DEEP LEARNING

PARKINSON DISEASE: OFF-PERIOD PATTERN ANALYSIS

LUCA LABOCCETTA
GIORGIO ANDRONICO

Indice

1 Introduzione

1.1 Background

La malattia di Parkinson è il secondo disturbo neurologico più comune che causa una disabilità significativa e riduce la qualità della vita e non ha una cura.
Circa il 90% delle persone affette da Parkinson presenta disturbi del linguaggio. Attualmente non esiste una cura definitiva per questa malattia, ma esistono farmaci, come la *levodopa*, che possono alleviare i sintomi. L'effetto di questo farmaco, tuttavia, non è del tutto preciso e affidabile. Ci sono dei periodi di tempo in cui il paziente non è più coperto dall'effetto del farmaco che vengono chiamati periodi di sospensione, o *off-periods*.
I periodi di sospensione comprendono sia sintomi motori, come tremori e rigidità, sia sintomi sinottici non motori, come l'ansia. Quando si monitora un paziente, è importante essere in grado di identificare questi periodi off per pianificare il trattamento in modo diverso. Identificare questi periodi non è facile.
Nel corso degli anni sono state cercate diverse soluzioni. Una di queste prevede l'uso di un orologio in grado di monitorare alcuni parametri del paziente. Il punto debole di questa soluzione risiede nel fatto che i periodi di riposo dovrebbero teoricamente essere segnalati manualmente. Questa procedura è inaffidabile perché spesso il paziente, in preda ai sintomi, non è nemmeno in grado di effettuare la segnalazione. Per non parlare del fatto che i pazienti spesso rimuovono il dispositivo. Per migliorare questa soluzione sarebbe necessario che il dispositivo sia in grado, sulla base dei parametri monitorati, di individuare autonomamente i periodi di spegnimento.

1.2 Presentazione Dataset

I dati raccolti provengono dall'azienda multinazionale *Healthware Group*. Questa compagnia ci ha permesso di utilizzare i dati con l'obiettivo di realizzare questo progetto. I dati sono stati campionati da un insieme eterogeneo di pazienti così suddiviso:

- 3 Pazienti in stato di Parkinson avanzato
- 24 Pazienti affetti da Parkinson
- 13 Pazienti senza Parkinson

Per ogni paziente i dati a disposizione sono:

- Coordinate spaziali di movimento di un accelerometro legato al paziente (colonne *x, y, z*)
- Battito cardiaco del paziente
- Istante di rilevazione

1.2.1 Dati usati per ciascun task

La versione del dataset per ogni task è differente. Sostanzialmente, entrambi consistono di time-series che risultano utili per risolvere i due differenti task.

- **Task 1**: In questo task il dataset è composto da tre coordinate spaziali: x,y,z. Sono le coordinate dell'accelerometro già introdotto in precedenza.
- **Task 2**: Qui il dataset presenta coordinate spaziali, battito cardiaco, *timestamp* che rappresenta il momento in cui sono stati raccolti i dati e dall'ID del paziente, per un totale di 18 pazienti.

1.3 Obiettivo finale

Gli obiettivi da raggiungere sono, principalmente, due.

- **Task 1**: Previsione del valore successivo
- **Task 2**: Rilevazione delle anomalie

1.3.1 Task 1 – Previsione del valore successivo

Il primo task si occupa di prevedere il valore successivo di una data sequenza temporale, facendo a monte un *windowing* del dataset.
Il numero di valori da prendere in considerazione è uno dei parametri che viene specificato durante la fase di sviluppo, insieme a quanti valori bisogna saltare per effettuare la predizione successiva.

1.3.2 Task 2 – Rilevazione degli off-period

Il task di rilevazione delle anomalie ha come fine ultimo quello di saper rilevare, in base ai dati registrati dal dispositivo medico, quando un paziente sta attraversando un *off-period*, ovvero un momento in cui la medicazione contro il Parkinson non sta agendo correttamente, con conseguente ritorno dei tremori.

2. Task 1

2.1 Data understanding

In questo insieme di dati ogni riga rappresenta un valore registrato ogni 10 secondi.
Il training set è composto dai seguenti dati:

- 144911 righe
- 3 colonne
- 434733 dati in totale

I seguenti numeri indicano il numero di righe rispettivamente di Test e Training.

Training set	Test set
144912	148372

Figura 1: Dimensione Dataset

In *fig. 2* è possibile visualizzare una porzione esemplare di dati del training set.

