Metodi di Ottimizzazione Per Big Data

Implementazione di una Rete Neurale Feed Forward

Marco Lorenzini

Università di Tor Vergata

1 Obiettivo

Il progetto è stato realizzato con l'obiettivo di implementare una Rete Neurale Feed Forward per affrontare problemi di Regressione e Classificazione. Il linguaggio scelto per la realizzazione è Python.

2 Requisiti

I pacchetti necessari per l'esecuzione del progetto sono:

- **Numpy**: libreria utilizzata per il calcolo scientifico in Python
- **Pandas**: libreria per la manipolazione e l'analisi dei dati.
- **Matplotlib**: creazione e visualizzazione di grafici
- **Sklearn**: contiene funzioni per il preprocessing di dati
- **Seaborn**: creazione e visualizzazione di grafici

E' possibile installare tutte le librerie necessarie tramite il file *requirements.txt*

3 Architettura

Andiamo ad analizzare l'architettura della rete, soffermandoci su tutte le tecniche implementate.

3.1 Rete Neurale di base

L'architettura di base che utilizzeremo nei test sarà così composta:

- **Input Layer**: riceve i dati in ingresso
- **2 Hidden Layer**: elaborano i dati
- **Output Layer**: strato finale, genera l'output

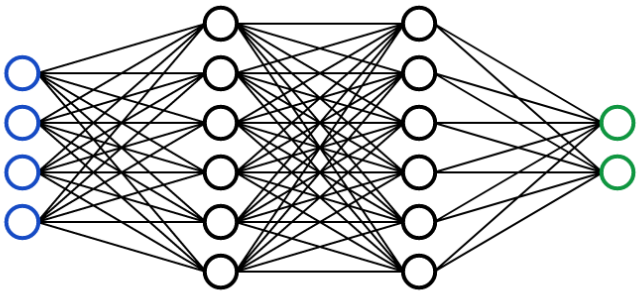


Figure 1: Esempio di Rete Neurale formata da due hidden layer

3.2 Funzioni di attivazione

Le funzioni di attivazione implementate nei modelli sono **ReLU** (Rectified Linear Unit) e **Tanh** (Tangente iperbolica), comunemente utilizzate nei livelli nascosti delle reti neurali per introdurre non linearità. Per lo strato di output vengono invece utilizzate: **Sigmoid** per la classificazione binaria, **Softmax** per la classificazione multiclasse e un'attivazione lineare (**Linear Activation**) per i problemi di regressione.

3.3 Inizializzazione dei pesi

All'interno della rete neurale, i pesi possono essere inizializzati tramite **He Initialization**, comunemente utilizzata con funzioni di attivazione non saturanti come **ReLU**, e **Glorot Initialization** che viene invece utilizzato con **Tanh**.

3.4 Algoritmi di ottimizzazione

Per l'addestramento è possibile scegliere tra i seguenti algoritmi di ottimizzazione:

- **Sgd**
- **Sgd con Momentum**
- **RmsProp**
- **Adagrad**
- **Adam**

Per tutti gli algoritmi è possibile selezionare un learning rate fisso o decrescente; nei nostri test, abbiamo optato per un learning rate decrescente. In questo approccio, il learning rate viene aggiornato ad ogni iterazione, riducendosi proporzionalmente al valore di **decay** e al numero di iterazioni **iteration**. Questo permette di controllare la velocità di apprendimento durante l'addestramento, adattandola dinamicamente nel corso del processo.

3.5 Funzioni di Loss

Le funzioni di loss implementate sono la **BinaryCrossEntropy** per la classificazione binaria, **CategoricalCrossEntropy** per la classificazione multiclasse e infine **MSE**, **MAE** e **RMSE** per la regressione.

3.6 Regolarizzatori

Come regolarizzatori sono stati implementati **Dropout**, **Early Stopping**, **L1** e **L2**. Il **Dropout** è una tecnica che consiste nell'escludere casualmente una percentuale di neuroni durante la fase di allenamento, riducendo la probabilità di overfitting e migliorando la capacità di generalizzazione del modello. L'**Early Stopping** invece arresta l'allenamento del modello quando le prestazioni, su un insieme di validazione, smettono di migliorare per evitare, come per il Dropout, l'overfitting. Le tecniche **L1** ed **L2** vanno invece a limitare i pesi tramite il parametro λ.

