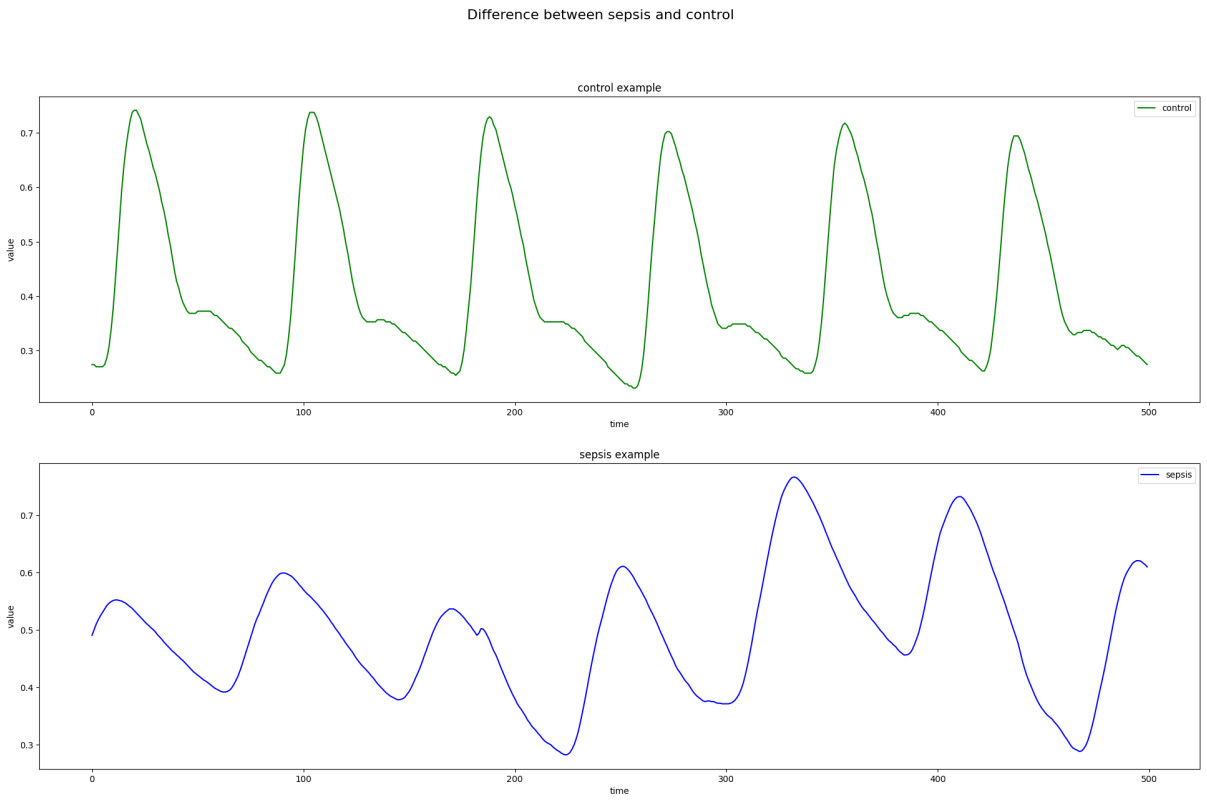
Relazione Progetto Machine Learning A.A. 2022/2023 - Luca Taverna

Realizzazione di un Autoencoder formato da layer LSTM per effettuare Anomaly Detection su serie temporali derivanti da acquisizione di segnali PPG al fine del riconoscimento preventivo della malattia Sepsi

1. Introduzione e descrizione del dataset utilizzato

La **Sepsi** o setticemia è una sindrome clinica che si verifica come rara complicazione di un'infezione. La sepsi è una condizione potenzialmente molto grave, che passa attraverso stadi di gravità crescente e come tale necessita di un immediato trattamento medico, senza una cura tempestiva infatti, può risultare in una malattia mortale.
L'obiettivo del progetto è quindi quello di sfruttare le tecniche di Machine Learning attuali per creare un modello che, ricevendo in **input segnali fotopletismografici (PPG)**, riesca a riconoscere, quanto prima, l'avvento di questa malattia.
Per questo progetto sono stati utilizzati i dati presenti nel **MIMIC-III Waveform Database Matched Subset**, una versione ridotta del dataset MIMIC-III Waveform Database, nel quale sono presenti 22137 record di segnali fisiologici e 22247 record numerici discreti da 10282 pazienti in terapia intensiva.
In questo progetto nello specifico sono stati utilizzati **dati pre-processati da alcuni colleghi**. Al dataset originale sono quindi state eliminate alcune istanze caratterizzate dall'assenza di dati (presumibilmente movimenti o perdita del sensore da parte del paziente) e applicate tecniche di correzione per ridurre l'impatto della presenza di rumore nei dati stessi. Successivamente i dati sono stati preventivamente suddivisi in training (70%), validation (10%) e test set (20%), in ognuno dei quali sono presenti i due gruppi di segnali, sepsis e control. Le **finestre temporali presenti nei vari sotto-dataset sono anche raggruppate per paziente**, ovvero ogni paziente presente in uno di essi avrà circa 30 finestre temporali (di *lunghezza 15.000 = 120 sec x 125 Hz*) associate. In questo modo sarà possibile anche valutare performance su quanti pazienti sono stati riconosciuti correttamente come malati/sani e non solo sulle singole finestre temporali.
Nell'immagine sottostante è possibile visualizzare la differenza tra un segnale di un paziente non affetto da sepsi (segnale di controllo) ed uno affetto dalla malattia (sepsi).



1.1. Descrizione del task di Anomaly Detection

Per affrontare il task in questione è stato sviluppato un **modello Autoencoder formato da layer LSTM** per effettuare tecniche di Anomaly Detection.
Questo modello verrà **addestrato sui dati di controllo** per imparare a riprodurre fedelmente tali segnali. Questo si traduce nel fatto che si cercherà di minimizzare la differenza dei valori delle funzioni di loss tra le istanze dei dati reali e le corrispondenti predizioni che il modello ha fornito in output per tali segnali. In questo modo il modello dovrebbe essere in grado di riprodurre perfettamente i segnali di controllo, mentre *dovrebbe produrre in output segnali poco coerenti con i corrispettivi valori forniti in input nel caso in cui gli si proponessero segnali diversi, come ad esempio quelli di sepsi*. In questo caso ci si aspetta infatti che la **differenza tra le loss di reali finestre temporali di sepsi e quelle delle rispettive predizioni sia più alta rispetto ai segnali di controllo**.
Grazie a ciò si può impostare una **soglia** attraverso la quale effettuare una **classificazione** della tipologia del segnale. Se infatti, fornita in input al modello una finestra temporale di un segnale PPG di un paziente, la differenza tra la loss del segnale reale e di quello riprodotto (fornito in output dal modello) è inferiore alla soglia impostata, tale istanza verrà categorizzata come “di controllo”. In caso contrario, ovvero se la differenza delle loss è superiore alla soglia impostata, il segnale verrà categorizzato come “sepsi”.
Sono state quindi utilizzate alcune metriche per valutare quanto il modello riproduce fedelmente i vari segnali e successivamente valutate le performance di alcune configurazioni di parametri del modello sul validation set. Per fare ciò sono state usate metriche quali accuracy, recall, precision e score F1 sia sulle singole finestre temporali (prendendo ogni segnale come indipendente dagli altri) sia sui vari pazienti (raggruppando i vari segnali dei

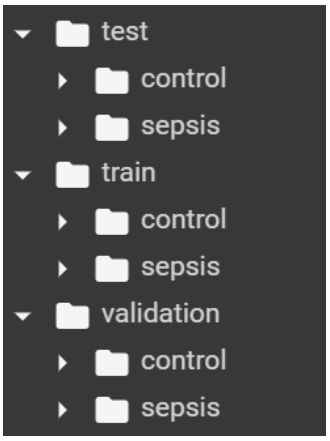
pazienti e considerando un'unica classificazione per singolo paziente, se ha la sepsi o meno). Per effettuare quest'ultimo punto è stata scelta un'ulteriore **soglia** numerica **che indicherà dopo quanti segnali categorizzati come "sepsi" del singolo paziente considerare quest'ultimo come "malato di sepsi"**.
Infine tutti questi parametri vengono anche utilizzati per valutare le performance finali sul test set del modello finale.

1.2. Tecnologie utilizzate

Per sviluppare il progetto è stato utilizzato **Python** come linguaggio di programmazione.
Per importare il dataset ed effettuare le operazioni di preprocessing sono state utilizzate le librerie **NumPy** e **scikit-learn**.
Il modello è stato creato invece con metodi e classi di **Keras** e **Tensorflow**.
La maggior parte delle metriche di valutazione sono state implementate manualmente seguendo la loro definizione matematica.
Matplotlib e scikit-learn sono infine state utilizzate per creare tutti i grafici relativi ai segnali, tutti i grafici relativi alle varie fasi di addestramento e di illustrazione delle varie performance del modello (come ad esempio le varie distribuzioni dei valori di loss ottenuti sui vari dataset) e le matrici di confusione.
Il codice è stato scritto ed eseguito su **Google Colab** dal momento che, nonostante le limitazioni temporali del piano gratuito, permette di avere una buona potenza computazionale per addestrare le reti ed una esecuzione del codice in tempo reale, al contrario per esempio di alcuni cluster hpc sui quali non sarebbe stato possibile effettuare modifiche sul momento del codice per effettuare varie prove.