X	Y	Z
-24	749	-626
-206	930	-63
-139	763	-577
-503	441	-557
-278	705	-396
240	839	-310
-671	318	-213
-45	296	-927
102	294	-888
15	635	-671

Figura 2: campione di dati dal training set

Le celle sono riempite da soli valori interi e non è stato necessario effettuare preprocessing per migliorare la qualità del dato, in quanto non erano presenti valori nulli. In *fig. 3* vi è l'evidenza di ciò.

	Training Set	Test Set
X	0	0
Y	0	0
Z	0	0

Fig. 3: numero di valori nulli per ciascuna colonna

I dati a disposizione indicano la posizione del dispositivo nello spazio, tramite i valori delle coordinate x, y, e z.

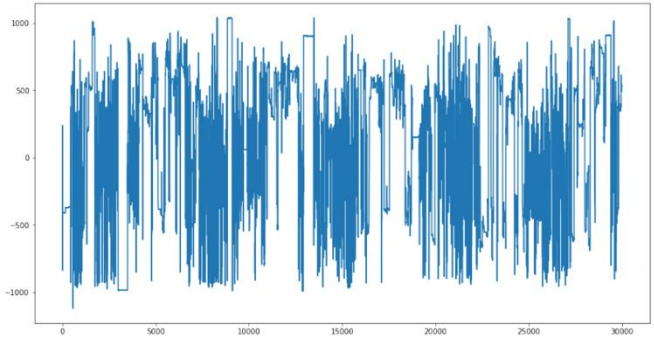


Fig. 4: valori colonna X

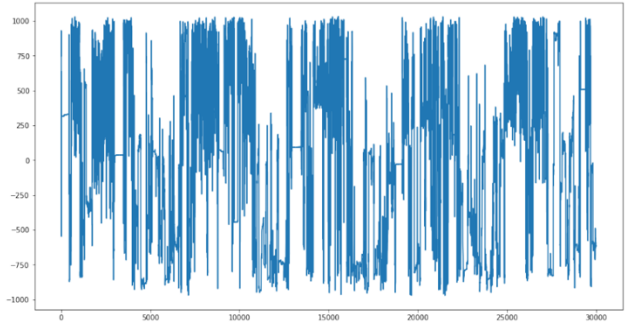


Fig. 5: valori colonna Y

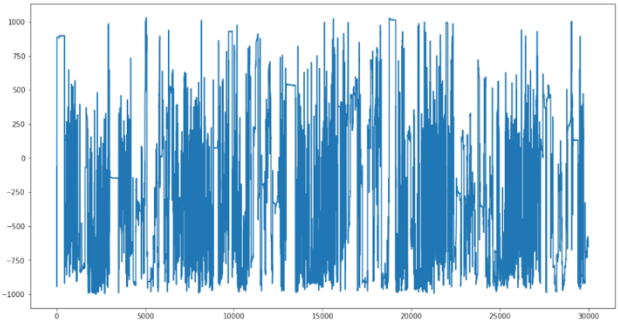


Fig. 6: valori colonna Z

2.2 Data Preparation

2.2.1 Normalizzazione

È stata effettuata una normalizzazione di ciascuna colonna del dataset in modo da ottenere valori più piccoli, ridurre le varie feature sulla stessa scala 0-1, e rendere la fase di training più efficiente. Per normalizzare i dati, la media viene sottratta da ogni dato e divisa dalla deviazione standard.

$$\frac{trainData - mean}{std}$$

Fig. 7: formula di normalizzazione impiegata

Ogni riga del dataset rappresenta una rilevazione, e come menzionato sopra, in questo dataset le rilevazioni vengono fatte **ogni dieci secondi**.
Ciò che è stato fatto in fase di preparazione di dati, è stato di creare una funzione chiamata *generate_windows,* con l'obiettivo di creare delle sequenze.
I parametri determinano la dimensione della finestra da creare, e di quanto spostarsi all'interno del dataset; sono riportati di seguito.

- *window_size*: Lunghezza di ogni sequenza
- *window_shift*: Quante posizioni saltare per la creazione della sequenza successiva

In particolare, scegliendo *window_size = 30* e *window_shift = 6*, ogni finestra raccoglierà **(30*10) / 60 = 5** minuti di rilevazioni per ogni **(6*10 = 60)** minuto successivo. In altre parole, la funzione quindi raccoglie i dati relativi a 30 righe spostandosi all'interno del dataset di 6 righe alla volta. Questo, naturalmente, genera ridondanza, ma è un effetto atteso.
L' operazione è stata applicata sia sul file .csv di test che su quello di train, che a sua volta è stato diviso per il 70% in "actual" training set e il restante 30% in validation set.

