MULTICLASS MNIST DIGITS CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK WITH OPTIMIZATION

STEFANO BIONDI

1. Dataset e Preprocessing

Il dataset è il MNIST digits. Importandolo direttamente da Keras si hanno un training set ed un test set formato rispettivamente da 60000 e 10000 immagini in scala di grigio, 28x28 pixel. Ogni immagine rappresenta una delle 10 cifre decimali. Il problema è quindi un problema di classificazione multiclasse a 10 classi.

Come mostrato in figura (1) la distribuzione di frequenza delle classi è bilanciata, quindi nessuna tecnica di over o under sampling è stata applicata.

```
[[0.         1.         2.         3.         4.         5.         6.         7.         8.         9.        ]
 [0.09871667 0.11236667 0.0993     0.10218333 0.09736667 0.09035    0.09863333 0.10441667 0.09751667 0.09915   ]]
```

Figura 1. Distribuzione di Frequenza Classe Target

Ogni valore della matrice 28x28 è un numero intero compreso tra 0 e 255. I dati, quindi, sono stati riscalati tra 0 e 1. Le immagini, infine, sono state mescolate randomicamente e divise in Training e Validation Set, nelle percentuali di 80% e 20% rispettivamente utilizzando uno stratified sampling. Infine la variabile target, associata al training set, è stata categorizzata.

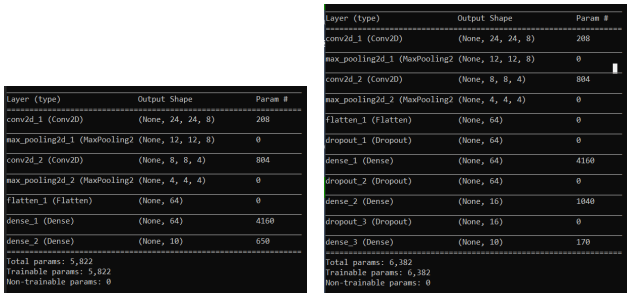
2. Models

Il modello costruito è un *Convolutional Neural Network* con un massimo di 7500 parametri.

Per rispettare la constraint è stato deciso di creare un modello con due strati convoluzionali, il primo a 8 filtri ed il secondo a 4 filtri, entrambi con *kernel size* 5x5 e *strides* $1x1$. Dopo ognuno dei due strati convoluzionali, per ridurre ulteriormente la dimensionalità è stato aggiunto uno strato di *max pooling* di grandezza 2x2, il primo con strides 2x2. Infine, con l'output della rete convoluzionale è stato generato un vettore di 64 valori che è diventato l'input di una rete neurale artificiale con uno strato denso nascosto di 64 nodi e uno strato di output formato dai 10 nodi che rappresentano le 10 classi che il modello andrà a predire.

Come si può osservare nella figura 2(a) la dimensionalità dell'input, (28x28), è stata ridotta dal primo strato convoluzionale a 8 filtri in (24x24), ulteriormente ridotta dal max pooling a (12x12). Sucessivamente gli 8 filtri sono stati ridotti a 4 di dimensionalità (8x8) dal secondo strato convoluzionale che combinato con il seguente max pooling ha ridotto la dimensionalità a 4 filtri (4x4) per un totale di 64 valori. Il numero totale di parametri da stimare per il modello è quindi di $5822 < 7500$.

Essendo un problema di classificazione multicalsse è stata utilizzata, come funzione di attivazione dell'output layer, la *Softmax* e, come funzione di loss, la *Categorical Crossentropy*.
Ogni addestramento è stato impostato per arrivare ad un massimo di 200 *epoche* e, per evitare problemi di overfitting è stato inserito un *early stopping* sull'invarianza per 15 epoche del minimo della loss function del validation set. Infine, oltre agli andamenti di loss e accuracy in funzione delle epoche, sono state calcolate



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 24, 24, 8)	208
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 8)	0
conv2d_2 (Conv2D)	(None, 8, 8, 4)	804
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 4)	0
flatten_1 (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 10)	650

Total params: 5,822
Trainable params: 5,822
Non-trainable params: 0

(a) Signolo Strato Nascosto

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 24, 24, 8)	208
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 8)	0
conv2d_2 (Conv2D)	(None, 8, 8, 4)	804
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 4)	0
flatten_1 (Flatten)	(None, 64)	0
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4160
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 16)	1040
dropout_3 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 10)	170

Total params: 6,382
Trainable params: 6,382
Non-trainable params: 0

(b) 2 Strati Nascosti

FIGURA 2. Architettura modelli e conteggio dei parametri

la confuzion matrix e le misure di Accuracy, Precision, Recall e F-measure per il modello che, in fase di addestramento, ha manifestato l'accuracy maggiore per il validation test.

