UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea magistrale in Ingegneria Informatica e dell'Automazione

Algoritmi di apprendimento supervisionato per la classificazione del comportamento alla guida di autoveicoli

Supervised learning algorithms for the classification of car driving behavior

Relatore: Chiar.mo

Prof. Aldo Franco Dragoni

Tesi di Laurea di:

Nico Osimani

A.A. 2021/2022

Indice

Introduzione

Lo sviluppo ed il miglioramento dei sistemi di identificazione degli incidenti negli ultimi decenni è dovuto alla diffusione di nuove tecnologie, nuovi sensori e allo sviluppo di nuovi algoritmi di rilevazione.
Identificare in maniera automatica gli incidenti automobilistici è utile per diversi motivi e in diversi ambiti: ad esempio, per la fornitura automatica di soccorso stradale, o più semplicemente, per il monitoraggio del comportamento alla guida.
In questi casi l'identificazione degli incidenti automobilistici risulta utile alle compagnie assicurative, che sono in grado di fornire soluzioni automatizzate ed efficienti ai loro clienti, sia in termini di servizi che di costi.
Il seguente lavoro è stato svolto in collaborazione con l'azienda FairConnect [1], che opera nel mondo della Connected Life per offrire servizi tecnologici avanzati alle compagnie assicurative, mettendo a loro disposizione le proprie piattaforme IoT e Big Data per lo sviluppo di programmi assicurativi innovativi.
Il loro scopo è quello di fornire soluzioni connesse e personalizzate alle compagnie assicurative per migliorare la qualità della vita delle persone.
Nella seguente tesi verranno descritte le metodologie applicate, i dataset, gli algoritmi utilizzati, e gli esperimenti eseguiti.
Infine, verrà descritta l'implementazione di applicazioni per microcontrollore STM32 per l'uso di intelligenza artificiale direttamente su dispositivi a basso consumo.
Il lavoro svolto è disponibile al repository GitHub [2].

Capitolo 1

Identificazione di incidenti automobilistici

In questo capitolo verranno definiti gli obiettivi e le metodologie del lavoro svolto, gli algoritmi di apprendimento supervisionato utilizzati, e i dataset utilizzati per il training e per il test.

1.1 Obiettivi e metodologie

Gli obiettivi di questo lavoro sono molteplici:

- Sviluppare metodologie e algoritmi di machine learning per la classificazione degli incidenti automobilistici: si usano dei classificatori binari (classe 0 - non c'è stato incidente, classe 1 - c'è stato incidente), che utilizzano le serie temporali registrate dai sensori inerziali del veicolo (accelerometri) lungo i tre assi x, y, e z, come dati di addestramento.

  Il campionamento viene effettuato tramite apposite black box installate sul veicolo, dopo di che i dati vengono inviati ad un server che ha il compito di processare e rielaborare i dati opportunamente, per poi classificarli tramite algoritmi di machine learning precedentemente allenati.

- Sviluppare metodologie per risolvere il problema delle rotazioni del sistema di riferimento: in ottica di sviluppo dei sistemi di sensori per il campionamento delle accelerazioni, è utile cercare di rendere il sistema di riferimento indipendente dalle rotazioni spaziali, ovvero avere la possibilità di campionare le accelerazioni con sensori che non siano solidali con l'automobile come nel caso di una black box.

  Un'applicazione di questo scenario è, ad esempio, l'utilizzo di uno smartphone semplicemente posto nell'automobile per il campionamento delle accelerazioni.

- Sviluppare applicazioni per microcontrollore STM32 che permettano la classificazione direttamente in automobile: uno sviluppo del lavoro è quello di portare la fase di processamento e di classificazione direttamente in automobile, e di utilizzare il lato server del sistema solo per la ricezione della risposta finale. Questo porta numerosi vantaggi in termini di consumo.

  Una soluzione è appunto quella di usare un dispositivo della famiglia STM32 per la realizzazione di questo tipo di applicazioni. La pratica di implementare funzionalità basate sull'intelligenza artificiale in prossimità del punto di acquisizione dai dati è nota come Edge AI.

1.2 Dispositivi e dataset

Le registrazioni sono state generate da due diversi dispositivi: Microbox e Infobox. Ogni dispositivo inizia a registrare ogni volta che rileva, in base al suo algoritmo decisionale, un incidente.
Ciò viene stabilito in termini di intensità delle accelerazioni, e vengono quindi memorizzate tutte le misurazioni inerziali in una finestra temporale in cui il picco più alto cade più o meno in la metà. Le accelerazioni vengono misurate in g (accelerazione di gravità).
Queste registrazioni vengono poi analizzate dall'algoritmo di classificazione, ovviamente più preciso dell'algoritmo decisionale con il quale il dispositivo decide di iniziare a registrare, per classificare la registrazione come incidente o meno.
A seconda del tipo di dispositivo e della loro specifica revisione hardware, i dati dei sensori vengono campionati in modi diversi, come verrà mostrato in seguito.

1.2.1 Infobox

Il dispositivo Infobox ha 3 revisioni hardware, le cui caratteristiche sono illustrate nella Tabella 1.1.

Revisione	Numero di campioni per registrazione	Numero di registrazioni (false)	Numero di registrazioni (vere)	Frequenze di campionamento
1	3100	2881	3019	[0:499] -> 50Hz [500:2900] -> 400Hz [2900:3099] -> 50Hz
2	3060	162	26	[0:59] -> 10Hz [60:3059] -> 1000Hz
3	3000	2	0	[0:2999] -> 1000Hz
Totale	-	3045	3045	-

Tabella 1.1: Caratteristiche del dispositivo Infobox

Il dispositivo Infobox memorizza anche i valori prima dell'incidente, oltre ai dati sull'incidente e dopo l'incidente.

1.2.2 Microbox

Il dispositivo Microbox ha una sola revisioni hardware, le cui caratteristiche sono illustrate nella Tabella 1.2.

Numero di campioni per registrazione	Numero di registrazioni (false)	Numero di registrazioni (vere)	Frequenza di campionamento
3000	962	962	[0:2999] -> 100Hz sovracampionati a 1000Hz

Tabella 1.2: Caratteristiche del dispositivo Microbox

La frequenza di campionamento originale del dispositivo Microbox è 100Hz, ma viene effettuato un processo di sovracampionamento per interpolazione (con un fattore x10) eseguito dal dispositivo quando i dati del sensore vengono esportati e memorizzati.
Microbox memorizza solo i dati durante e dopo l'incidente. Non fornisce informazioni prima di un incidente.

1.3 Reti neurali convoluzionali

Come algoritmi per la classificazione sono state utilizzate delle reti neurali convoluzionali (CNN), che sono un tipo di reti neurali artificiali feed-forward in cui il pattern di connettività tra i neuroni è ispirato dall'organizzazione della

corteccia visiva animale, i cui neuroni individuali sono disposti in maniera tale da rispondere alle regioni di sovrapposizione che tassellano il campo visivo.
Hanno diverse applicazioni nel riconoscimento di immagini e video, ma possono anche essere utilizzate, come nel nostro caso, per classificare dati monodimensionali [3].
Per implementare le reti neurale è stato utilizzato il framework TensorFlow [4].

1.3.1 1D CNN Microbox

Per il dataset proveniente dal dispositivo Microbox, è stata utilizzata una rete convoluzionale monodimensionale, i cui layer sono elencati nella Tabella 1.3.

Layer	**Caratteristiche**
Conv1D	Numero di kernel: 100; dimensione: 10
MaxPooling1D	Dimensione: 20
Dropout	20%
Conv1D	Numero di kernel: 50; dimensione: 2
MaxPooling1D	Dimensione: 5
Dropout	10%
Conv1D	Numero di kernel: 25, dimensione: 2
GlobalMaxPooling1D	-
Dropout	10%
Dense	Classificazione tramite softmax

Tabella 1.3: Layer della rete neurale utilizzata con il dataset relativo a Microbox

Gli strati convoluzionali applicano un certo numero di filtri convoluzionali (numero di kernel) di dimensioni sopra descritte, e utilizzano come funzione di attivazione la ReLU (Rectified Linear Unit), una funzione di attivazione definita come la parte positiva del suo argomento (Equazione 1.1):

$$f(x) = \max(0, x) \tag{1.1}$$

Gli strati di pooling riducono il numero di parametri, applicando filtri di Max Pooling di dimensioni sopra descritte (o di dimensioni globali nel caso del Global Max Pooling).
Gli strati di dropout effettuano un drop casuale di una percentuale dei parametri.
Infine, lo strato denso è uno strato completamente connesso che riduce il numero di nodi in uscita a 2, ovvero il numero di classi, e applica usa come funzione per la classificazione una softmax, che è una generalizzazione di una funzione logistica che comprime un vettore k dimensionale z di valori reali arbitrari in un vettore k dimensionale $\sigma(z)$ di valori compresi in un intervallo (0, 1) la cui somma è 1 (Equazione 1.2).

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{1.2}$$

Gli indici k e j rappresentano le classi. La classificazione avviene associando a un campione la classe il cui valore di softmax in uscita alla rete è maggiore.
Come ottimizzatore è stato usato SGD (stochastic gradient descent), con un learning rate fisso a 0.005 e una Binary Crossentropy loss.

1.3.2 1D CNN Infobox

Per il dataset proveniente dal dispositivo Infobox, è stata ancora utilizzata una rete convoluzionale monodimensionale, i cui layer sono elencati nella Tabella 1.4.

Layer	**Caratteristiche**
Conv1D	Numero di kernel: 110; dimensione: 20
MaxPooling1D	Dimensione: 16
Dropout	50%
Conv1D	Numero di kernel: 100; dimensione: 10
GlobalAveragePooling1D	-
Dropout	50%
Dense	Classificazione tramite softmax

Tabella 1.4: Layer della rete neurale utilizzata con il dataset relativo a Infobox

Come per la rete utilizzata con Microbox, gli strati convoluzionali utilizzano come funzione di attivazione la ReLU.
Nel caso di applicazione di pooling globale, viene applicato un Average Pooling anzichè un Max Pooling.
Su questa rete è stato effettuato un processo di hypertuning dei parametri, per trovare la configurazione dei parametri ottimale cercando di mantenere il numero di parametri totali del modello non troppo elevato (sotto i 120000).
Come ottimizzatore è stato usato Adam (Adaptive moment estimation) con un learning rate variabile, partendo da 0.001 e decadendo di 0.96 ad ogni step per un totale di 100000 step in maniera esponenziale, e una Binary Crossentropy loss.

1.3.3 2D CNN Microbox

Sono state effettuate diverse trasformazioni delle serie temporali ricavate da Microbox per la visualizzazione delle stesse come immagini. Per questi tipi di esperimenti si è utilizzata una rete neurale convoluzionale 2D.
Per via della pesantezza del modello, si è utilizzato solo il dataset proveniente da Microbox in questo caso, essendo queste serie temporali più piccole rispetto a quelle provenienti da Infobox.

Le caratteristiche dei layer di questa rete sono visibili nella Tabella 1.5.

Layer	**Caratteristiche**
Conv2D	Numero di kernel: 64; dimensioni: 6x6 con dilation rate 2
BatchNormalization	-
MaxPooling2D	Dimensioni: 6x6
Conv2D	Numero di kernel: 64; dimensioni: 3x3
MaxPooling2D	Dimensioni: 3x3
Flatten	-
Dense	Numero di nodi: 256
Dropout	50%
Dense	Classificazione tramite softmax

Tabella 1.5: Layer della rete neurale utilizzata nel caso 2D con il dataset relativo a Microbox

Gli strati convoluzionali utilizzano come funzione di attivazione la ELU (Exponential Linear Unit), una variazione di ReLU che cerca di rendere le attivazioni più vicine allo zero e che accelera l'apprendimento. Viene definita come (Equazione 1.3):

$$f(x) = \begin{cases} x & \text{se } x > 0 \\ a\,(e^x - 1) & \text{altrimenti} \end{cases} \tag{1.3}$$

dove a è un iperparametro e $a \geq 0$.
Anche il penultimo layer Dense usa come funzione di attivazione la ELU.
Il primo strato convoluzionale ha un dilation rate di 2, ovvero il filtro convoluzionale viene esteso di 2 volte ad ogni passo (si sposta lungo l'immagine come un filtro 6x6 ma esegue la convoluzione come se fosse un filtro 11x11).
L'applicazione di un layer di Batch Normalization è un metodo utilizzato per rendere più veloce e stabile l'addestramento delle reti neurali attraverso la normalizzazione degli input mediante ricentraggio e ridimensionamento.
Il layer Flatten è un layer completamente connesso che non fa altro che ridistribuire i nodi del layer precedenti lungo una sola dimensione (si comporta come un layer Dense di dimensioni il prodotto delle dimensioni del layer precedente).
Come ottimizzatore è stato utilizzato Adam con un learning rate fisso di 0.00005 e una Binary Crossentropy loss.