2. Preparazione del dataset e preprocessing

Il primo passo da eseguire per preparare il dataset al preprocessing e poi al successivo addestramento del modello è stato quello di caricare i file contenenti le varie finestre temporali nel codice python.
Il dataset che mi è stato fornito è strutturato secondo questa **gerarchia di cartelle**:



All'interno di ognuna delle cartelle "control" o "sepsis" sono presenti alcuni file aventi una notazione del tipo: "*p008281_#13.npy*", in ognuno di questi file è presente una singola finestra temporale di un paziente avente la registrazione di *15.000 valori = 120 sec x 125 Hz*.

- La parte di stringa che precede l'underscore identifica l'**ID del paziente** in questione, in questo caso: "*p008281*";
- Il numero presente dopo il cancelletto "*#13*" identifica la **finestra temporale** in questione nell'insieme delle finestre temporali di quel particolare paziente, in questo caso la tredicesima.
- L'estensione "*.npy*" indica che il file contiene un **array numpy** che può essere caricato tramite il metodo *np.load('filename.npy')*.

Dopo aver creato, per ognuna delle sottocartelle, un array contenente tutti i nomi dei file di quella determinata cartella, tramite una *map* ed il metodo *load* sopracitato, è stato possibile caricare tutti i dati delle finestre temporali nell'array stesso.
Dato che le finestre temporali sono già raggruppate in variabili diverse sia per quanto riguarda il dataset di appartenenza (*train, validation e test*), sia per quanto riguarda la classe (*control e sepsis*), ai vari dati è stata rimossa la colonna contenente la *label*. L'intento è infatti quello di far apprendere al modello la ricostruzione del segnale e non di associare una label ad una determinata tipologia di segnale.

2.1. Raggruppamento delle finestre temporali per i vari pazienti

I dati presenti nei vari array sono poi stati riordinati in modo tale da avere **tutte le finestre temporali di un determinato paziente contigue e nell'ordine corretto**. In questo modo sarà poi possibile calcolare con quante finestre temporali di anticipo è stata rilevata la sepsi in un determinato paziente.
Sono anche stati creati alcuni array contenenti tuple del tipo:
`[patient_id, number_of_windows_of_the_patient],`
che permetteranno di *tenere traccia del numero di finestre temporali dei vari pazienti* per tutto il resto del progetto.

2.2. Normalizzazione

Tutte le finestre temporali sono state in seguito normalizzate (per portare tutti i valori nell'internvallo compreso tra 0 e 1) tramite un **MinMaxScaler**. Non tutti i dati erano infatti compresi nello stesso intervallo di valori e tramite la normalizzazione viene quindi facilitato l'apprendimento del modello in fase d'addestramento.

2.3. Aggiunta pazienti ai dataset di validation e test (sepsi)

Ricordando che, come affermato nell'introduzione, il modello verrà addestrato esclusivamente sulle finestre temporali di controllo, il **dataset di training-sepsis non verrebbe utilizzato**. Dal momento che però i dataset di validation e test erano sbilanciati verso i dati di controllo (erano presenti più finestre temporali di controllo rispetto a quelle di sepsi), e dato che l'obiettivo principale del progetto era quello riuscire ad identificare quanto più possibile i pazienti aventi la sepsi, è stata aggiunta nel codice la possibilità di prendere un determinato numero di pazienti presenti nel dataset *training-sepsis* ed aggiungere tutte le loro finestre temporali nel dataset di *validation* o di *test*.
Sono quindi state aggiunte tutte le finestre temporali di 4 pazienti al *validation-sepsis dataset* e 10 al *test-sepsis dataset*.

2.4. Dimensioni finali dei vari dataset

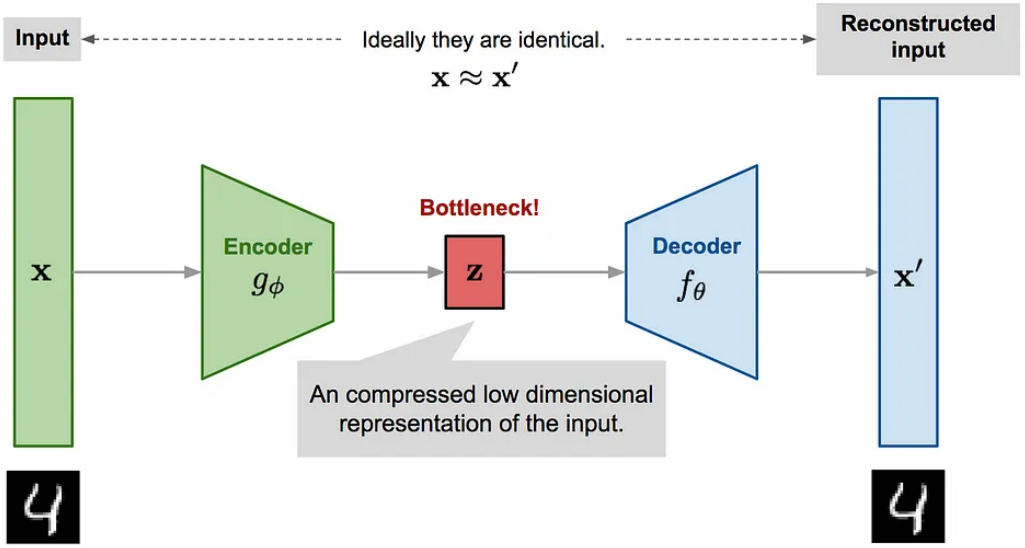
I dataset utilizzati al fine del progetto saranno quindi così composti:

```
training (control) dataset shape: (2772, 15000, 1)
consisting of 95 patients

validation (control) dataset shape: (390, 15000, 1)
consisting of 13 patients

validation (sepsis) dataset shape: (420, 15000, 1)
consisting of 14 patients

test (control) dataset shape: (810, 15000, 1)
consisting of 27 patients

test (sepsis) dataset shape: (873, 15000, 1)
consisting of 30 patients
```

Il primo numero dentro le parentesi identifica la lunghezza del dataset in questione dal punto di vista del numero delle finestre temporali totali, mentre sotto è specificato il numero di pazienti da cui sono prese le stesse.

3. Descrizione del modello Autoencoder utilizzato

Un autoencoder è un tipo di rete neurale artificiale utilizzato per apprendere codifiche efficienti di dati non etichettati, rientrando quindi nel campo dell'apprendimento non supervisionato. In generale un **autoencoder è formato da due componenti:** *l'encoder* e il *decoder*. L'encoder si occupa di prendere la sequenza di dati in input e creare una rappresentazione compressa degli stessi in modo tale da ridurre la dimensionalità pur mantenendo le loro caratteristiche salienti/importanti. Il decoder si occupa, al contrario, di ricreare il segnale originale partendo dalla rappresentazione compressa fornita dall'encoder.



Per realizzare encoder e decoder è stato utilizzato il modello *Sequential* (`tensorflow.keras.models.Sequential`) ed i layer presenti nella libreria `tensorflow.keras.layers`.

3.1. Encoder

L'encoder è stato così realizzato:

```
encoder = Sequential([
    Input(shape=(N_SAMPLES, 1)),
    LSTM(units=outer_lstm_units, return_sequences=True),
    Dropout(rate=0.2),
    LSTM(units=inner_lstm_units, return_sequences=True),
    Dropout(rate=0.2)
  ], name="encoder")
```

Nel primo layer di input è stata specificata la dimensione dei dati di input, in questo caso *(N_SAMPLES, 1)*, in cui ***N_SAMPLES** = 15.000 (120 sec x 125 Hz)*.
Successivamente sono stati inseriti due layer LSTM tra i quali è presente un layer di dropout con *dropout_rate = 0.2*.
I due layer LSTM hanno un numero di *units* differenti in quanto, come detto nell'introduzione di questo capitolo, l'obiettivo dell'encoder è quello di ridurre la dimensionalità dei dati, quindi si avrà che: **`outer_lstm_units > inner_lstm_units`**. In particolare valori possibili potrebbero essere per esempio:

`outer_lstm_units = 128` e `inner_lstm_units = 64`

oppure

`outer_lstm_units = 64` e `inner_lstm_units = 32`.

Un ulteriore dettaglio a cui va prestata attenzione è il settaggio del flag **`return_sequences`** a ***True***. Questo permette di fornire in output l'intera sequenza di hidden states, necessaria per poter ricostruire il segnale nei layer successivi.

3.2. Decoder

Il decoder invece è stato realizzato in questo modo:

```
decoder = Sequential([
    LSTM(units=inner_lstm_units, return_sequences=True),
    Dropout(rate=0.2),
    LSTM(units=outer_lstm_units, return_sequences=True),
    Dropout(rate=0.2),
    TimeDistributed(Dense(units=1))
  ], name="decoder")
```

Come l'encoder anche il decoder presenta due layer LSTM separati da un layer di dropout. In questo caso, come si può però notare, `inner_lstm_units` ed `outer_lstm_units` sono invertiti. Questo avviene poiché la funzione del decoder è quella di ricostruire il segnale originale partendo da una sua versione compressa e quindi la dimensionalità dello stesso dovrà aumentare.

