# MULTICLASS IMAGES PREDICTION W/O CONVOLUTIONAL LAYERS AND DIMENSIONALITY REDUCTION THANKS TO AUTOENCODERS

STEFANO BIONDI

## 1. Dataset e Preprocessing

Il training set è formato da 14000 immagini in scala di grigio, 28x28 pixel. Ogni immagine rappresenta una lettera dell'alfabeto tra la lettera P e la lettera Z. Il problema è quindi una classificazione multiclasse ad 11 classi. Come mostrato in figura (1) la distribuzione di frequenza delle classi è bilanciate e nessuna tecnica di over o under sampling è stata applicata.

```
[[16.        17.        18.        19.        20.        21.        22.        23.        24.        25.        26.       ]
 [ 0.0925     0.09035714  0.09614286  0.09492857  0.09542857  0.09264286  0.09064286  0.09478571  0.09442857  0.09435714  0.06378571]]
```

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 e la matrice 28x28 è stata trasformata in un vettore di 784 valori. I dati sono stati mescolati randomicamente e divisi in Training e Validation Set, nelle percentuali di 80% e 20% rispettivamente. Infine la variabile target, associata al training set, è stata categorizzata.

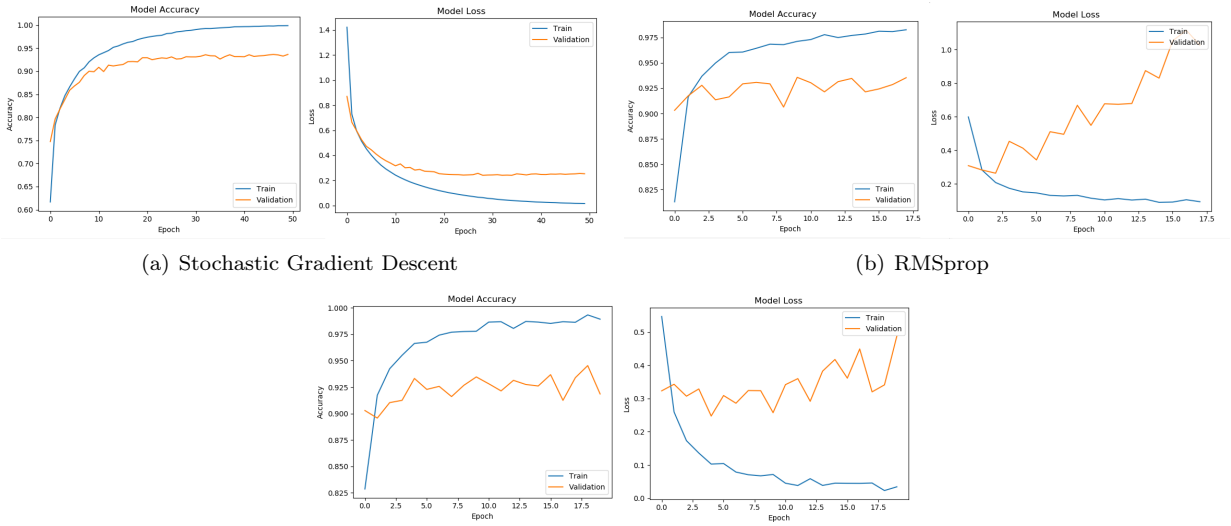
## 2. Models

Essendo un problema di classificazione multicalsse sono state considerate reti neurali dense che avessero 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 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*. La rete addestrata è stata così composta: 3 strati nascosti. I primi due layer composti da 1024 nodi, visto che lo strato di input contiene 784 nodi e l'ultimo composto da 256 nodi, visto che lo strato di output contiene 11 nodi.