Capitolo 2

Esperimenti e fase di training

In questo capitolo verranno elencati i vari esperimenti effettuati, con particolare attenzione alla fase di training delle varie reti neurali. In tutti gli esperimenti sono state settate 1000 epoche, implementando un early stopping per ognuno dei training.
Per implementare gli esperimenti e i test è stato utilizzato Google Colab [5].
Di seguito una parte di codice utilizzata per l'esecuzione del train e per il plot dell'accuracy di train e di validation e della loss di train e di validation:

```
checkpoint = tf.keras.callbacks.ModelCheckpoint(results_path + '/' +
                                                model_name + '.h5',
                                                monitor='val_accuracy',
                                                verbose=1,
                                                save_best_only=True,
                                                save_weights_only=False,
                                                mode='auto')
early = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                         min_delta=min_delta,
                                         patience=patience,
                                         verbose=1, mode='auto')

history = model.fit(X_train, y_train, validation_split=validation_split,
                    epochs=max_epoch, callbacks=[checkpoint, early],
                    shuffle=True)

# Accuracy
fig1 = plt.figure()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.grid()
fig1.savefig(results_path + '/' + model_name + '_model_accuracy.png')
plt.show()
# Loss
fig2 = plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.grid()
fig2.savefig(results_path + '/' + model_name + '_model_loss.png')
plt.show()
```

2.1 1D nel dominio del tempo

Il primo esperimento eseguito tratta l'uso delle serie temporali nel dominio del tempo, così come sono state campionate dai dispositivi.

2.1.1 Microbox

Nel caso di uso del dataset proveniente da Microbox, le serie temporali sono state processate per ottenere serie di 300 campioni lungo 3 secondi. Per questo, si è quindi effettuato un sottocampionamento della serie, passando da 1000Hz di frequenza a 100Hz.
Per Microbox vengono presi in considerazione tutti e tre gli assi, x, y, e z.
Per eseguire la lettura e l'importazione del dataset, sono state utilizzate le librerie Pandas [6] e NumPy [7].
Di seguito delle parti di codice utilizzate per il caricamento e il processamento del dataset Microbox:

```
def get_dataset(path):

  entry_list = []

  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)
    ds_entry_processed = preprocess_microbox(ds_entry)

    entry_list.append(ds_entry_processed)

  # Convert list to a numpy array. Cast to float32 to avoid errors on fit
  ds_array = np.asarray(entry_list).astype('float32')

  return ds_array

def preprocess_microbox(entry):
  loaded_entry = entry.values[1:3000,4:7]
  loaded_entry_subsamp = loaded_entry[::10].copy()

  return loaded_entry_subsamp
```

I dati di input di dimensioni 300x3 (lunghezza di ogni serie 300, per i 3 canali x, y, e z) sono stati utilizzati con la rete neurale 1D CNN Microbox descritta sopra.
In Figura 2.1 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

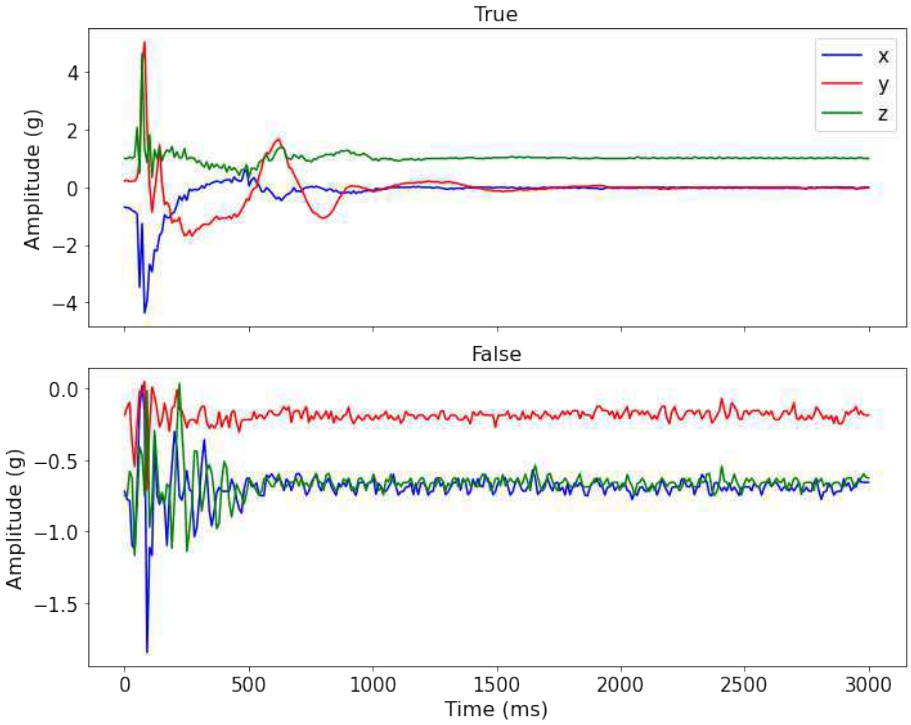

Figura 2.1: Classe vera e classe falsa Microbox 1D nel dominio del tempo

È stata utilizzata la libreria Scikit-learn [8] perla divisione del dataset in set di training e set di test. L'80% del dataset è stato utilizzato per il training e il 20% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 60 epoche.
In Figure 2.2 e 2.3 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

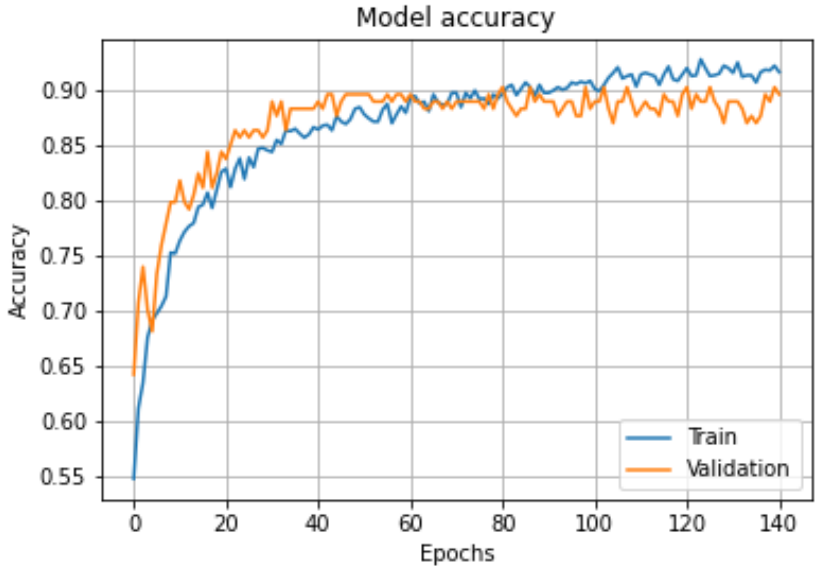

Figura 2.2: Train e validation accuracy Microbox 1D nel dominio del tempo

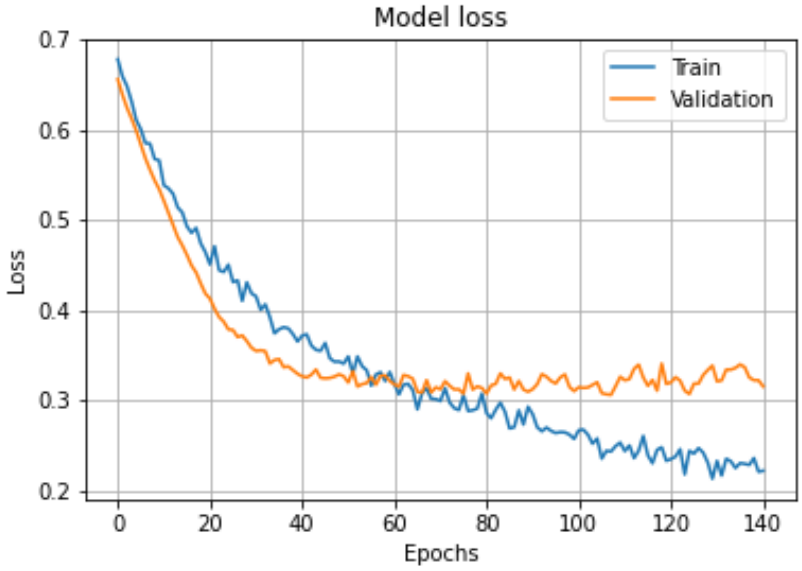

Figura 2.3: Train e validation loss Microbox 1D nel dominio del tempo

2.1.2 Infobox

Nel caso di uso del dataset proveniente da Infobox, le serie temporali sono state processate per ottenere serie di 1200 campioni lungo 3 secondi. A seconda delle revisioni hardware, si sono effettuate diverse procedure:

- Per la revisione 1, si sono presi 1200 campioni dalla parte centrale della registrazione, campionata a 400Hz di frequenza.
- Per la revisione 2, si è presa tutta la parte campionata a 1000Hz (3000 campioni) e si è sottocampionata a 400Hz, per ottenere 1200 campioni.
- Per la revisione 3, si è presa tutta la registrazione (3000 campioni a 1000Hz) e si è sottocampionata a 400Hz, per ottenere 1200 campioni.

Per ricampionare le serie, è stata usata la libreria Samplerate [9].
Per Infobox vengono presi in considerazione solamente gli assi x ed y per rendere il modello più leggero, non essendoci state particolari differenze di risultati nell'utilizzo di tutti e tre gli assi.
Di seguito una parte di codice utilizzata per il processamento del dataset Infobox:

```
def preprocess_infobox(entry):
  # Decide hardware version based on recording lenght
  # rev 1
  if (len(entry)) > 3100:
    # Get records for which sampling freq is 400Hz
    # Get only col 4 and 5 (filtered x and y accelerometers)
    # Make infobox sample similar to microbox
    loaded_entry = entry.values[900:2100,4:6]

  # rev 2,3
  else:
    if (len(entry)) > 3050:
      loaded_entry = entry.values[59:3059,4:6]
    else:
      loaded_entry = entry.values[0:3000,4:6]

    # All records have sampling freq of 1000Hz
    # Resample 1000Hz to 400Hz -> ratio = out_freq / in_freq
    ratio = 0.4
    converter = 'sinc_best'  # Or 'sinc_fastest', ...
    loaded_entry = samplerate.resample(loaded_entry, ratio, converter)

  return loaded_entry
```

I dati di input di dimensioni 1200x2 sono stati utilizzati con la rete neurale 1D CNN Infobox descritta sopra.

In Figura 2.4 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

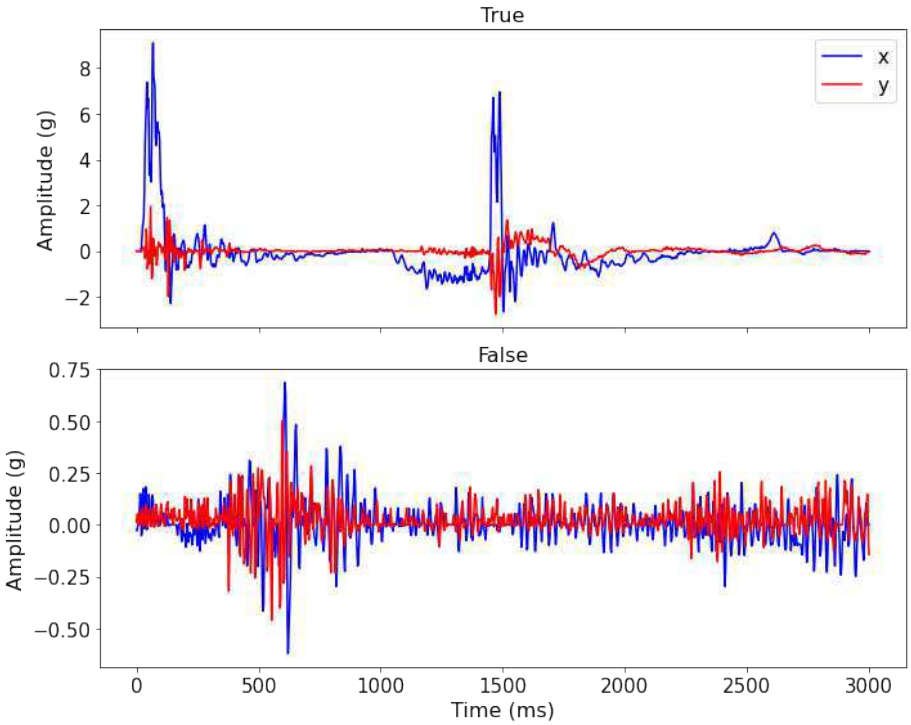

Figura 2.4: Classe vera e classe falsa Infobox 1D nel dominio del tempo

Il 95% del dataset è stato utilizzato per il training e il 5% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 100 epoche.
In Figure 2.5 e 2.6 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

