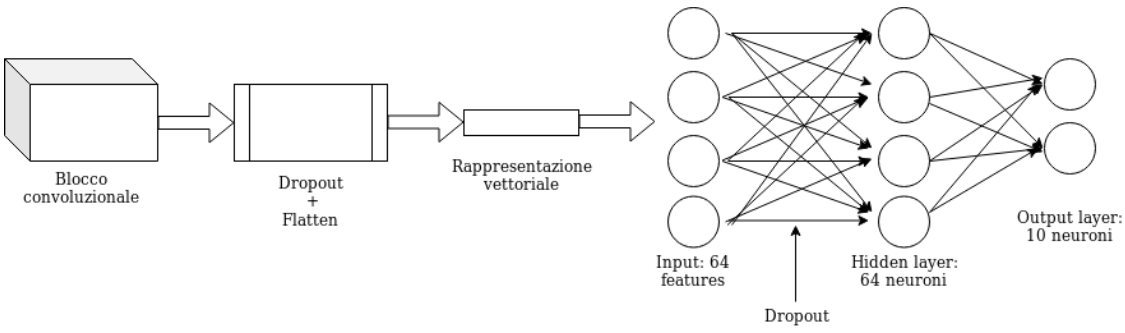

Figura 5: Schema del blocco fully-connected.

In figura 5 è possibile osservare una rappresentazione grafica del blocco fully-connected, con aggiunta anche la parte di *flattening* delle matrici.

3.4 Calcolo dei parametri e vincolo realizzativo

Come anticipato nell'introduzione, l'assignment presenta un vincolo sul numero di parametri, ovvero è possibile utilizzare un modello con al massimo 7.500 parametri. Viene di seguito riportata una tabella che mostra il *summary* del modello e successivamente verranno spiegati alcuni aspetti.

Layer	# Parametri	Output
Conv1	80	26x26x8
MaxPooling1	0	13x13x8
BatchNormalization	32	13x13x8
Conv2	1.168	11x11x16
Conv3	580	9x9x4
MaxPooling2	0	4x4x4
Dropout1	0	4x4x4
Flatten	0	64
Dense1	4.160	64
Dropout2	0	64
Dense2	650	10

Tabella 1: Summary del modello.

Come è possibile calcolare, la somma dei parametri utilizzati dal modello è **6.670**. Di questi però, vi sono 16 parametri classificati come *non-trainable* per via del BatchNormalization. I parametri allenabili sono quindi 6.654.

Entrando nel dettaglio del numero dei parametri, il layer Conv1 presenta 80 parametri che sono ottenuti dal numero di filtri moltiplicato per la loro dimensione più un bias dovuto al numero di filtri: 8x3x3x1 + 8 = 80. Successivamente, il MaxPooling 2x2 con Stride 2 ha permesso di dimezzare la dimensione della matrice, ottenendo una 13x13.

Il layer Conv2 presenta 1.168 parametri, ottenuti banalmente, come per Conv1, dalla moltiplicazione più somma: 16x3x3x8 + 16 = 1.168, dove 16 è il numero di filtri che a loro volta introducono un bias di 16 e 3x3(x8) è la dimensione dei filtri. L'output di questo layer è 11x11x16, misura ottenuta dallo Stride 1 su matrici 13x13 con filtri 3x3 e 16 canali dovuti al numero di filtri utilizzati.

Lo stesso calcolo può essere applicato al layer Conv3, che moltiplica 4 filtri di dimensione 3x3(x16) al quale risultato va aggiunto il bias introdotto dai filtri: 4x3x3x16 + 4 = 580. L'output di questo layer è 9x9x4, misura ottenuta applicando filtri 3x3 con Stride 1 a matrici 11x11 e 4 canali dovuti al numero di filtri. Il MaxPooling 2x2 in questo caso va a più che dimezzare la dimensione delle matrici, portando l'output ad un 4x4x4.

Il Flatten non ha fatto altro che appiattire la matrice 4x4x4 in un vettore di 64 elementi.

Il numero di parametri del primo layer Dense è stato ottenuto moltiplicando il numero di parametri del vettore per il numero di neuroni, più un bias unitario introdotto da ogni neurone: 64x64 + 64 = 4.160. In questo caso l'output non subisce variazioni e corrisponde al numero di neuroni, 64.

Il numero di parametri del secondo layer Dense, ovvero l'output è stato ottenuto simmetricamente andando a moltiplicare il numero di parametri dei vettori per il numero di neuroni, più il bias introdotto da ogni neurone: 64x10 + 10 = 650.

