Anomaly Detection by Autoencoders

Ferdinando Cozzolino

1 Introduzione

Un autoencoder è un particolare tipo di rete neurale che ha come scopo quello di replicare, nel modo più fedele possibile, l'input fornito. Ogni autoencoder è costituito da due parti:

- Encoder: Rete neurale che prende l'input originale e lo codifica.
- Decoder: Rete neurale che prende in input l'output dell'encoder e prova a ricostruire l'input originale.

. In termini matematici, denotando con:

- $D = \{x_1, x_2, ..., x_n\}$ con $x_i \in R^m$ l'insieme dei dati.
- g_ϕ la funzione di encoding (Dove con ϕ denotiamo il vettore dei pesi della rete neurale di encoding).
- f_α la funzione di decoding (Dove con α denotiamo il vettore dei pesi della rete neurale di decoding).

Possiamo pensare all'autoencoder come la funzione composta $f_\alpha(g_\phi(x))$. Stabilendo quindi una loss function L, l'obiettivo dell'addestramento di un autoencoder è quello di trovare i pesi ϕ e α per cui sia minima la seguente quantità:

$$\sum_{x \in D} L(f_\alpha(g_\phi(x)), x)$$

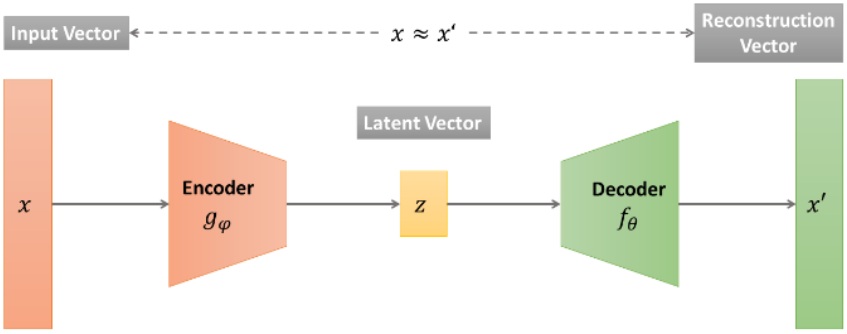


Figura 1: Immagine riassuntiva dell'architettura di un autoencoder.
L'immagine è reperibile qui [3]

Se un autoencoder riesce semplicemente ad imparare la funzione identica (cioè pone $f(g(x)) = x \; \forall \; x \in D$), allora non è particolarmente utile. Di conseguenza, durante la costruzione di questo tipo di reti, vengono implementate delle limitazioni che consentono di copiare solo approssimativamente l'ipnput o di copiare solo input che "assomigliano" ai dati di addestramento. Poiché il modello è costretto a dare la priorità a quali aspetti dell'input devono essere copiati, spesso apprende proprietà utili dei dati.

2 Tipologie di autoencoders

In questa sezione vedremo varie tipologie di autoencoder. In particolare vedremo le varie tecniche, citate in precedenza, che permettono di forzare l'apprendimento della rete (caratteristica che ne differenzia i vari tipi [1]).

2.1 Undercomplete Autoencoders

Un modo per far apprendere ad un autoencoder le features dei dati è quella di costruire la rete di encoding in modo tale il suo output risulti di dimensione minore rispetto a quello dell'input originale. Questo tipo di architettura prende il nome di "collo di bottiglia". In questo modo non si permette all'autoencoder di avere troppa potenza e quindi non può riprodurre semplicemente la funzione identica, di conseguenza riesce a catturare le caratteristiche salienti dei dati. Gli autoencoder con questo tipo di architettura vengono chiamati Undercomplete.

