# Report dell'Assignment 2 di Advanced Machine Learning

Paolo Mariani 800307

Ottobre 2019

## 1 Introduzione

Lo scopo dell'assignment è quello di svolgere tre compiti importanti di classificazione e codifica di immagini a partire dal dataset EMNIST Letters, contenente (nella versione fornita appositamente per l'assignment) immagini di lettere scritte a mano (dalla P alla Z) in scala di grigi.
Gli obiettivi fondamentali che si devono raggiungere sono i seguenti:

- Classificazione delle immagini attraverso una Rete Neurale (non convoluzionale) per riconoscere le lettere scritte a mano contenute in esse.
- Creazione di una struttura Autoencoder in grado di codificare un'immagine e decodificarla con la minore perdita di informazioni possibile.
- Classificazione delle immagini del dataset attraverso la creazione di una struttura "ibrida" che, fornito un dataset, sappia codificare le sue osservazioni (via Encoder, la componente di codifica dell'Autoencoder) e procedere con la classificazione delle immagini a partire dalla codifica ottenuta.

Si procede quindi a introdurre il dataset, le operazioni di preprocessing effettuate su di esso e le risoluzioni dei compiti assegnati nell'ordine in cui sono stati presentati.

## 2 Dati e Preprocessing

Il Dataset "EMNIST Letters" (opportunamente modificato) contenente immagini rappresentanti lettere scritte a mano dalla P alla Z, è stato caricato nell'ambiente Google Colab via Drive, da qui si è proceduto con una sua breve analisi.
Si presenta composto da 14000 immagini in scala di grigi e di dimensione 28x28, cioè in formato matriciale. Ogni singolo pixel dei 784 totali è rappresentato da una "cella" della matrice, ed esprime tramite un valore compreso tra 0 e 255 l'intensità del grigio situato nel pixel corrispondente (0 indica la presenza di nero, 255 la presenza di bianco: i valori intermedi formano la scala di grigi).
Ogni valore che la classe target assume è un intero che corrisponde alla posizione della lettera nell'alfabeto (da 16 a 26).

### 2.1 Overview del Dataset

Si mostrano alcune immagini di esempio per comprendere come sia formato il Dataset:

Figura 1: Esempio di lettere

### 2.2 Split dei dati e Downsampling

Il dataset originale si presenta in due componenti:

- **x_train** e **y_train** che rapppresentano il training set (immagine e label rappresentante la lettera corrispondente)
- **x_test** utilizzato per effettuare su di esso una previsione della label senza avere a disposizione il valore reale da confrontare.

Partendo dalla considerazione che si vuole ottenere un modello validato nei parametri e capace di classificare correttamente immagini mai osservate (ad esclusione della fase di training) è stato deciso di effettuare una separazione ulteriore di x_train e y_train in due porzioni:

- **80%** dei dati per il training del modello, assegnati alle variabili **X_train** e **y_train**. *11200 osservazioni.*
- **20%** dei dati per il test (validazione del modello complessivo) del modello, assegnati alle variabili **X_test** e **y_test**. *2800 osservazioni.*

La porzione **x_test** originale non viene modificata, sarà fornita al modello solo in fase di classificazione finale per poter fornire i risultati nel file *Paolo_Mariani_800307_score2.txt*

### 2.3 Preprocessing

Per garantire al modello un buon apprendimento dei dati è stata effettuata una semplice normalizzazione dei valori contenuti in ogni pixel-cella dell'immagine, riducendo lo spazio in cui prendono i valori tra 0 e 1.
È stato sottratto, ad ogni valore che la colonna target assume, il valore 16 in quanto le prime 16 lettere non sono considerate: ora la lettera P viene riconosciuta con 0 e a seguire le altre fino a 11 per la lettera Z.
Una ulteriore modifica del Dataset è stata applicata a tutti i valori della classe target (lettera corrispondente all'immagine): è stato deciso di implementare una **codifica One-Hot** tramite la funzione di Keras *to_categorical* che ha permesso di rappresentare tramite un vettore binario (composto da zeri e il valore 1 solo nella posizione corrispondente alla lettera rappresentata) per poter effettuare una classificazione non binaria sfruttando la Loss Function *Categorical Crossentropy* che sarà spiegata in seguito.

