Advanced Machine Learning - Assignment 2

Luca Gandolfi, matricola 807485

October 2019

1 Introduzione

L'assignment consiste nel predire le rappresentazioni in greyscale delle lettere dalla P alla Z. Verrà inoltre ispezionata la capacità di codifica e la seguente ricostruzione di tali lettere tramite un Autoencoder. In questo report, sarà quindi discussa la creazione dell'autoencoder e del suo utilizzo per migliorare il task di prediction, il quale verrà svolto tramite una Feed Forward network. Verranno poi confrontate le performance con una rete costruita per predire le lettere non codificate e discusse le problematiche di quest'ultima.

2 Analisi preliminari

Prima di svolgere l'assignment, sono state svolte alcune analisi preliminari sui dataset. Per non divagare troppo dallo scopo dell'assignment, vengono qui riportate soltanto due di queste. La prima riguarda la distribuzione delle lettere nel train set: osservabile in Figura 1, il train set offre un equo bilanciamento dei record, ad eccezione per la lettera Z, anche se di poco.

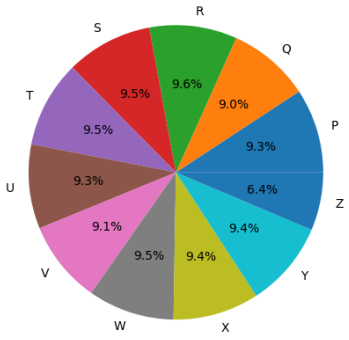

Figura 1: Torta indicante la distribuzione del train set.

La seconda analisi riguarda la presenza di immagini di lettere abbastanza strane, osservabili in Figura 2, che potrebbero portare a diverse interpretazioni a seconda della persona che le osserva.

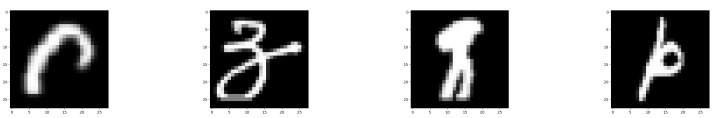

Figura 2: Esempi di lettere strane nel test set.

3 Autoencoder

Come prima operazione, i record 28x28 del train e del test sono stati ri-scalati e rappresentati come vettori di 784 attributi. A questo punto, tramite la funzione *train_test_split* di *sklearn*, il train è stato suddiviso in train e validation set, per permettere il tuning dei parametri del modello di autoencoder.

3.1 Encode

La fase di Encode ha visto la costruzione di 3 hidden layer, rispettivamente di dimensioni 128, 64, 32 neuroni. Tutti questi sono stati costruiti con una Rectified linear Unit - ReLU activation function, la quale permette una buona velocità di apprendimento andando a non considerare neuroni con pesi troppo bassi. Il primo hidden layer prende in input la rappresentazione in 784 feature delle immagini e viene allenato per ottenere una rappresentazione di, circa, 6 volte più piccola, ovvero 128 feature. Lo scopo di utilizzare più livelli con dimensioni sempre inferiori è quello di cercare di avvicinarsi al *bottleneck* del dataset, ovvero la più piccola rappresentazione dalla quale è possibile ricostruire l'input originale. In questo caso, la scelta del numero di neuroni è una scelta abbastanza standard trattando immagini 28x28.

3.2 Decode

La fase di Decode ha visto la ricostruzione della rappresentazione latente di 32 attributi in modo simmetrico alla fase di Encode. In particolare, sono stati costruiti 2 hidden layer + 1 output layer, rispettivamente di dimensioni 64, 128 e 784 neuroni. I due hidden layer riprendono simmetricamente anche le ReLU activation function, mentre l'ultimo layer, l'output, è stato costruito con una Sigmoid activation function, per ricostruire il vettore binario.

3.3 Fit e Performance

Il modello autoencoder, costituito dalla parte di encode direttamente collegata alla parte di decode, è stato compilato utilizzando come optimizer *Adadelta* e come Loss Function *binary_crossentropy*, ovvero parametri standard per questo tipo di task. La crossentropy binaria è scelta soprattutto per la funzione di attivazione sigmoidale, mentre Adadelta funziona molto bene per questi dataset.

Successivamente, il modello è stato allenato per 200 epochs, un valore non eccessivo che ha permesso di ottenere buoni risultati in termini di validation loss.

Epoch	Loss	Val Loss
100	0.1594	0.1624
200	0.1463	0.1498

Tabella 1: Performance dopo 100 e 200 epochs.

