Anselmo Raffaele
Matricola 846842
r.anselmo@campus.unimib.it

ASSIGNMENT 2

ADVANCED MACHINE LEARNING

L'assignment consiste nella predizione di immagini delle lettere P-Z in scala di grigi. Dopo una breve esplorazione dei dati e alcune operazioni di preprocessing si sviluppa una rete neurale tradizionale e se ne valutano le performances.
Successivamente si analizzano le performances di un autoencoder a livello grafico ed infine si utilizza la rappresentazione *encoded* generata da quest'ultimo per risolvere il problema della classificazione supervisionata.

1. Dataset Overview and Preprocessing

Il dataset consiste di 14000 osservazioni di training etichettate e 8800 di test senza etichetta. Queste sono immagini a colori delle lettere P-Z scritte a mano. Le classi risultano abbastanza bilanciate.

Il training set viene a sua volta suddiviso in train set (80%) e validation set (20%).

Prima di procedere con le fase di learning, vengono eseguite delle operazioni sul dataset per renderlo più per l'algoritmo. Tutti i valori vengono normalizzati fra 0 e 1 e si esegue un *reshape* delle immagini che le appiattisce in vettori di dimensione 784.

Le immagini, ora in bianco e nero, sono pronte per essere usate come input della rete neurale.

Ai fini di valutazione dei modelli vengono definite le funzioni ***plot_metrics***, ***plot_loss*** e ***plot_roc***.

2. Neural Network Model

Con la funzione ***make_model*** definiamo la rete neurale che consiste in 2 Hidden Layers da 512 neuroni ciascuno e due Layers di Dropout con rate = 0.5.

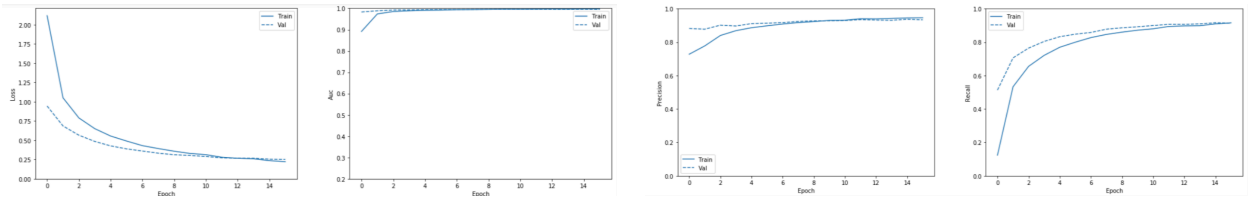
Negli hidden layer la funzione di attivazione è la ***ReLU*** *(Rectified Linear Unit),* che è efficiente nella back propagation degli errori e ha la caratteristica di attivare pochi neuroni in momenti diversi rendendo la rete sparsa e di facile computazione*,* mentre per il layer di output si è scelta la ***softmax***, molto utile nei problemi di classificazione con *n* classi ($n > 2$) anche perché continua e differenziabile (a differenza della *argmax*).

Per inizializzare i pesi si è scelta la funzione **TruncatedNormal** che genera una distribuzione Normale troncata e forzano la stessa varianza forward/backward tra i layers. Come ottimizzatore è stato usato un algoritmo con learning rate adattivo, **Adam** (Adaptive Moments), che è una variazione di RMSProp + Momentum.

La funzione di perdita da ottimizzare è rappresentata dalla **Categorical CrossEntropy**.

L'algoritmo sembra apprendere bene e le performance nel validation set sono buone. Non si manifesta particolare overfitting grazie ai layers di dropout.

L'***Early Stopping*** monitora i valori dell'Auc nel validation test, che convergono presto intorno a 1 (come si vede nella figura 1b), facendo terminare l'algoritmo in anticipo rispetto alle 70 epoche precedentemente impostate.



a) Loss **b)** Auc **c)** Precision **d)** Recall

Fig. 1 Model evaluation

3. Autoencoder

"Autoencoding" è un algoritmo di compressione dati dove la compressione e la decompressione sono *data-specific,* ovvero possono comprimere solo dati simili a quelli con cui l'autoencoder è stato allenato.

La più interessante applicazione degli autoencoders consiste invece nel ***data denoising***, in quanto gli autoencoders selezionano autonomamente quali aspetti preservare dell'informazione e questo può portare all'apprendimento di proprietà utili dei dati, nascoste nella loro forma originaria.

Nel nostro caso siamo interessati a valutare le performances a livello grafico dell'autoencoder nel replicare le immagini di input dopo averne ridotto la dimensione per un fattore di compressione pari a 24.5 (si passa da dimensione 784 a 32).