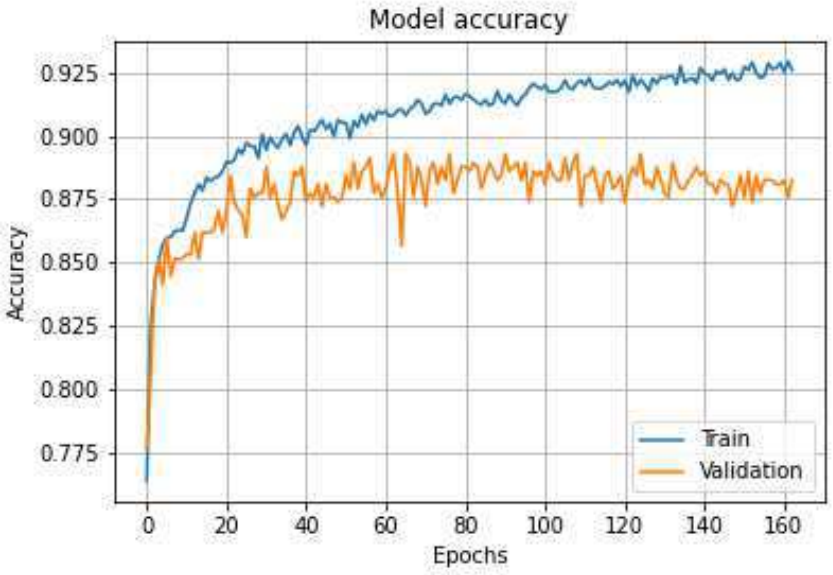

Figura 2.5: Train e validation accuracy Infobox 1D nel dominio del tempo

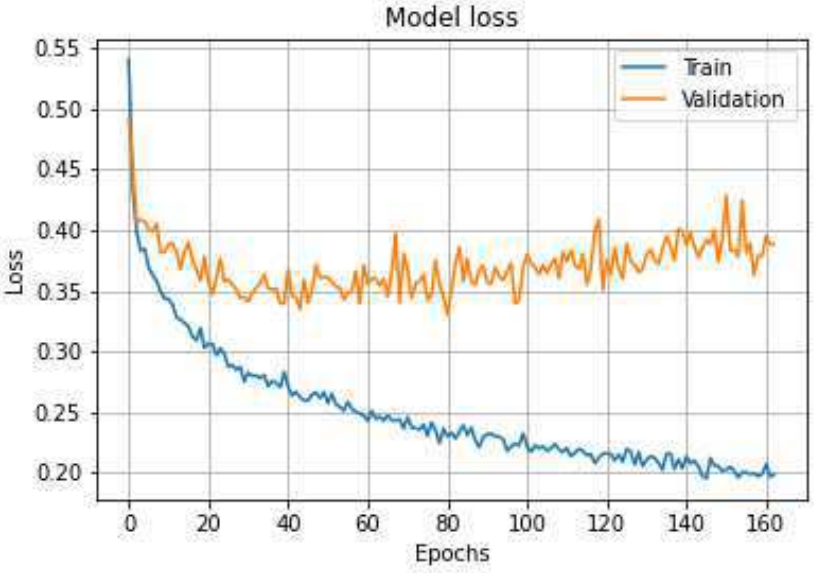

Figura 2.6: Train e validation loss Infobox 1D nel dominio del tempo

2.2 1D nel dominio della frequenza

Il secondo esperimento eseguito tratta l'uso delle serie temporali trasformate nel dominio della frequenza tramite la trasformata di Fourier (Equazione 2.1):

$$\mathcal{F}(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-j\omega t}\,dt \tag{2.1}$$

É stata utilizzata la libreria SciPy [10] per l'implementazione della Fast Fourier transform per l'esecuzione della trasformata di Fourier (Equazione 2.2):

$$\mathcal{F}(n) = \sum_{h=0}^{N-1} y(h)e^{-j\frac{2\pi}{N}nh} \tag{2.2}$$

Dopo l'esecuzione delle trasformata di Fourier è stato applicato il modulo della trasformata punto per punto, per ottenere così una trasformata reale partendo dalla trasformata complessa ottenuta con la Fast Fourier transform.
Per un numero complesso $z = a + jb$, il suo modulo è definito come (Formula 2.3):

$$|z| = \sqrt{a^2 + b^2} \tag{2.3}$$

2.2.1 Microbox

Di seguito una parte di codice utilizzata per la trasformazione nel dominio della frequenza del dataset Microbox:

```
def get_dataset(path):

  entry_list = []

  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)
    ds_entry_processed = preprocess_microbox(ds_entry)

    fft_x = fft(ds_entry_processed[:,0])
    fft_y = fft(ds_entry_processed[:,1])
    fft_z = fft(ds_entry_processed[:,2])

    d_x = np.absolute(fft_x)
    d_y = np.absolute(fft_y)
    d_z = np.absolute(fft_z)

    ds_entry_fft = np.stack([d_x, d_y, d_z], axis=1)
    entry_list.append(ds_entry_fft)

  # Convert list to a numpy array. Cast to float32 to avoid errors on fit
  ds_array = np.asarray(entry_list).astype('float32')

  return ds_array
```

I dati di input di dimensioni 300x3 sono stati utilizzati con la rete neurale 1D CNN Microbox.
In Figura 2.7 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa. Il range di frequenze si estende da 0 a 50Hz.

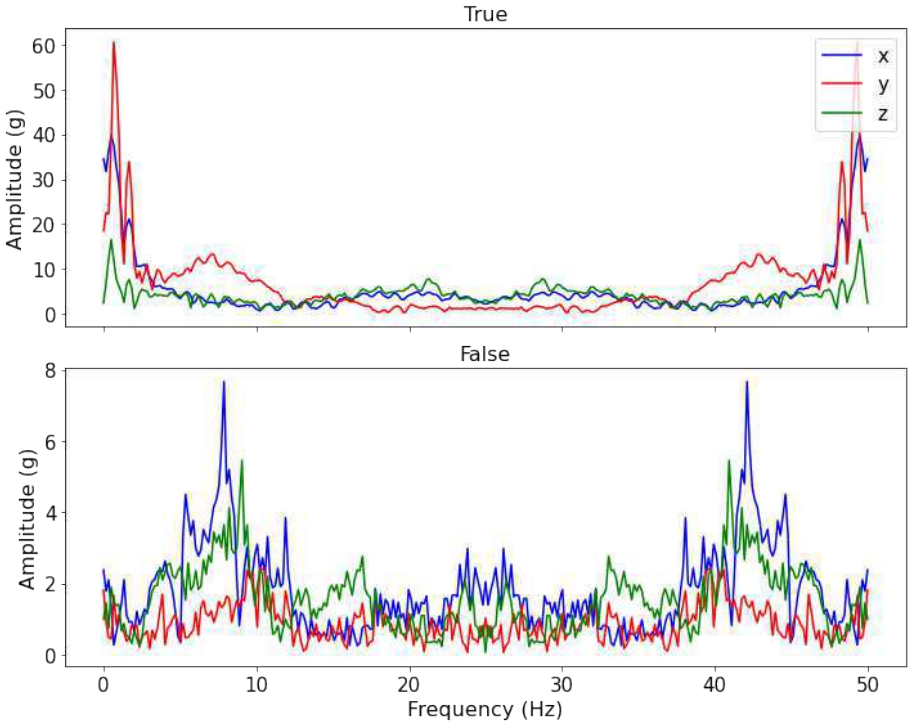

Figura 2.7: Classe vera e classe falsa Microbox 1D nel dominio della frequenza

L'80% del dataset è stato utilizzato per il training e il 20% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 60 epoche.
In Figure 2.8 e 2.9 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

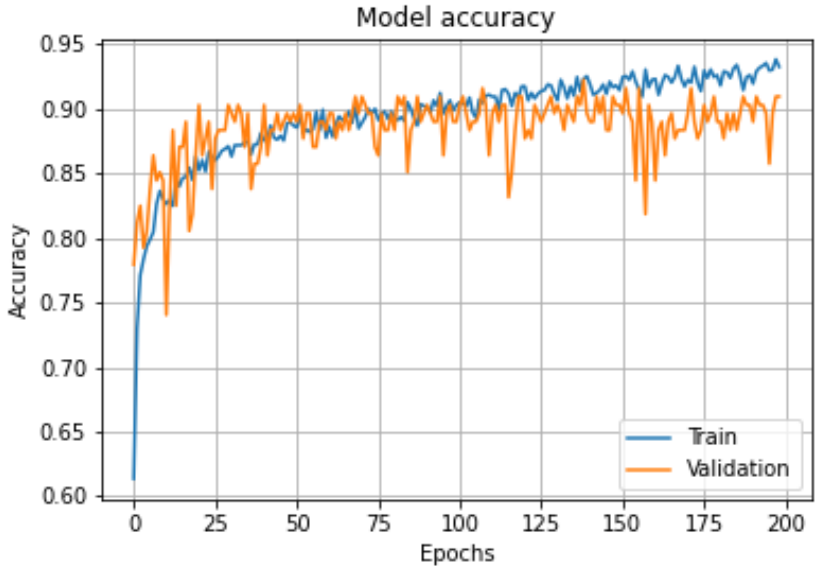

Figura 2.8: Train e validation accuracy Microbox 1D nel dominio della frequenza

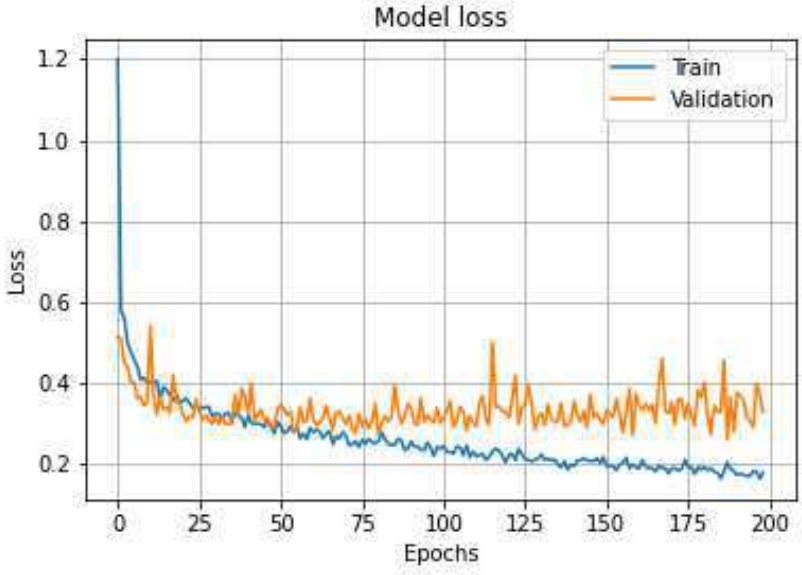

Figura 2.9: Train e validation loss Microbox 1D nel dominio della frequenza

Si può vedere che l'algoritmo impiega più tempo a convergere rispetto al caso

nel dominio del tempo.

2.2.2 Infobox

Di seguito una parte di codice utilizzata per la trasformazione nel dominio della frequenza del dataset Infobox:

```
def get_dataset(path):

  entry_list = []

  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)
    ds_entry_processed = preprocess_infobox(ds_entry)

    fft_x = fft(ds_entry_processed[:,0])
    fft_y = fft(ds_entry_processed[:,1])

    d_x = np.absolute(fft_x)
    d_y = np.absolute(fft_y)

    ds_entry_fft = np.stack([d_x, d_y], axis=1)
    entry_list.append(ds_entry_fft)

  # Convert list to a numpy array. Cast to float32 to avoid errors on fit
  ds_array = np.asarray(entry_list).astype('float32')

  return ds_array
```

I dati di input di dimensioni 1200x2 sono stati utilizzati con la rete neurale 1D CNN Infobox.
In Figura 2.10 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa. Il range di frequenze si estende da 0 a 200Hz.

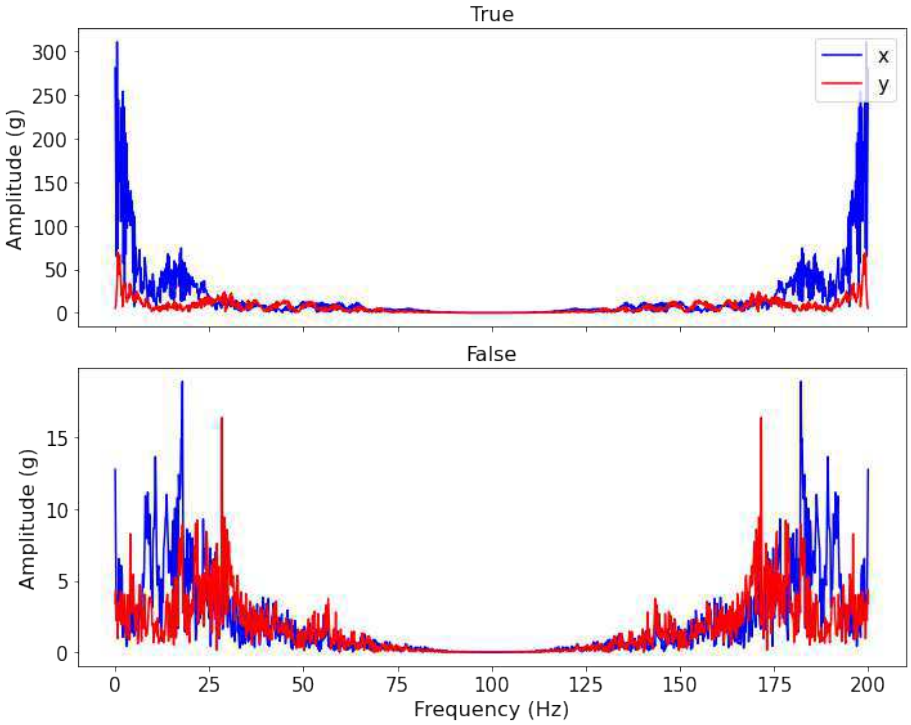

Figura 2.10: Classe vera e classe falsa Infobox 1D nel dominio della frequenza

Il 95% del dataset è stato utilizzato per il training e il 5% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 100 epoche.
In Figure 2.11 e 2.12 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