Inoltre, è stata impostata la dimensione del batch a 64, in quanto è abbastanza piccola da permettere diversi aggiustamenti dei pesi durante le varie epochs e ha riscontrato risultati migliori rispetto a dimensioni più grandi. Un confronto tra i valori di Loss e Validation Loss ottenuti dopo 200 epochs tra diverse dimensioni di batch è rappresentato in Tabella 2.

Batch size	Loss	Val Loss
64	0.1463	0.1498
128	0.1616	0.1649
256	0.1770	0.1785

Tabella 2: Dimensioni di batch diverse a confronto dopo 200 epochs.

La curva della Loss Function durante la fase di training è rappresentata in Figura 3. Ad occhio, è possibile intuire come un allenamento più prolungato avrebbe portato ad un leggero overfit.

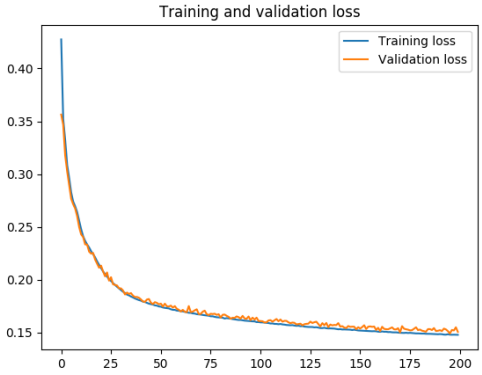

Figura 3: Grafico evoluzione della Loss e Validation Loss durante le epochs.

Dato il leggero andamento altalenante riscontrato, soprattutto verso le epochs vicine alla 200, è stato implementato un callback per salvare i pesi con performance migliori. In particolare, è stato utilizzato un *ModelCheckpoint*. Al termine del training, quindi, i pesi salvati sono stati caricati nel modello di autoencoder.

3.4 Rappresentazione latente e analisi visuale

I layer di Encode e Decode sono stati utilizzati anche come modelli separati. In particolare, il modello di Encode può essere utilizzato per ottenere una rappresentazione latente del dataset in input, mentre il modello di Decode può essere utilizzato per ispezionare la precisione di ricostruzione dell'input codificato.
Utilizzando l'algoritmo t-SNE, è possibile osservare in una dimensione inferiore, come il modello di encode abbia rappresentato i record. In particolare, il dataset di train (completo, ovvero senza la suddivisione in validation e train), è stato dato in pasto al modello di Encode. L'output di tale modello è un dataset contenente lo stesso numero di record ma soltanto 32 attributi, poichè l'ultimo layer di Encode presenta 32 neuroni. In Figura 4, è possibile osservare i risultati di t-SNE applicato a questo nuovo dataset, con una proiezione in bi-dimensione. Ogni colore rappresenta una lettera dalla P alla Z. Più un'area è densa di un colore e più significa che i record relativi alla stessa lettera sono stati codificati in modo simile. Più le aree dense di un colore sono separare tra loro e più il modello è stato in grado di codificare in modo diverso lettere diverse.

Dai risultati emersi è possibile osservare come alcune lettere siano rappresentate in modo simile tra loro, come ad esempio la U con la V o la V con la Y, le quali sono posizionate molto vicine nello spazio. Altre lettere, tipo la R, sono state più difficili da apprendere, come è possibile osservare dalle diverse aree BLU nel grafico. In linea generale però, i risultati sono stati soddisfacenti e il modello, tenendo conto delle immagini difficilmente riconoscibili anche ad occhio umano, è stato in grado di raggruppare e differenziare le lettere.