3.5 Scelte in fase di training

Dopo aver definito la struttura del modello, sono stati impostati alcuni parametri fondamentali durante il train del modello quali il numero di epochs, la dimensione del batch, funzione di Loss, callbacks e optimizer.

- Il numero di epochs è stato influenzato dal monitor di EarlyStop utilizzato. In particolare, è stato definito 35 come numero massimo di epochs in cui allenare il modello, ma grazie

al EarlyStop impostato dopo 3 epochs nelle quali la *validation loss* non migliora, tale cifra non è stata quasi mai raggiunta nel corso dei test. In media infatti, il monitor interrompeva l'apprendimento intorno alle 25 epochs.

- Per la dimensione di batch è stato scelto di utilizzare l'approccio a mini-batch con 64 record per iterazione. Questa dimensione ha riscontrato, in media, risultati migliori rispetto a dimensioni maggiori come 128 e 256, permettendo più aggiustamenti dei pesi durante le varie epochs.

- Come funzione di Loss è stata utilizzata la *categorical crossentropy* che, spesso, in coppia con la funzione di attivazione Softmax, permette ottimi risultati in problemi di classificazione multi-classe.

- Come callback è stato utilizzato soltanto il monitor di EarlyStop già descritto, con la funzione molto importante di *restore_best_weights* abilitata. Questa funzione ha permesso di recuperare i pesi che hanno prodotto la più bassa validation loss e impostare il modello con essi.

- L'optimizer utilizzato è stato Adam, con learning rate 0.001 e variante AMSGrad abilitata, la quale permette una miglior convergenza in alcune situazioni. Differenti misure di learning rate sono state sperimentate come 0.01, 0.002 e 0.0001. Nel primo caso, utilizzando 0.01 è stato visto come la loss tendeva a scendere in poco tempo ma a risalire poco dopo, segno che il modello non era stato capace di trovare "il varco" corretto tra i minimi locali. Si è scelto quindi di utilizzare un learning rate più basso. 0.001 ha mostrato i risultati migliori valutando sia le tempistiche di learning e sia le performance. Questo, anche rispetto al più simile 0.002 ma soprattutto rispetto al più lento 0.0001, che dopo 35 epochs non aveva ancora trovato un minimo presentando performance non soddisfacenti.

  Anche Adadelta è stato sperimentato, mantenendo però i parametri di default. Non sono state notate differenze con rilevanza statistica tra i due optimizer, per cui si è scelto di utilizzare Adam.

  Prima della release finale del progetto, è stato sperimentato un monitor chiamato *ReduceLROnPlateau* per ridurre gradualmente il learning rate qualora la validation loss non migliorasse per più di due epochs. Partendo quindi da un learning rate iniziale di 0.01, si arriva gradualmente fino a 0.0001. Ovviamente con questa tecnica sono state ottenute performance migliori, tuttavia è stato scelto di non utilizzarla nella release finale per due motivi: il numero maggiore di epochs necessarie, che comporta una maggiore lentezza della fase di apprendimento, e le migliorie apportate rispetto alla versione descritta nel report, nel dettaglio si parla di 0.001 in più sulla accuracy e circa la stessa quantità in meno sulla loss, ovvero cifre non troppo significative.

4 Analisi dei risultati

I risultati si sono rivelati davvero soddisfacenti tenendo conto del vincolo importante presente sul numero di parametri. Infatti, pur considerando la stocasticità del modello che implica un possibile risultato diverso dopo ogni allenamento del modello, la validation loss è sempre scesa sotto lo 0.045 e la validation accuracy è sempre salita sopra lo 0.98, toccando in alcune esecuzioni anche lo 0.99.

I risultati mostrati di seguito si riferiscono ad una esecuzione che è stata interrotta dal monitor di Early Stopping dopo 23 epochs. In Tabella 2 vengono mostrate le ultime epochs di tale esecuzione, che hanno portato all'interruzione dell'apprendimento dopo non aver migliorato il valore di Validation Loss ottenuto alla ventesima epoch. In particolare, è possibile osservare come l'accuracy sul validation set sia in realtà migliorata, tuttavia siamo interessati a minimizzare la loss il più possibile e per questo sono stati salvati i pesi ottenuti con la epoch numero 20.