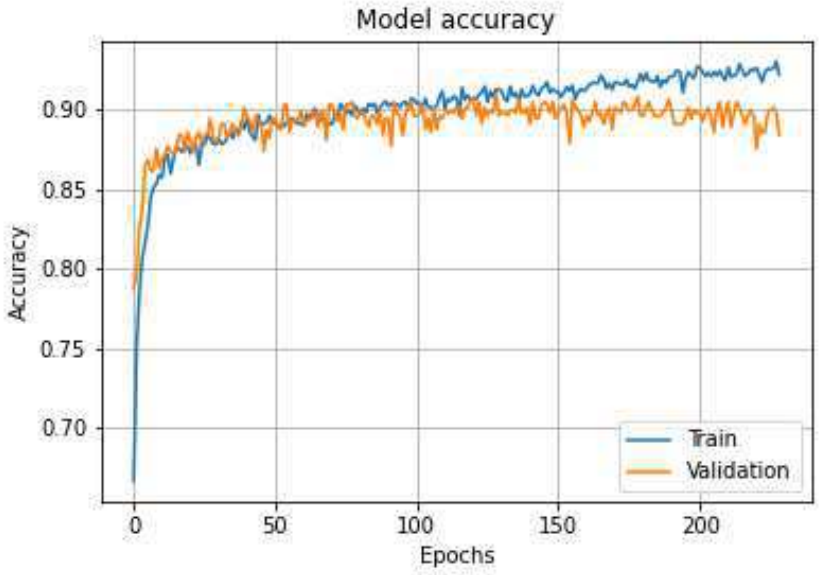

Figura 2.11: Train e validation accuracy Infobox 1D nel dominio della frequenza

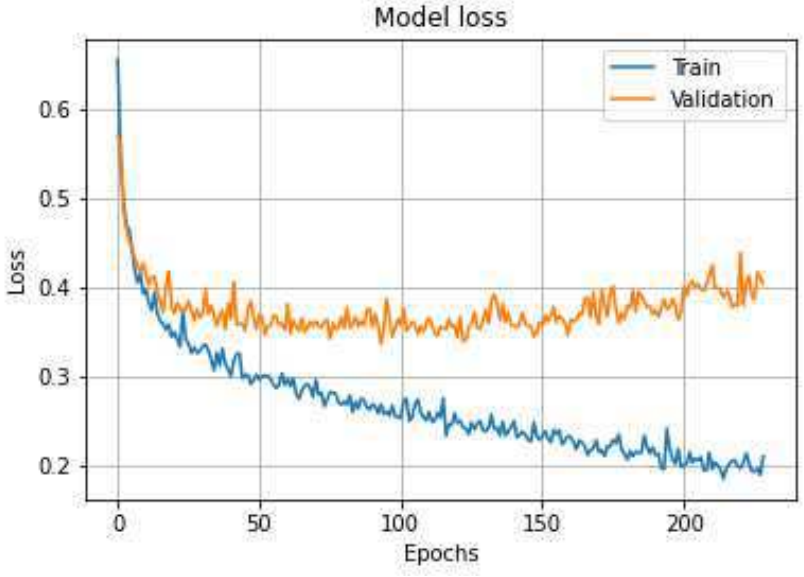

Figura 2.12: Train e validation loss Infobox 1D nel dominio della frequenza

Come nel caso Microbox, l'algoritmo impiega più tempo a convergere rispetto

al caso nel dominio del tempo.

2.3 1D con norma nel dominio del tempo

Il terzo esperimento eseguito tratta l'uso delle serie temporali dopo l'applicazione della norma punto per punto, in modo da ottenere una sola serie temporale, che risulti quindi indipendente dal sistema di riferimento dei sensori.
Dato un vettore $v = [x\ y\ z]^T$ (nel nostro caso ogni punto delle serie temporali è descritto da un vettore tridimensionale, nel caso Microbox, o bidimensionale, nel caso Infobox), la sua norma è (Formula 2.4):

$$|v| = \sqrt{x^2 + y^2 + z^2} \tag{2.4}$$

Questo è stato il primo esperimento per cercare di ottenere una metodologia che possa essere utilizzata per il campionamento delle accelerazioni tramite uno smartphone posto nell'automobile.

2.3.1 Microbox

Di seguito una parte di codice utilizzata per l'applicazione della norma nel dataset Microbox:

```
def get_dataset(path):

  entry_list = []

  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)
    ds_entry_processed = preprocess_microbox(ds_entry)

    ds_entry_norm = np.empty([len(ds_entry_processed), 1])
    for i in range (0, len(ds_entry_processed)):
      ds_entry_norm[i] = np.linalg.norm(ds_entry_processed[i,:])

    entry_list.append(ds_entry_norm)

  # Convert list to a numpy array. Cast to float32 to avoid errors on fit
  ds_array = np.asarray(entry_list).astype('float32')

  return ds_array
```

I dati di input di dimensioni 300x1 sono stati utilizzati con la rete neurale 1D CNN Microbox.
In Figura 2.13 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

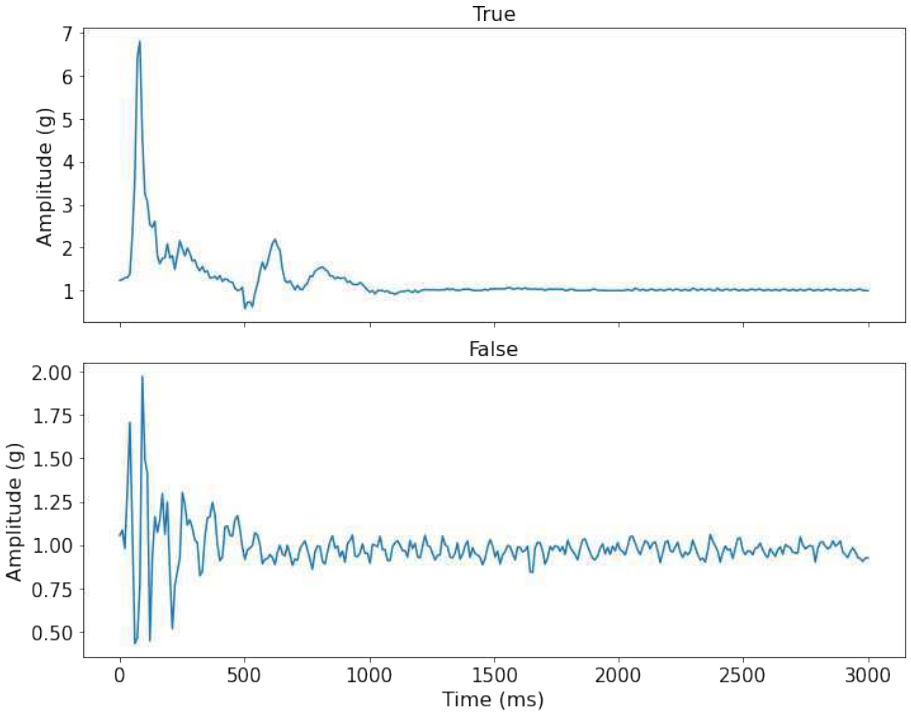

Figura 2.13: Classe vera e classe falsa Microbox 1D nel dominio del tempo con applicazione della norma

L'80% del dataset è stato utilizzato per il training e il 20% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 60 epoche.
In Figure 2.14 e 2.15 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

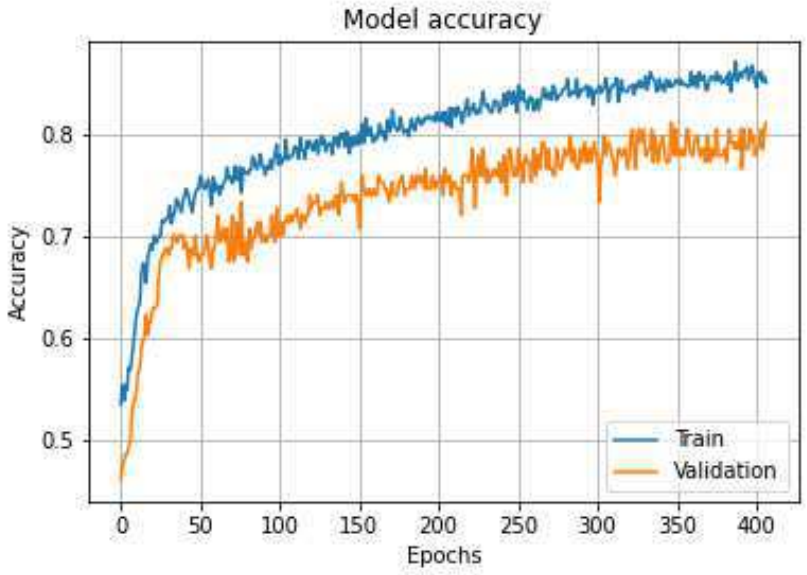

Figura 2.14: Train e validation accuracy Microbox 1D nel dominio del tempo con applicazione della norma

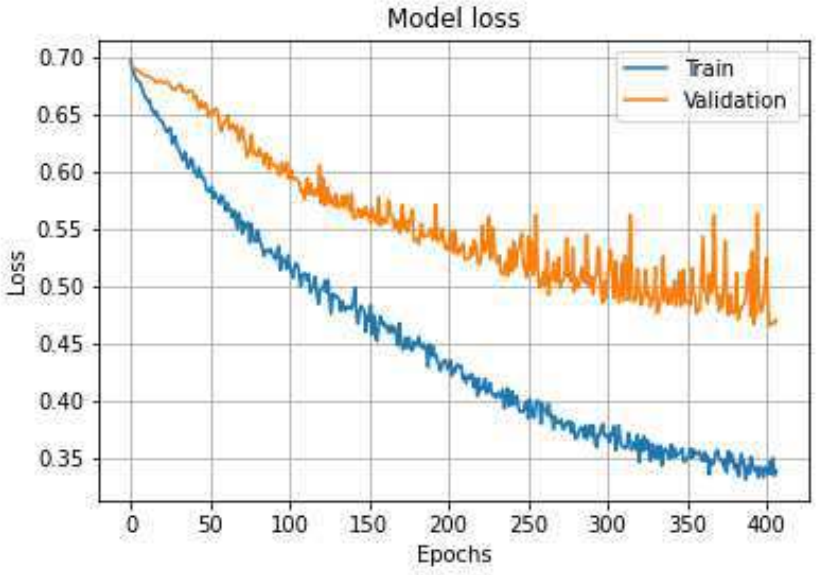

Figura 2.15: Train e validation loss Microbox 1D nel dominio del tempo con applicazione della norma

Si può vedere che l'algoritmo impiega considerevolmente più epoche a convergere rispetto al caso senza applicazione della norma.

2.3.2 Infobox

I dati di input di dimensioni 1200x1 sono stati utilizzati con la rete neurale 1D CNN Infobox.
In Figura 2.16 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

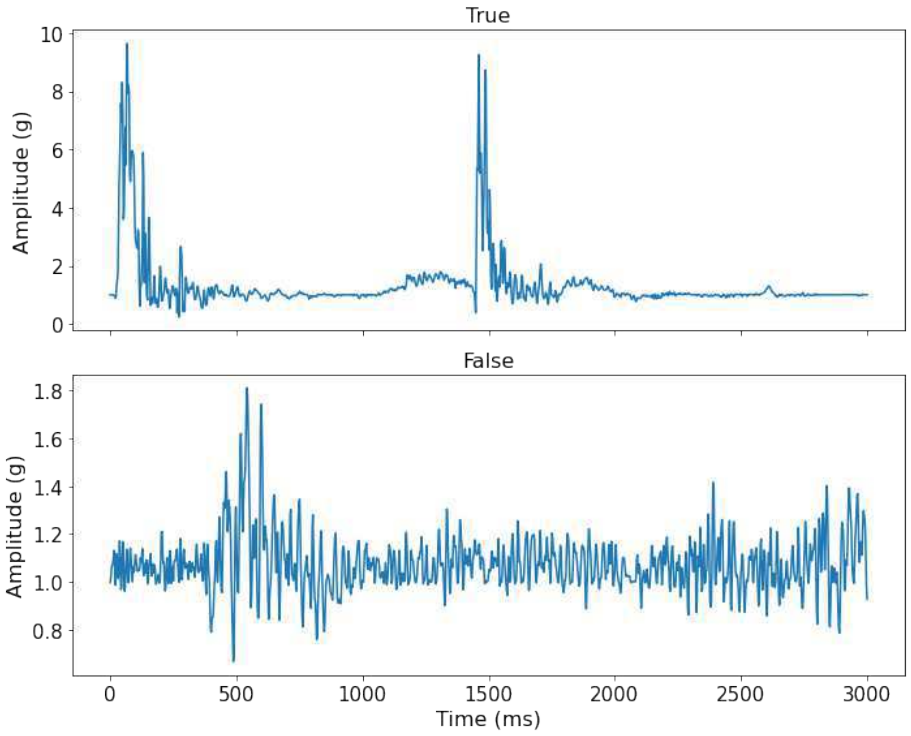

Figura 2.16: Classe vera e classe falsa Infobox 1D nel dominio del tempo con applicazione della norma

Il 95% del dataset è stato utilizzato per il training e il 5% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 100 epoche.
In Figure 2.17 e 2.18 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

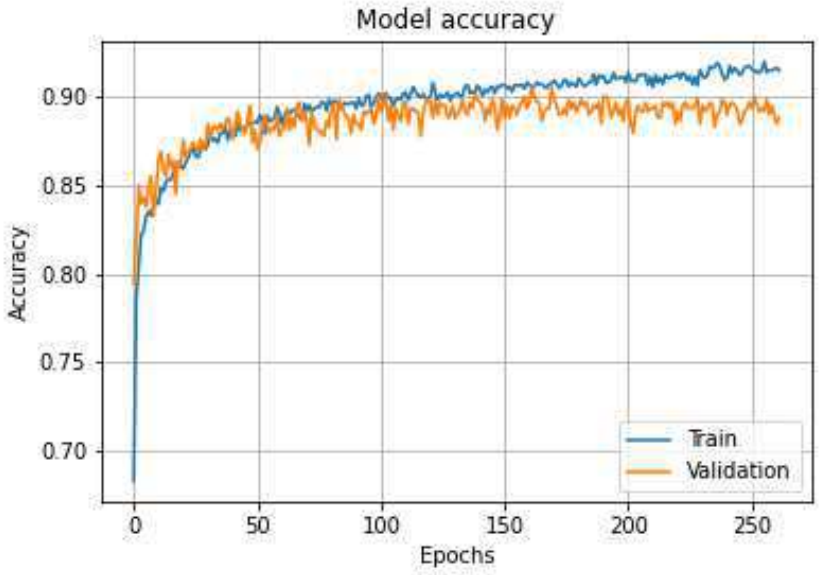

Figura 2.17: Train e validation accuracy Infobox 1D nel dominio del tempo con applicazione della norma