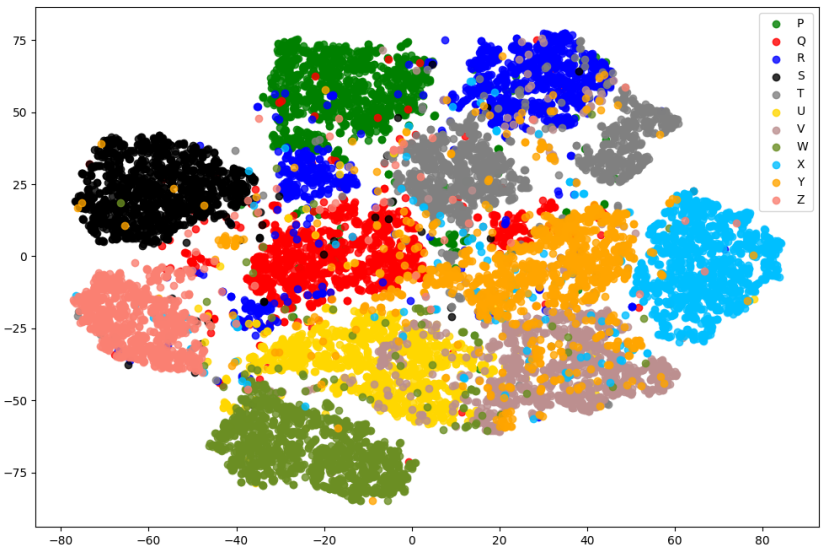

Figura 4: Proiezione dei record in 2 dimensioni con relative etichette svolta con t-SNE.

In Figura 5, è possibile osservare un confronto tra la rappresentazione ricostruita e le immagini originali. In particolare, è stato applicato il modello di Encode sul test set e l'output ottenuto è stato dato in pasto al modello di Decode. Il risultato per le prime 12 immagini è mostrato nella seconda riga dell'immagine. La prima riga mostra l'immagine originale presa dal test set. Come è possibile osservare, le ricostruzioni sono abbastanza fedeli. Vi sono però, tra queste, 3 lettere in particolare che ad ogni esecuzione hanno riscontrato problemi: la T in quinta posizione, la P in ottava posizione e la S in nona posizione. Queste sono dovute ad un particolare tratto più marcato o diverso rispetto al normale.

Figura 5: Confronto lettere originali (sopra) con rappresentazione ricostruita dal modello di Autoencoder (sotto).

4 Feed Forward Network

Per svolgere il task di predizione, è stato scelto di utilizzare il modello di Encode come Feature Extraction e quindi costruire una rete sequenziale Feed Forward che prendesse in input il train set codificato a soli 32 attributi, invece degli originali 784. Un'ulteriore opzione presa in considerazione inizialmente consisteva nell'aggiungere dei layer al modello di Encode e settare i pesi di quest'ultimo come non allenabili. Tuttavia, per rendere più snello il codice, è stato scelto di sfruttare la funzione *predict* del modello di Encode, applicata al train set originale (ovvero il train dopo essere stato ri-scalato e portato in valori compresi tra 0 e 1). Un'altra possibile soluzione al problema avrebbe visto la costruzione da zero di una rete sequenziale. Questa strada è stata intrapresa in seguito e discussa nel paragrafo 5.2, per poter confrontare i risultati ottenuti utilizzando un Autoencoder come feature extraction con una rete senza aiuti.

4.1 Preprocessing

Come prima cosa, le etichette sono state processate con la funzione *label_encoder* che, nonostante generalmente venga applicata ad attributi qualitativi, è stata utilizzata per eliminare eventuali ordinamenti tra le classi numeriche. Successivamente, è stata applicata anche la funzione *to_categorical* per binarizzare il target. Questa funzione ha permesso di ottenere 11 nuovi attributi (uno per lettera dalla P alla Z) dei quali solo uno è settato a 1 per ogni record. Infine, è stata applicata una divisione del train set tra train e validation set, utilizzando un rapporto 80-20, ovvero 80% per il train e 20% per il validation. Questa divisione è stata scelta visto il numero relativamente basso di dati, che avrebbe portato ad un train molto piccolo utilizzando percentuali maggiori per il validation. La funzione utilizzata è la *train_test_split* di *sklearn*.

4.2 Scelte in fase di progettazione

Come già detto, l'input del modello sequenziale erano vettori di 32 dimensioni. Per classificare l'input e ottenere quindi una etichetta riguardante la lettera corrispondente, è stato scelto di costruire una rete sequenziale composta da 2 hidden layer + 1 output layer, rispettivamente di 32, 64 e 11 neuroni. In aggiunta, è stato inserito un Dropout del 20% dopo il secondo hidden layer. Il Dropout è una tecnica che serve principalmente per evitare overfitting, spegnendo e quindi settando a zero il (in questo caso) 20% dei neuroni del layer prima di esso. Mantenere attivo l'80% dei neuroni è una misura suggerita in letteratura. Di seguito, è possibile osservare un confronto tra le performance ottenute con un Dropout come descritto e con due Dropout, uno dopo il primo hidden layer e uno dopo il secondo. Il numero di epochs dopo il quale l'apprendimento si arresta varia di esecuzione in esecuzione, anche in base ai pesi casuali che vengono assegnati inizialmente. Osservando i risultati, è possibile concludere che utilizzando una rappresentazione compatta come le 32 feature ottenute dal modello di Encode e un numero di neuroni relativamente basso, probabilmente non sarebbe stato necessario l'uso di nessun Dropout. Si è scelto di utilizzarlo per completezza e per poter sperimentare tecniche di ottimizzazione.

