Compressione di immagini con gli AutoEncoders

***Abstract*—La compressione delle immagini è un'importante area di ricerca in quanto può ridurre significativamente il costo di archiviazione e di trasmissione delle immagini. In questo lavoro, viene presentato l'utilizzo degli *autoencoder* per la compressione delle immagini. Gli autoencoder sono una classe di reti neurali che possono apprendere una rappresentazione compressa delle immagini. In particolare, la rete viene addestrata a codificare l'immagine di input in una rappresentazione compressa a bassa dimensionalità, che può essere quindi decodificata per ottenere l'immagine originale. In questo modo, è possibile ridurre la dimensione dell'immagine senza perdere importanti dettagli e caratteristiche. In questo studio, abbiamo eseguito un'analisi approfondita dei risultati ottenuti dalla compressione delle immagini utilizzando autoencoder su tre differenti dataset di immagini di test.**

I. Introduzione

A. Autoencoders

Gli **autoencoder** [1] sono una classe di reti neurali artificiali utilizzate principalmente per l'apprendimento *non supervisionato* di rappresentazioni di dati ad alta dimensionalità. L'obiettivo principale degli autoencoder è quello di apprendere una rappresentazione compressa, o codifica, di un insieme di dati di input, in modo tale da ricostruire fedelmente i dati originali attraverso una decodifica inversa. In altre parole, gli autoencoder cercano di comprimere le informazioni contenute nei dati di input in una rappresentazione a bassa dimensionalità e poi di ricostruire i dati di input originali a partire dalla rappresentazione compressa.

Gli autoencoder sono composti da due parti principali:

- **Encoder** che prende in input i dati ad alta dimensionalità e li trasforma in una rappresentazione compressa.
- **Decoder** che rende la rappresentazione compressa e la trasforma nuovamente nei dati di input originali.

La struttura dell'autoencoder è dunque simile ad una rete neurale *feedforward*, ma con una struttura "a specchio" (o "speculare"), in cui le prime fasi dell'*encoder* sono simmetriche alle fasi del *decoder*.

Gli autoencoder sono utilizzati per una vasta gamma di applicazioni, tra cui la compressione di dati, il filtraggio dei dati, la riduzione della dimensionalità, la generazione di dati e il miglioramento delle prestazioni di classificazione. Una delle principali applicazioni degli autoencoder è la compressione delle immagini, dove gli autoencoder sono utilizzati per apprendere una rappresentazione compressa dell'immagine originale, riducendo così la quantità di dati necessari per rappresentare l'immagine. In generale, gli autoencoder sono stati ampiamente utilizzati per l'apprendimento non supervisionato, il pretraining delle reti neurali e la generazione di dati sintetici. Nella Fig. 1 viene riportato un esempio dell'architettura.

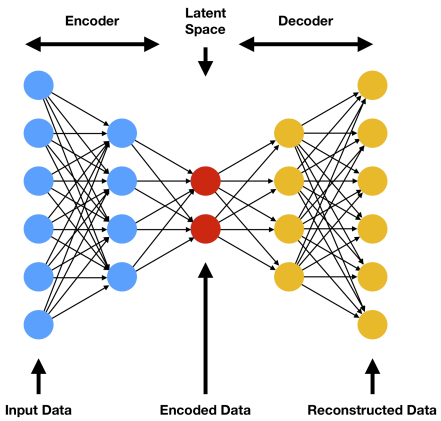


Fig. 1: Architecture of an autoencoder.

II. Implementazione

A. Librerie Usate

Per l'implementazione di questo progetto sono state utilizzate le seguenti librerie:

- `pytorch` libreria per il deep learning.
- Le librerie `transforms` e `datasets` di **torchvision** che vengono utilizzate per effettuare alcune trasformazioni sulle immagini e per caricare il dataset MNIST.
- `hydra` che viene utilizzata per gestire la configurazione del modello e dei parametri di addestramento in un file di configurazione esterno.

B. Dataset

I dataset utilizzati durante lo sviluppo del progetto sono:

- **MNIST**: è uno dei dataset più famosi e utilizzati nel campo dell'apprendimento automatico. Contiene immagini di cifre scritte a mano (da 0 a 9) in scala di grigi di dimensioni 28x28 pixel [2].
- **FashionMNIST**: è stato introdotto come alternativa al dataset MNIST, e contiene immagini di abiti di varie categorie, come camicie, pantaloni, vestiti, scarpe e borse. Anche questo dataset contiene immagini in scala di grigi di dimensioni 28x28 pixel [3].
- **EMNIST**: il dataset EMNIST (Extended MNIST) è una versione estesa del dataset MNIST che contiene sia cifre che caratteri alfabetici in maiuscolo e minuscolo [4].