2.3 Modelling e training

Questa è la fase principale di ogni attività di Deep Learning. Modellare significa trovare la miglior architettura per costruire il modello, in modo da avere i risultati migliori per le analisi da effettuare. L'allenamento del modello sarà fatto per **tutte e 3** le dimensioni spaziali considerate *(x,y,z)*, divisi in tre file *.ipynb* separati per comodità. Il modello sarà di tipo sequenziale con i seguenti layer:

- Un layer **LSTM** (Long Short Term Memory), un particolare tipo di Recurrent Neural Network (RNN), in grado di imparare relazioni fra i dati per lunghi periodi di tempo (sequenze).
    - In particolare, questo layer è utile perché per predire uno step successivo, bisogna guardare a quelli passati.
    - Il numero di celle del layer LSTM è derivato da una ricerca iperparametrica applicata tramite ***Keras Tuner.***
- Un layer **Dense** con un solo nodo, in quanto vogliamo predire un solo elemento successivo ad ogni sequenza.

```
model = Sequential()
LSTM_1_units = hp.Int('LSTM_1_units', min_value=16, max_value=128, step=16)
dropout_rate = 0.1
model.add(LSTM(LSTM_1_units,input_shape=(window_size, X_train.shape[2])))
model.add(Dense(1))
```

Fig. 8: struttura del modello in Keras

Da come si evince in *Fig.* 8, durante la ricerca del numero ottimale di *units* del primo layer, il numero effettivo può variare da 16 a 128 con una step-size di 16. Abbiamo osservato che la *step-size* uguale a 16 fosse lo "sweet spot", in quanto size ulteriormente più basse o alte portano a un maggior numero di ricerche, e in ultimo, un minore tasso di efficienza. Come *loss function* viene utilizzata la **Mean Absolute Error (MAE).**
L'ottimizzatore scelto è "Adam" e il parametro di learning rate varia tra 1e-2 ,1e-3 e 1e-4.
Il Keras Tuner lancerà una ricerca, facendo multipli fit dello stesso modello usando valori diversi di learning rate e units del layer di LSTM. L'output sarà un insieme di iperparametri ottimali su cui poi verrà allenato il modello finale.

Il modello viene allenato utilizzando i parametri ottimi per 150 epoche. Successivamente viene scelto come numero di epoche ottime quella in cui la *validation loss* ha raggiunto il valore più piccolo. Per fare inferenza, viene successivamente scelto il set di pesi relativi all'epoca in cui si riscontra il minimo valore di *validation loss*. Il risultato della fase di addestramento per ogni variabile è riportato nei grafici di seguito.

Fig. 9: loss per colonna x

Fig. 10: loss per colonna y

Fig. 11: loss per colonna z

Dal grafico sopra raffigurato è possibile notare come:

- Il miglioramento della *training loss* è costante e uniforme
- La *validation loss* ha una linea meno uniforme, ma la tendenza è una diminuzione costante.
    - Inoltre, i valori sono inferiori rispetto a quelli della training loss, ciò vuol dire che non è presente l'overfitting.

2.4 Evaluation

In quest'ultima fase viene effettuata un'operazione di inferenza sul *test set*. In particolare, valutiamo quanto è "bravo" il modello a eseguire *next value prediction* su dati che non ha mai visto. Così come l'allenamento è stato effettuato per tutte e tre le dimensioni spaziali prese singolarmente (x, y, z), anche l'inferenza verrà effettuata seguendo questo schema.
Ovviamente, come fatto nella fase precedente, i dati di test verranno normalizzati e divisi a sequenze utilizzando lo stesso metodo di *windowing* descritto nella sezione 2.2.1. Di seguito i valori di Mean Absolute Error ottenuti per ogni feature con la rispettiva baseline fornita. E' possibile denotare come i valori di MAE, per ciascuna delle colonne, siano ben al di sotto delle soglie da rispettare.

	MAE	BASELINE
X	73.57	81.06
Y	80.22	85.26
Z	73.06	79.94

Fig. 12: Valori di MAE e baselines

2.5 Task 1.2

L'obiettivo di questo task è quello di analizzare come cambiano le metriche di prestazione al variare dei parametri *window_size* e *window_shift*.
Il dataset utilizzato è quello del Task 1, e lo è anche la fase di preparazione dei dati. In questa fase non è stata effettuata la ricerca automatica degli iperparametri, bensì è stato creato un modello con valori predeterminati. I valori scelti sono quelli ottenuti dalla ricerca iperparametrica effettuata nel Task 1, e lo stesso vale per il numero di epoche.