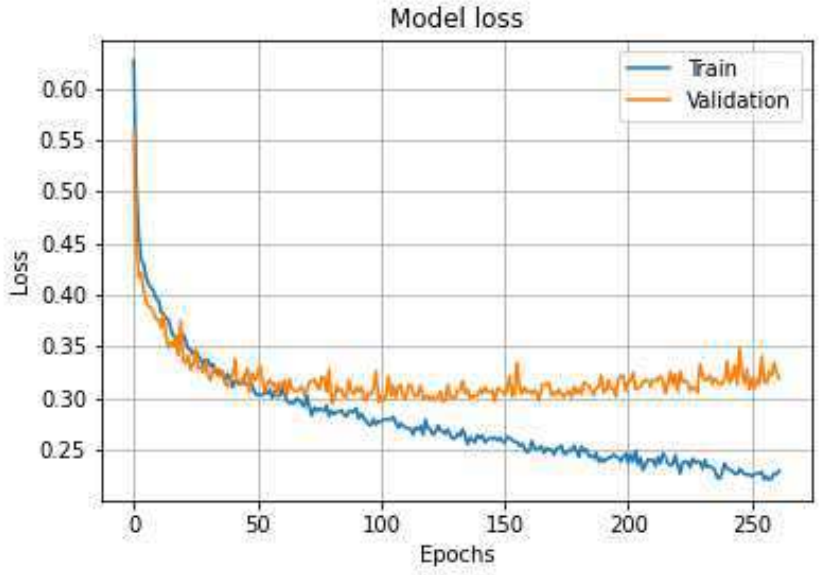

Figura 2.18: Train e validation loss Infobox 1D nel dominio del tempo con applicazione della norma

I tempi di convergenza sono simili al caso nel dominio della frequenza utilizzando gli assi x ed y, pur avendo un modello più leggero in termini di parametri totali.

2.4 1D con norma nel dominio della frequenza

Il quarto esperimento eseguito tratta l'uso delle serie temporali dopo l'applicazione della norma punto per punto, e dopo la trasformazione nel dominio della frequenza tramite Fast Fourier transform.

2.4.1 Microbox

Di seguito una parte di codice utilizzata per l'applicazione della norma e per la trasformazione nel dominio della frequenza del dataset Microbox:

```
def get_dataset(path):

  entry_list = []

  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)
    ds_entry_processed = preprocess_microbox(ds_entry)

    ds_entry_norm = np.empty([len(ds_entry_processed), 1])
    for i in range (0, len(ds_entry_processed)):
      ds_entry_norm[i] = np.linalg.norm(ds_entry_processed[i,:])

    fft_ = fft(ds_entry_norm[:,0])

    d_ = np.absolute(fft_)

    ds_entry_fft = np.stack([d_], axis=1)
    entry_list.append(ds_entry_fft)

  # Convert list to a numpy array. Cast to float32 to avoid errors on fit
  ds_array = np.asarray(entry_list).astype('float32')

  return ds_array
```

I dati di input di dimensioni 300x1 sono stati utilizzati con la rete neurale 1D CNN Microbox.
In Figura 2.19 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

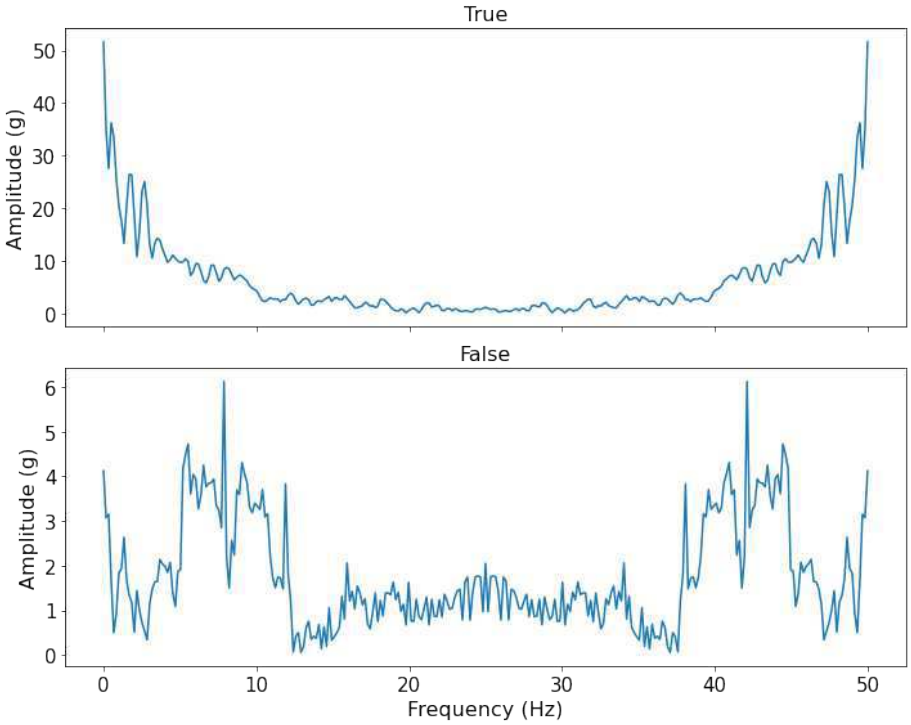

Figura 2.19: Classe vera e classe falsa Microbox 1D nel dominio della frequenza con applicazione della norma

L'80% del dataset è stato utilizzato per il training e il 20% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 60 epoche.
In Figure 2.20 e 2.21 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

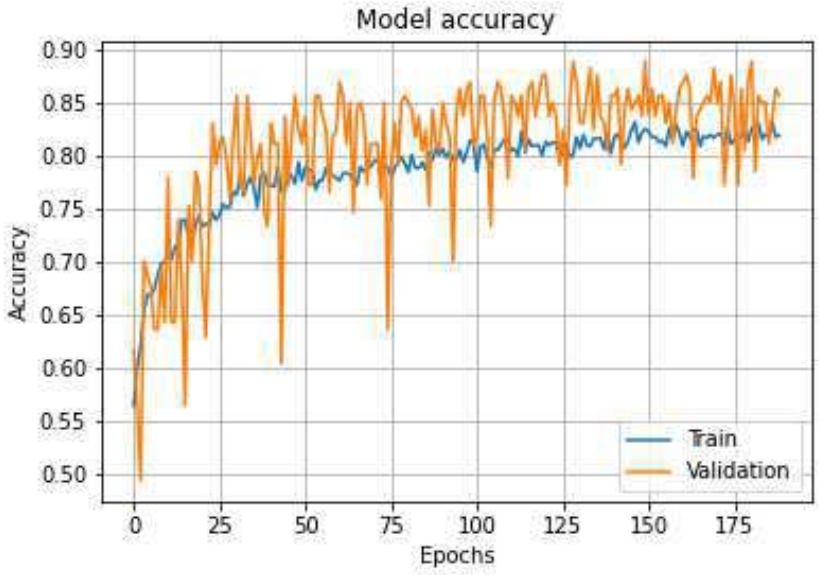

Figura 2.20: Train e validation accuracy Microbox 1D nel dominio della frequenza con applicazione della norma

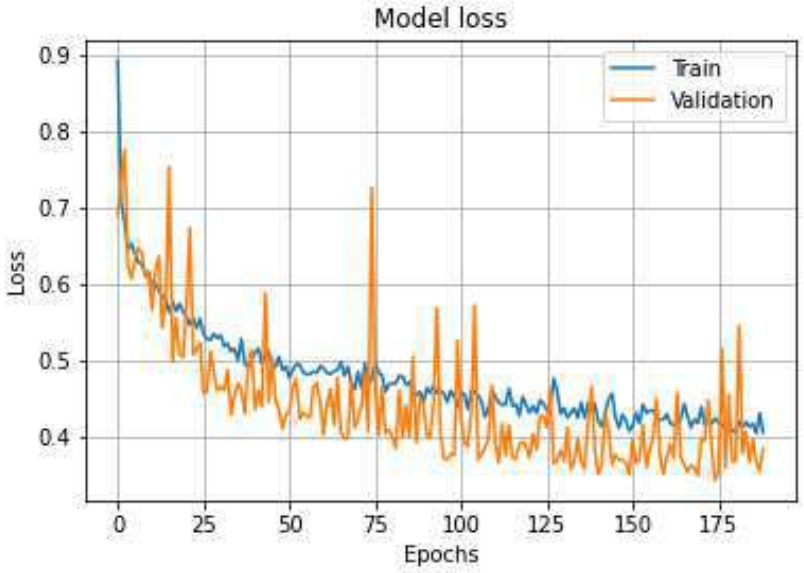

Figura 2.21: Train e validation loss Microbox 1D nel dominio della frequenza con applicazione della norma

L'algoritmo impiega meno epoche a convergere rispetto al caso nel dominio del tempo, pur avendo un andamento dell'accuracy e della loss molto più frastagliati.

2.4.2 Infobox

I dati di input di dimensioni 1200x1 sono stati utilizzati con la rete neurale 1D CNN Infobox.
In Figura 2.22 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

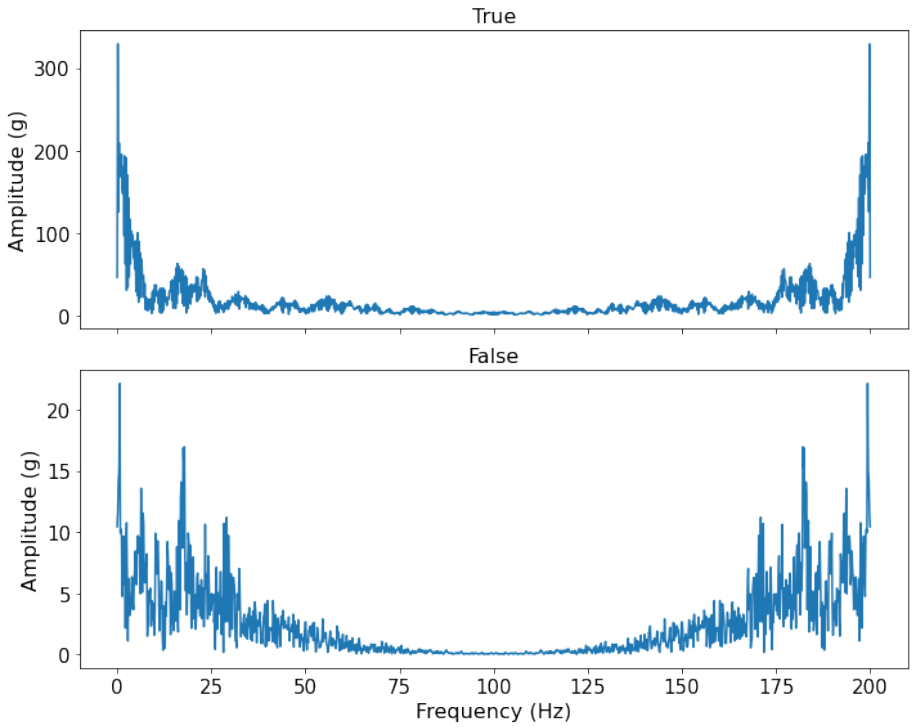

Figura 2.22: Classe vera e classe falsa Infobox 1D nel dominio della frequenza con applicazione della norma

Il 95% del dataset è stato utilizzato per il training e il 5% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 100 epoche.
In Figure 2.23 e 2.24 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

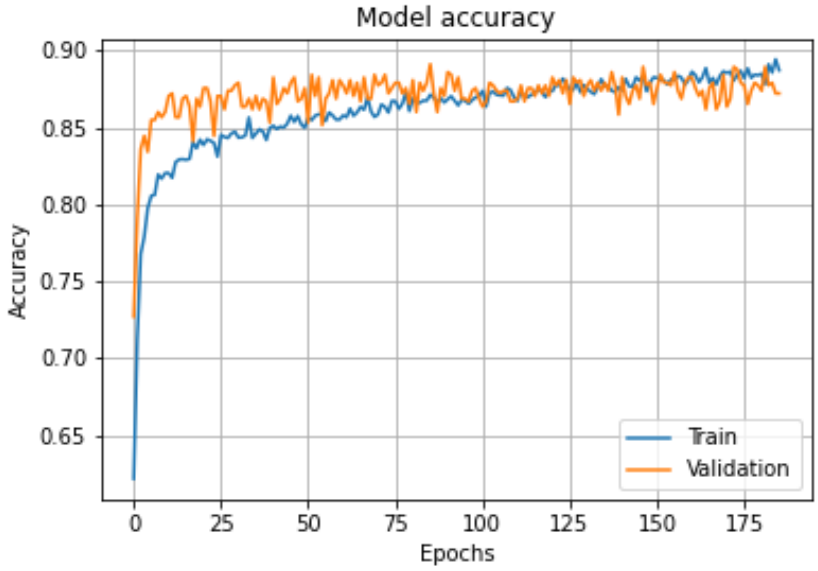

Figura 2.23: Train e validation accuracy Infobox 1D nel dominio della frequenza con applicazione della norma