Numero Dropout	Early Stopped Epoch	Val Loss	Val Accuracy
1	42	0.2955	0.9125
2	31	0.3858	0.8825

Tabella 3: Confronto modello senza Dropout e modello con Dropout.

La scelta del numero di neuroni, invece, è dovuta principalmente alle performance registrate.

Come Activation Function sono state utilizzate la ReLU nei due hidden layer, per lo stesso motivo spiegato nel paragrafo 3.1, e la Softmax nel layer di output, ottima per classificazione multi-classe.

La rete è stata allenata per poco meno di 70 epochs, variabile a seconda dell'esecuzione, con un monitor di Early Stopping, settato a interrompere il training dopo 2 epochs nelle quali la

validation loss non migliora. Il monitor è stato impostato inoltre, per salvare i pesi migliori ottenuti nel corso del training.

5 Analisi dei risultati

Le performance del modello di classificazione sono state misurate attraverso metriche come la Loss, l'Accuracy, ma anche con metriche come la Precision, la Recall e la f1-Measure. In Figura 6 e Figura 7 è possibile osservare le curve relative all'andamento della Loss e dell'Accuracy nel corso delle varie epochs. In questo caso, i dati di validation hanno sempre riscontrato risultati migliori che, nel caso dell'utilizzo di due Dropout, vedevano addirittura un margine marcato tra il training e il validation, pur ottenendo risultati leggermente più bassi.

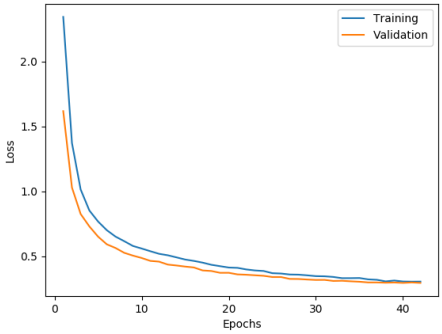

Figura 6: Curva della Loss del train e del validation.

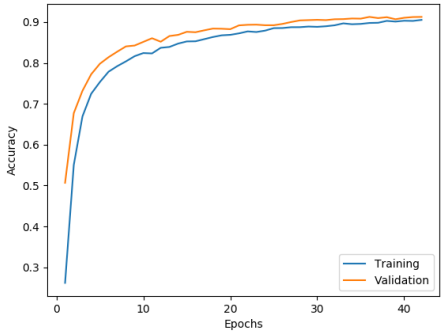

Figura 7: Curva dell'Accuracy del train e del validation.

I valori precisi ottenuti durante l'ultima esecuzione possono essere osservati nella tabella sottostante, dove sono mostrate le performance nel momento in cui si è raggiunto il valore più basso di Val Loss e le performance due epochs dopo, ovvero il momento in cui si è interrotto il training per via del monitor di Early Stop.

Epoch	Loss	Val Loss	Accuracy	Val Accuracy
40	0.3053	0.2955	0.9031	0.9104
42	0.3058	0.2955	0.9052	0.9125

Tabella 4: Performance ottenute in fase di training e validation.

Data la grande quantità di classi, risulta interessante osservare quanto bene vengono classificate singolarmente le classi. Per fare ciò, è stata utilizzata la Precision con un approccio *micro-average*, ovvero nessuna classe è considerata meno importante delle altre e la misura risultate è ottenuta considerando anche il contributo di ciascuna classe. In questo caso, è stata ottenuta una Precision media del 97%, un valore molto buono se considerato da solo, ma che andrebbe considerato insieme alla misura di Recall. In Figura 8 è possibile confrontare le misure di Precision e Recall per ogni classe. Il grafico è abbastanza intuitivo: sull'asse delle ascisse è descritta la Recall mentre sull'asse delle ordinate la Precision, più ci si avvicina a 1 da entrambe le parti e più il modello classifica bene. E' possibile osservare come i risultati siano soddisfacenti anche grazie alla f1 Measure, ovvero la media armonica tra le due metriche.