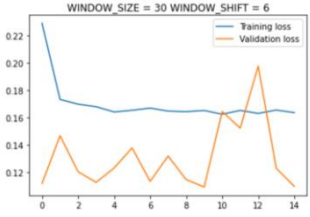
```
model = Sequential()
model.add(LSTM(128, return_sequences = True, input_shape = (seq_len, num_feat)))
model.add(LSTM(64))
model.add(Dense(32, activation = 'tanh'))
model.add(Dropout(0.1))
model.add(Dense(16, activation = 'tanh'))
model.add(Dropout(0.1))
model.add(Dense(1))
```

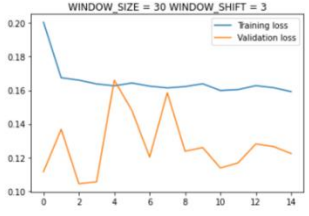
Fig. 13: Struttura in Keras del modello per 1.2

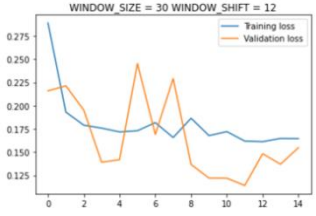
Il modello è stato addestrato con le seguenti configurazioni di *window_size* e *window_shift*:

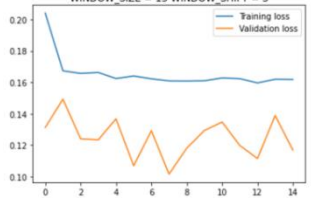
- *window_shift = 3, window_size = 15*
- *window_shift = 3, window_size = 30*
- *window_shift = 3, window_size = 60*
- *window_shift = 6, window_size = 15*
- *window_shift = 6, window_size = 30*
- *window_shift = 6, window_size = 60*
- *window_shift = 12, window_size = 15*
- *window_shift = 12, window_size = 30*
- *window_shift = 12, window_size = 60*

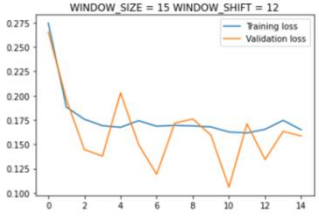
Di seguito sono riportati i risultati ottenuti in plot.

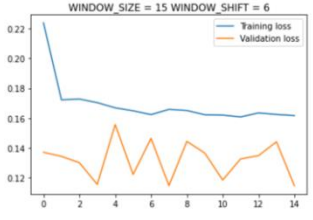


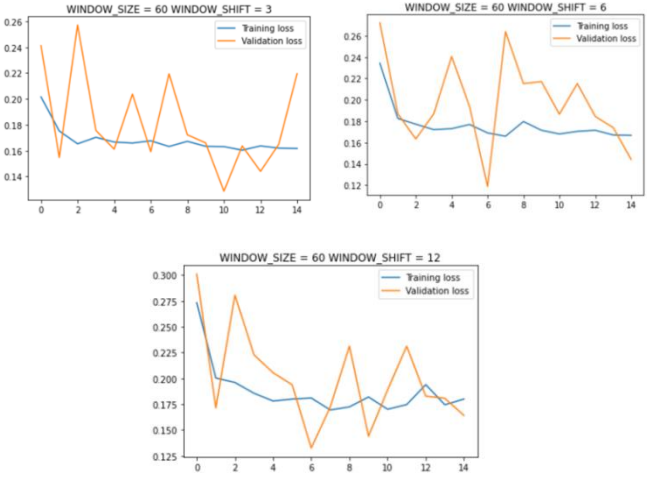












In *Fig.* 14 è possibile visualizzare le combinazioni migliori mostrando come varia il Mean Absolute Error.

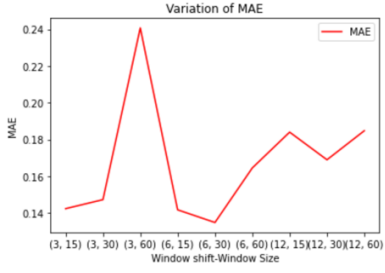


Fig. 14: Variazione della MAE

Dal grafico si evince che la combinazione migliore risulta essere quella con *window_shift = 6* e *window_size = 30*.
Tuttavia, i modelli sono stati allenati con sole 15 epoche e i parametri non sono stati scelti utilizzando l'hypertuner. Per questo motivo, molti modelli tendono ad avere overfitting o apprendimento non costante.