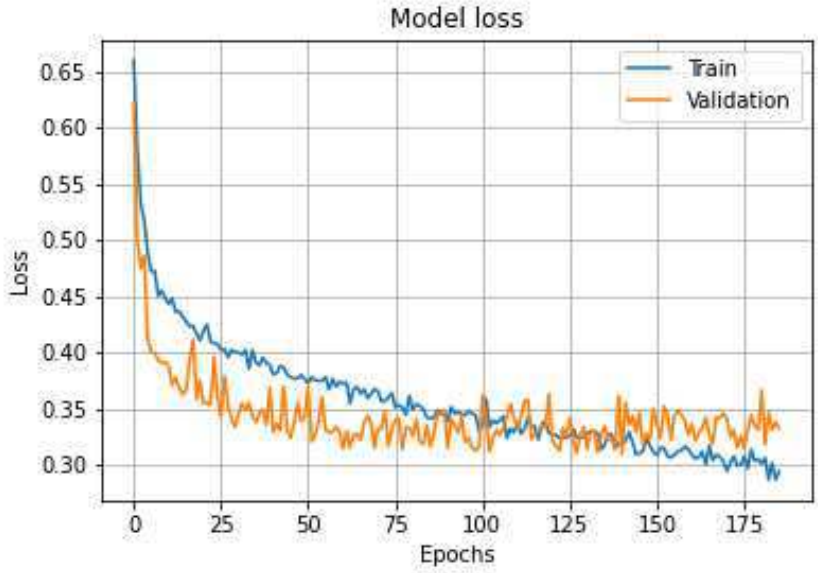

Figura 2.24: Train e validation loss Infobox 1D nel dominio della frequenza con applicazione della norma

I tempi di convergenza risultano più brevi rispetto al caso nel dominio del tempo, e con un andamento dell'accuracy e della loss più frastagliato.

2.5 1D con rotazioni nel dominio della frequenza

Il quinto esperimento eseguito tratta l'uso delle serie temporali dopo la rotazione casuale del sistema di riferimento, e dopo la trasformazione nel dominio della frequenza tramite Fast Fourier transform.
L'utilizzo delle serie nel dominio del tempo dopo la rotazione risulta impossibile, in quanto l'algoritmo tende a non apprendere o, in alternativa, ad overfittare.
Questo esperimento propone un'altra metodologia per cercare di campionare le accelerazioni tramite uno smartphone posto nell'automobile, il cui sistema di riferimento è ovviamente ruotato nello spazio rispetto al sistema di riferimento dei device Microbox e Infobox.
Una rotazione generale in 3 dimensioni può essere espressa come una composizione di 3 rotazioni intorno a tre assi indipendenti, ovvero gli assi x, y, e z. Quindi, dati tre angoli α, β, γ che indicano rispettivamente di quanto si deve ruotare intorno a ognuno degli assi, la matrice di rotazione risulta (Matrice 2.5):

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Ogni punto delle serie, espresso come vettore tridimensionale, è stato post-moltiplicato alla matrice sopra descritta, per ottenere il suo corrispettivo ruotato.
Ogni serie del set di training è stata sostituita con 10 rotazioni casuali della stessa, mentre ogni serie del set di validation e del set di test è stata sostituita con una rotazione casuale della stessa.
Gli angoli α, β, γ possono assumere valori che vanno da $-\frac{\pi}{4}$ a $\frac{\pi}{4}$, con sensibilità di 1 grado.
Di seguito delle parti di codice che definiscono le matrici di rotazione e le possibili combinazioni:

```
def Rx(theta):
  return np.matrix([[ 1, 0           , 0           ],
                    [ 0, m.cos(theta),-m.sin(theta)],
                    [ 0, m.sin(theta), m.cos(theta)]])

def Ry(theta):
  return np.matrix([[ m.cos(theta), 0, m.sin(theta)],
                    [ 0           , 1, 0           ],
                    [-m.sin(theta), 0, m.cos(theta)]])

def Rz(theta):
  return np.matrix([[ m.cos(theta), -m.sin(theta), 0 ],
                    [ m.sin(theta), m.cos(theta) , 0 ],
                    [ 0           , 0            , 1 ]])

steps = 181 # Equal steps between -max angle to +max angle (including angle 0)
angle = m.pi/4 # +/-max angle
rotations = 10 # Number of rotations to use for training set data augmentation

R = []
for i in range(-int((steps - 1)/2), int((steps - 1)/2) + 1):
  a = i*angle*(1/((steps - 1)/2))
  for j in range(-int((steps - 1)/2), int((steps - 1)/2) + 1):
    b = j*angle*(1/((steps - 1)/2))
    for z in range(-int((steps - 1)/2), int((steps - 1)/2) + 1):
      c = z*angle*(1/((steps - 1)/2))
      if not (a == 0 and b == 0 and c == 0):
        R_ = Rz(a) * Ry(b) * Rx(c)
        R.append(R_)
```

2.5.1 Microbox

Di seguito una parte di codice utilizzata per l'applicazione delle rotazioni alle serie del set di training, ovvero data augmentation del set di training del dataset Microbox:

```
# Data augmentation for ds_true train
ds_aug = ds_true_no_aug[0,:,:].copy()
ds_true_aug = []
for j in range(0, ds_true_no_aug.shape[0]):
  randomlist = []
  while len(randomlist) < rotations: # Fill with random rotations
    n = random.randint(0,len(R))
    if n not in randomlist:
      randomlist.append(n)
  for k in range (0, len(randomlist)):
    if randomlist[k] == len(R): # Null rotation
      ds_aug = ds_true_no_aug[j,:,:]

      fft_x = fft(ds_aug[:,0])
      fft_y = fft(ds_aug[:,1])
      fft_z = fft(ds_aug[:,2])

      d_x = np.absolute(fft_x)
      d_y = np.absolute(fft_y)
      d_z = np.absolute(fft_z)

      ds_entry_fft = np.stack([d_x, d_y, d_z], axis=1)

    else: # Apply rotations
      for i in range(0, ds_true_no_aug.shape[1]):
        v1 = np.array([[ds_true_no_aug[j,i,0]],
                       [ds_true_no_aug[j,i,1]],
                       [ds_true_no_aug[j,i,2]]])
        v2 = R[randomlist[k]] * v1
        ds_aug[i, 0] = v2[0,0]
        ds_aug[i, 1] = v2[1,0]
        ds_aug[i, 2] = v2[2,0]

      fft_x = fft(ds_aug[:,0])
      fft_y = fft(ds_aug[:,1])
      fft_z = fft(ds_aug[:,2])

      d_x = np.absolute(fft_x)
      d_y = np.absolute(fft_y)
      d_z = np.absolute(fft_z)

      ds_entry_fft = np.stack([d_x, d_y, d_z], axis=1)

    ds_true_aug.append(ds_entry_fft)
# Convert list to a numpy array. Cast to float32 to avoid errors on fit
ds_true = np.asarray(ds_true_aug).astype('float32')
```

I dati di input di dimensioni 300x3 sono stati utilizzati con la rete neurale 1D CNN Microbox.
In Figura 2.25 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

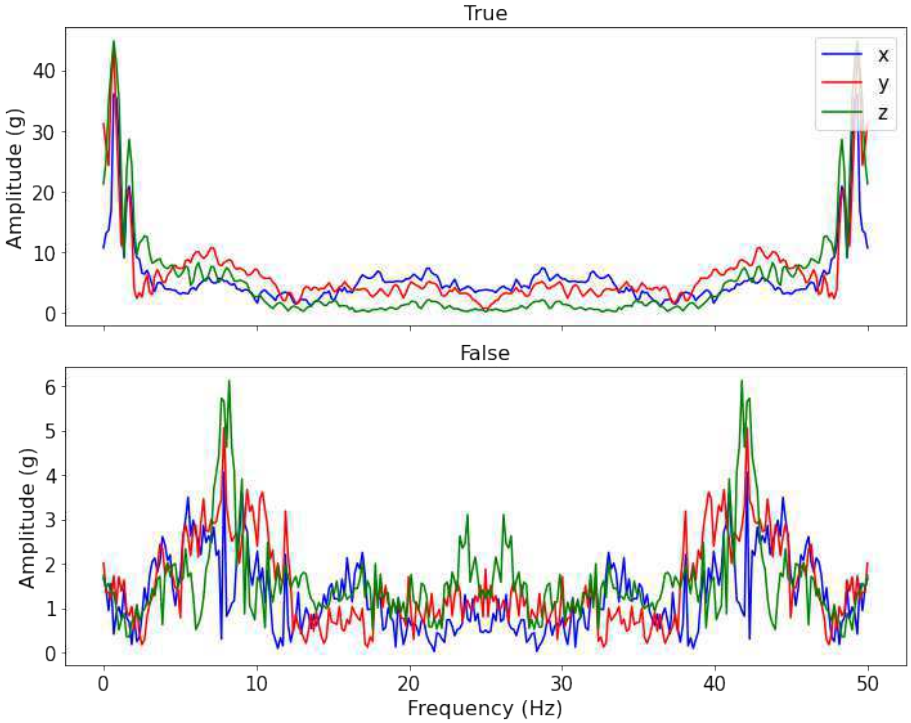

Figura 2.25: Classe vera e classe falsa Microbox 1D con rotazioni nel dominio della frequenza

L'80% del dataset è stato utilizzato per il training e il 20% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 60 epoche.
In Figure 2.26 e 2.27 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

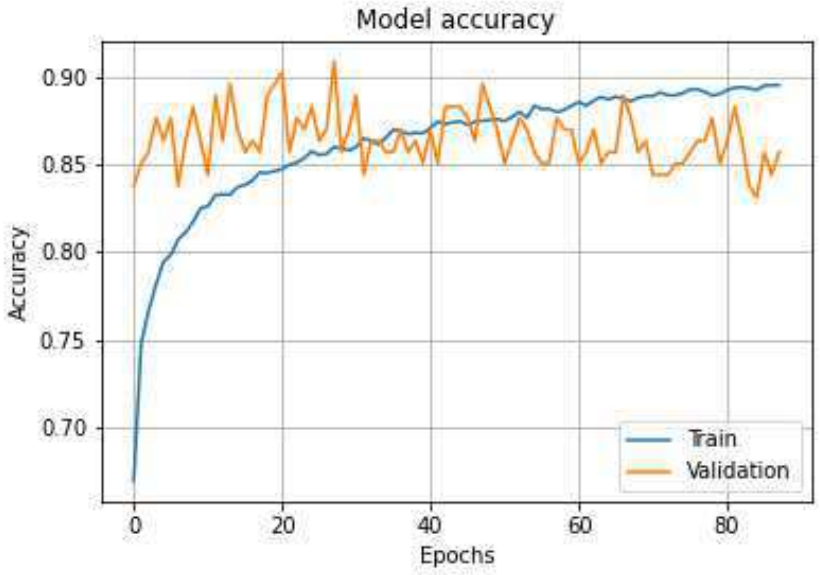

Figura 2.26: Train e validation accuracy Microbox 1D con rotazioni nel dominio della frequenza

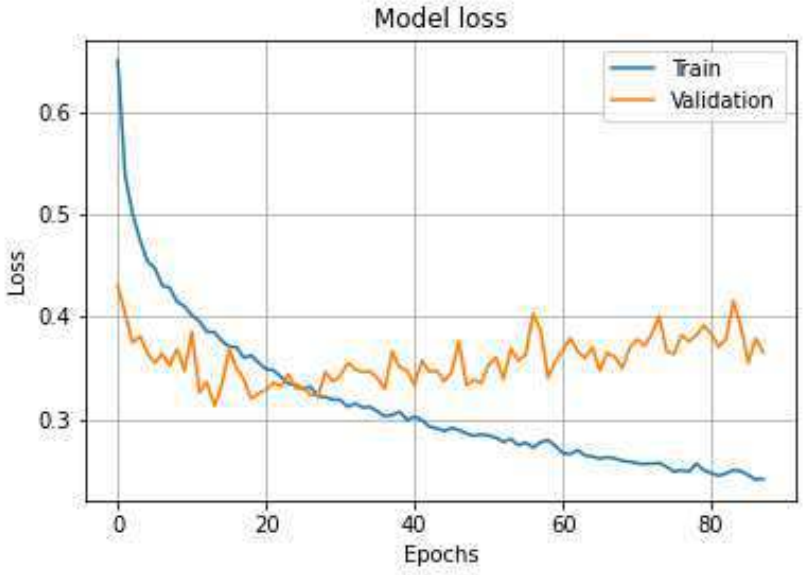

Figura 2.27: Train e validation loss Microbox 1D con rotazioni nel dominio della frequenza

Si nota che il numero di epoche è molto inferiore agli altri casi, dovuto alla data augmentation che moltiplica per 10 le dimensioni del set di training.