Questa riduzione avviene tramite due layers, il primo con 128 neuroni ed il secondo con, appunto, 32 neuroni. I neuroni degli hidden layers hanno funzione di attivazione ReLU, mentre per il layer di output la funzione ***sigmoid*** si è rivelata quella con le performances migliori.

La funzione di ottimizzazione è ***Adadelta,*** un estensione più robusta di *Adagrad*, che adatta il learning rate sulla base di una finestra mobile degli aggiornamenti del gradiente.

La funzione di perdita è invece la ***Binary CrossEntropy***. Di seguito il summary dell'autoencoder:

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	(None, 784)	0
dense_9 (Dense)	(None, 128)	100480
dense_10 (Dense)	(None, 32)	4128
dense_11 (Dense)	(None, 128)	4224
dense_12 (Dense)	(None, 784)	101136

Total params: 209,968
Trainable params: 209,968
Non-trainable params: 0

Fig. 2 Autoencoder Summary

Dopo 70 epoche, con batch size pari a 256, l'autoencoder raggiunge ottime performances facendo convergere la loss function (binary_crossentropy) intorno a 0.19 sia nel train set che nel validation set (Fig. 3).

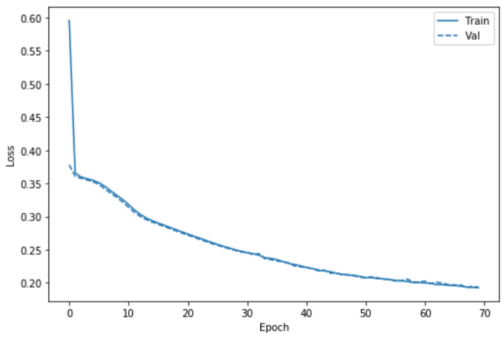


Fig. 3 Autoencoder Loss plot

L'architettura dell'autoencoder riesce quindi a comprimere e ricostruire le immagini con una perdita minima, come si può vedere nella figura 4:

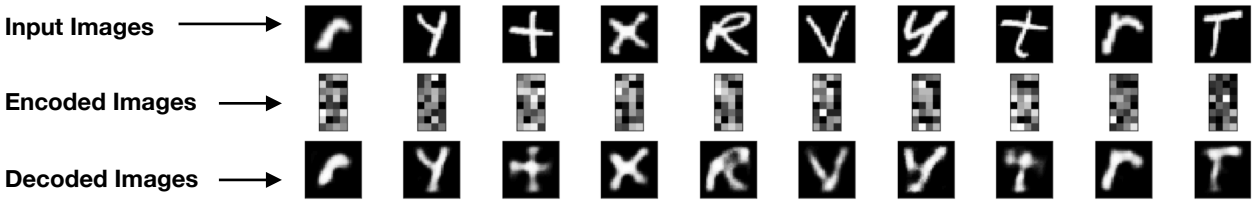


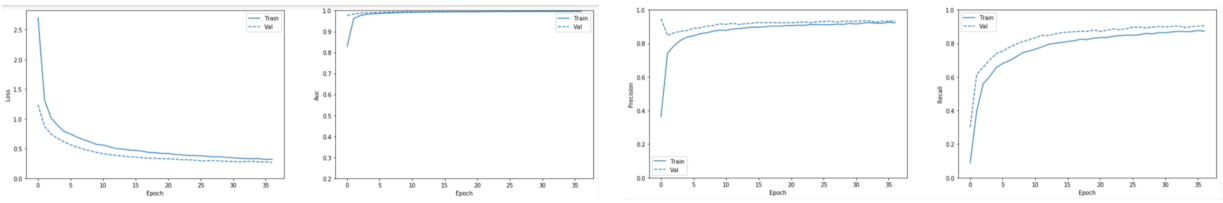
Fig. 4 Autoencoder visual investigation

4. Neural Network Model with encoded input

Si valutano le performances della rete neurale utilizzando come input le immagini compresse dall'autoencoder.

A tal proposito si modifica la funzione *make_model* in ***make_model_2*** con dimensione di Input pari a 32.

L'algoritmo performa in maniera molto simile a quanto osservato precedentemente con l'Input originale, a conferma del fatto che, malgrado la compressione per un fattore di 24.5, l'autoencoder ha mantenuto le caratteristiche principali delle immagini di Input.



a) Loss **b)** Auc **c)** Precision **d)** Recall

Fig. 5 Model with encoded Input evaluation

Una volta compilato e validato il modello si procede con l'encoding delle immagini di test e con la successiva predizione delle labels ad esse associate.

Le predizioni vengono riportate in allegato nel file *"Raffaele_Anselmo_846842_Score2.txt"*.