Sono stati testati i principali ottimizzatori utilizzati nel deep learning, lo *Stochastic Gradient Descent*, lo *RMSprop* e la sua versione con momentum integrato: l'*Adam*.
Inizialmente si è utilizzata la batch size di default del metodo fit della libreria Keras che è pari a 32.
Tutti e tre gli ottimizzatori hanno dimostrato ottime performance:

Ottimizzatore	Accuracy	F-measure	Epoch
SGD	0.99	$0.98 - 1.00$	54
RMSprop	0.99	$0.98 - 0.99$	30
Adam	0.98	$0.97 - 0.99$	27

TABELLA 1. Performance degli ottimimzzatori per il Validation Set dopo la divisione dal Training Set, Batch Size 32

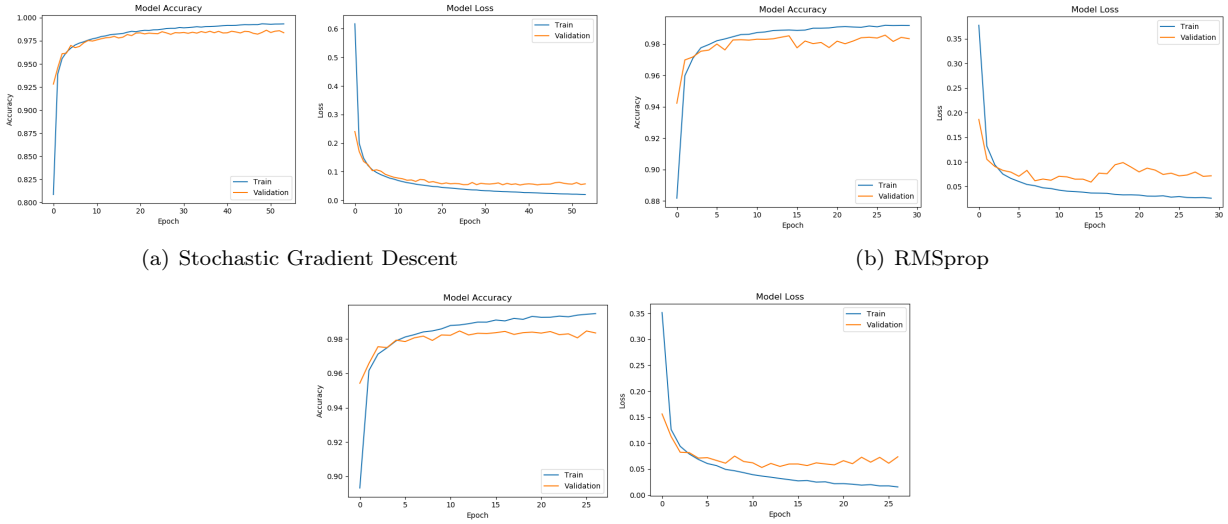
Come si può osservare nella figura (3) tutti e tre i modelli presentano dell'overfitting e, considernado che l'early stop ha interrotto l'addestramento della rete alla 54^{a} iterazione e che un basso numero di batch size può comportare effetti di regolarizzazione, si è deciso di aumentare la batch size per il modello con ottimizzatore Stochastic Gradient Descent. Questo infatti è il modello che ha ottenuto le performance migliori.

Con una batch size di 1024 l'early stop ha interrotto l'apprendimento dopo 76 epoche ma le misure di *Accuracy* e *F-Measure* non hanno raggiunto i livelli precedenti con rispettivamente 0.98 e $0.97 - 0.99$, mantenendo comunque ancora degli effetti di overfitting.

Per distendere l'apprendimento ed evitare l'overfitting si è deciso di aggiungere al layer denso la regolarizzazione $L1(0.0001)$ e la regolarizzazione $L2(0.001)$. Il modello con la regolarizzazione $L1$ ha performato meglio. In 139 epoche di addestramento ha evidenziato un'accuracy di 0.98 e una F-measure compresa in $0.98 - 0.99$. Gli effetti di overfitting però non sono stati ridotti.

Si è quindi deciso di applicare un *Dropout* tra l'input layer dopo gli strati convoluzionali e il layer nascosto e tra quest'ultimo e l'output layer con un rate del 10%. In 159 epoche di addestramento è stata raggiunta un'accuracy pari a 0.98 e una F-measure che varia in un intervallo compreso tra 0.98 e 0.99 senza però diminuzione dell'overfitting.