2.5.2 Infobox

Per Infobox vengono presi in considerazione prima tutti e tre gli assi, e dopo la rotazione spaziale, solamente gli assi x ed y.
Di seguito una parte di codice utilizzata per l'applicazione delle rotazioni alle serie del set di training del dataset Infobox:

```
# Data augmentation for ds_true train (only x and y)
ds_aug = ds_true_no_aug[0,:,:2].copy()
ds_true_aug = []
for j in range(0, ds_true_no_aug.shape[0]):
  randomlist = []
  while len(randomlist) < rotations: # Fill with random rotations
    n = random.randint(0,len(R))
    if n not in randomlist:
      randomlist.append(n)
  for k in range (0, len(randomlist)):
    if randomlist[k] == len(R): # Null rotation
      ds_aug = ds_true_no_aug[j,:,:2]

      fft_x = fft(ds_aug[:,0])
      fft_y = fft(ds_aug[:,1])

      d_x = np.absolute(fft_x)
      d_y = np.absolute(fft_y)

      ds_entry_fft = np.stack([d_x, d_y], axis=1)

    else: # Apply rotations
      for i in range(0, ds_true_no_aug.shape[1]):
        v1 = np.array([[ds_true_no_aug[j,i,0]],
                       [ds_true_no_aug[j,i,1]],
                       [ds_true_no_aug[j,i,2]]])
        v2 = R[randomlist[k]] * v1
        ds_aug[i, 0] = v2[0,0]
        ds_aug[i, 1] = v2[1,0]

      fft_x = fft(ds_aug[:,0])
      fft_y = fft(ds_aug[:,1])

      d_x = np.absolute(fft_x)
      d_y = np.absolute(fft_y)

      ds_entry_fft = np.stack([d_x, d_y], axis=1)

    ds_true_aug.append(ds_entry_fft)
# Convert list to a numpy array. Cast to float32 to avoid errors on fit
ds_true = np.asarray(ds_true_aug).astype('float32')
```

I dati di input di dimensioni 1200x2 sono stati utilizzati con la rete neurale 1D CNN Infobox.
In Figura 2.28 si può vedere un esempio di serie appartenente alla classe vera e una appartenente alla classe falsa.

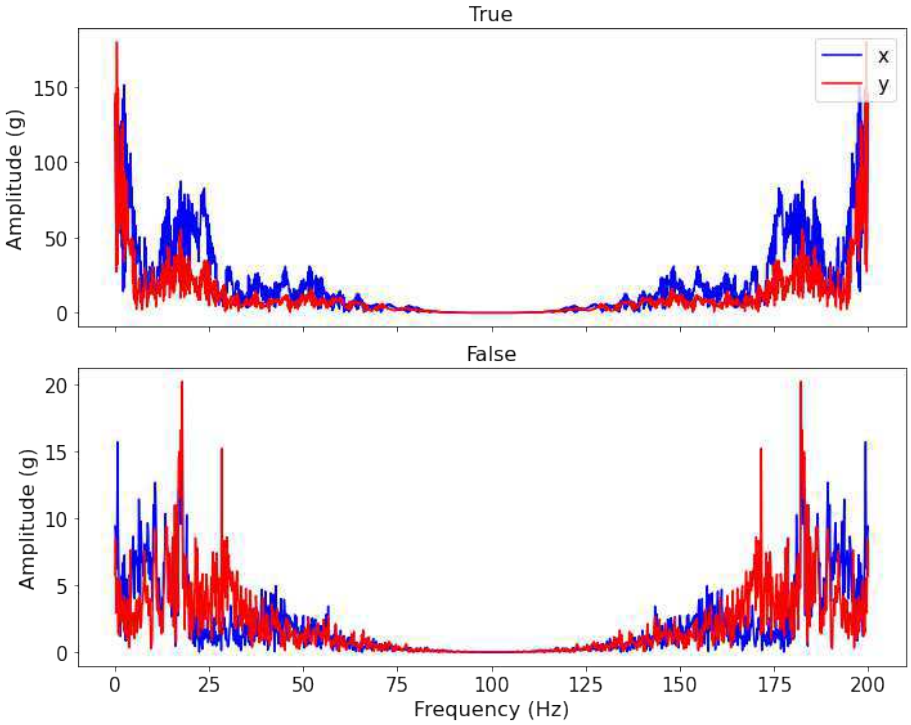

Figura 2.28: Classe vera e classe falsa Infobox 1D con rotazioni nel dominio della frequenza

Il 95% del dataset è stato utilizzato per il training e il 5% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 32.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 100 epoche.
In Figure 2.29 e 2.30 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

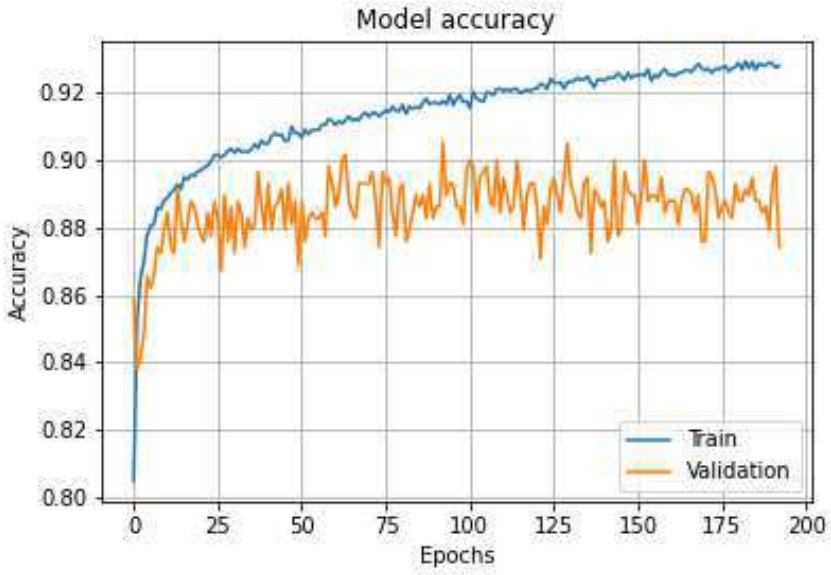

Figura 2.29: Train e validation accuracy Infobox 1D con rotazioni nel dominio della frequenza

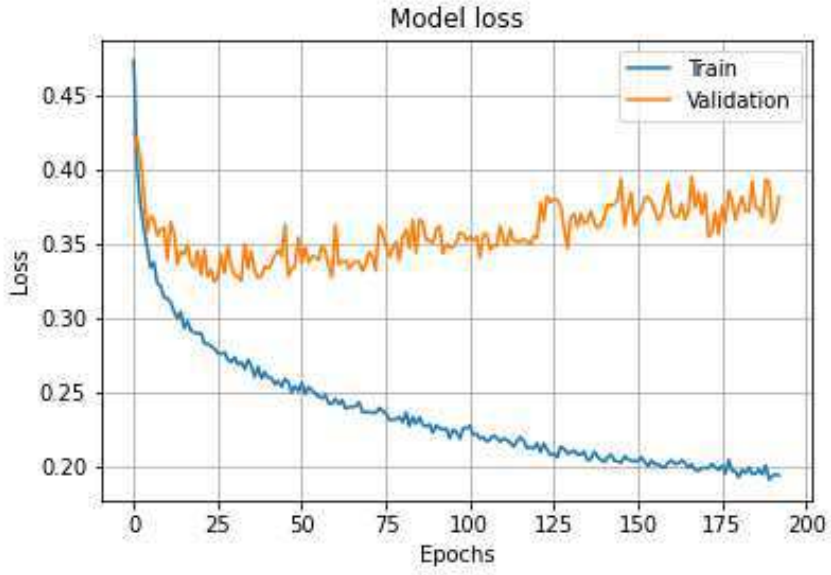

Figura 2.30: Train e validation loss Infobox 1D con rotazioni nel dominio della frequenza

In questo caso, le epoche necessarie sono paragonabili agli esperimenti precedenti.

2.6 Esperimenti con trasformazioni 2D delle serie temporali

Gli ultimi esperimenti riguardano l'utilizzo di metodi 2D per la classificazione. Sono stati utilizzati diversi metodi per la trasformazione delle serie temporali in immagini, con lo scopo di migliorare i risultati ottenuti con la classificazione monodimensionale. Questo comporta ovviamente avere modelli molto più pesanti.
Per questo motivo, questi esperimenti sono stati effettuati utilizzando solamente il dataset proveniente da Microbox.
Per tutti gli esperimenti descritti di seguito, è stato utilizzato l'80% del dataset per il training e il 20% per il test. Del set di training, il 10% è stato utilizzato per la validation. Il training è stato effettuato utilizzando un batch size di 16.
É stato utilizzato un early stopping che tiene in considerazione la validation accuracy come indicatore per lo stop, con un delta minimo di 0 e una patience di 30 epoche.

2.6.1 Recurrence plot

Il primo degli esperimenti 2D utilizza il recurrence plot per rappresentare le serie temporali come immagini.
Un recurrence plot è un diagramma che mostra, per ogni istante i nel tempo, gli istanti in cui una traiettoria dello spazio delle fasi visita all'incirca la stessa area nello spazio delle fasi dell'istante j. Lo spazio delle fasi è uno spazio in cui sono rappresentati tutti i possibili stati di un sistema, con ogni possibile stato corrispondente a un punto univoco.
Può essere espresso come (Formula 2.6):

$$\vec{x}(i) \approx \vec{x}(j) \tag{2.6}$$

mostrando i sull'asse orizzontale e j sull'asse verticale, dove $\vec{x}$ è una traiettoria dello spazio delle fasi.
Se $\vec{x}$ rappresenta la traiettoria dello spazio delle fasi della serie temporale relativa all'asse x, allora $\vec{y}$ e $\vec{z}$ rappresentano, rispettivamente, le traiettorie dello spazio delle fasi delle serie temporali relative agli assi y e z.
Di seguito delle parti di codice utilizzate per il caricamento del dataset e per la realizzazione del recurrence plot:

```
def get_dataset(path, threshold=-1):
  entry_list = []
  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)

    loaded_entry = ds_entry.values[1:3000,4:7]
    # Subsample 1 every 30 samples
    ds_entry_processed = loaded_entry[::30].copy()

    d_x = recurrence_plot(ds_entry_processed[:,0,None], None, threshold)
    d_y = recurrence_plot(ds_entry_processed[:,1,None], None, threshold)
    d_z = recurrence_plot(ds_entry_processed[:,2,None], None, threshold)

    d_all = np.stack([d_x, d_y, d_z], axis=2)
    entry_list.append(d_all)
  ds_array = np.stack(entry_list)

  return ds_array

def recurrence_plot(s1, threshold):
  eps=0.1 # Scaling factor
  d = sklearn.metrics.pairwise.pairwise_distances(s1)
  d = np.floor(d / eps)
  if threshold > 0:
    d = np.where(d > threshold,1,0)
  return d
```

I dati di input di dimensioni 100x100x3 sono stati utilizzati con la rete neurale 2D CNN Microbox descritta sopra.
In Figura 2.31 si può vedere degli esempi di recurrence plot appartenenti alla classe vera e alla classe falsa.