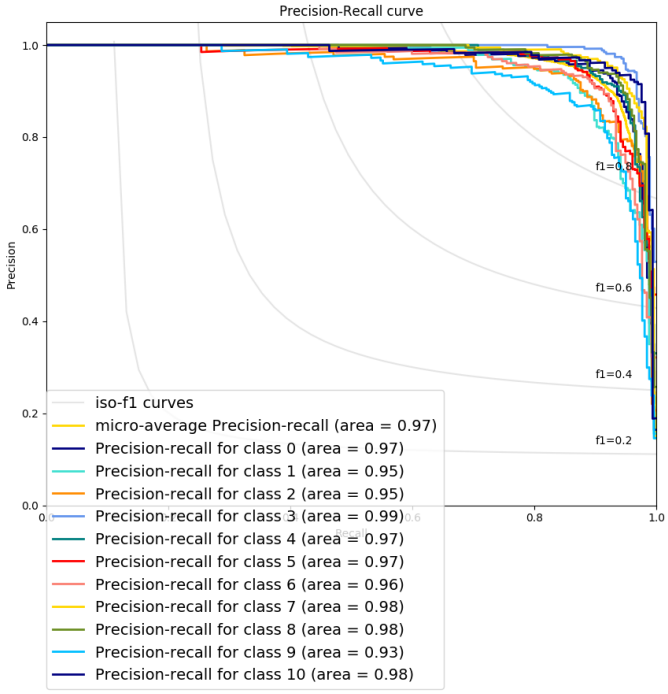

Figura 8: Curva Precision-Recall. Le classi vanno dalla 0 alla 10 e ogni classe indica una lettera: 0 = P e 10 = Z.

5.1 Predict sul test

Per analizzare ulteriormente le capacità del modello, è stata eseguita una predict sul test set. I risultati delle prime 12 lettere sono stati visualizzati insieme all'immagine di esse. E' bene notare che i risultati sono espressi numericamente, indicando la posizione della lettera nell'alfabeto. Per semplificare la lettura viene di seguito inserita una tabella.

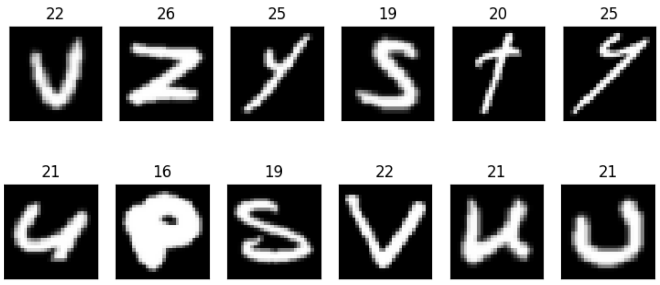

Figura 9: Immagini in input con relative classi predette.

P: 16	Q: 17	R: 18
S: 19	T: 20	U: 21
V: 22	W: 23	X: 24
Y: 25	Z: 26	

Tabella 5: Legenda classi del modello.

5.2 Confronto con una nuova rete

Per rendere più solide le considerazioni sui risultati ottenuti, è stata costruita una seconda rete neurale multi-layer, con lo scopo di provare a classificare l'input originale dopo il pre-processing, ovvero vettori di 784 feature. In questo caso, le scelte in fase di costruzione del modello sono state diverse, poichè una dimensione così ampia dei record portava il modello ad andare in overfit molto facilmente.

La prima soluzione è stata quella di rimpicciolire gradualmente il numero di neuroni tra i vari hidden layer. Sono stati quindi utilizzati 3 hidden layer + 1 output layer, con numero di neuroni rispettivamente di 512, 256, 125 e 11. Non sono invece state toccate le funzioni di attivazione, ovvero è stata mantenuta la ReLU negli hidden layer e la softmax nell'output.

La seconda soluzione adottata è stata quella di inserire dei Dropout tra i vari layer, in particolare sono stati ottenuti i risultati migliori inserendo un Dropout del 40% dopo il primo hidden layer, uno del 30% dopo il secondo e uno del 20% dopo il terzo hidden layer.

Per tamponare l'overfit è stato utilizzato anche un monitor di Early Stop, con attesa di 2 epoch sulla validation loss, come spiegato nel paragrafo 4.2, e la funzione di *restore_best_weights* abilitata.

Infine, il modello costruito è stato allenato prendendo una dimensione di batch pari a 64, dimensione relativamente piccola ma che ha permesso di ottenere i risultati migliori con i suoi frequenti aggiustamenti dei pesi.

I risultati ottenuti si sono rivelati veramente buoni, nonostante il modello tendesse ad andare in overfit dopo poche epochs, soprattutto in termini di loss e accuracy sul validation set, che è stato ottenuto utilizzando la stessa porzione del dataset originale. Nella seguente tabella sono messi a confronto i due modelli, riportando i valori due epochs prima dell'early stop.