4 Cross-Validation

Per ricavare la miglior NNFF per il nostro dataset è stata implementata una Cross-Validation. La Cross-Validation è un metodo di valutazione del modello che divide il dataset in più gruppi, chiamati **fold**. Per ogni iterazione, un fold specifico viene utilizzato come set di test, mentre gli altri $k-1$ fold vengono utilizzati per addestrare il modello. Questa tecnica ci permette di ottenere una valutazione più robusta del modello rispetto alla semplice suddivisione dei dati in un set di addestramento e uno di test. In particolare, evita il rischio di sovra-adattamento fornendo una stima più affidabile delle prestazioni del modello su dati non visti. Per limitazioni di tempo e di hardware, non è stato possibile esplorare tutte le possibili combinazioni di parametri; pertanto, l'analisi si è concentrata sulle combinazioni delle seguenti configurazioni:

- **Ottimizzatori**: Adam, RmsProp, Sgd con Momentum
- **Regolarizzatori sui pesi**: l1, l2, None

- **Dropout**: True, False
- **Combinazioni dei layer**: Variano in base al dataset
- **Funzioni di Attivazione**: Relu, Tanh

Si osserva che, impostando il parametro booleano **multithread**, è possibile eseguire in parallelo le diverse combinazioni dei fold sfruttando il multithreading.
Le metriche prese in considerazione per la classificazione sono:

- **Accuracy**: rapporto tra previsioni corrette e il totale delle previsioni effettuate.
- **Recall**: misura la capacità del modello di identificare correttamente tutte le istanze positive. Si calcola come il rapporto tra i veri positivi e la somma dei veri positivi e falsi negativi.
- **Precision**: misura l'accuratezza delle previsioni positive del modello. Si calcola come il rapporto tra i veri positivi e la somma di veri positivi e falsi positivi.
- **F1 Score**: media armonica tra precision e recall. F1 Score è utile quando si desidera bilanciare sia il numero di falsi positivi che quelli negativi.

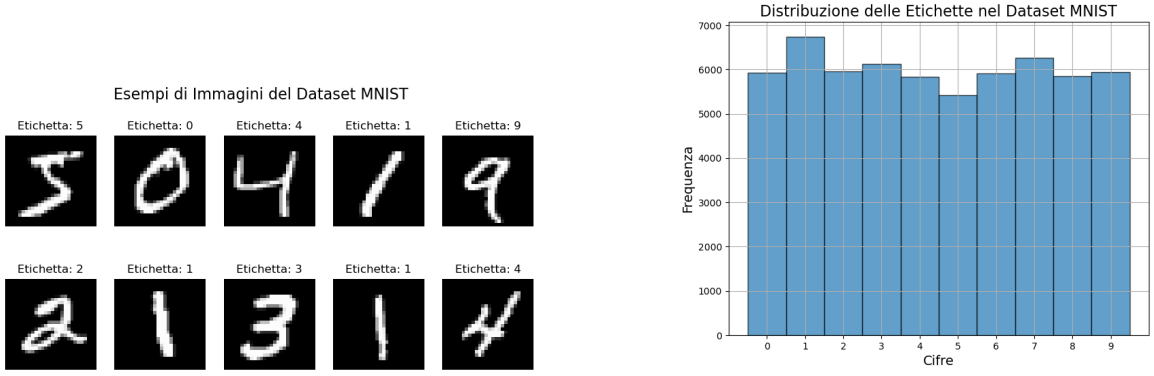
Mentre per la regressione ci baseremo sul valore della **loss**.

5 Analisi dei Dataset

In questa sezione verranno presentati i dataset che utilizzeremo per testare la Rete Neurale.