C. Codice

Il codice è divisio in script:

- `autoencoder.py`: questo codice definisce la struttura di un **autoencoder**.

L'*encoder* viene definito nella classe `Encoder`. L'encoder è composto da una sequenza di quattro layer lineari con funzione di attivazione ReLU, che riducono gradualmente la dimensione dell'immagine fino a ottenere un vettore di 12 elementi.

Il *decoder* viene definito nella classe `Decoder`. Il decoder è composto da una sequenza di quattro layer lineari con funzione di attivazione ReLU, che ricostruiscono gradualmente l'immagine a partire dal vettore di 12 elementi prodotto dall'encoder.

L'*autoencoder* completo viene definito nella classe `Autoencoder`. L'autoencoder è costituito da un'istanza dell'encoder e del decoder, che sono inizializzati nel metodo `init`. Il metodo `forward` implementa il passaggio in avanti attraverso l'encoder e il decoder, ovvero l'elaborazione dell'immagine in ingresso fino alla sua ricostruzione.

- `main.py`: questo codice definisce una funzione di train per un autoencoder che utilizza i dataset MNIST per ricostruire immagini di numeri scritti a mano.

  Durante il training, il modello viene addestrato per ricostruire l'immagine originale data in input all'autoencoder, utilizzando la Mean Squared Error (MSE) come funzione di loss e l'ottimizzatore Adam per la backpropagation.

  La funzione di train effettua inoltre la normalizzazione dei pixel delle immagini e la ridimensionamento delle immagini stesse, e salva le immagini ricostruite in una specifica cartella del file system.

  Infine, il codice è scritto per essere configurato tramite un file di configurazione in formato YAML, che definisce i parametri di training come il percorso delle immagini, la dimensione del batch, il learning rate, il numero di epoche e la cartella di output per le immagini ricostruite.

III. Risultati

A. Indice di Compressione

L'**indice di compressione** di un autoencoder è definito come il rapporto tra la dimensione dell'input originale e la dimensione della rappresentazione compressa. I tre dataset utilizzati hanno tutti e tre immagini di 28x28 = 784 pixel e la dimensione della rappresentazione compressa è 12. Il calcolo dell'indice di compressione viene effettuato dividendo la dimensione dell'input originale per la dimensione della rappresentazione compressa, vedi Eq. 1. Il risultato è l'indice di compressione, che indica quante volte l'input originale è stato compresso rispetto alla sua dimensione originale.

$$Indice\ di\ compressione = \frac{784}{12} = 65.3 \tag{1}$$

B. Metriche

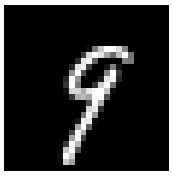
I risultati sono riportati nella Tabella I dopo 5 epoche di training per ogni dataset. Oltre al SSIM è stata utilizzata un ulteriore metrica di compressione: MSE. Anche chiamato **Mean Square Error**:

TABLE I: Result for each Dataset.

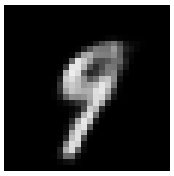
MNIST	FashionMNST	EMNIST
0.0195	0.0168	0.0240

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{2}$$

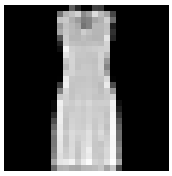
Dove y_i è il pixel dell'imagine originale e $\hat{y_i}$ rappresenta un pixel dell'immagine ricostruita. Nelle figure sottostanti vi sono gli esempi, a sinistra l'immagine originale e adestra l'immagine ricostruita.



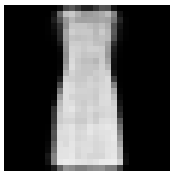
(a) Immagine originale del MNIST.



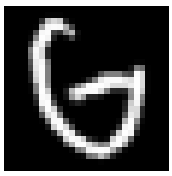
(b) FashionMNIST.



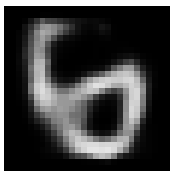
(a) Originale del FashionMNIST.



(b) Ricostruita del FashionMNIST.



(a) Originale del EMNIST.



(b) Ricostruita del EMNIST.

References

[1] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507.

[2] Yann LeCun, Corinna Cortes, and Christopher JC Burges. "The MNIST database of handwritten digits". In: *www.cs.nyu.edu/˜ylclab/data/nips2001.ps.gz* 2 (1998).

[3] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[4] Gregory Cohen et al. "EMNIST: an extension of MNIST to handwritten letters". In: *arXiv preprint arXiv:1702.05373* (2017).