2.2 Sparse Autoencoders

Questo tipo di autoencoder si distingue per l'aggiunta di un termine di penalità di sparsità durante l'addestramento. Questo permette alla rete di ridurre il numero di neuroni attivi (molti pesi tenderanno a 0) e quindi di "catturare"

solo le informazioni salienti dei dati. Mantenendo le notazioni adottate in precedenza e denotando con Ω la funzione di penalità, durante l'addestramento si minimizzerà la seguente quantità:

$$\sum_{x \in D} L(f_\alpha(g_\phi(x)), x) + \Omega(f(x))$$

2.3 Denoising autoencoders

Un altro approccio possibile è quello di, anziché intervenire sulla loss function da minimizzare, modificare il termine dell'errore di ricostruzione. Questo significa che l'addestramento viene fatto su un input corrotto da un qualche tipo di rumore (come ad esempio un rumore Gaussiano) e quindi l'obiettivo dell'autoencoder è di ricostruire l'input originale. Matematicamente quindi verrà minimizzata la seguente funzione:

$$\sum_{x \in D} L(f_\alpha(g_\phi(\tilde{x}), x)$$

dove $\tilde{x}$ rappresenta l'input corrotto. Questa procedura forza l'autoencoder a ricostruire la vera distribuzione dei dati [1].

2.4 Contractive autoencoders

Un'altra categoria di autoencoders con regolarizzazione sono i cosiddetti contractive autoencoders. In questo tipo di rete la penalizzazione è della seguente forma:

$$\Omega(x, f(x)) = \lambda \sum_i ||\nabla_x f(x)_i||^2$$

Come si vede dalla precedente formula, in questo caso la regolarizzazione viene applicata alle derivate parziali della funzione di encoding. Questa penalità forza il modello ad apprendere una funzione che non modifica di molto l'output quando l'input varia in maniera molto significativa.

3 Applicazioni degli autoencoders

In questa sezione approfondiremo i vari utilizzi degli autoencoders.

3.1 Riduzione della dimensionalità

La riduzione della dimensionalità consiste nella trasformazione dei dati da uno spazio ad alta dimensione in uno spazio di dimensione minore, in modo che quest'ultimo mantenga le proprietà più significative dei dati originali. Lavorare con spazi ad alta dimensionalità risulta spesso sconveniente per molte ragioni; i dati di questi tipo sono spesso sparsi e la loro analisi è solitamente computazionalmente costosa.

3.1.1 Analisi delle componenti principali

Una delle tecniche più utilizzate per la riduzione della dimensione è sicuramente l'analisi delle componenti principali (PCA). Lo scopo di quest'analisi è descrivere la variabilità in un insieme di variabili $x_1, x_2, ..., x_p$ attraverso un nuovo insieme di variabili incorrelate $y_1, y_2, ..., y_q$. La prima variabile y_1 è pari alla seguente combinazione linerare:

$$y_1 = a_{11}x_1 + a_{12}x_2 + ... + a_{1p}x_p$$

dove gli a_{1i} sono tali che la varianza di y_1 sia massima e il modulo del vettore $a_1 = (a_{11}, a_{12}, ..., a_{1p})$ sia pari ad 1. Successivamente si va ad individuare la variabile

$$y_2 = a_{21}x_1 + a_{22}x_2 + ... + a_{2p}x_p$$

in modo tale da massimizzare la varianza residua dei dati (cioè quella rimasta eliminando la parte di varianza coperta da y_1). Al vettore $a_2 = (a_{21}, a_{22}, ..., a_{2p})$ viene richiesto, come in precedenza, di essere di modulo unitario e di essere ortogonale con il vettore a_1. Procedendo in questa maniera si individuano le nuove variabili. Osserviamo che, dato il modo in cui sono costruite le variabili y_i, queste sono tutte incorrelate tra loro(grazie alla condizione di ortogonalità tra gli a_i) e sono ordinate in maniera decrescente in base alla percentuale di varianza spiegata.

3.1.2 Riduzione della dimensionalità con gli autoencoders

Iniziamo osservando che quando il decoder è lineare e la funzione d'errore L è l'errore quadratico medio un autoencoder di tipo undercomplete impara a generare uno spazio equivalente a quello della PCA. Di conseguenza un autoencoder risulta essere uno strumento più potente rispetto ad una PCA in quanto riesce a catturare anche relazioni non lineari tra i dati. Di contro un autoencoder ha un costo computazionale più alto rispetto alla PCA e comporta numerosi iperparametri da settare(numero di layer, numero di neuroni, funzioni di attivazione per i vari strati). Numerosi sono gli articoli che mostrano come un autoencoder possa surclassare le prestazioni dell'analisi delle componenti principali. Come esempio riportiamo l'elaborato [2]. In questo articolo l'autore lavora con il dataset MNIST (dataset contentente cifre da 0 a 9 scritte a mano molto noto in letteratura) e mostra come le cifre proiettate nelle prime due dimensioni della PCA diventino poco riconoscibili mentre addestrando un autoencoder che produce uno spazio a due dimensioni le cifre proiettate risultano molto simili alle originali. Questo dimostra come appunto un autoencoder riesca a catturare meglio i pattern interni ai dati anche se non lineari.