# Epoch	Train Loss	Train Accuracy	Val Loss	Val Accuracy
20	0.0880	0.9724	0.0419	0.9858
21	0.0838	0.9734	0.0427	0.9860
22	0.0864	0.9733	0.0458	0.9855
23	0.0834	0.9734	0.0423	0.9863

Tabella 2: Risultati ottenuti in fase di training e validation.

I risultati sono mostrati graficamente in Figura 6 e in Figura 7. E' possibile osservare come, grazie soprattutto ai Dropout inseriti, il modello sia stato in grado di ottenere performance migliori sul validation set rispetto al training set, evitando quindi l'overfit in maniera impeccabile.

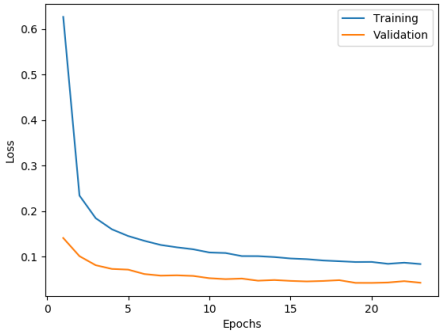

Figura 6: Curva della Loss del train e del validation set.

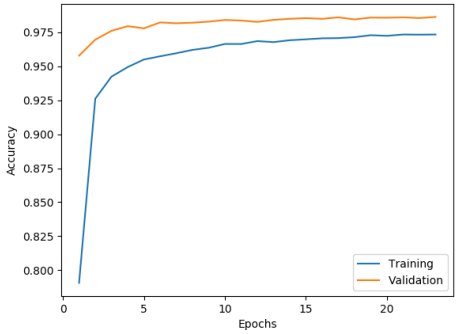

Figura 7: Curva dell'Accuracy del train e del validation set.

Dopo la fase di training, il modello è stato utilizzato sul test set, il quale è provvisto di etichette per permettere un controllo delle performance ottenute. Sorprendentemente, i risultati sono stati addirittura migliori rispetto a quelli ottenuti con il validation set. Nella seguente tabella è possibile osservare i valori di Loss e di Accuracy sul test set.

Loss	Accuracy
0.0372	0.9886

Tabella 3: Risultati sul test set.

Oltre all'accuratezza del modello, in un problema di classificazione multi-classe è spesso conveniente misurare anche le performance con metriche diverse, quali la Precision, la Recall e la f1-Measure. Anche in questo caso, i risultati si sono rivelati ottimi, raggiungendo addirittura il 100% di Recall (arrotondato) sulla classe rappresentante la cifra 8, il che vuol dire che tutte le immagini raffiguranti il numero 8 sono state classificate correttamente. Le classi rappresentati le cifre 0, 1 e 7 invece, hanno ottenuto il 100% di Precision (arrotondato), il che vuol dire che non vi sono state immagini di altre cifre ad essere classificate come 0, 1 o 7 (falsi positivi). Per una visione estesa delle misure per classe, viene riportata la seguente tabella.

Classe	Precision	Recall	f1-Score	# Immagini
0	1.00	0.99	0.99	986
1	1.00	0.99	1.00	1 142
2	0.98	0.99	0.99	1 024
3	0.99	0.99	0.99	1 007
4	0.99	0.98	0.99	994
5	0.99	0.98	0.99	895
6	0.98	0.99	0.99	947
7	1.00	0.98	0.99	1 046
8	0.98	1.00	0.99	955
9	0.98	0.98	0.98	1 004

Tabella 4: Metriche Precision, Recall, f1-Measure per classe.

Globalmente, in termini di Macro-Average e di Weighted-Average, sia la Precision che la Recall (e di conseguenza anche la f1-Measure) hanno ottenuto 0.99.

5 Conclusioni

Il vincolo sul numero di parametri agisce sicuramente da limitatore di performance sul modello. Tuttavia, come emerge dai risultati ottenuti e analizzati, è possibile ottenere comunque delle performance quasi perfette per problemi semplici. Un esercizio simile può essere utile ovviamente per situazioni reali, dove per eventuali limitazioni hardware o per altri motivi, sono richiesti modelli vincolati su alcuni aspetti.

Per quanto riguarda le dimensioni dei filtri, numero di layer convoluzionali, numero di neuroni etc., non esistono purtroppo linee guida ben precise. E' possibile, tuttavia, ispirarsi a soluzioni famose oppure provare differenti e numerosi settaggi, ovviamente dopo aver utilizzato le corrette funzioni e ottimizzazioni, per individuare i parametri che permettono le migliori performance.