Inizialmente si è utilizzata la batch size di default del metodo fit della libreria Keras che è pari a 32.
Lo SGD con i parametri di default ha dimostrato una buona performance con un'*Accuracy* di 0.94 e una *F-measure* che oscilla tra lo 0.91 e lo 0.98. Lo *RMSprop* ha avuto performance molto simili al precedente sia in termini di accuracy che di F-measure, però come possiamo vedere in figura (2) il valore di *Loss* del validation test è divergente, il che ci fa escludere che lo RMSprop in queste condizioni sia un buon ottimizzatore per la funzione di loss apllicata al nostro problema. L'*Adam* ha mostrato performance migliori con un'accuracy pari a 0.95 ma, come per lo RMSprop, ha mostrato effetti di overfitting avendo il valore di loss del validation test non convergente.

Per tutti e 3 gli ottimizzatori si sono avute performance migliori utilizzando la *Relu* come funzione di attivazione degli hidden layer anzichè la *LeakyReLU*. Non sono state considerate altre funzioni di attivazione in quanto, empiricamente, queste sono preferibili per gli strati nascosti.




(a) Stochastic Gradient Descent

(b) RMSprop

(c) Adam

FIGURA 2. Performance degli Otimizzatori con parametri di default

Visti i risultati si è deciso di proseguire nell'ottimizzazione degli iperparametri per l'ottimizzatore *Stochastic Gradient Descent*. L'early stop ha interrotto l'addestramento della rete alla 32ª iterazione e, sapendo che un basso numero di batch size può comportare effetti di regolarizzazione sulla loss function si è deciso di aumentare la batch size a 256.
L'addestramento è risultato molto più veloce, si è passati da una media di tempo di addestramento di $6s$ per epoca ad una media di $2s$ per epoca. L'early stop in questo caso non è intervenuto quindi non abbiamo avuto stazionarietà del valore di loss, nell'evaluation set, per almeno 15 epoche. Inoltre l'accuracy è risultata pari a 0.93 quindi possiamo ipotizzare che l'algoritmo non sia stato addestrato a sufficienza. Per accelerare la convergenza del SGD si è utilizzato il momentum. In accordo con le euristiche si sono utilizzati i valori di 0.5, 0.9 e 0.99.

| Momentum | Accuracy | F-measure | Epoch |
|---|---|---|---|
| 0.50 | 0.93 | 0.89 – 0.98 | 120 |
| 0.90 | 0.93 | 0.89 – 0.97 | 45 |
| 0.99 | 0.94 | 0.90 – 0.97 | 24 |

TABELLA 1. Performance SGD al variare del Momentum, Batch Size 256

Si è quindi deciso di aumetare la batch size a 512 e 1024 mantenendo il momentum fisso a 0.99 e di abbassare il momentum aumentando conteporaneamente il learning rate.

| Learning Rate | Batch Size | Momentum | Accuracy | F-measure | Epoch |
|---|---|---|---|---|---|
| 0.01 | 512 | 0.99 | 0.94 | 0.89 – 0.98 | 30 |
| 0.01 | 1024 | 0.99 | 0.94 | 0.90 – 0.98 | 42 |
| 0.05 | 512 | 0.90 | 0.94 | 0.90 – 0.98 | 26 |
| 0.05 | 1024 | 0.90 | 0.94 | 0.90 – 0.97 | 40 |
| 0.05 | 512 | 0.50 | 0.93 | 0.89 – 0.97 | 64 |
| 0.05 | 1024 | 0.50 | 0.93 | 0.88 – 0.97 | 40 |

TABELLA 2. Performance SGD al variare di Learning Rate, Momentum e Batch Size

Si osserva che le combinazioni di learning rate 0.01, batch size 1024, momentum 0.99 e learning rate 0.05, batch size 512, momentum 0.90 sono quelle con performance migliori. Per distendere l'apprendimento ed evitare l'overfitting si è deciso di aggiungere ai due modelli migliori la regolarizzazione $L1(0.0001)$ e la regolarizzazione $L2(0.001)$. Il secondo modello, con la regolarizzazione $L1$ ha performato meglio degli altri. In 69 epoche di addestramento ha evidenziato un'accuracy di 0.94 e una F-measure compresa in $0.91 - 0.98$.