Particolare attenzione si deve infine prestare al `TimeDistributed` layer che funge da wrapper per il layer `Dense` con un'unità. Questo permette di applicare il `Dense` layer ad ogni time-step della sequenza di input e quindi di ricostruire il segnale fornendo in output una finestra temporale avente le dimensioni iniziali.

3.3. Composizione dell'Autoencoder e sommario dei modelli

Encoder e decoder vengono successivamente così combinati:

```
model = Sequential([
    encoder,
    decoder
  ], name="LSTMAutoencoder")
```

Anticipando quella che sarà la configurazione di parametri ottimale scelta, impostando `outer_lstm_units = 128` e `inner_lstm_units = 64`, si ottiene un modello **Autoencoder** così composto:

```
Model: "LSTMAutoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder (Sequential)        (None, 15000, 64)         115968

 decoder (Sequential)        (None, 15000, 1)          131969

=================================================================
Total params: 247937 (968.50 KB)
Trainable params: 247937 (968.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In cui **l'encoder** è formato da:

```
Model: "encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 15000, 128)        66560

 dropout (Dropout)           (None, 15000, 128)        0

 lstm_1 (LSTM)               (None, 15000, 64)         49408

 dropout_1 (Dropout)         (None, 15000, 64)         0

=================================================================
```

```
Total params: 115968 (453.00 KB)
Trainable params: 115968 (453.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Ed il **decoder** è formato da:

```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 15000, 64)         33024

 dropout_2 (Dropout)         (None, 15000, 64)         0

 lstm_3 (LSTM)               (None, 15000, 128)        98816

 dropout_3 (Dropout)         (None, 15000, 128)        0

 time_distributed (TimeDist  (None, 15000, 1)          129
 ributed)

=================================================================
Total params: 131969 (515.50 KB)
Trainable params: 131969 (515.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4. Addestramento del modello e validazione dei migliori iperparametri

Nella progettazione del modello e della sua fase di addestramento è stato necessario cercare ed impostare molti iperparametri, un esempio sono il *learning rate*, la *funzione di loss*, le *epoche di addestramento*, il *batch size*, il *numero di unità del modello* e le varie *soglie* precedentemente nominate.
Su alcuni di essi è stata svolta una piccola **grid search** per valutare quale combinazione di essi fosse la migliore.
Su altri parametri, per esigenze temporali, computazionali (date dal piano gratuito di Google Colab) e prestazionali (come verrà illustrato in seguito), sono state effettuate scelte “arbitrarie”. Questi, dopo una fase di analisi manuale, sono quindi stati mantenuti fissi a valori predefiniti durante la fase di validazione di quelli su cui è invece stata effettuata la grid search.

4.1. Scelte degli iperparametri effettuate

Il primo parametro che non è stato sottoposto a grid search è il **numero di epoche di addestramento**. Il valore fissato nella versione definitiva del progetto è infatti `EPOCHS = 50`.
Dopo alcune prove manuali si è infatti visto che, dato l'elevato numero di `Trainable params` presenti nel modello (come visto nella sezione 3), un numero di epoche più elevato rendeva le tempistiche di addestramento troppo onerose per un ambiente interattivo come Colab e contemporaneamente non portava a grandi miglioramenti in termini di performance.
Per evitare l'overfitting senza ridurre il numero di epoche è invece stato introdotto il meccanismo dell' "**early stopping***". Grazie* alla callback inserita all'interno del metodo `fit`:

```
callbacks=[EarlyStopping(monitor="val_loss", patience=5, mode="min")]
```

è infatti stato possibile monitorare il cambiamento del valore di loss in validation e terminare l'addestramento nel caso si presentasse overfitting, identificato nel momento in cui per più di *5* volte, valore di *patience* impostato, il valore di loss di un'epoca fossa maggiore di quello della precedente.
Proprio per quanto riguarda l'addestramento, come detto in precedenza esclusivamente effettuato sui dati di *control*, è stato impostato il parametro **`validation_split`** del metodo `fit` uguale a *0.2*. Effettuando qualche prova manuale e seguendo altri esempi presenti in letteratura questo, infatti, si è dimostrato il valore che ha permesso di ottenere la loss minore dopo la fase di addestramento.
Come **loss function** è invece stato utilizzato il *Mean Squared Error* o *MSE*. Dopo un confronto con il *Mean Absolute Error* è infatti emerso come l'MSE consentisse una riproduzione più fedele dei segnali da parte del modello.
L'ultimo parametro mantenuto ad un valore predefinito per quanto riguarda l'addestramento vero e proprio del modello è il **batch size**. Si è notato come un *batch size* uguale a *64* portasse infatti ad un buon bilanciamento tra velocità di addestramento e performance predittive.
Riepilogando sono stati impostati questi parametri:

```
BATCH_SIZE = 64
EPOCHS = 50
VALIDATION_SPLIT = 0.2
LOSS_FUNCTION = "mse"
EarlyStopping(monitor="val_loss", patience=5, mode="min")
```

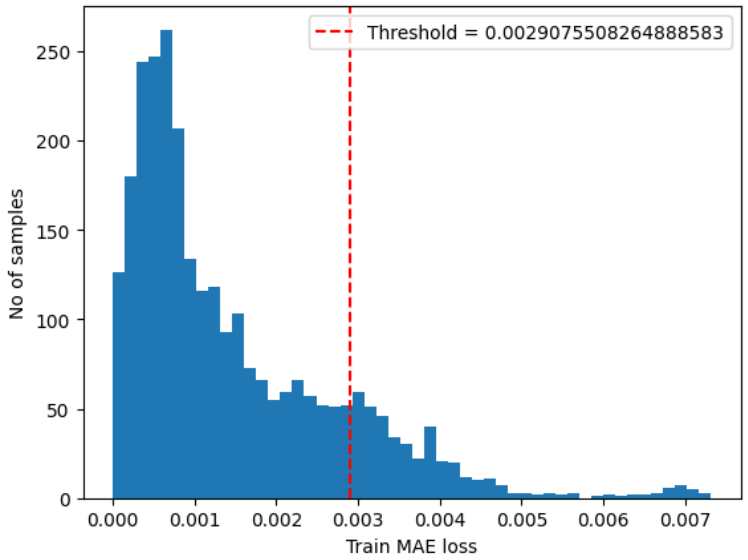
Infine per automatizzare la fase di validazione, e poter effettuare una grid search sugli ultimi parametri rimasti, è stato necessario impostare ulteriori due parametri soglia. Si è dovuto infatti innanzitutto stabilire la **soglia** in termini di **valore di loss** superata la quale una certa finestra temporale veniva considerata come sepsi. Dal momento che ogni addestramento forniva ovviamente una diversa scala di valori di loss sulle varie istanze non era sicuramente possibile impostare un valore fisso e prestabilito, ma questo doveva variare di esecuzione in esecuzione. Per fare ciò, una volta finito l'addestramento, viene fatto predire al modello il dataset di training e successivamente viene scelto come valore soglia il valore che divide gli output del modello ad esattamente una pre-determinata percentuale. Ad esempio, si supponga che questa percentuale sia stata *fissata al 90%*, il procedimento che verrà seguito dal sistema è il seguente:

- verrà innanzitutto chiesto al modello di effettuare la predizione del dataset di training,

- successivamente le predizioni verranno confrontate con le istanze reali tramite l'RMSE,
- tutte le differenze in termini di RMSE tra finestre temporali reali e le rispettive fornite in output dal modello verranno salvate in una lista che avrà quindi le stesse dimensioni di quelle di input,
- successivamente i valori di loss all'interno di questa lista verranno ordinati e verrà scelto un valore di loss che dividerà esattamente i valori ad una determinata percentuale,
- avendo impostato 90% in questo esempio, si avrà che il 90% dei valori presenti nella lista saranno inferiori al valore di soglia impostato, mentre il 10% dei valori sarà superiore a quest'ultima.

Effettuando varie prove, ed anticipando la difficoltà che avrà il modello nel predire la label di sepsi, è stata impostata una soglia che divide i valori di loss del training set in **85% e 15%**.

Seguendo l'esempio in figura, che mostra la distribuzione dei valori di loss sul training set, l'85% dei valori di loss presenti nella lista saranno inferiori alla soglia (a sinistra della linea tratteggiata rossa), mentre il rimanente 15% saranno superiori alla soglia (alla destra della linea rossa).



Infine è stato necessario effettuare le predizioni di "sano" ed "avente la sepsi" sui singoli pazienti. Avendo ogni paziente circa 30 finestre temporali associate, si è dovuto stabilire dopo quante finestre temporali predette come sepsi effettuare la decisione che per il sistema il paziente fosse malato. Sempre per facilitare il riconoscimento della sepsi, questo valore è stato impostato ad **1 finestra temporale**. Se, quindi, 1 finestra temporale di un paziente viene predetta come sepsi, il paziente viene considerato come malato di sepsi.