5.1 Classificazione

MNIST: Il dataset **MNIST** è composto da immagini di cifre che vanno da 0 a 9. L'obiettivo è classificare correttamente le immagini, ossia riconoscere quale cifra numerica rappresenta ogni immagine. Vediamo qualche esempio e la frequenza delle istanze del dataset rispetto alle labels:



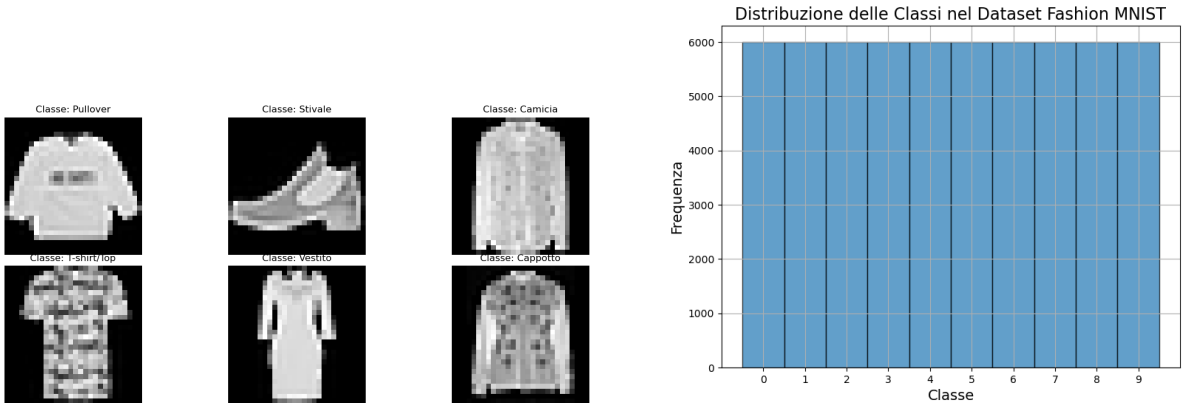
(a) Esempio di istanze del Dataset

(b) Distribuzione delle istanze

Figure 2: Esempi e distribuzione delle istanze del dataset

Inizialmente, il training set è stato ridotto da 60.000 a 40.000 campioni, mantenendo invariato il numero di campioni per ciascuna classe. Successivamente, i dati sono stati normalizzati.

Fashion MNIST: il dataset **Fashion MNIST** è composto da immagini di articoli di abbigliamento e accessori, organizzato in 10 categorie diverse. Come prima l'obiettivo è quindi una classificazione multiclasse. Vediamo anche qui qualche esempio e la frequenza delle istanze del dataset rispetto alle labels:



(a) Esempi del Fashion MNIST

(b) Frequenza dei dati nel Fashion MNIST

Figure 3: Esempi e distribuzione delle classi nel dataset Fashion MNIST

Anche in questo caso, diminuiamo i campioni del training set a 40.000 e effettuiamo normalizzazione dei dati.

Breast Cancer: Il dataset **Breast Cancer** contiene informazioni diagnostiche basate su una serie di caratteristiche misurate su immagini digitalizzate di biopsie. Le caratteristiche includono dimensioni, forma, texture e altre proprietà dei nuclei cellulari. L'obiettivo è quindi quello di costruire un modello di classificazione binaria per distinguere tumori maligni e benigni. Possiamo inanzitutto andare ad osservare la distribuzione dei dati:

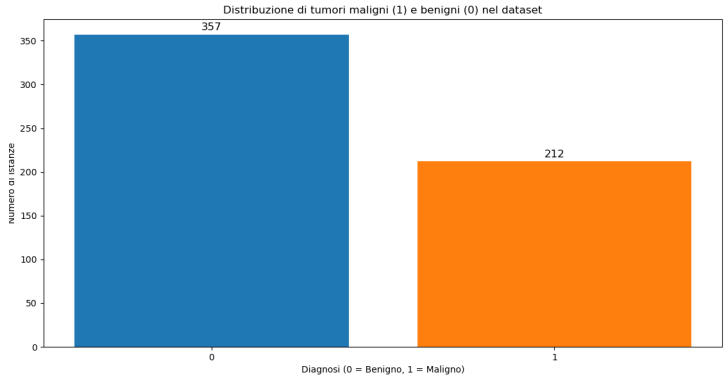


Figure 4: Distribuzione dei dati in Breath Cancer

Effettuiamo sul dataset l'operazione di rimozione della colonna **id**, poichè non è rilevante per il compito di classificazione e trasformiamo poi i valori della colonna **diagnosis** in valori numerici. Successivamente vengono rimosse le colonne che presentano una **bassa correlazione** con il target.