3.2 Anomaly Detection

Data la loro caratteristica di apprendere i pattern interni ai dati gli autoencoders si prestano molto bene a trovare eventuali valori anomali. L'idea chiave per

trovare le anomalie è quella di andare a vedere l'errore di ricostruzione compiuto dal modello sulle varie istanze, infatti, addestrando l'autoencoder sui valori "normali" sarà in grado di ricostruire questi valori in maniera abbastanza fedele, al contrario le anomalie le ricostruirà commettendo un errore più grande. Fissando quindi una threshold a partire dall'errore commesso sul training set si classificano i valori come anomali se sforano la threshold. Un utilizzo di questo tipo in un caso reale può essere trovato qui [3]. In quest'articolo viene riportato come l'azienda Airbus controlla le condizioni del laboratorio Columbus(Uno dei laboratori principali della Stazione Spaziale Internazionale). Per assicurare la sicurezza dello staff che lavora nel laboratorio e dei sistemi dello stesso, Airbus tiene traccia di svariati flussi di dati che vengono trasmessi sulla Terra. Per monitorare quindi la situazione, sono stati addestrati svariati autoencoders (uno per ogni "regione" di competenza) che monitorano lo status del laboratorio ed individuano eventuali anomalie con un procedimento molto simile a quello descritto in precedenza.

3.3 Image denoising

L'image denoising è uno dei principali problemi studiati in computer vision; di conseguenza in letteratura sono presenti svariate tecniche che cercano di recuperare un immagine corrotta. Matematicamente il problema si può modellare come segue:

$$z = x + y$$

Dove z è l'immagine corrotta, y il rumore e x l'immagine originale. L'obiettivo è quindi quello di stimare x partendo da z. Nella maggior parte dei casi si assume che y sia un rumore strutturato(proveniente da un processo ben definito). Uno dei metodi più utilizzati per l'image denoising sono appunto gli autoencoders. Ad esempio, nell'articolo [4] vediamo come questo tipo di rete viene utilizzata per ricostruire immagini corrotte di tipo medico ottenendo prestazioni molto importanti. L'autore utilizza un denoising autoencoder in cui l'input viene corrotto tramite un processo stocastico che porta a 0 un numero randomico di bit delle immagini. Come layer l'autore utilizza dei layer convoluzionali (che meglio si adattano all'elaborazione di immagini). Questo tipo di layer sono semplicemente degli strati in cui anzichè effettuare una classica moltiplicazione tra matrici viene utilizzata una convoluzione. Uno dei risultati più importanti mostrati nel paper è che, al contrario di come spesso è richiesto in molti problemi in data science, anche con un piccolo dataset l'autoencoder riesca a raggiungere un buon livello di ricostruzione, segno che questo tipo di modello riesce molto bene a risolvere il problema dell'image denoising.

4 Case Study

In questa sezione approfondiremo come effettuare anomaly detection con gli autoencoders utilizzando un dataset reale.

4.1 Dataset

Il dataset contiene circa 280000 transazioni bancarie effettuate con carte di credito in Europa. Tutte le transazioni sono state effettuate nell'arco di due giornate. Il dataset presenta 28 variabili numeriche che sono il risultato di una PCA effettuata sui dati originali dei clienti della banca. I dati veri non sono disponibili per motivi di privacy. Oltre le 28 dimensioni della PCA sono presenti altre 3 variabili, Amount (ammontare della transazione), Time (contatore che parte da un tempo arbitrario 0) e la variabile target Fraud (vale 1 se la transazione è fraudolenta e 0 altrimenti). Obiettivo del modello che costruiremo è quindi individuare le transazioni illegittime.