Infine si è quindi deciso di aggiungere un ulteriore layer nascosto formato da 16 nodi con regolarizzazione $L1(0.0001)$ e dropout di 0.1, portando così i parametri ad un totale di $6382 < 7500$, figura 2(b). In questo caso l'early stopping non è intervenuto ed al termine delle 200 epoche il modello migliore ha raggiunto un'accuracy pari a 0.99 con F1-Measure compresa tra $0.98 - 0.99$ e, come si può vedere in figura 4(a), non si sono avuti effetti di overfitting.

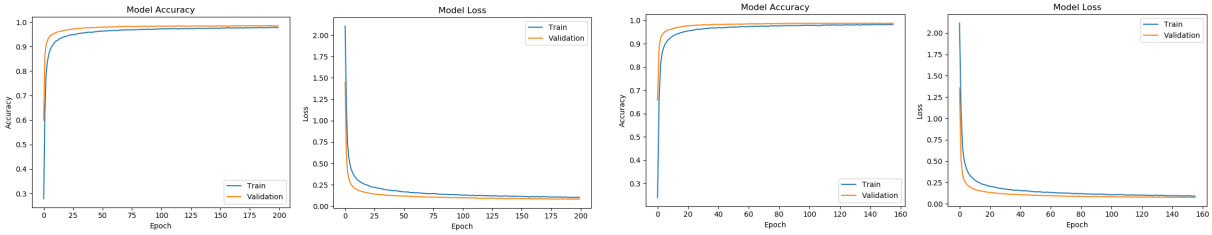


(a) Stochastic Gradient Descent

(b) RMSprop

(c) Adam

FIGURA 3. Performance degli Otimizzatori con parametri di default, Validation Set splittato dal Training Set



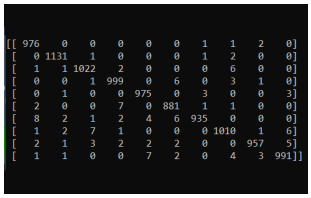
(a) Momentum = 0.00, Learning Rate = 0.01

(b) Momentum = 0.90, Learning Rate = 0.05

FIGURA 4. Accuracy e Loss dei modelli con 2 Strati Densi, Validation Set splittato dal Training Set

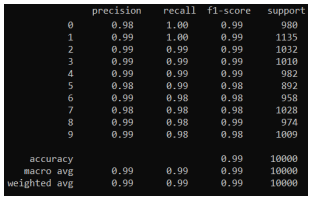
Visto che l'apprendimento non è stato terminato si è deciso di aumentare il momentum dell'ottimizzatore a 0.99 e il learning rate a 0.05 così da accelerare la convergenza al minimo della funzione di loss. Il risultato è un modello che in 156 epoche ha raggiunto una accuracy del 0.99 e una F-measure del $0.98 - 0.99$ senza manifestare fenomeni di overfitting, figura 4(b).

Per concludere è stato testato il modello sul test set fornito da Keras e il risultato è stata la conferma di un modello con ottime performance e senza la presenza di fenomeni di overfitting, figura 5.



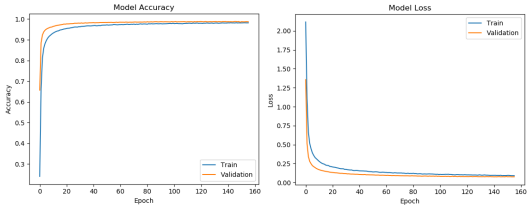
```
[[ 976    0    0    0    0    0    1    1    2    0]
 [   0 1131    1    0    0    0    1    2    0    0]
 [   1    1 1022    2    0    0    0    6    0    0]
 [   0    0    1  999    0    6    0    3    1    0]
 [   0    1    0    0  975    0    3    0    0    3]
 [   2    0    0    7    0  881    1    1    0    0]
 [   8    2    1    2    4    6  935    0    0    0]
 [   1    2    7    1    0    0    0 1010    1    6]
 [   2    1    3    2    2    2    0    0  957    5]
 [   1    1    0    0    7    2    0    4    3  991]]
```

(a) Confusion Matrix



	precision	recall	f1-score	support
0	0.98	1.00	0.99	980
1	0.99	1.00	0.99	1135
2	0.99	0.99	0.99	1032
3	0.99	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.98	0.99	0.98	892
6	0.99	0.98	0.98	958
7	0.98	0.98	0.98	1028
8	0.99	0.98	0.99	974
9	0.99	0.98	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

(b) Misure di Performance



(c) Accuracy e Loss del Training e del Test Set

FIGURA 5. Confusion Matrix e Misure di Performance per la Valutazione del Test Set fornito da Keras