Dim Input	Epoch	Val Loss	Val Accuracy
32	40	0.2955	0.9104
784	9	0.2124	0.9425

Tabella 6: Performance a confronto tra i due modelli di classificazione.

Vengono infine mostrati i grafici relativi a questo nuovo modello: come è possibile notare, il modello tende ad overfit anche con le relative precauzioni adottate. Si può osservare come anche la precision sia però migliorata, raggiungendo il 98%.

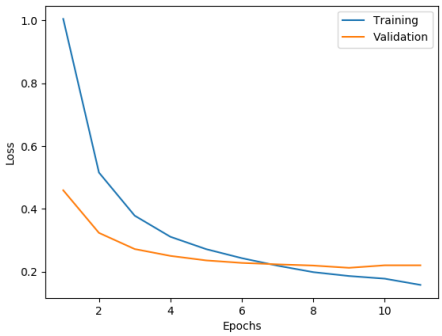

Figura 10: Curva della Loss del train e del validation.

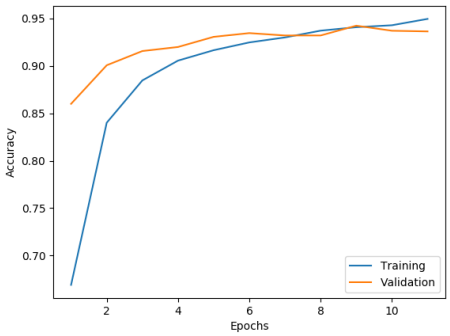

Figura 11: Curva dell'Accuracy del train e del validation.

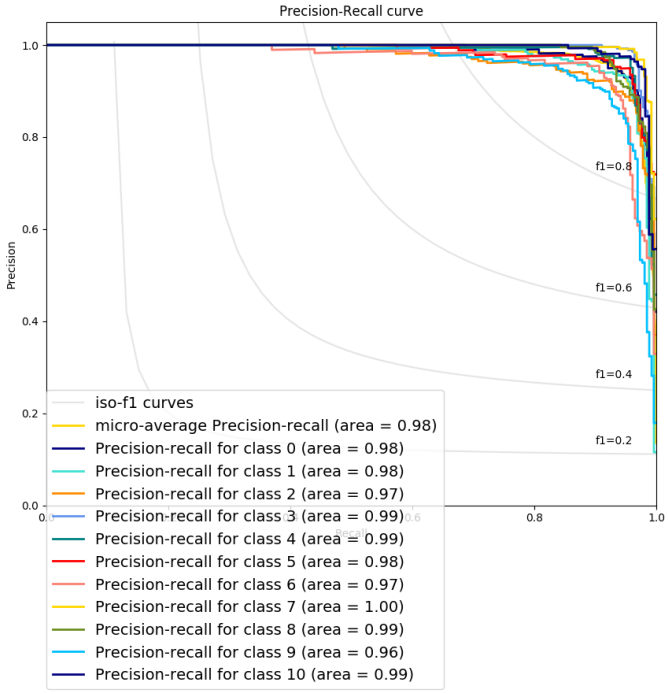

Figura 12: Curva Precision-Recall del nuovo modello.

6 Conclusioni

Come è stato possibile osservare, i risultati ottenuti sono stati molto soddisfacenti. Tuttavia, è bene ricordare che l'utilizzo di una tecnica di feature extraction come un modello di encode, può introdurre un errore, che ovviamente si cerca di minimizzare migliorando le performance dell'autoencoder. I dati classificati e sui quali sono state svolte le misure di performance, non sono quindi i dati originali, bensì sono delle ricostruzioni in dimensione minore di essi. Come già detto, la scelta di utilizzare l'encoder è stata presa per sperimentare e per osservare le capacità di un autoencoder. In una situazione reale, probabilmente la scelta migliore sarebbe quella di utilizzare una rete che prenda in input i vettori originali. La rete in questione, descritta nel paragrafo 5.2, pur presentando problematiche di overfit abbastanza importanti, è riuscita ad ottenere performance migliori rispetto alla prima descritta. Questa differenza potrebbe avere molteplici ragioni, tuttavia un'ipotesi potrebbe essere il fattore di compressione utilizzato nell'autoencoder: per esempio, utilizzare una rappresentazione codificata più grande permetterebbe di mantenere più informazioni necessarie a discriminare le classi.