4.2 Caratteristiche del dataset

Iniziamo un'analisi preliminare per esplorare le proprietà del dataset. Innanzitutto osserviamo, come si vede dal barplot seguente, che la variabile target è estremamente sbilanciata. Le frodi costituiscono circa il 0.002% delle transazioni totali. Questa caratteristica della variabile target giustifica l'utilizzo di un autoencoder, tutti i metodi supervisionati avrebbero bisogno di un undersampling o di qualche altra tecnica per bilanciare il dataset costruendo dati "artificiali".

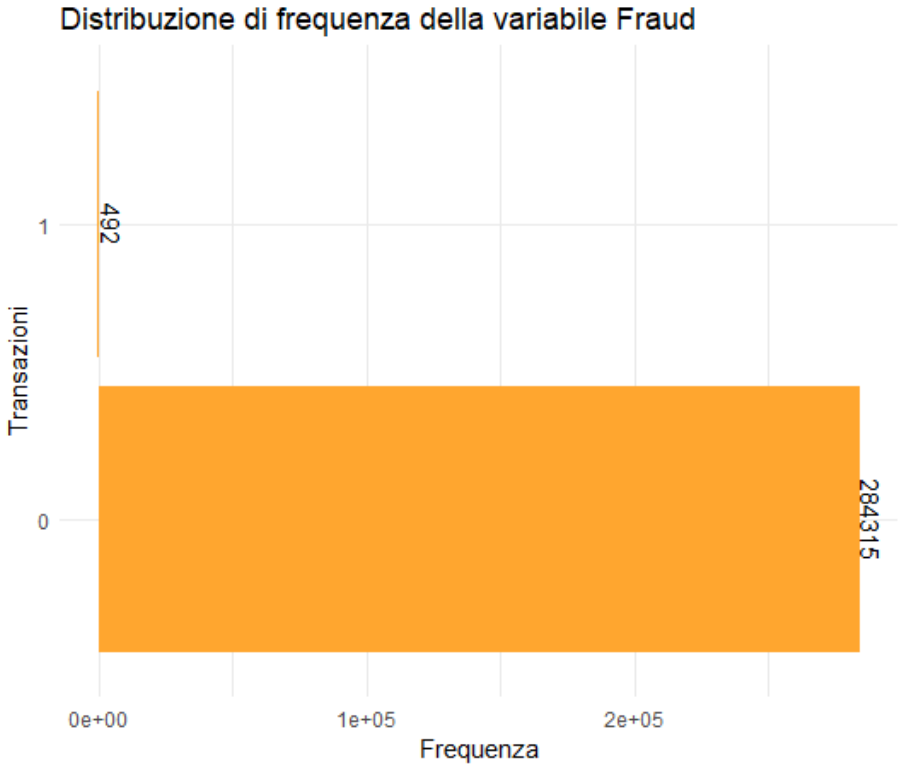


Figura 2: Conteggio delle due classi della variabile target "Fraud".

Vediamo adesso come è distribuita la variabile "Amount", in particolare vedremo se la sua distribuzione cambia in base al tipo di transazione(fraudolenta o onesta). Tracceremo, in realtà, l'ammontare logaritmico per rendere il grafico più leggibile.