Ricapitolando, la soglia per la predizione delle singole finestre è stata impostata al valore che divide i valori di loss del training set in **85% e 15%**, mentre il numero di predizioni di sepsi dopo il quale il paziente viene considerato malato è stato impostato ad **1**.

4.2. Grid search

Al contrario di quelli scelti nella sezione precedente, **alcuni iperparametri sono stati selezionati tramite una grid search**. In particolar modo si è scelto di valutare la miglior

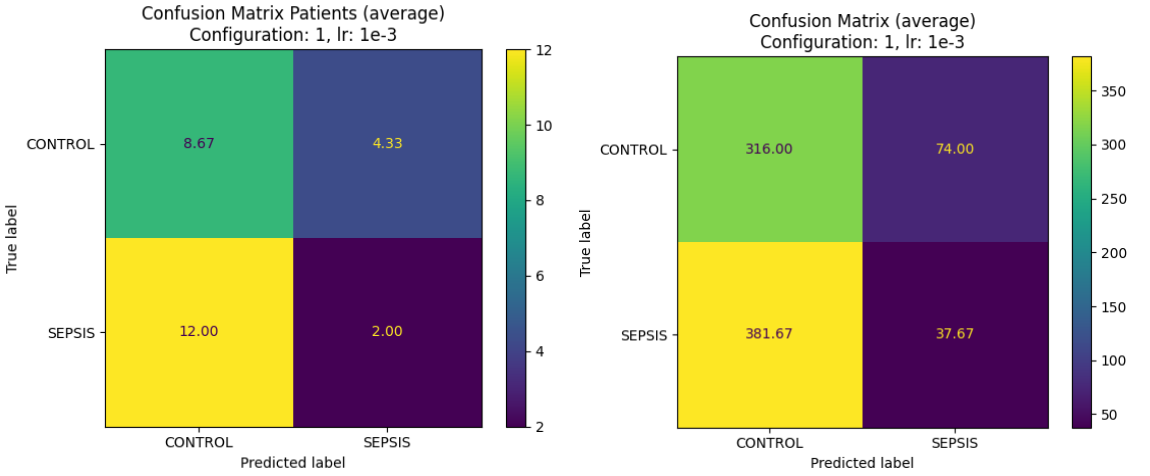
combinazione di *<numero di unità all'interno dei layer lstm, learning rate>* tramite questa metodologia.
Sono state valutate 2 configurazioni delle units dei layer LSTM:

- **configurazione 1:** `outer_lstm_units = 64` e `inner_lstm_units = 32`
- **configurazione 2:** `outer_lstm_units = 128` e `inner_lstm_units = 64`,

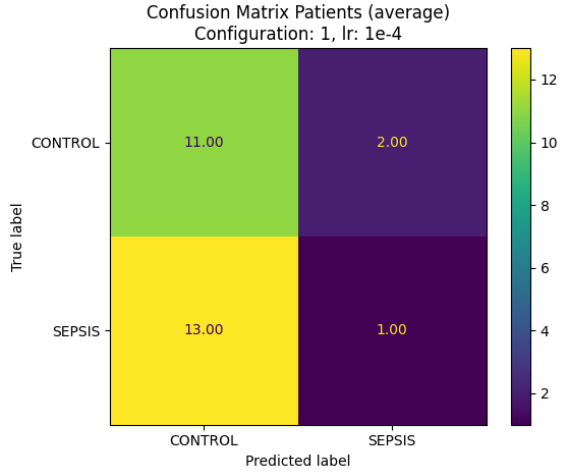
mentre sono stati valutati 4 valori per il learning rate: `1e-3, 1e-4, 1e-5, 1e-6.`

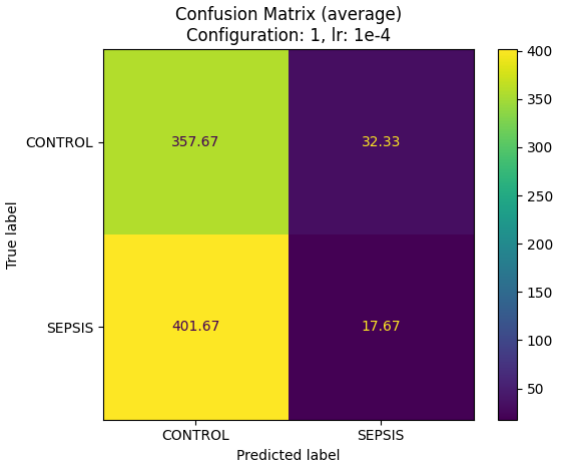
Per ogni possibile combinazione di: `learning_rate - configurazione_units` sono state effettuate **5 esecuzioni** ed infine è stata scelta la combinazione che forniva le migliori **performance** (come media delle 5 esecuzioni) **sul validation set**. Come metrica di valutazione per la scelta della combinazione ottimale è stata utilizzata la **Recall sulla label di sepsi** in quanto l'obiettivo del progetto è proprio riuscire ad identificare quanti più malati di sepsi possibile. Come prima valutazione è stato quindi utilizzato il valore dei True Positive delle matrici di confusione riguardanti le predizioni sui singoli pazienti (paziente malato / paziente sano). Nel caso in cui vi fosse un'uguaglianza di questo valore, a causa del basso numero di pazienti, è stato invece considerato il valore dei veri positivi ma della matrice di confusione generale (ottenuta considerando ogni istanza indipendente dalle altre e quindi non suddividendo le varie predizioni per singolo paziente).
Sono di seguito riportate le matrici di confusione ottenute sul validation set **combinando i risultati delle 5 esecuzioni** per ogni combinazione sia riguardanti i pazienti sani/malati sia riguardanti le singole finestre temporali (control/sepsis).

- `outer_lstm_units = 64, inner_lstm_units = 32, learning_rate = 1e-3:`

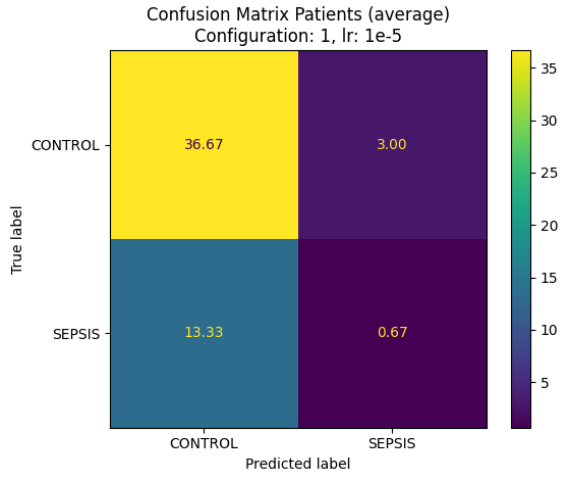


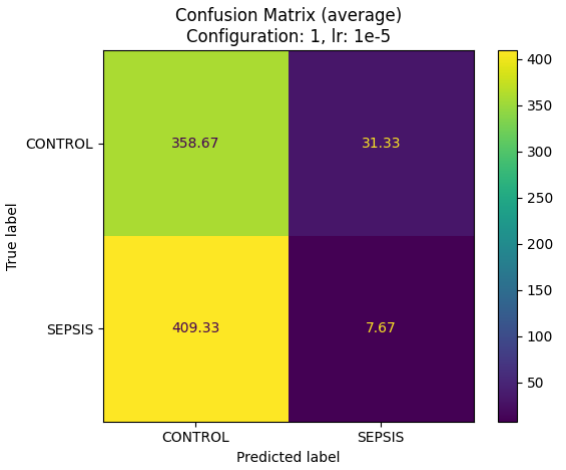
- outer_lstm_units = 64, inner_lstm_units = 32, learning_rate = 1e-4:



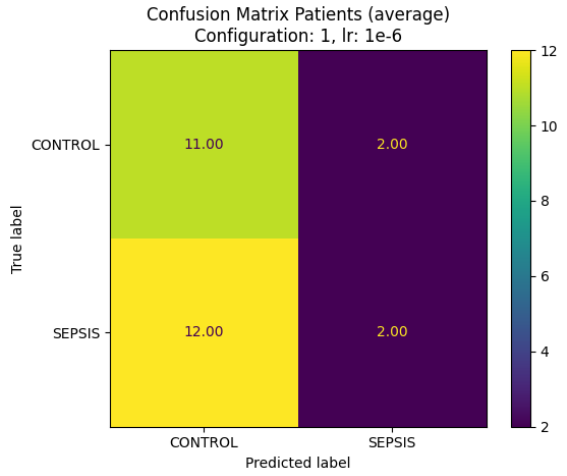


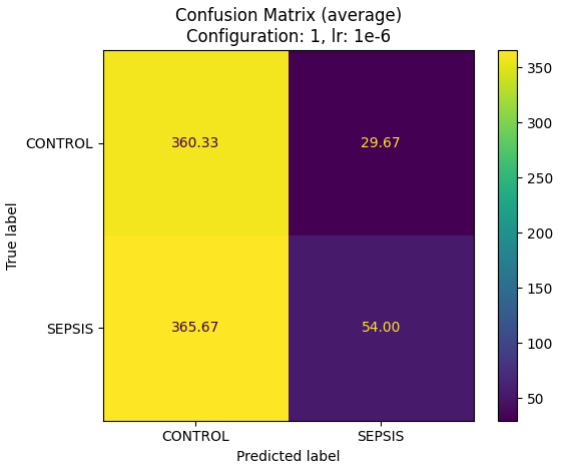
- outer_lstm_units = 64, inner_lstm_units = 32, learning_rate = 1e-5:



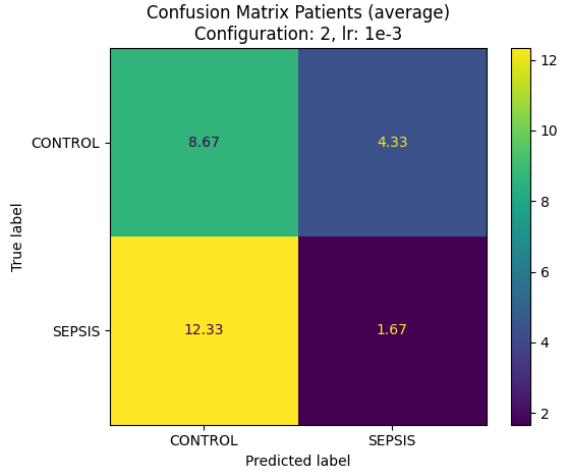


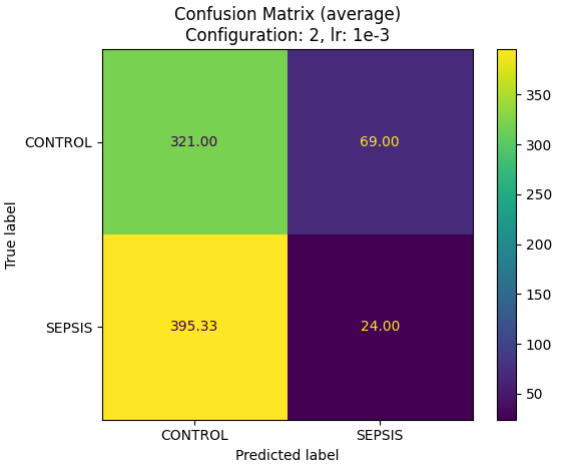
- outer_lstm_units = 64, inner_lstm_units = 32, learning_rate = 1e-6:



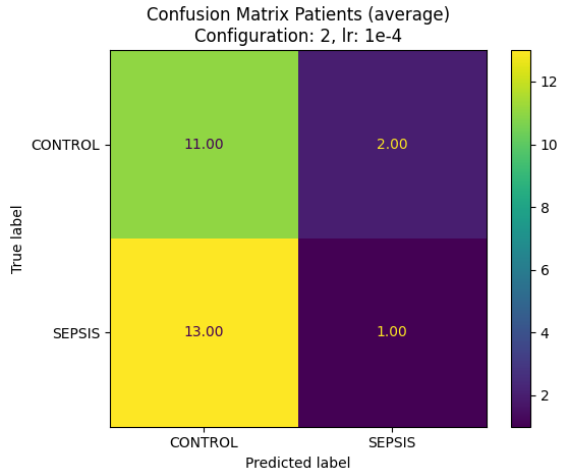


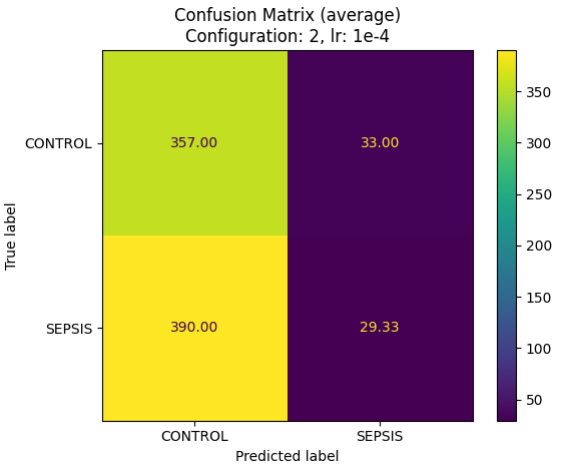
- outer_lstm_units = 128, inner_lstm_units = 64, learning_rate = 1e-3:



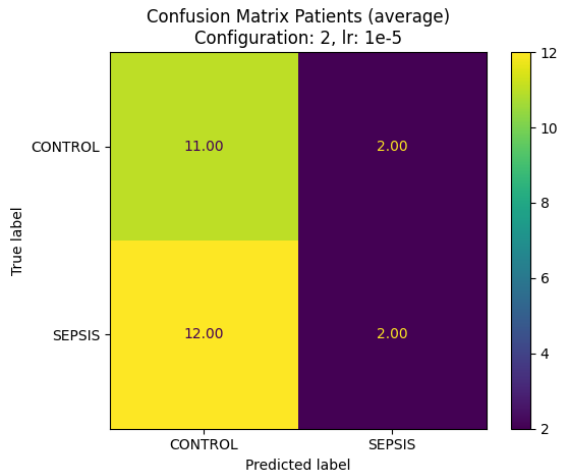


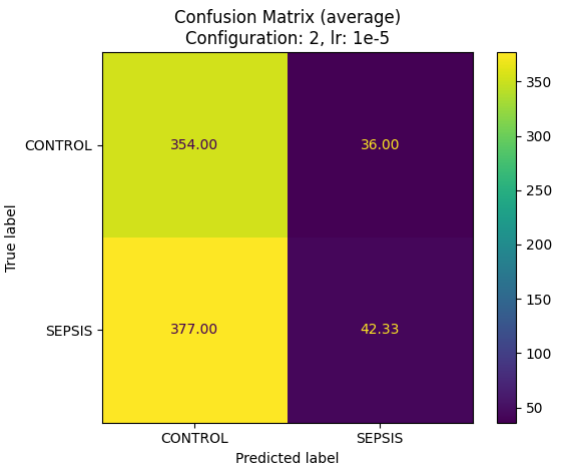
- outer_lstm_units = 128, inner_lstm_units = 64, learning_rate = 1e-4:



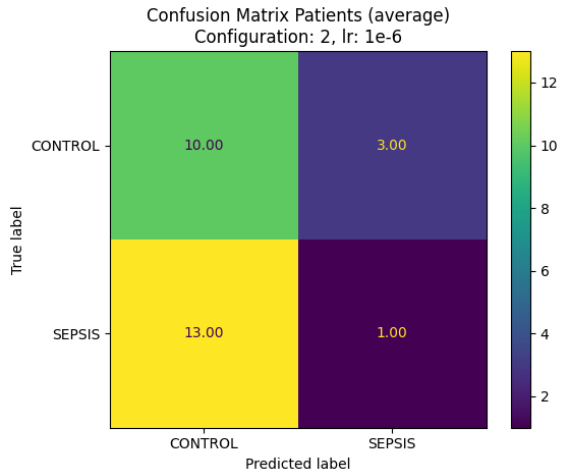


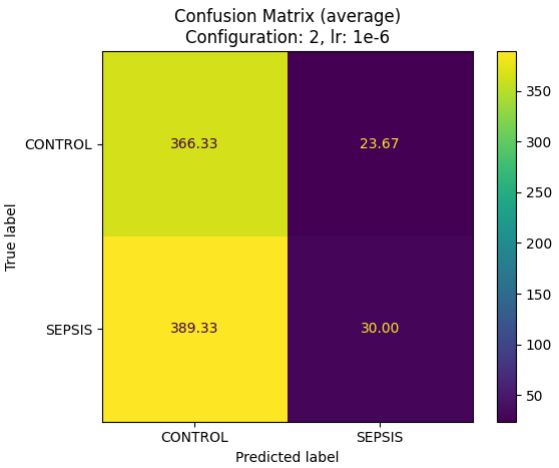
- outer_lstm_units = 128, inner_lstm_units = 64, learning_rate = 1e-5:



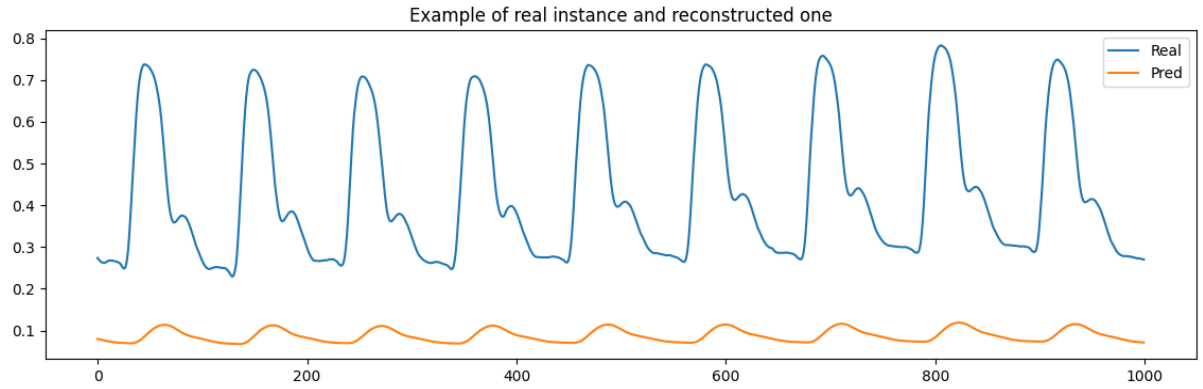


- `outer_lstm_units = 128, inner_lstm_units = 64, learning_rate = 1e-6:`





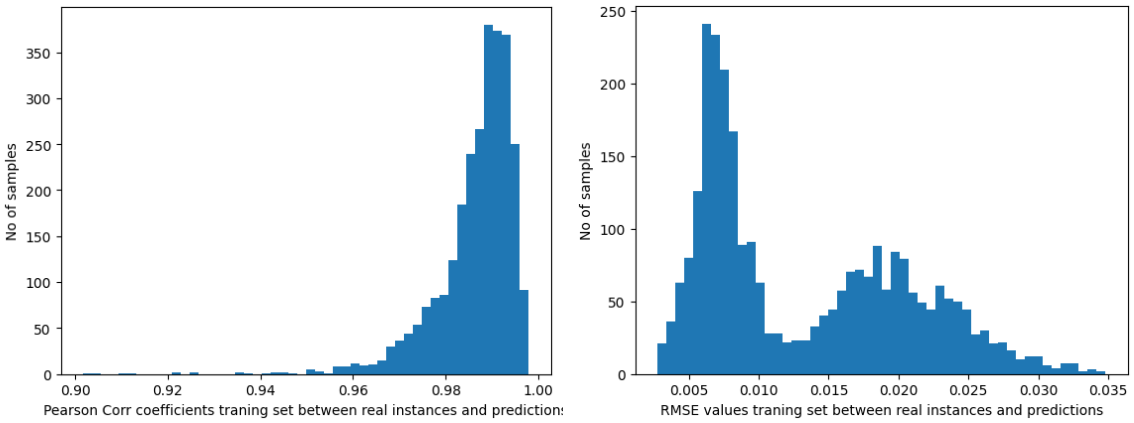
Come si può notare, a parità di performance per quanto riguarda le predizioni sui pazienti, la configurazione: `outer_lstm_units = 64, inner_lstm_units = 32, learning_rate = 1e-6` è la migliore nella predizione sulle singole finestre temporali.
La scelta fatta per identificare la miglior combinazione tiene però in considerazione esclusivamente le performance predittive, ma non di come queste sono ottenute. Non è ad esempio valutata la fedeltà di riproduzione dei vari segnali. Se, infatti, si va a prendere un esempio di segnale riprodotto dal modello addestrato con questa configurazione si ottiene un segnale ricostruito che non assomiglia minimamente a quello originale, come riportato nell'esempio.



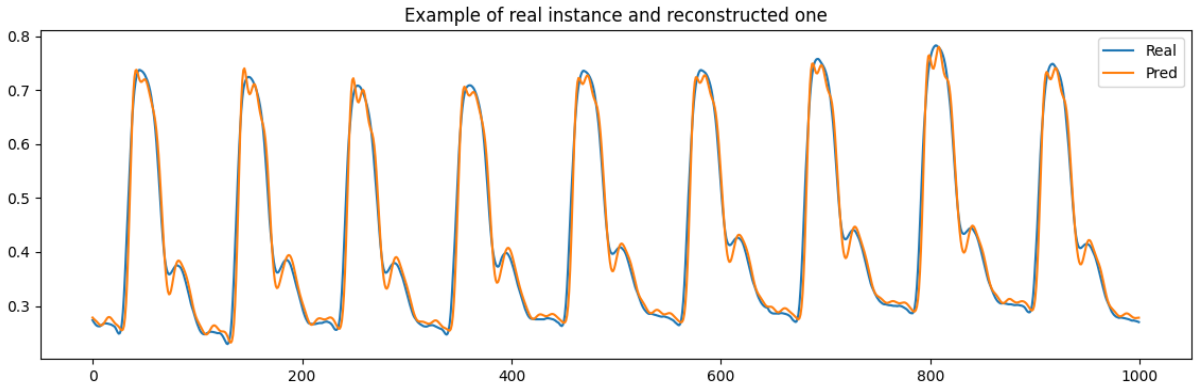
Per valutare quindi la miglior combinazione di parametri, sono state introdotte ulteriori due metriche che indicano il livello di fedeltà di riproduzione del segnale, il **Coefficiente di correlazione di Pearson** e il **Root Mean Squared Error (RMSE)**, calcolate sulle predizioni effettuate sul training set.
Tenendo in considerazione anche queste metriche, il **modello che generalmente si comporta meglio è quello avente questi parametri:**

- **`outer_lstm_units = 128,`**
- **`inner_lstm_units = 64,`**
- **`learning_rate = 1e-4.`**

Addestrando il modello utilizzando questi valori si ottengono infatti le seguenti *distribuzioni di valori di coefficiente di correlazione di Pearson ed RMSE sul training set*:

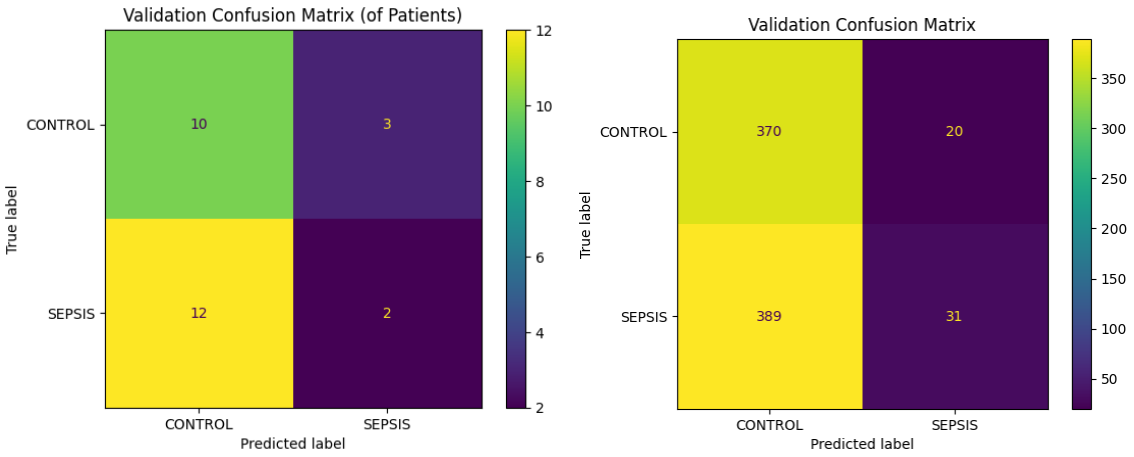


Questo dimostra una ricostruzione ottimale dei segnali in ingresso. La maggior parte dei segnali ha infatti un *valore di correlazione di Pearson molto vicino ad 1* (valore ottimale di questa metrica) mentre un *valore di RMSE molto vicino a 0* (valore ottimale).
Si riporta infatti un esempio di ricostruzione di segnale.



4.3. Performance della configurazione scelta sul validation set

Si riportano di seguito le matrici di confusione ed alcune metriche di performance (accuracy, recall, precision, score F1) ottenute sul validation set addestrando il modello utilizzando la combinazione di parametri ottimale illustrata nel paragrafo precedente.

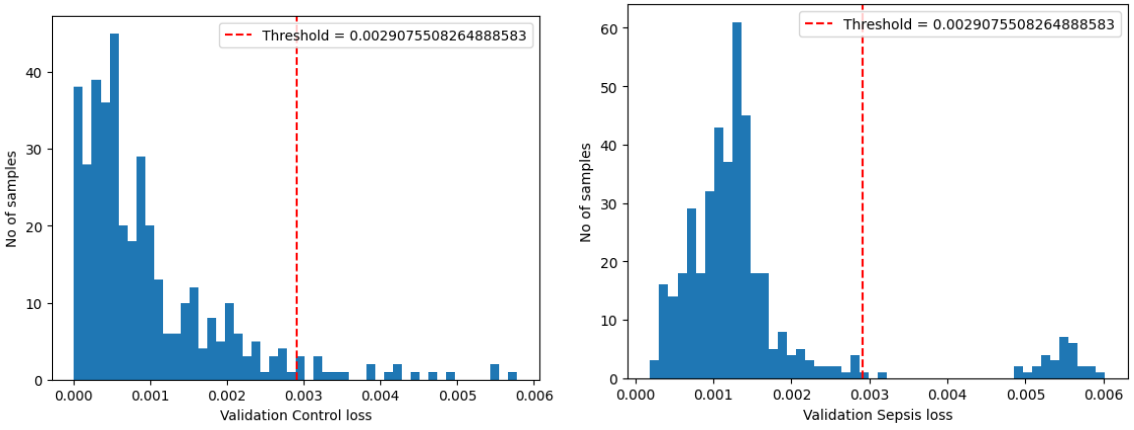