5.2 Regressione

California Housing Prices Dataset: questo dataset contiene informazioni socioeconomiche e demografiche raccolte in diverse aree della California, il nostro obiettivo è quello di prevedere il valore mediano delle case in ciascun distretto basandosi sulle caratteristiche disponibili. Prima di testare i modelli sono state utilizzate tecniche di **Scaling** e **Normalizzazione** delle feature. Vediamo la distribuzione della variabile target prezzi:

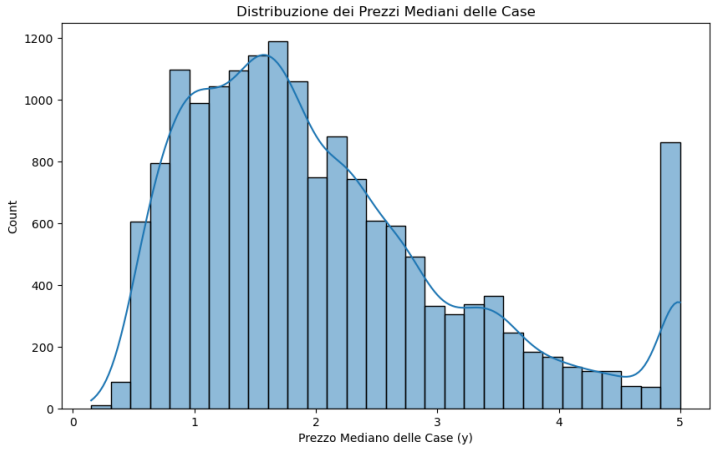


Figure 5: Distribuzione dei prezzi

6 Risultati

I risultati seguenti, sono ottenuti andando ad eseguire la Cross-Validation multithread con 5 fold su ogni dataset e successivamente riaddestrando il modello sull'intero training set, prima di testarlo sul test set.

6.1 Fashion MNIST

Dato il numero di feature, proviamo due combinazioni di neuroni:

- (256,128)
- (128,64)

Data la grande quantità di tempo necessaria, riduciamo gli ottimizzatori a **Adam** e **RmsProp**. La Cross-Validation ha impiegato 2:51:14 ore. Il miglior modello è composto da un numero di neuroni pari a: **(128,64)**, con attivazione **Tanh** e ottimizzatore **RmsProp** con un learning rate pari a 0.01 e un tasso di decadimento pari a 0.0001. Risultati ottenuti dalla Cross-Validation:

Metrica	**Valore**
Accuracy	0.8610
Precision	0.8623
Recall	0.8612
F1 Score	0.8617

Table 1: Prestazioni del miglior modello

A questo punto, dopo aver addestrato il modello sull'intero training set, abbiamo ottenuto i seguenti risultati sul test set:

Metrica	Valore
Accuracy	0.8685
Precision	0.8697
Recall	0.8684
F1 Score	0.8690

Figure 6: Prestazioni sul test set

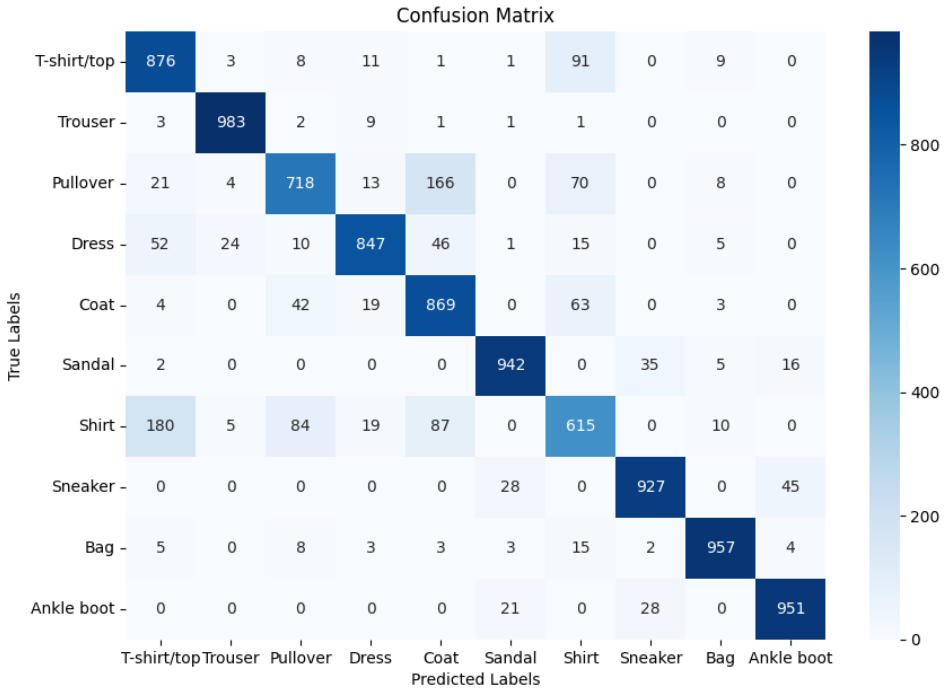


Figure 7: Matrice di Confusione relativa al Test Set

6.2 MNIST

Dato il grande numero di features, proviamo solo due combinazioni di neuroni:

- (256,128)
- (128,64)

Anche in questo caso, vista la quantità di tempo necessaria, riduciamo gli ottimizzatori a **Adam** e **RmsProp**. La Cross-Validation ha impiegato 2:50:14 ore. Il miglior modello è composto da un numero di neuroni pari a: **(256,128)**, con attivazione **Tanh** e ottimizzatore **RmsProp** con un learning rate pari a 0.01 e un tasso di decadimento pari a 0.0001. Risultati ottenuti dalla Cross-Validation:

Metrica	Valore
Accuracy	0.9694
Precision	0.9684
Recall	0.9690
F1 Score	0.9687

Table 2: Prestazioni del miglior modello

A questo punto, dopo aver addestrato il modello sull'intero training set, abbiamo ottenuto i seguenti risultati sul test set:

Metrica	Valore
Accuracy	0.963
Precision	0.9628
Recall	0.9627
F1 Score	0.9627

Figure 8: Prestazioni sul test set

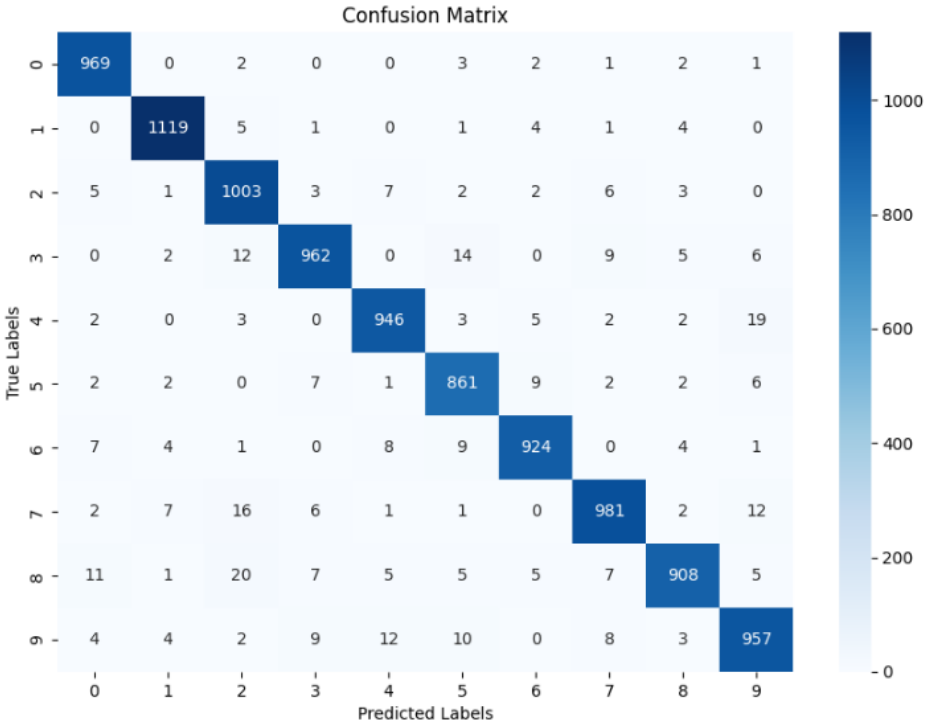


Figure 9: Matrice di Confusione relativa al Test Set

6.3 Breast Cancer

Dato il ridotto numero di features, il numero di neuroni degli hidden layer scelti per la Cross-Validation sono:

- (32,16)
- (64,32)
- (32,64)
- (128,64)

La Cross-Validation ha impiegato 19:40 minuti. Il miglior modello ottenuto è composto da un numero di neuroni pari a: **(32, 16)** sfruttando la regolarizzazione **l1** (valore 0.001 sia per i pesi sia per i bias), con attivazione **Tanh**, ottimizzatore **Sgd con Momentum** con un learning rate pari a 0.01, un tasso di decadimento pari a 0.0001 e momentum 0.9, infine **Dropout** con un rate pari a 0.2 Risultati ottenuti dalla Cross-Validation:

Metrica	Valore
Accuracy	0.9780
Precision	0.9777
Recall	0.9756
F1 Score	0.9765

Table 3: Prestazioni del miglior modello

A questo punto, dopo aver addestrato il modello sull'intero training set, abbiamo ottenuto i seguenti risultati sul test set:

Metrica	Valore
Accuracy	0.9824
Precision	1
Recall	0.9523
F1 Score	0.9756

Figure 10: Prestazioni sul test set

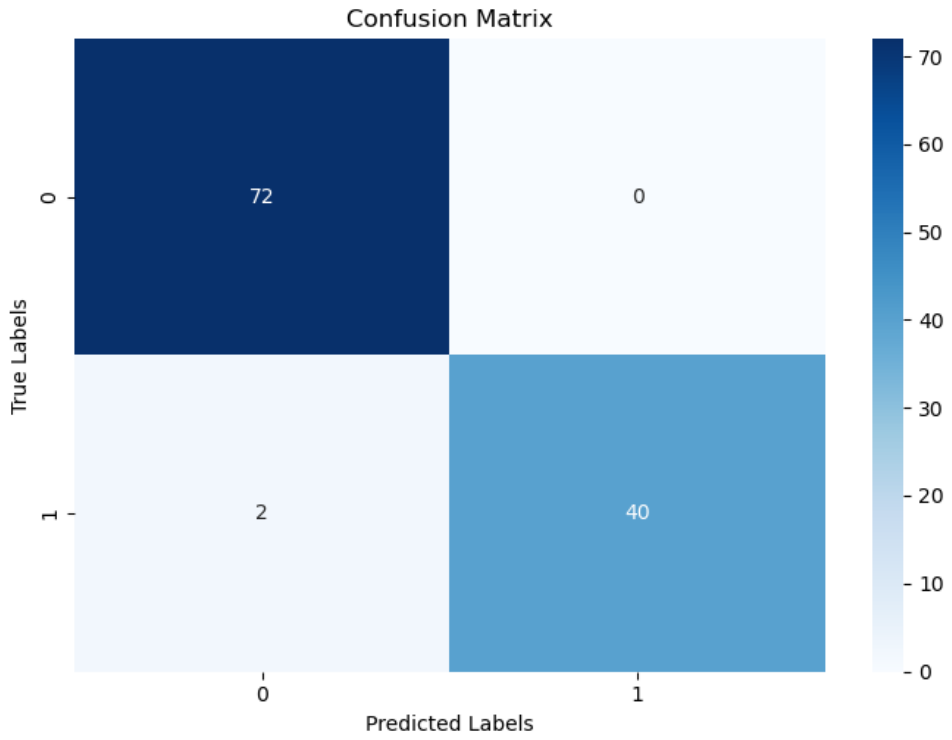


Figure 11: Matrice di Confusione relativa al Test Set

6.4 California Housing Prices Dataset

Il numero di neuroni scelti per gli hidden layer scelti per la Cross-Validation sono:

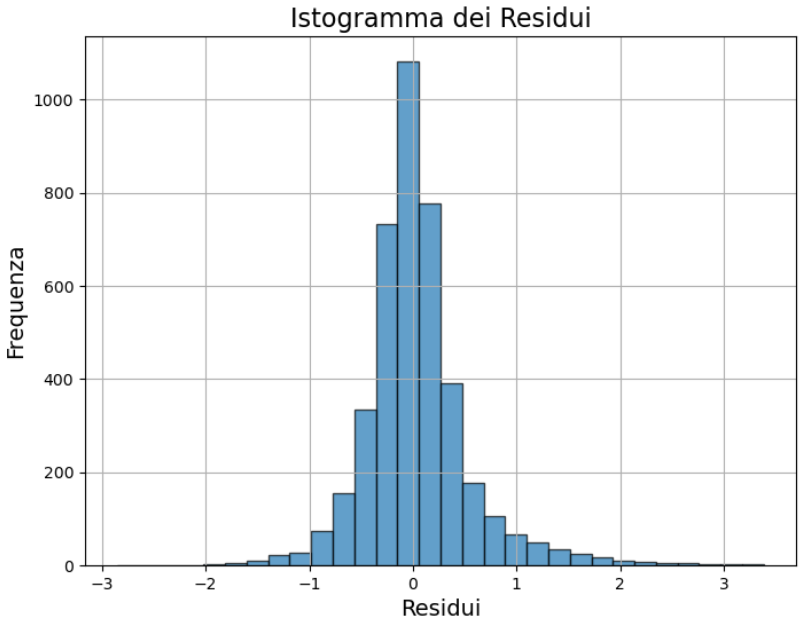
- (16,8)
- (32,16)
- (64,32)

La Cross-Validation ha impiegato 15:25 minuti. Il miglior modello ottenuto è composto da un numero di neuroni pari a: **(64,32)**, con attivazione **Tanh** e ottimizzatore **Adam** con un learning rate pari a 0.01 e un tasso di decadimento pari a 0.001. Risultati ottenuti dalla Cross-Validation:

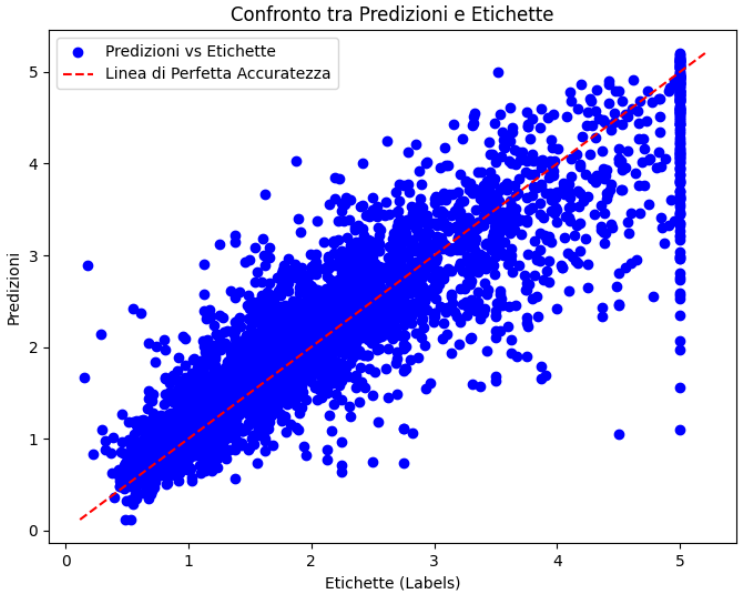
- **Mean Loss**: 0.3559

A questo punto, dopo aver addestrato il modello sull'intero training set, abbiamo ottenuto i seguenti risultati sul test set:

- **Mean Loss**: 0.344



(a) Istogramma dei Residui



(b) Grafico della dispersione

Figure 12: Confronto tra l'istogramma dei residui e il grafico di dispersione

6.5 Osservazioni finali

Possiamo fare alcune osservazioni sui dati appena analizzati. Innanzitutto, i risultati relativi a **Fashion MNIST** mostrano che la rete neurale ha maggiori difficoltà a fare previsioni su questo dataset rispetto ad altri. Questa differenza è principalmente attribuibile alla complessità delle immagini contenute nel dataset, che risultano significativamente più elaborate rispetto a quelle del classico **MNIST**. Infatti, caratteristiche come bordi e forme possono risultare più difficili da riconoscere per un modello di machine learning. Inoltre, è importante notare la similarità visiva tra alcune classi, come maglioni e t-shirt, che può portare a previsioni imprecise. In secondo luogo, è fondamentale sottolineare che, in diverse esecuzioni del programma, sia il modello ottimale che le sue performance possono variare, questo comportamento è dato dalla variabile suddivisione dei vari set (train, validation, test) e dall'inizializzazione dei pesi.