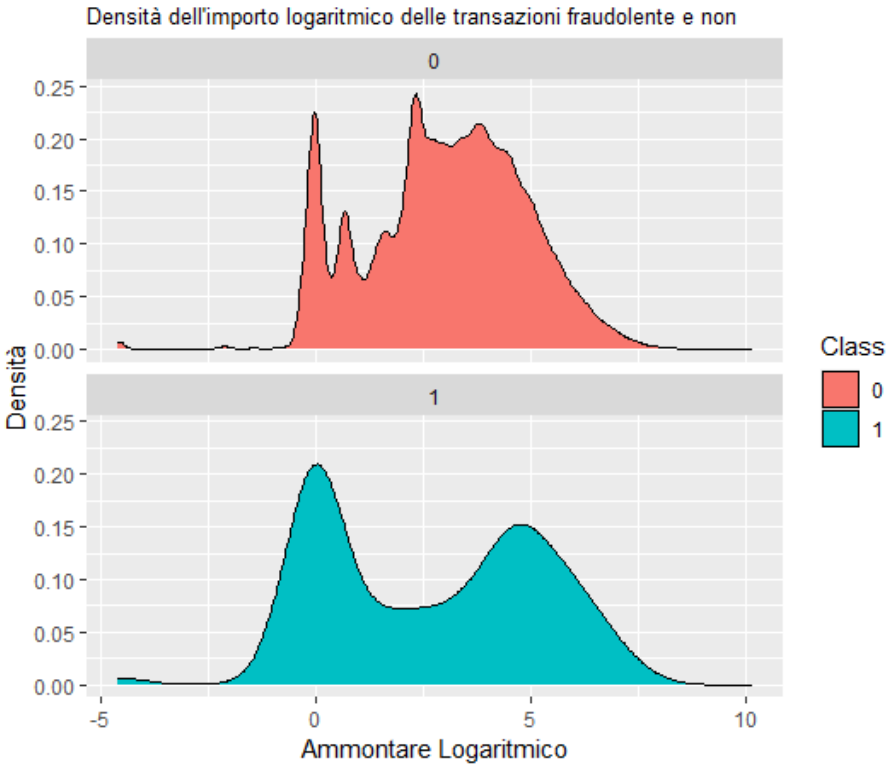


Figura 3: Distribuzione dell'ammontare logaritmico delle transazioni.

Dai grafici notiamo che la distribuzione per le transazioni fraudolente sia a due picchi(la maggior parte delle transazioni si accentra attorno a due valori) mentre quella delle transazioni oneste sia più variegata(questo potrebbe essere dovuto al fatto che le transazioni oneste siano molto maggiori in numero rispetto a quelle illegali). Tracciamo adesso uno scatterplot delle transazioni rispetto alle variabili della PCA per individuare eventuali pattern. Utilizzeremo solo le prime dimensioni in quanto più significative (Le prime dimensioni di una PCA sono quelle che spiegano più varianza nei dati).

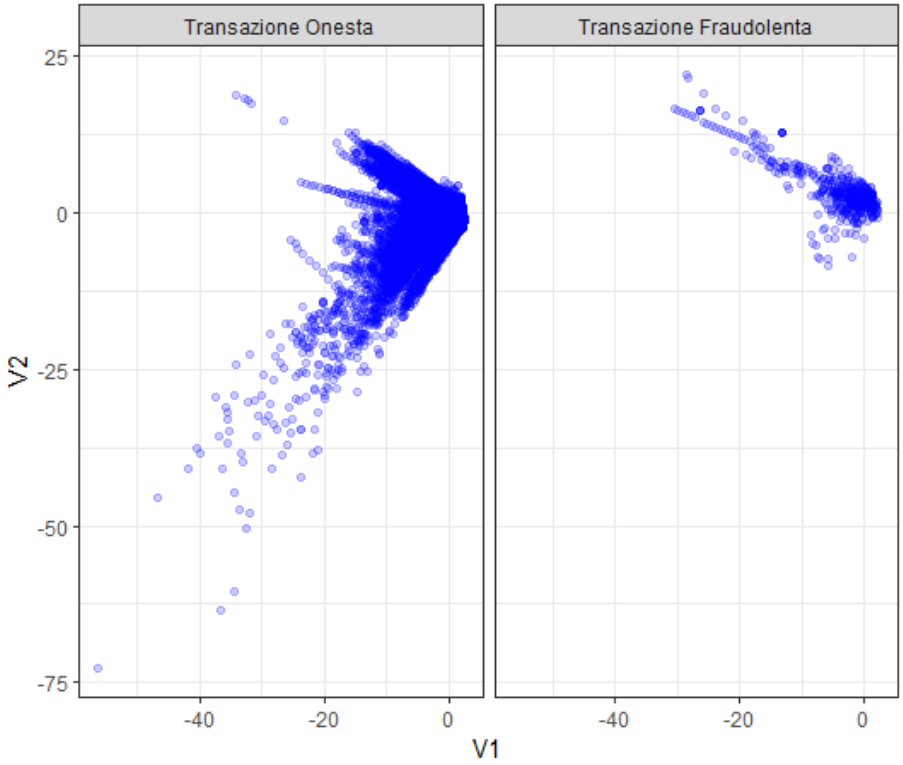


Figura 4: Scatterplot delle transazioni rispetto alle prime due dimensioni della PCA.