3 Task 2 – Anomaly Detection

3.1 Data Understanding

I dati dei pazienti che sono stati forniti per questo task sono stati divisi in due file, “ad_train.csv” e “ad_test.csv”. Il training set contiene i dati di pazienti non affetti dalla sindrome di Parkinson mentre il test set contiene dati dei pazienti affetti dalla sindrome di Parkinson. Nella seguente tabelle possiamo visualizzare le informazioni del training set:

Feature	Descrizione
Paziente	Identificativo del paziente
X	Accelerometro letto in x
Y	Accelerometro letto in y
Z	Accelerometro letto in z
Heart Rate	Battito cardiaco
Timestamp	Timestamp in cui sono state effettuate le letture
tsDate	Data in cui sono sono state effettuate le letture

Fig. 15: descrizione colonne

Per quanto riguarda la fase di controllo qualità dei dati, la problematica principale che risulta è una. In alcune righe la colonna *HeartRate* presenta il valore -1, indicato da *Healthware* stessa come un modo per evidenziare un dato mancante.

Feature	Training Set	Test Set
Paziente	0	0
X	0	0
Y	0	0
Z	0	0
Heart Rate	39029	0
Timestamp	0	0
tsDate	0	0

Fig. 16: valori nulli

Per fornire un campione dei dati, nell'immagine sottostante è possibile visualizzare l'andamento del battito cardiaco per il paziente 4504 e 4505. Si noti come alcune larghe zone del grafico presentano un valore costante a -1. Questo fa capire come i valori nulli non siano distribuiti in maniera del tutto casuale nel dataset, bensì si presentino "a gruppi".

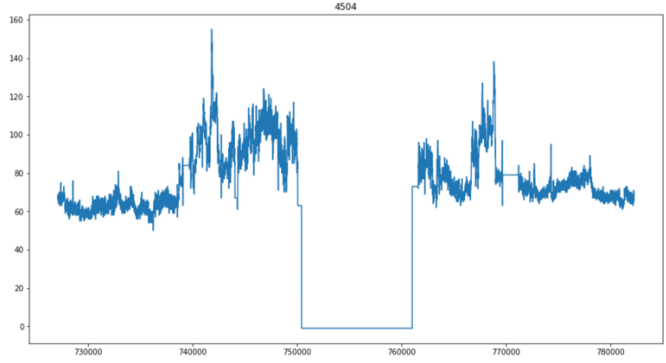


Fig. 17: battito cardiaco paziente 4504

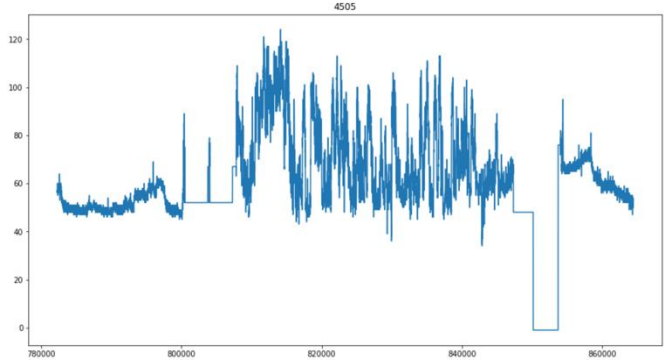


Fig. 18: battito cardiaco paziente 4505

Il motivo per cui è presente questa anomalia di dato non è noto, ma si può suppore che sia dovuto ad un malfunzionamento del dispositivo di acquisizione. Ad ogni modo, per ovviare al problema, le righe che presentavano queste anomalie sono state rimosse.
Analizzando nel training set la colonna relativa all'identificativo del paziente, ne sono stati rilevati 13, ciascuno con un numero piuttosto grande di righe. Questi pazienti, come dettagliato sopra, non soffrono del morbo di Parkinson.

ID Paziente	Numero di righe
1502	85244
1503	82640
1505	84171
1507	79815
1509	72130
3506	46527
3507	62583
3508	85968
3509	45673
4502	82361
4504	55153
4505	82184
4506	79073

Fig.19: Rilevazioni per paziente

Un'analisi è stata effettuata anche sulla coordinata spaziale x, come per il Task 1. Di seguito è riportata la rappresentazione grafica della coordinata x per il paziente 1503, per fornire una visione panoramica dei dati.