## 3 Parte Uno: Implementazione Rete Neurale

La rete neurale implementata risulta così composta:

- **Strato Iniziale Dense 1**: 128 neuroni, dimensione dell'input pari al numero di colonne-pixel dell'immagine (784).
- **Strato Dropout 1**: il 40% di unità sono spente casualmente in fase di training.
- **Strato Dense 2**: 64 neuroni, funzione di attivazione "ReLU".
- **Strato Dropout 2**: il 30% di unità sono spente casualmente in fase di training.
- **Strato Dense 3**: 64 neuroni, funzione di attivazione "ReLU".
- **Strato Dropout 3**: il 20% di unità sono spente casualmente in fase di training.
- **Strato Finale Dense 4**: 11 neuroni, funzione di attivazione "softmax".

Si effettua una breve descrizione della rete spiegando i motivi per cui è stata definita questa struttura:

La rete è composta da **4 layer Dense** che si occupano di approssimare la funzione di apprendimento tramite la composizione dei risultati layer dopo layer tramite pesi e funzioni di attivazione che sono associati ad ogni strato, la dimensione iniziale di ognuna di essi era di 64 neuroni, poi convertita solo per il primo layer in 128 in quanto ha permesso di incrementare le performance del modello in fase di test della struttura.
Si utilizza inoltre un **layer di Dropout** appena dopo ogni layer Dense per poter "proteggere" la rete da un eventuale caso di ***overfitting***, in cui il modello comincia a specializzarsi sulle osservazioni del training set e non riesce più a fornire buoni risultati per la classificazione sulla porzione di dati dedicata al test set: implementando questo layer è possibile simulare il training della rete neurale sui dati come se fossero disponibili più versioni differenti della stessa rete. Il layer permette di spegnere in maniera randomica una determinata percentuale di unità (nel modello in analisi il 40, 30 e 20%) che favorisce un apprendimento migliore ed evita di creare una specializzazione della rete.

### 3.1 Funzioni di attivazione

Tutte le funzioni di attivazione che sono state associate a ciascun layer (ad esclusione di quello finale) sono di tipo "***ReLU***", cioè una funzione di attiva-

zione "Rectified Linear Unit" che restituisce, fornito un valore in input, un risultato equivalente se il valore è positivo (cioè identità) o un valore pari a 0 se invece l'input risulta essere negativo; È stata scelta poichè:

- se i valori di input sono negativi la funzione non si attiva, quindi non tutti i neuroni sono attivati e la rete diventa sparsa; ciò garantisce una buona velocità di computazione del modello.
- il gradiente della funzione è zero per i valori negativi, quindi durante il meccanismo di backpropagation non avviene un aggiornamento di quei pesi associati.

La funzione di attivazione dell'ultimo layer è di tipo ***Softmax***, una funzione di trasferimento che permette di rappresentare la distribuzione di probabilità su di una variabile discreta che può assumere più valori, nel caso analizzato 11.
La funzione è continua e differenziabile, rappresenta la funzione esponenziale generalizzata e permette di indicare la probabilità legata ad ogni esito possibile per l'osservazione, la somma di tutte le probabilità associate ad ogni esito dev'essere pari a 1.

### 3.2 Optimizer

È stato scelto ***Adam*** come ottimizzatore, permette di ottenere risultati ottimi (non sempre migliori di SGD) in tempi di calcolo ridotti

### 3.3 Loss Function

La Loss Function scelta è la "***Categorical Crossentropy***", una funzione di perdita consigliata dalla letteratura per i casi di classificazione non binaria. È stata scelta poichè si utilizza nei casi in cui solo un possibile esito-categoria si può associare a ciascuna osservazione, e permette di cooperare con la funzione di attivazione del layer di Output "softmax" in maniera ottimale.
Il comportamento di questa Loss function è tale per cui quando vi si fornisce un vettore predetto in codifica one-hot, essa calcola il valore di quanto differisce dalla codifica one-hot del vettore reale; più sarà diverso il valore della codifica, maggiore sarà la Loss.

### 3.4 Batch Size, Epoche ed Early Stopping

Per la fase di Training si è scelto di utilizzare un **Batch Size** pari a 64, combinato con una durata delle **epoche** pari a 100. In aggiunta è stato implementato il meccanismo di **Early Stopping**, una funzione di *Callback* che rimane in ascolto dei risultati che il modello sta garantendo in fase di Training e Validation (25% dei dati forniti in fase di Training) per decretare uno stop dell'apprendimento dei parametri quando la Validation Loss incrementa per un numero di volte superiore a quelle indicate dalla *Patience*, in questo caso pari a 6: il modello quindi salva gli ultimi parametri che ha appreso e la fase di training si conclude.