Come vediamo, le transazioni fraudolente hanno un valore poco variabile rispetto alla variabile V2 (a differenza di quelle oneste) segno che questa variabile è quella che caratterizza questo tipo di transazioni. La PCA riesce quindi ad individuare una differenza tra le due transazioni. La distinzione è però grossolana in quanto comunque la zona in cui sono presenti le transazioni fraudolente è ricca anche di quelle legittime.

4.3 Pre-processing

Una volta effettuata l'esplorazione dei dati inizia la costruzione effettiva del modello. La prima fase, fondamentale per tutti i modelli di machine learning, è quella di pre-processing. In questa fase i dati vengono standardizzati e si effettua l'imputazione dei valori mancanti. La standardizzazione è necessaria per rendere l'addestramento più efficiente e per non permettere alle variabili con varianza più elevata di prendere il "sopravvento" durante l'addestramento. Nella maggior parte dei casi sono due le tecniche di standardizzazione utilizzate.

- Standardizzazione Standard: Data una variabile X e chiamati x_i i suoi valori, questi vengono trasformati secondo la seguente formula:

  $$z_i = \frac{x_i - \mu}{\sigma}$$

  Dove μ è la media di X e σ la sua deviazione standard. Con questa trasformazione la nuova variabile Z ha media 0 e varianza 1.

- Standardizzazione MinMax: Mantenendo le notazioni della definizione precedente, questa volta i valori vengono trasformati nel seguente modo:

  $$z_i = \frac{x_i - m}{M - m}$$

  Dove m è il valore minimo assunto dalla variabile X e M il suo massimo. Con questa trasformazione la nuova variabile Z ha tutti i valori compresi nell'intervallo $[0, 1]$.

La scelta su quale tipo di standardizzazione effettuare è del tutto empirica e quindi va fatta valutando le prestazioni dei modelli addestrati nei due casi. Tornando adesso al nostro dataset, iniziamo osservando che i dati non presentano valori mancanti, non è stata necessaria, quindi, nessuna imputazione o eliminazione di righe del dataset. Per prima cosa i dati sono stati divisi in due, da una parte le transazioni oneste e dall'altra quelle fraudolente. L'80% delle transazioni legali (sorteggiate casualmente) sono state utilizzate come training set mentre il restante 20% insieme alle transazioni fraudolente sono state usate come test set. Infine entrambi i set sono stato normalizzati con una trasformazione di tipo MinMax (la scelta è stata effettuata in maniera empirica come detto in precedenza).

4.4 Addestramento del modello

Terminata la fase di pre-processing inizia la fase di addestramento del modello. Per scegliere gli iperparametri è stata effettuata una 5-fold Cross-Validation. Il training set è stato diviso quindi in 5 e a turno 4 delle 5 parti sono state utilizzate come set di training e la restante come test. Alla fine di ogni addestramento è stata calcolata la loss (si è optato per il mean-squared-error) del modello sul test set. Infine lo score di ogni modello è calcolato come la media delle 5 loss calcolate in precedenza. Effettuato questo procedimento su tutti i modelli candidati si è scelto, come modello migliore, quello con lo score più basso (avere score più basso significa in media ricostruire meglio i dati). I modelli migliori sono risultati i seguenti:

- Autoencoder a 5 strati con numero di neuroni [29,20,15,20,29]
- Autoencoder a 5 strati con numero di neuroni [29,25,20,25,29]
- Autoencoder a 5 strati con numero di neuroni [29,25,15,25,29]

- Autoencoder a 5 strati con numero di neuroni [29,25,10,25,29]

Tutti questi modelli hanno come funzione di attivazione nei vari layer(apparte i primo che accetta solo in input i dati) una sigmoide. Osserviamo inoltre che tutti i modelli sono autoencoders di tipo undercomplete. Per ogni modello è stato testato anche una versione analoga ma avente per funzione di attivazione dei vari strati la tangente iperbolica. Quest'ultima ha offerto però prestazioni significativamente peggiori (Questo è probabilmente dovuto al fatto che i dati sono stati scalati con un MinMax e quindi portati nel range [0,1] che coincide con il codominio della sigmoide).

Vediamo adesso l'andamento della loss function durante uno degli addestramenti per ogni modello.