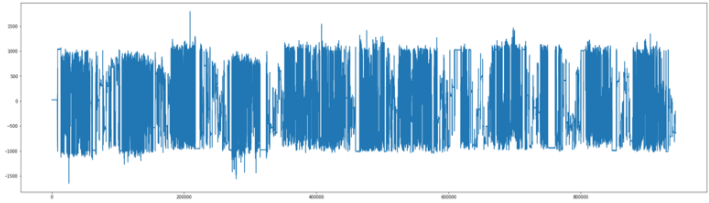


Fig. 20: colonna X paziente 1503

Come per il Task 1, anche qui è possibile notare come ci siano dei valori costanti della coordinata x per un lasso di tempo. Una possibile spiegazione è analoga a quella data in precedenza, ovvero che il paziente stava dormendo oppure il dispositivo non stava funzionando correttamente.

3.2 Data Preparation

3.2.1 Data Cleaning

Come spiegato precedentemente, sono state eliminate le righe in cui il battito cardiaco presenta valore uguale a -1:

```
train = train[train.heartRate != -1]
```

Fig. 21: frammento di codice rimozione

Successivamente la colonna tsDate è stata convertita nel formato *datetime64[s]*. Questo è propedeutico alla fase di resampling. Infatti, nel training set sono registrati dati acquisiti secondo uno schema di una rilevazione al secondo, mentre nel test set, viene effettuata una rilevazione solo ogni dieci secondi. Per omologare la natura dei dati di train e test, ogni gruppo di dieci righe consecutive viene sostituito da una riga singola, rappresentante la media delle osservazioni in quei dieci secondi.

```
train.tsDate = train.tsDate.astype('datetime64[s]')

train = train.groupby(['patient', pd.Grouper(freq='10s', key='tsDate')]).mean()
train.reset_index(level=0, inplace=True)
train.reset_index(level=0, inplace=True)
```

Fig. 22: frammento di codice resampling

3.2.2 Normalizzazione

Come nel task 1, anche qui i dati sono stati normalizzati in modo da migliorare l'efficienza del processo di allenamento.

```
def normalize_df(df):
  normalized_df = df.copy()
  for feature_name in ['x', 'y', 'z', 'heartRate']:
    col_mean = normalized_df[feature_name].mean(axis = 0)
    col_std = normalized_df[feature_name].std(axis = 0)
    normalized_df[feature_name] = (normalized_df[feature_name] - col_mean) / col_std
  return normalized_df
```

Fig. 23: frammento di codice normalizzazione

3.2.3 Generazione sequenze e creazione validation set

La funzione di generazione delle finestre nel Task 2 segue la scia della funzione usata nel Task 1, ma con delle sostanziali differenze. La prima operazione che viene effettuata è quella di suddividere il dataset in porzioni disgiunte, ognuna delle quali è relativa ad un singolo paziente. Questo è stato fatto per evitare che, dando in input alla funzione di windowing il dataset per intero, venissero create finestre che andassero a mischiare dati di pazienti diversi. Avendo 13 pazienti nel nostro training set, scegliamo di costruire il validation set prendendo il 20% del training set. In particolare, il 20% di 13 è 2.6, quindi approssimando per eccesso otterremo tre pazienti nel validation set e 10 nel training set.

```
for i in range(0, len(train_patients)):
  if i < 10:
    train_windows.append(generate_windows(
        normalized_train[normalized_train.patient == train_patients[i]],
        window_size,
        window_shift)[1])
  else:
    val_windows.append(generate_windows(
        normalized_train[normalized_train.patient == train_patients[i]],
        window_size,
        window_shift)[1])
```

Fig. 24: frammento di codice windowing

Questo, però, genera un problema ulteriore: potrebbe succedere che alcune finestre siano più corte di altre, in quanto il numero di righe non è equamente distribuito fra

pazienti. Per esempio, per quanto riguarda il training set, 85422 righe sono relative al paziente *1502*, mentre 82640 sono relative al paziente *1503*.
In particolare, viene effettuata una operazione di padding per garantire che le sequenze generate siano tutte della stessa lunghezza.

Qui sotto è riportato un esempio fittizio di dataset di 227 righe. Immaginando di avere una *window_size* pari a 30, la finestra finale dovrebbe coprire gli indici da 200 a 227; si verrebbe a creare una finestra di 27 elementi e non di 30. Per ovviare a questo problema, gli ultimi 3 elementi della finestra vuoti vengono riempiti con una riga di padding che ha 0 come valore di ciascuna colonna.