### 3.5 Risultati

Utilizzando le metriche di Accuracy, Loss e F-Measure (la cui implementazione può essere osservata nel notebook consegnato) sono stati ottenuti i seguenti risultati in fase di Training e Validation (sul 25% dei dati di Training):

- un valore di **Loss** pari a 0.38 nel training, 0,36 nella fase di validation.
- un valore di **Accuracy** pari a 0.88 nel training, 0,89 nella fase di validation
- un valore di **F-measure** pari a 0.88 nel training, 0,90 nella fase di validation.

I risultati sono da riferirsi ad un completamento della fase di Training limitata a 52 epoche, è avvenuta infatti una interruzione da parte del meccanismo di Early Stopping.
Una volta convalidata la struttura del modello, ha ottenuto in fase di Test sui dati di X_test e y_test (20% dei dati ottenuti a seguito dello split) la seguente performance riassunta dall'immagine:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 16 | 0.92 | 0.93 | 0.93 | 256 |
| 17 | 0.88 | 0.92 | 0.90 | 259 |
| 18 | 0.89 | 0.91 | 0.90 | 279 |
| 19 | 0.95 | 0.98 | 0.96 | 276 |
| 20 | 0.91 | 0.92 | 0.91 | 261 |
| 21 | 0.95 | 0.87 | 0.91 | 280 |
| 22 | 0.86 | 0.87 | 0.86 | 220 |
| 23 | 0.93 | 0.93 | 0.93 | 258 |
| 24 | 0.91 | 0.91 | 0.91 | 252 |
| 25 | 0.91 | 0.84 | 0.87 | 285 |
| 26 | 0.89 | 0.92 | 0.91 | 174 |
| accuracy | | | 0.91 | 2800 |
| macro avg | 0.91 | 0.91 | 0.91 | 2800 |
| weighted avg | 0.91 | 0.91 | 0.91 | 2800 |

Figura 2: Valori delle metriche in fase di Test

L'ultima operazione svolta per questa parte di lavoro consiste nella classificazione delle osservazioni del set x_test e la scrittura dei risultati ottenuti con la funzione *predict* del modello nel file *PaoloMariani800307score2.txt*

# 4 Parte Due: Implementazione dell'Autoencoder

L'Autoencoder che è stato implementato per codificare le immagini del dataset e successivamente decodificarle (in un'unica struttura) possiede questa forma:

- **Strato encoded1**: 392 neuroni, dimensione dell'input pari al numero di colonne-pixel dell'immagine (784). Produce una codifica temporanea di dimensione 392 passata al layer successivo.
- **Strato encoded2**: 196 neuroni, dimensione dell'input pari alla codifica dello strato precedente di 392 elementi. Produce una codifica temporanea di dimensione 196 passata al layer successivo.
- **Strato encoded3**: 98 neuroni, dimensione dell'input pari alla codifica dello strato precedente di 196 elementi. Produce una codifica temporanea di dimensione 98 passata al layer successivo.
- **Strato encoded4**: 49 neuroni, dimensione dell'input pari alla codifica dello strato precedente di 98 elementi. Produce una codifica definitiva di dimensione 49.

Questa prima porzione di rete prende il nome di **Encoder**, il risultato della sua applicazione ai dati è il seguente:

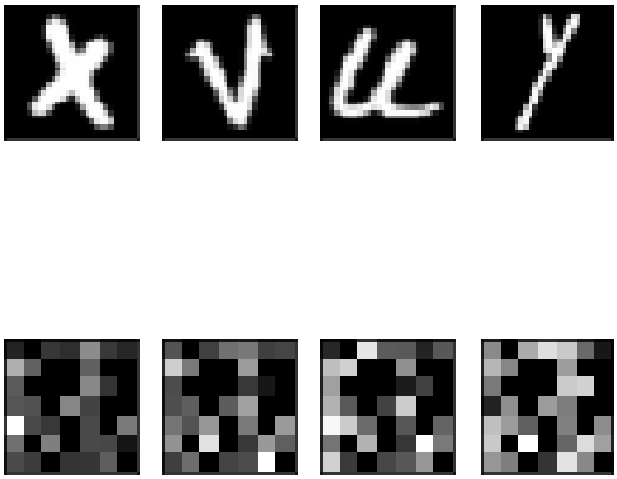

Figura 3: Esempio di codifica delle immagini

La seconda porzione dell'Autoencoder si chiama **Decoder**:

- **Strato decoded1**: 98 neuroni, dimensione dell'input pari alla codifica a 49 elementi dell'immagine ottenuta dall'Encoder. Produce una decodifica temporanea di dimensione 98 passata al layer successivo.
- **Strato decoded2**: 196 neuroni, dimensione dell'input pari alla codifica a 98. Produce una decodifica temporanea di dimensione 196 passata al layer successivo.
- **Strato decoded2**: 392 neuroni, dimensione dell'input pari alla codifica a 196. Produce una decodifica temporanea di dimensione 392

passata al layer successivo.