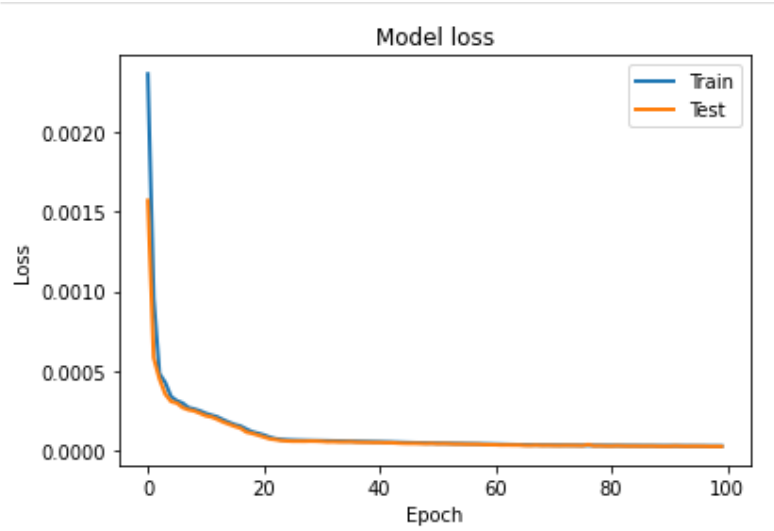


Figura 5: Andamento della loss function durante l'addestramento del modello [29,25,20,25,29].

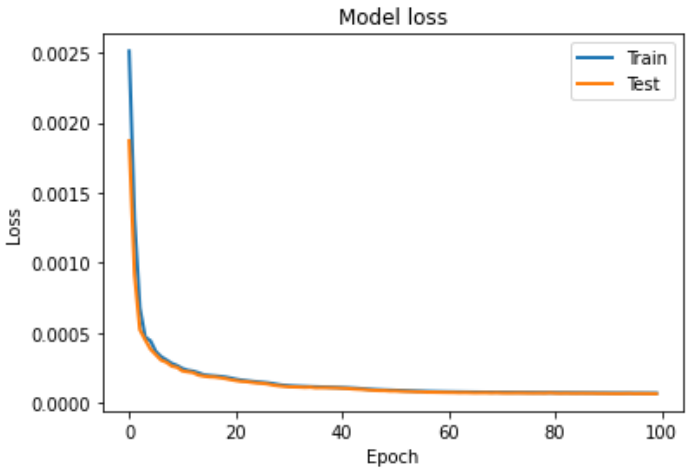


Figura 6: Andamento della loss function durante l'addestramento del modello [29,20,15,20,29].

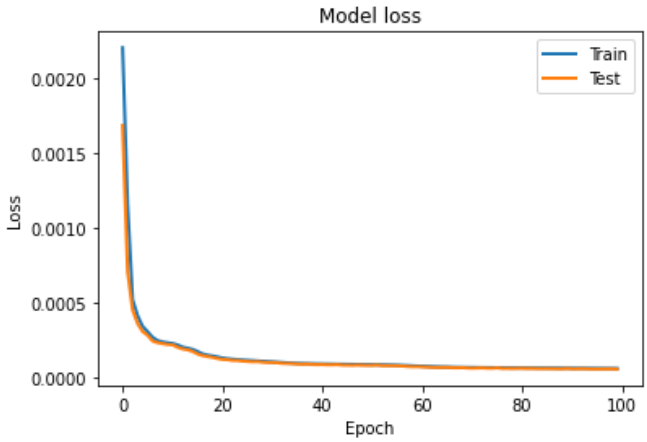


Figura 7: Andamento della loss function durante l'addestramento del modello [29,25,15,25,29].

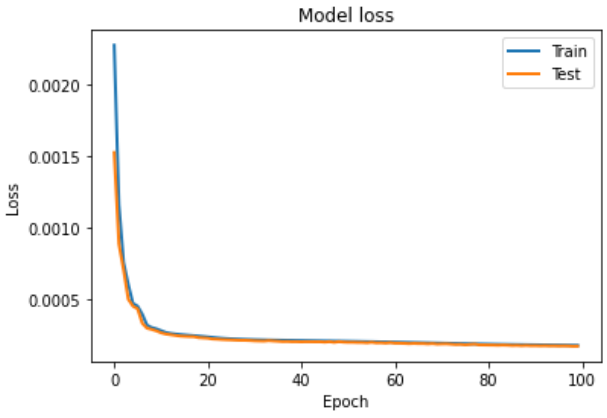


Figura 8: Andamento della loss function durante l'addestramento del modello [29,25,10,25,29].