...	i :224	i :225	i :226	i :227	out	out	out
...	r224	r225	r226	r227	x	x	x
...	r224	r225	r226	r227	row0	row0	row0

Fig. 25: esempio windowing

3.3 Modelling e training

Per questo tipo di task di apprendimento non supervisionato, l'architettura scelta è stata un **Autoencoder.**
Un autoencoder è un tipo di rete neurale artificiale usata per imparare codifiche efficienti di dati non etichettati. L'autoencoder si presta perfettamente al nostro caso d'uso, in quanto è in letteratura viene spesso utilizzato per il rilevamento delle anomalie.

Il modello è composto da due parti, una di encoding e una di decoding. La porzione di encoding si occupa di imparare una rappresentazione compressa dei dati in ingresso, mentre la parte di decoding si occupa di ricostruire il dato originale a partire dalla rappresentazione compressa. Quindi, la label in questo caso sarebbe il dato stesso.
In essenza, il modello impara a ricostruire il dato sempre meglio. Per questo task viene allenato a ricostruire sequenze temporali relative a pazienti non affetti da Parkinson, ovvero il gruppo di controllo. Ci aspettiamo quindi che, provando a fare inferenza su dati relativi a persone affette dal morbo di Parkinson (dati presenti nel test set), l'errore di ricostruzione su alcune finestre sia particolarmente alto, ad indicare la presenza di un dato anomalo.
Il modello che abbiamo realizzato è composto dalle seguenti caratteristiche:

```
model = Sequential()
LSTM_1_4_units = hp.Int('LSTM_1_4_units', min_value=64, max_value=128, step=32)
LSTM_2_3_units = hp.Int('LSTM_2_3_units', min_value=32, max_value=64, step=32)
dropout_rate = 0.2

model.add(Masking(mask_value=0.0,
                  input_shape=(windowed_train.shape[1],
                               windowed_train.shape[2])))
model.add(LSTM(LSTM_1_4_units,
               activation='tanh',
               return_sequences=True,
               input_shape=(windowed_train.shape[1],
                            windowed_train.shape[2])))
model.add(Dropout(rate=dropout_rate))
model.add(LSTM(LSTM_2_3_units,
               activation='tanh'))
model.add(RepeatVector(windowed_train.shape[1]))
model.add(LSTM(LSTM_2_3_units,
               activation='tanh',
               return_sequences=True))
model.add(Dropout(rate=dropout_rate))
model.add(LSTM(LSTM_1_4_units,
               activation='tanh',
               return_sequences=True))
model.add(TimeDistributed(Dense(windowed_train.shape[2])))
```

Fig. 26: struttura Keras autoencoder

Il modello è un **LSTM Autoencoder** composto da:

- Layer **Masking,** per ignorare le righe "finte" con valori 0 che abbiamo inserito nella fase di generazione delle sequenze.
- **Due** layer **LSTM** che formano la porzione di encoding
    - Il primo layer avrà un numero x di unità, mentre il secondo avrà un numero y, con y < x
- Layer **RepeatVector**, ripete l'input in arrivo un numero specifico di volte, nel nostro caso la dimensione della finestra.
    - Serve perché l'encoder comprime l'input in un vettore a singola feature, e per fare in modo che il decoder rigeneri l'input secondo la dimensione originale, il tensore va "riconvertito".
- **Due** layer **LSTM** che formano la porzione di decoding
    - Il terzo layer avrà un numero x di unità, mentre il quarto avrà un numero y, con y > x
- Layer **TimeDistributed**, un wrapper che consente di applicare un layer Dense a ogni finestra.

Questi layer sono intervallati da diversi Dropout, con un rate fisso pari a 0.2, usato per prevenire overfitting.
Non solo per il numero di nodi e del learning rate è stato utilizzato l'hypertuner, ma anche per il numero di epoche. Il modello viene allenato utilizzando i parametri ottimi per cento epoche.
Per fare inferenza viene scelto il set di pesi dell'epoca in cui abbiamo il minimo valore di *validation loss*, in questo caso 97. Il risultato della fase di addestramento è possibile visualizzarlo nell'immagine sottostante.

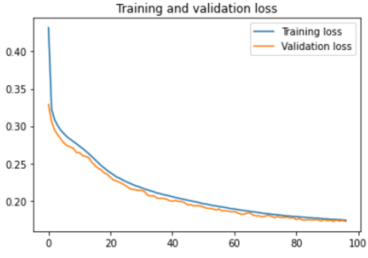


Fig. 27: grafico loss per autoencoder

Dalla figura è possibile vedere come il decremento delle loss è netto e costante.

3.4 Evaluation

Per la fase di test è stato utilizzato il dataset "ad_test.csv". I dati di test sono stati preparati e processati allo stesso modo di quelli di train; dunque, si può far riferimento alla sezione 3.2, eccetto per la fase di raggruppamento dei dati ogni 10 secondi. Infatti, la frequenza della rilevazione per il test set è già di dieci secondi.