- **Strato encoded4**: 784 neuroni, dimensione dell'input pari alla codifica dello strato precedente di 392 elementi. Produce una decodifica finale che consiste nella rappresentazione dell'immagine originale.

L'Encoder, come si può osservare dalle dimensioni dei dati in ingresso per ogni layer, effettua un dimezzamento della codifica dell'immagine fino a raggiungere la dimensione minima di 49 elementi; nella successiva fase di decodifica il Decoder riproduce l'immagine, a partire dalla codifica, espandendola del doppio ad ogni strato fino ad arrivare alla dimensione originale di 784 elementi.

### 4.1 Risultati

Come si può osservare avviene una perdita di informazioni nella fase di codifica e decodifica, ma è accettabile e non compromette la comprensione del suo contenuto;
Si osserva il risultato nella seguente immagine:

Figura 4: Risultato della codifica e decodifica dell'immagine

## 5 Parte Tre: Implementazione del modello ibrido

L'ultimo compito da svolgere riguarda l'implementazione di un modello ibrido che permetta di classificare le immagini a partire da una loro codifica ridotta (non dalla loro forma originale); il procedimento è utilizzato per poter ridurre il numero di features in ingresso del modello, riducendo quindi la complessità computazionale associata al compito ma anche il tempo necessario per svolgere il task.
Il modello è composto quindi da due parti fondamentali:

- L'**Encoder**, ottenuto coi passaggi descritti nella sezione precedente, che si occupa di codificare le immagini del Training e Test set.
- Una **Rete Neurale** che si occupa di effettuare classificazione su una rappresentazione codificata delle immagini; si utilizza la medesima illustrata nella prima sezione modificata opportunamente per garantire compatibilità con le codifiche.

Per primo l'**Encoder** ha codificato tutte le istanze di **X_train**, permettendo di ottenere per ogni immagine la sua versione di dimensione ridotta a 49 elementi;
Successivamente la **Rete Neurale** è stata modificata nel primo layer Dense per concedere all'input di 49 elementi di essere appreso correttamente: la dimensione accettata dell'input diventa 49 anzichè 784. Gli altri parametri della rete rimangono invariati.

### 5.1 Risultati

Si procede ad osservare i risultati ottenuti dalla rete in fase di Training, Validation e Test:

- un valore di **Loss** pari a 0.48 nel training, 0,33 nella fase di validation.
- un valore di **Accuracy** pari a 0.85 nel training, 0,90 nella fase di validation
- un valore di **F-measure** pari a 0.84 nel training, 0,90 nella fase di validation.

Il risultato è stato ottenuto attraverso 53 epoche, poi il meccanismo di Early Stopping ha interrotto la fase di apprendimento. I risultati in fase di Training e Validation sono molto positivi considerando che si tratta di una classificazione di immagini codificate; Si osserva che i valori delle metriche sono migliori nella fase di Validation, questo avviene in quanto il meccanismo di Dropout agisce modificando la struttura della rete solo in fase di Training e non di Validation, peggiorandone i risultati.
Dopo aver convalidato i parametri del modello e la struttura, è stata effettuata la validazione complessiva sui dati di **X_test** e **y_test**, ottenendo i seguenti valori delle metriche:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 16 | 0.94 | 0.94 | 0.94 | 259 |
| 17 | 0.86 | 0.94 | 0.90 | 246 |
| 18 | 0.91 | 0.88 | 0.89 | 295 |
| 19 | 0.96 | 0.95 | 0.95 | 290 |
| 20 | 0.91 | 0.91 | 0.91 | 263 |
| 21 | 0.91 | 0.90 | 0.91 | 262 |
| 22 | 0.85 | 0.88 | 0.86 | 216 |
| 23 | 0.94 | 0.85 | 0.89 | 285 |
| 24 | 0.91 | 0.92 | 0.92 | 249 |
| 25 | 0.86 | 0.90 | 0.88 | 250 |
| 26 | 0.93 | 0.90 | 0.92 | 185 |
| accuracy | | | 0.91 | 2800 |
| macro avg | 0.91 | 0.91 | 0.91 | 2800 |
| weighted avg | 0.91 | 0.91 | 0.91 | 2800 |

Figura 5: Valori delle metriche in fase di Test