Infine si è deciso di applicare un *Dropout* tra gli hidden layer e tra l'ultimo hidden layer e l'output layer con un rate del 40%. In 159 epoche di addestramento è stata raggiunta un'accuracy pari a 0.95 e una F-measure che varia in un intervallo compreso tra 0.92 e 0.98.

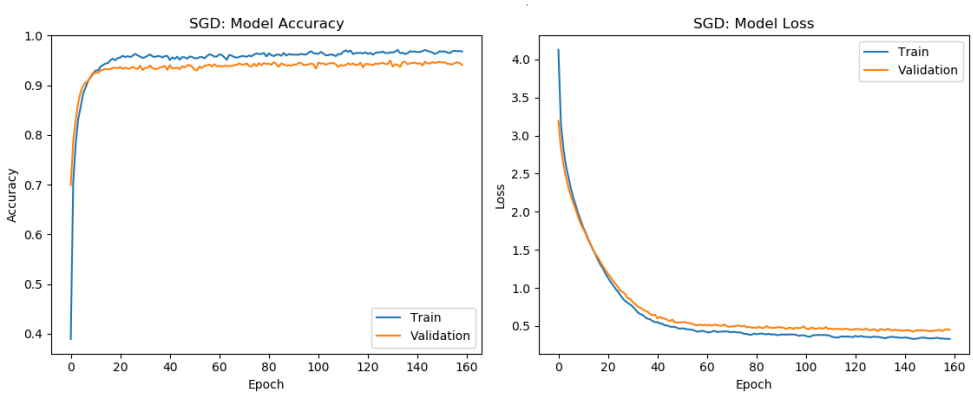



FIGURA 3. Accuracy e Loss del modello con performance migliore

## 3. AUTOENCODERS

Per diminuire la dimensionalità delle immagini mantenendo alta la quantità di informazione contenuta nella compressione si è costruito un autoencoder.

E' stato implementato un singolo strato fully-connected come encoder e decoder ed è stato addestrato per ricostruire le immagini del training set. E' stato utilizzato un ottimizzatore *adadelta* con loss *binary_crossentropy*. E' stata impostata una compressione di fattore 24.5. L'output dell'encoding avrà così una dimensionalità di 32. In figura (4) si può notare che la perdita di informazione tra l'input e l'input ricostruito decodificando l'output dell'encoder è minima. Infatti è possibile riconoscere la maggior parte delle lettere.

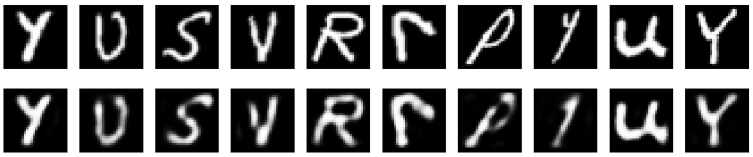


FIGURA 4. Prime 10 lettere del training set con rispettive ricostruzioni dell'autoencoder

Infine, dopo aver diminuito la dimensionalità del training e del validation set grazie all'encoder, è stata addestrata una rete neurale con 2 strati nascosti con 64 nodi e funzione di attivazione *Relu* ciascuno. Seguendo i ragionamenti fatti precedentemente si sono scelti, come funzione di attivazione dello strato di output, la *Softmax* e, come ottimizzatore. lo *Stochastic Gradient Descent* con learning rate pari a 0.05. La rete è stata addestrata per 121 epoche e come risultato si è avuta un'accuracy pari a 0.91 e una F-measure che oscilla tra 0.86 e 0.96.

Come ci si poteva aspettare il modello che ha come input i dati codificati non riesce a performare come il modello che ha come input i dati originali. Questo perchè nella fase di encoding si perde dell'informazione, non molta come visto nella figura (4), ma a sufficienza da avere differenze negative di performance nella valutazione del modello.