Dopo aver effettuato la generazione delle finestre anche per il dataset di test, è stata effettuata l'operazione di inferenza, tramite la funzione *predict* messa a disposizione da Keras.

```
predictions = model.predict(windowed_test)
```

Fig. 28: frammento codice predictions

Successivamente è stato calcolato il Mean Absolute Error per ogni finestra generata nel dataset di test, e poi per ogni riga: l'obiettivo infatti è quello di rilevare l'anomalia sulla singola riga, più che sulla finestra.

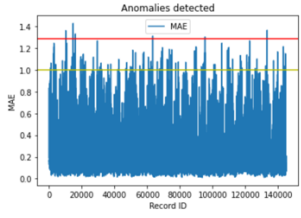
Quando viene generata una finestra di 30 righe con uno spostamento di 6 righe, alcune di esse verranno ripetute fra una finestra e l'altra; ciò andrà ad influenzare il calcolo della MAE.
Per ovviare a questo problema, in fase di finestraggio vengono tenute in memoria alcune informazioni. Per ogni sotto-dataset di test relativo ad un singolo paziente, a tempo di windowing vengono memorizzate quattro informazioni principali.

- Indice di riga dove inizia il sotto-dataset relativo al paziente
- Indice di riga dove finisce il sotto-dataset relativo al paziente
- Numero di finestre generate
- Numero di righe padding

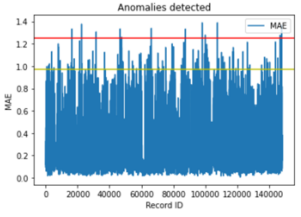
Grazie a queste informazioni, è possibile costruire una struttura di tipo *HashMap*. La chiave è l'indice della riga del dataset originale che stiamo analizzando (da 0 a 722786), mentre il valore è un insieme di indici nei quali si ripete la riga. Infatti, ad esempio, la riga "reale" 25 corrisponderà alla riga duplicata 25 (sua occorrenza nella prima finestra in cui appare), 49 (occorrenza nella seconda finestra in cui appare), 73 (occorrenza nella terza finestra in cui appare), 97 (occorrenza nella quarta finestra in cui appare), e 121 (occorrenza nella quinta ed ultima finestra in cui appare). Avendo questa corrispondenza, è possibile calcolare per ogni riga "vera" di ogni sotto-dataset relativo ad un paziente, la media delle MAE di tutti i duplicati per quella riga.

Infine, riguardo le anomalie (verificate per paziente) sono state scelte due "soglie di sbarramento" che rispettivamente verificheranno se una misurazione è un'anomalia quasi certamente o meno certamente. Una misurazione viene classificata anomala se il valore di Mean Absolute Error supera il 90% del valore massimo di Mean Absolute Error generato per quel paziente. Una misurazione viene classificate probabilmente anomala se il valore di Mean Absolute Error supera il 70% del valore massimo di Mean Absolute Error per quel paziente.
Di seguito vengono mostrati i valori di Mean Absolute Error per ogni paziente.

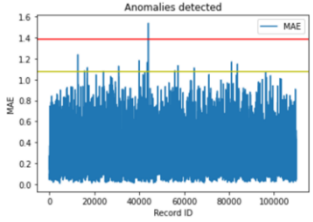
```
Paziente 1004:
Errore massimo: 1.4283496141433716
Soglia massima: 1.2855146527290344
```



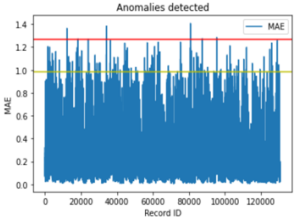

```
Paziente 1006:
Errore massimo: 1.3886370658874512
Soglia massima: 1.249773359298706
```



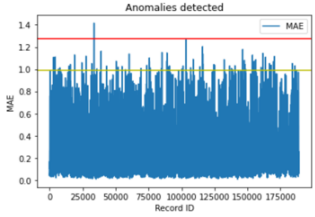
```
Paziente 3006:
Errore massimo: 1.5387420654296875
Soglia massima: 1.3848678588867187
```




```
Paziente 4002:
Errore massimo: 1.40720534324646
Soglia massima: 1.266484808921814
```




```
Paziente 3001:
Errore massimo: 1.415687918663025
Soglia massima: 1.2741191267967225
```



Infine, tutte le anomalie sono state salvate in un file .csv contenente anche le date di rilevazione.