accuracy validation set: **0.49**506172839506174
precision validation set: **0.60**78431372549019
recall validation set: **0.07**380952380952381
F1 score validation set: **0.13**163481953290873

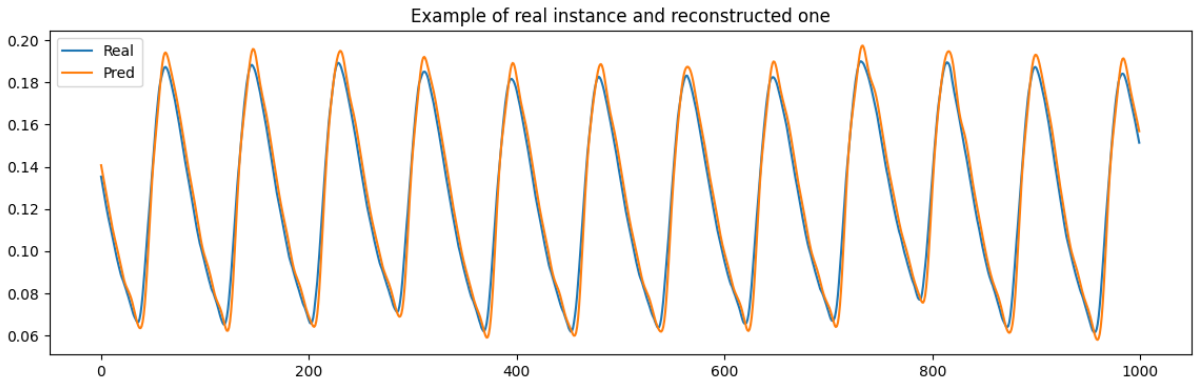
accuracy validation set **(of patients): 0.44**44444444444444
precision validation set **(of patients): 0.4**
recall validation set **(of patients): 0.14**285714285714285
F1 score validation set **(of patients): 0.21**052631578947364

sepsis patients (validation set) are recognized with an average **anticipation of 29 time windows**

Come si può notare dalle matrici di confusione e dagli indicatori di performance, il modello non riesce a riconoscere come dovrebbe i segnali di sepsi. Si riportano i grafici delle distribuzioni dei valori di loss delle istanze di controllo e delle istanze sepsi dove si può infatti notare che non vi è particolare differenza tra i due.



Di seguito è mostrato un esempio di ricostruzione di un segnale di sepsi da parte del modello dove si può infatti notare che, **nonostante non abbia mai visto segnali di sepsi durante l'addestramento, il modello riesce a riprodurre fedelmente anche i segnali di tale classe.**

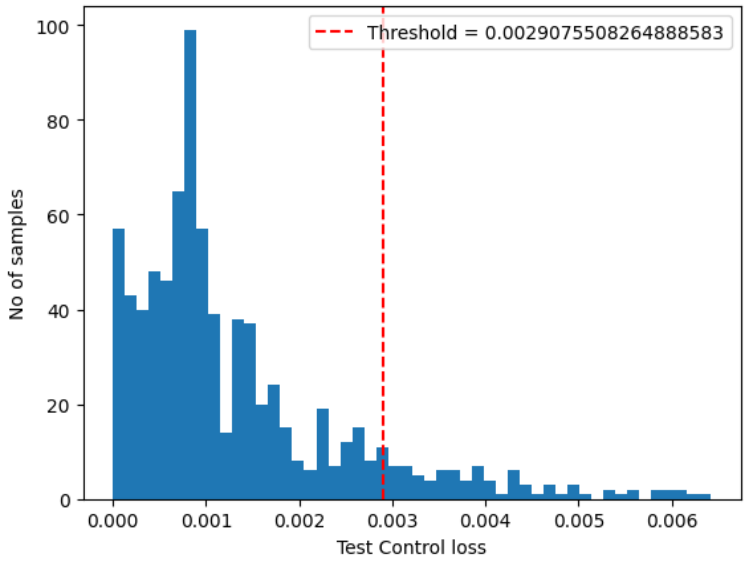


5. Risultati ottenuti sul test set

Una volta scelta la configurazione definitiva dei parametri del modello sono state valutate le sue **performance sul test set**.
Queste sono le **predizioni corrette sulle istanze di controllo** (ottenute considerando ogni finestra temporale indipendente):

`Correct control predictions` `on test set:` **`723/810`**

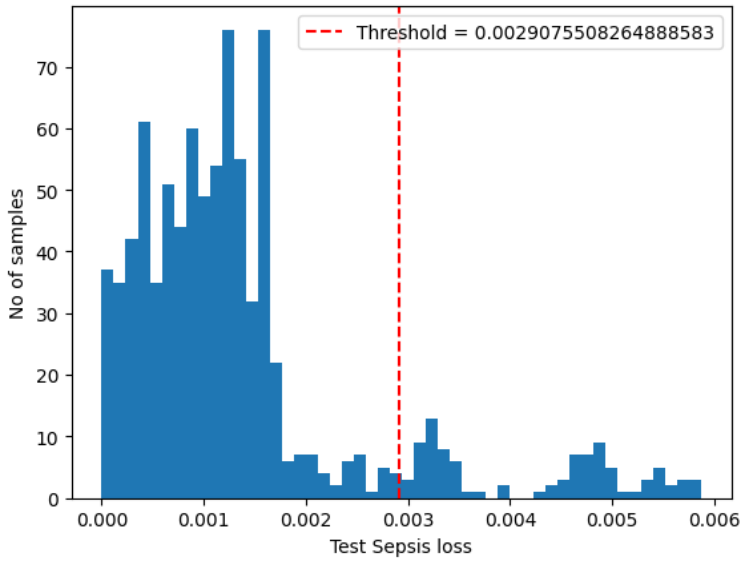
di cui si riporta anche la distribuzione dei valori di loss:



E queste le **predizioni corrette sulle istanze di sepsi:**

`Correct sepsis predictions` `on test set:` **`95/873`**

dove anche in questo caso si può osservare nel grafico sottostante la distribuzione dei valori di loss ottenuta:



Sono state quindi successivamente calcolate le **matrici di confusione**, *sia considerando le finestre temporali indipendenti*, *sia effettuando una singola predizione per paziente (sano/malato)* raggruppando le istanze che gli appartenevano.

Si riportano quindi accuracy, precision, recall e score F1 calcolate **considerando** tutte le **finestre indipendenti:**

accuracy test set: **0.48**60368389780155
precision test set: **0.52**1978021978022
recall test set: **0.10**882016036655212
F1 score test set: **0.18**009478672985785

e le stesse metriche ma **considerando i singoli pazienti:**
accuracy test set (of patients): **0.42**105263157894735
precision test set (of patients): **0.4**
recall test set (of patients): **0.2**
F1 score test set (of patients): **0.26**666666666666666

Si riporta infine **l'anticipazione**, in termini di numero di finestre temporali, **con la quale è stata identificata la sepsi nei pazienti effettivamente malati**:

```
sepsis patients (test set) are recognized with an average anticipation
of 23 time windows.
```

Questo indica che, nei pazienti in cui è stata effettivamente riconosciuta correttamente la malattia, il modello riesce ad identificare la sepsi con un'anticipazione media di 23 finestre temporali, che equivalgono a:

23 * 120 (durata finestre temporali) / 60 (durata in secondi di un minuto) =

46 minuti di anticipo.

Questo dato è indicativo del fatto che, nonostante il modello non riesca a riconoscere adeguatamente l'avvento della malattia, nei pazienti in cui effettivamente ci riesce lo fa utilizzando poche finestre temporali e quindi con largo anticipo rispetto al numero totale di quelle presenti nel dataset di quel paziente.

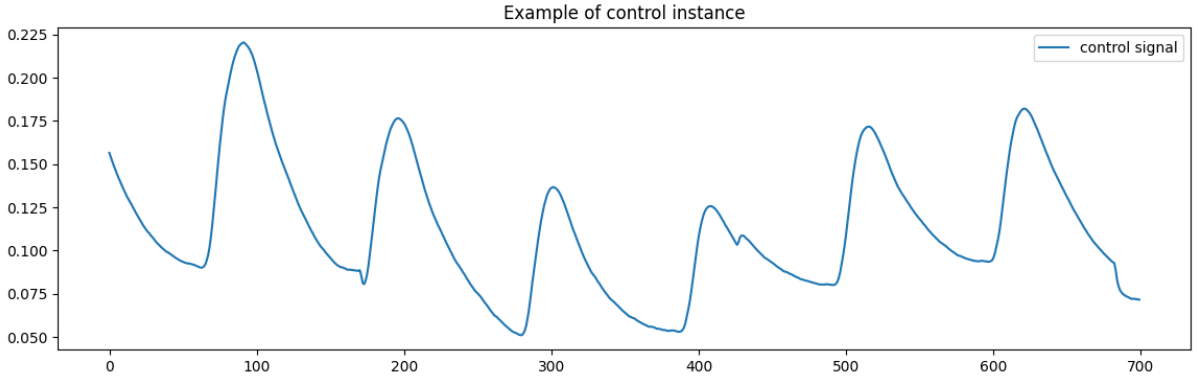
6. Conclusioni

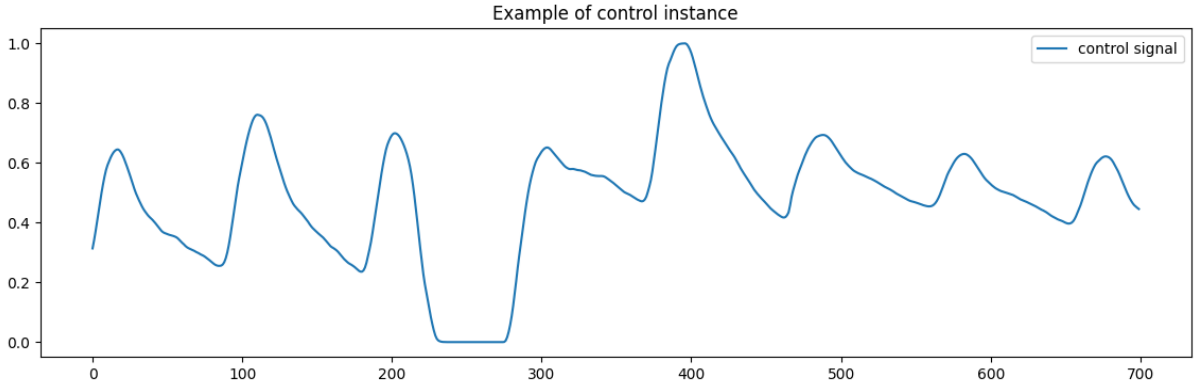
Riassumendo, in questo progetto si è cercato di **verificare le performance di un modello Autoencoder costituito da alcuni layer LSTM nello svolgimento di un task di anomaly detection al fine di riconoscere correttamente la malattia sepsi utilizzando i segnali PPG di alcuni pazienti**.
Come si può notare dai risultati ottenuti, le **performance sulla label di sepsi risultano non soddisfacenti**.
Nonostante il modello sia stato addestrato esclusivamente sulle istanze di controllo e riesca inoltre a ricostruire bene le finestre temporali appartenenti a questa classe in fase di predizione, si può notare come i valori di loss ottenuti quando si chiede al modello di predire istanze di sepsi sono pressoché simili a quelli ottenuti sui dati di controllo.
Da questo fatto si può dedurre che, probabilmente, il **modello utilizzato non è ottimale per il task in analisi**.
Nella sezione successiva tuttavia si cercherà di dare una spiegazione alle performance menzionate, illustrando alcune strategie che si potrebbero seguire per provare ad ottenere risultati migliori.

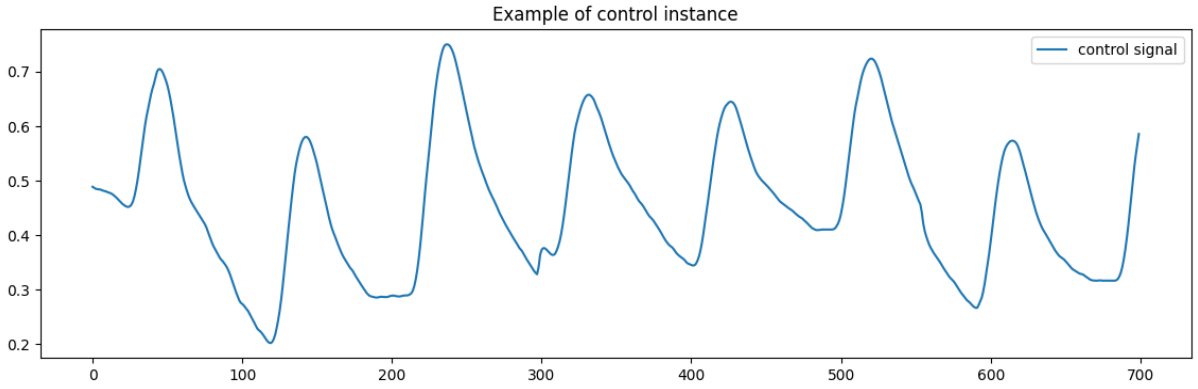
6.1. Sviluppi futuri

Analizzando manualmente le finestre temporali del dataset utilizzato si può notare che i **dati presenti in esso non sono certamente ottimali.**
Il *dispositivo utilizzato per la rilevazione dei segnali PPG* infatti non fornisce di per sè dati molto buoni per una successiva analisi a causa della *facilità con cui si presenta rumore* o errori di misurazione in essi. Infatti è sufficiente che il paziente si muova leggermente durante il processo per fare in modo che la finestra temporale risulti decisamente compromessa.
Inoltre ai dati utilizzati sono state applicate tecniche per la gestione dei valori mancanti e per la riduzione del rumore.

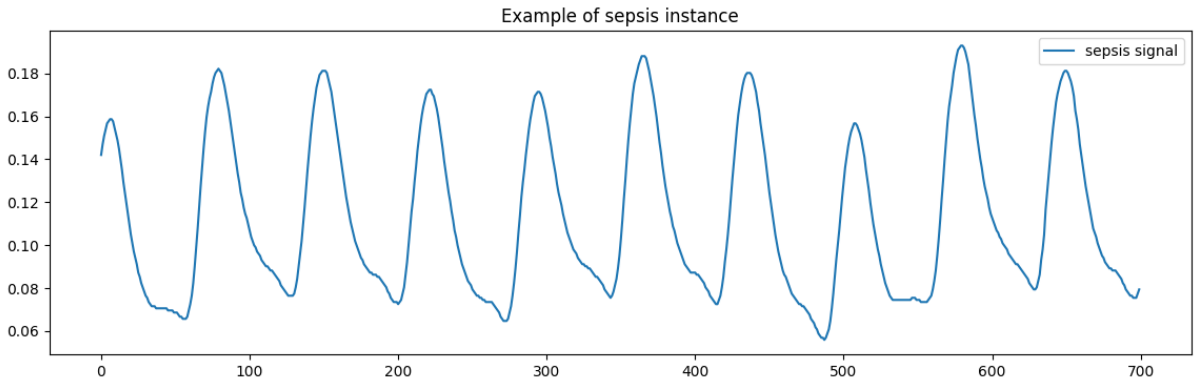
Sommando tutti questi fattori, si ottiene che **molti dati di controllo sono tutto fuorché segnali PPG eccellenti di un paziente sano** e questo potrebbe aver quindi compromesso la fase di addestramento.

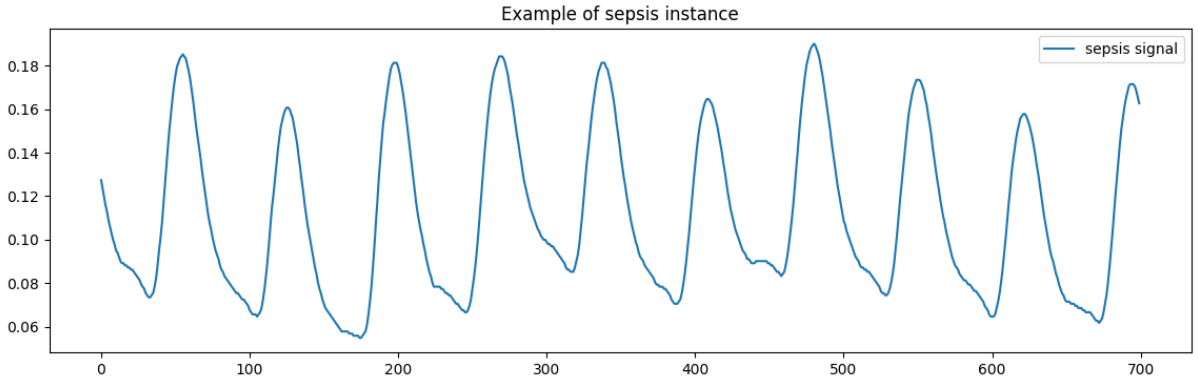


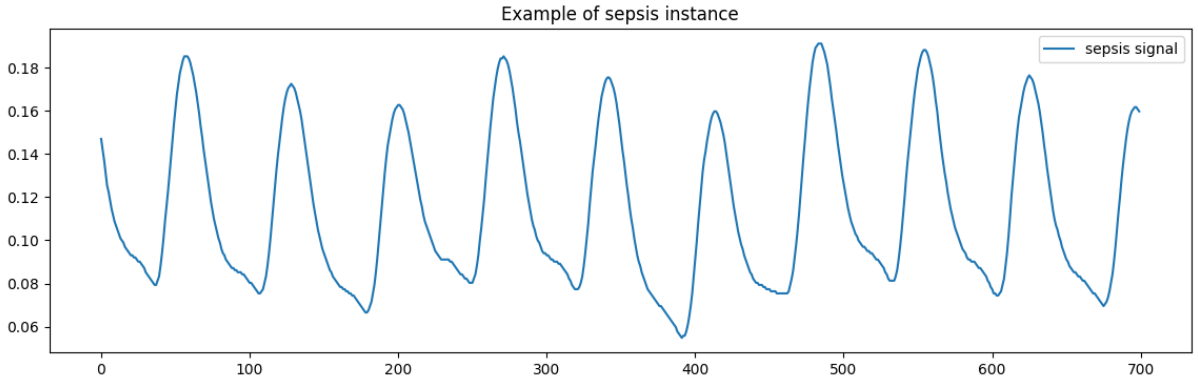




Al contrario invece sono presenti **istanze della classe di sepsi che presentano segnali PPG tipici di un paziente che non presenta la suddetta malattia** e che spiegano quindi i valori di loss così bassi su questa label.







Ricordando infatti che i **dati utilizzati sono stati presi da una versione ridotta del MIMIC-III Waveform Database** a cui successivamente sono **anche state eliminate alcune righe**, è possibile che di alcuni pazienti che hanno sviluppato la malattia sepsi non siano nemmeno presenti le finestre temporali in cui questa si è manifestata.
Sicuramente quindi una prima possibile metodologia per tentare di migliorare le performance sarebbe quella di **utilizzare un diverso dataset**, **o** di **prendere in analisi la sua versione completa e prestare successivamente particolare attenzione alle righe che vengono eliminate ed alle tecniche di preprocessing utilizzate**.

Nel caso in cui questa metodologia riveli performance decisamente più accettabili, bisognerebbe rivedere tutti gli iperparametri selezionati. Date le scarse performance riscontrate ed i limiti di utilizzo di Colab, non è infatti stata svolta una ricerca esaustiva di tutti i parametri del modello. In un eventuale aumento delle performance bisognerebbe, quindi, **sottoporre un maggior numero di parametri alla tecnica grid search** per trovare la migliore combinazione degli stessi.