Dai grafici si vede come dopo le 100 epoche tutti i modelli imparano a riprodurre le transazioni in input commettendo un errore di ordine più piccolo dei decimi di millesimo. Vediamo adesso gli score dei 4 modelli.

- Score del modello [29,25,20,25,29] : $3.31 \cdot 10^{-5}$
- Score del modello [29,20,15,20,29] : $6.74 \cdot 10^{-5}$
- Score del modello [29,25,15,25,29] : $5.67 \cdot 10^{-5}$
- Score del modello [29,25,10,25,29] : $1.75 \cdot 10^{-4}$

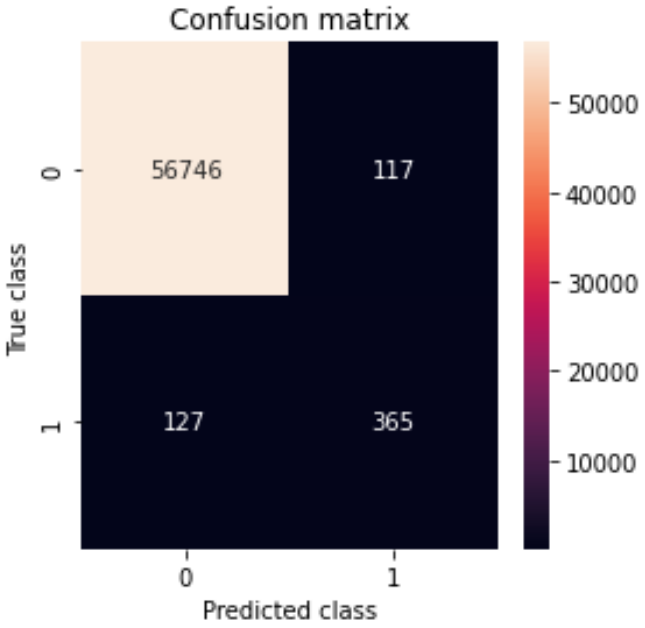
Tutti i modelli hanno un ottimo score, ma, il modello migliore risulta essere l'autoencoder con numero di neuroni [29,25,20,25,29].

4.5 Valutazione delle prestazioni

Terminata la fase di addestramento sono state quindi valutate le prestazioni del modello sul test set. Come prima cosa si è stabilita una threshold calcolata come il 0.998 (corrispondente alla percentuale di transazioni legali) percentile dell'errore di ricostruzione commesso dall'autoencoder sulle varie transazioni del traning set. Un elemento del test set viene quindi classificato come fraudolento se il suo errore di ricostruzione sfora la threshold o normale altrimenti. Iniziamo plottando graficamente la previsione del modello.

Figura 9: Plot delle transazioni del test set rispetto all'errore di ricostruzione dell'autoencoder.

Come si vede dal grafico la maggior parte delle transazioni fraudolente sfora la threshold mentre quasi tutte quelle oneste restano al di sotto. Quantifichiamo, infine, la qualità della previsione del modello. Tracceremo una confusion matrix e calcoleremo i principali score.




```
Accuracy:  0.9957457937407375
Recall:  0.741869918699187
Precision:  0.7572614107883817
F1:  0.7494866529774128
```

Figura 10: Confusion matrix della previsione e principali score associati.

Vediamo quindi che il modello effettivamente riesce a distinguere le due classi. Degno di nota tra gli score è l'F1 pari quasi al 75%. Questo è un ottimo risultato considerando l'alto sbilanciamento del dataset.

5 Bibliografia

[1] IAN GOODFELLOW, YOSHUA BENGIO, AARON COURVILLE: Deep Learning, MIT Press, 2016.
[2] RUKSHAN PRAMODITHA : How Autoencoders Outperform PCA in Dimensionality Reduction, 2019.
[3] PHILIPP GRASHORN, JONAS HANSEN, MARCEL RUMMENS: How Airbus Detects Anomalies in ISS Telemetry Data Using TFX, 2020.
[4] LOVEDEEP GONDARA: Medical image denoising using convolutional denoising autoencoders, 2016.