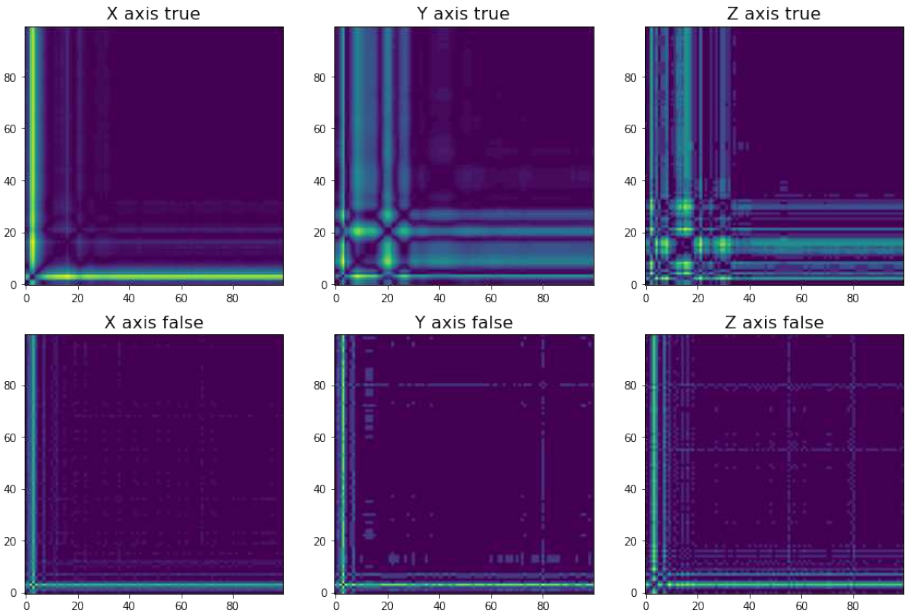

Figura 2.31: Recurrence plot, classe vera e classe falsa

In Figure 2.32 e 2.33 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

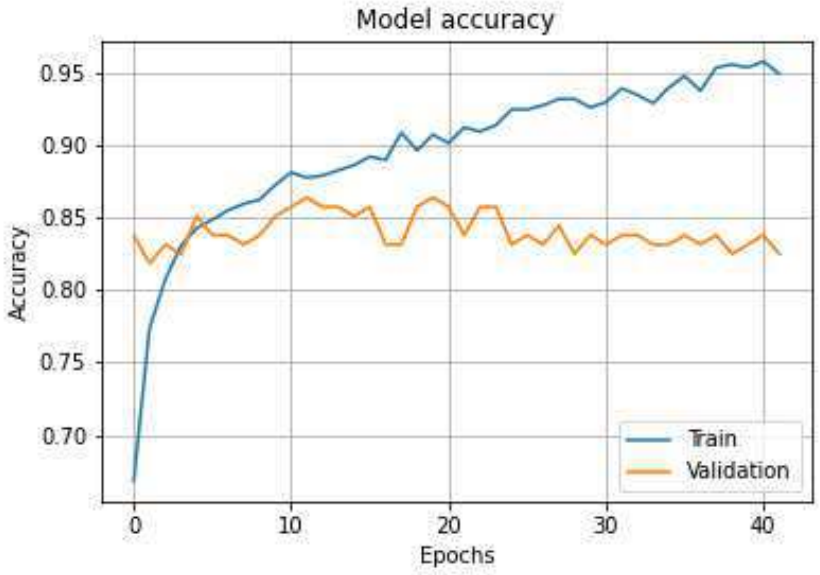

Figura 2.32: Train e validation accuracy recurrence plot

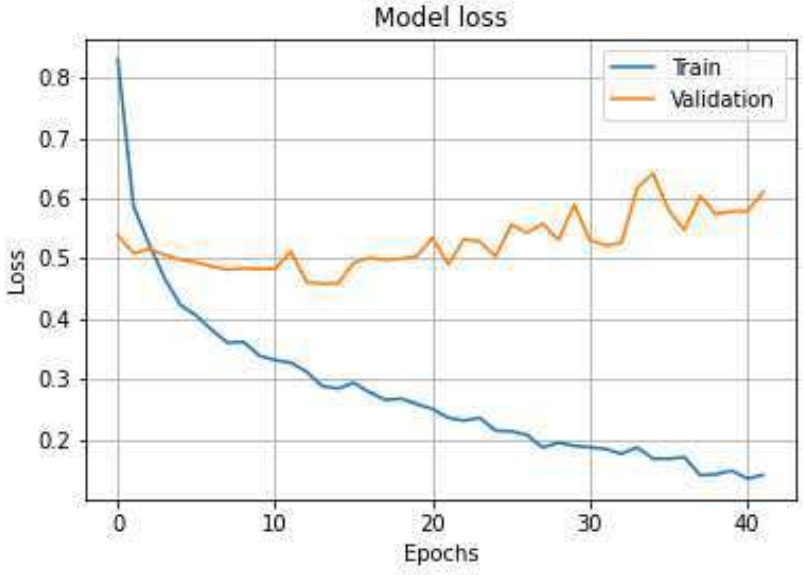

Figura 2.33: Train e validation loss recurrence plot

Si può notare che le epoche necessarie alla convergenza sono molto poche rispetto

ai casi monodimensionali.

2.6.2 Recurrence plot accoppiato

Nel secondo esperimento 2D è stato utilizzato un recurrence plot accoppiato per rappresentare le serie temporali come immagini.
Un recurrence plot accoppiato è un recurrence plot che non utilizza la stessa traiettoria dello spazio delle fasi per gli istanti i e j.
Per l'asse x è stata usata $\vec{x}$ per i e $\vec{y}$ per j (Formula 2.7):

$$\vec{x}(i) \approx \vec{y}(j) \tag{2.7}$$

per l'asse y è stata usata $\vec{y}$ per i e $\vec{z}$ per j (Formula 2.8):

$$\vec{y}(i) \approx \vec{z}(j) \tag{2.8}$$

e infine per l'asse z è stata usata $\vec{x}$ per i e $\vec{z}$ per j (Formula 2.9):

$$\vec{x}(i) \approx \vec{z}(j) \tag{2.9}$$

Di seguito delle parti di codice utilizzate per il caricamento del dataset e per la realizzazione del recurrence plot accoppiato:

```
def get_dataset(path, threshold=-1):
  entry_list = []
  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)

    loaded_entry = ds_entry.values[1:3000,4:7]
    # Subsample 1 every 30 samples
    ds_entry_processed = loaded_entry[::30].copy()

    d_x = recurrence_plot(ds_entry_processed[:,0,None],
                          ds_entry_processed[:,1,None], threshold) # X-Y
    d_y = recurrence_plot(ds_entry_processed[:,1,None],
                          ds_entry_processed[:,2,None], threshold) # Y-Z
    d_z = recurrence_plot(ds_entry_processed[:,0,None],
                          ds_entry_processed[:,2,None], threshold) # X-Z

    d_all = np.stack([d_x, d_y, d_z], axis=2)
    entry_list.append(d_all)
  ds_array = np.stack(entry_list)

  return ds_array
```

```
def recurrence_plot(s1, s2, threshold):
  eps=0.1 # Scaling factor
  # Paired recurrence matrix
  d = sklearn.metrics.pairwise.pairwise_distances(s1, s2)
  d = np.floor(d / eps)
  if threshold > 0:
    d = np.where(d > threshold,1,0)
  return d
```

I dati di input di dimensioni 100x100x3 sono stati utilizzati con la rete neurale 2D CNN Microbox.
In Figura 2.34 si può vedere degli esempi di recurrence plot accoppiato appartenenti alla classe vera e alla classe falsa.

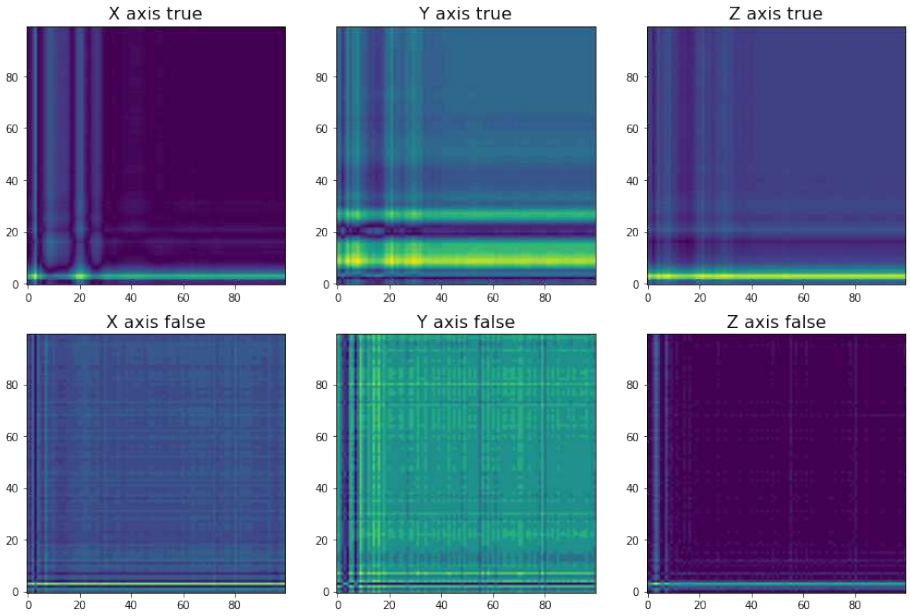

Figura 2.34: Recurrence plot accoppiato, classe vera e classe falsa

In Figure 2.35 e 2.36 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

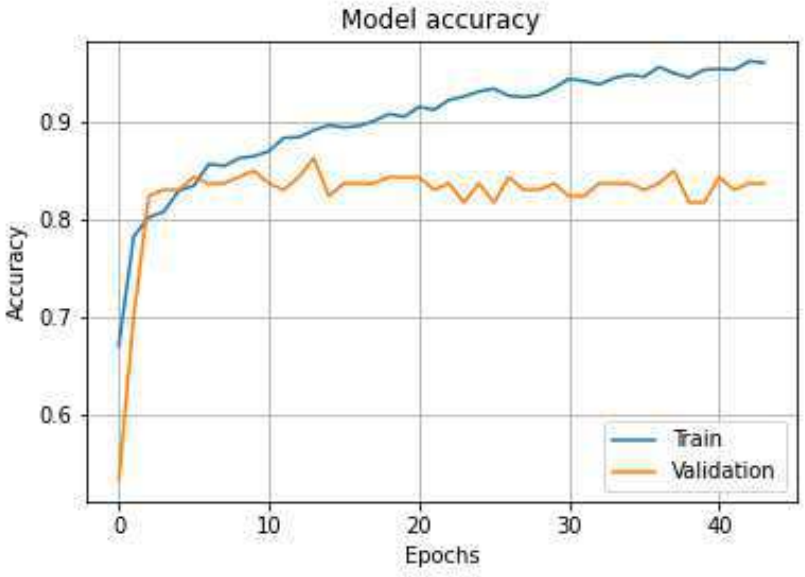

Figura 2.35: Train e validation accuracy recurrence plot accoppiato

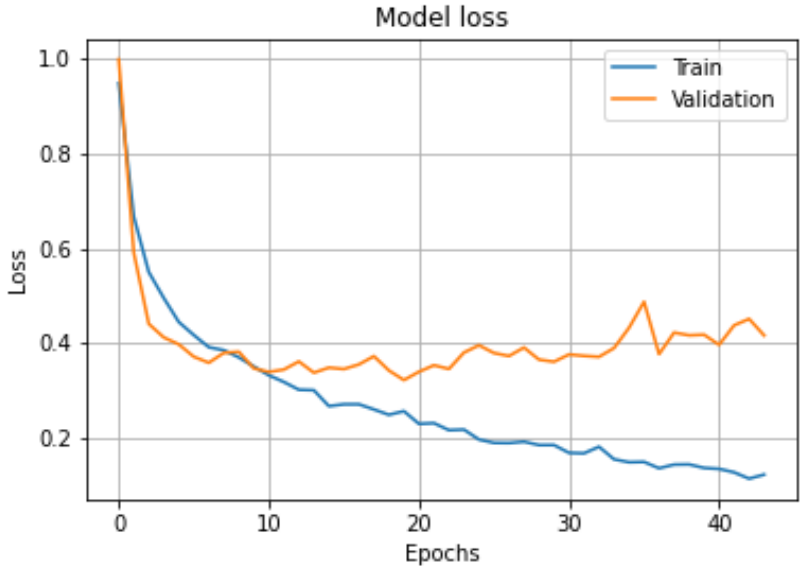

Figura 2.36: Train e validation loss recurrence plot accoppiato

I tempi di convergenza sono simili al recurrence plot non accoppiato.

2.6.3 Spettrogramma

Il terzo degli esperimenti 2D utilizza uno spettrogramma per rappresentare le serie temporali come immagini.
Uno spettrogramma è la rappresentazione grafica dell'intensità di un segnale in funzione del tempo e della frequenza o, in altre parole, è la rappresentazione grafica di $x(t)$ nelle variabili reali t ed ω (Formula 2.10):

$$f : x(t, \omega) \tag{2.10}$$

t rappresenta il tempo e ω la frequenza, con $x(t)$ funzione reale che rappresenta la serie temporale relativa all'asse x. Analogamente, $y(t)$ e $z(t)$ rappresentano le serie temporali relative agli assi y e z.
Il tempo viene visualizzato sull'asse orizzontale, mentre la frequenza sull'asse verticale.
Di seguito delle parti di codice utilizzate per il caricamento del dataset e per la realizzazione dello spettrogramma:

```
def get_dataset(path):
  entry_list = []
  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)

    loaded_entry = ds_entry.values[1:3000,4:7]
    # No subsample
    ds_entry_processed = loaded_entry[::1].copy()

    d_x = specg(ds_entry_processed[:,0].astype(float))
    d_y = specg(ds_entry_processed[:,1].astype(float))
    d_z = specg(ds_entry_processed[:,2].astype(float))

    d_all = np.stack([d_x, d_y, d_z], axis=2)
    entry_list.append(d_all)
  ds_array = np.stack(entry_list)

  return ds_array
```

```
def specg(s):
  eps = 1e-10
  fs = 1e3
  f, t, S = signal.spectrogram(s,
                               fs,
                               window = 'hamming',
                               nperseg = 128,
                               noverlap = 96,
                               nfft = 128)
  return np.log(S + eps)
```

I dati di input di dimensioni 65x90x3 sono stati utilizzati con la rete neurale 2D CNN Microbox.
In Figura 2.37 si può vedere degli esempi di spettrogramma appartenenti alla classe vera e alla classe falsa.

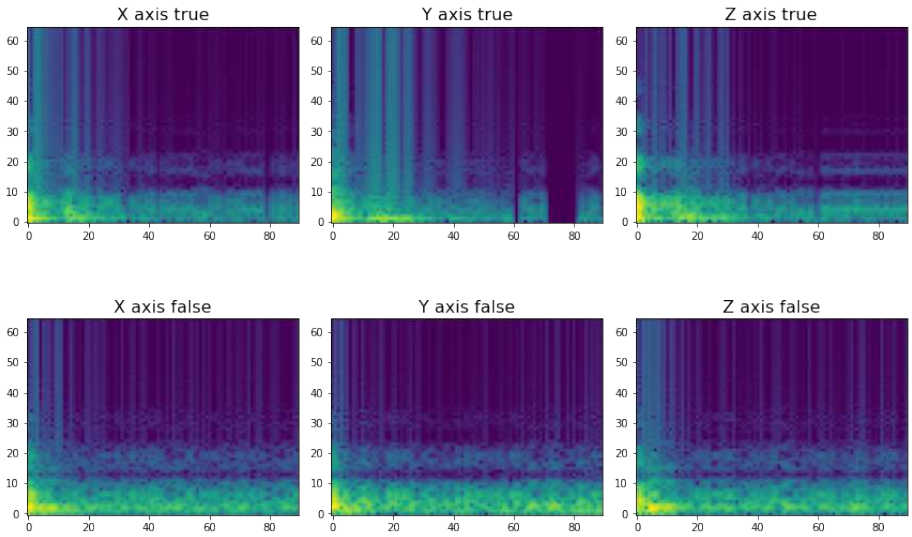

Figura 2.37: Spettrogramma, classe vera e classe falsa

In Figure 2.38 e 2.39 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

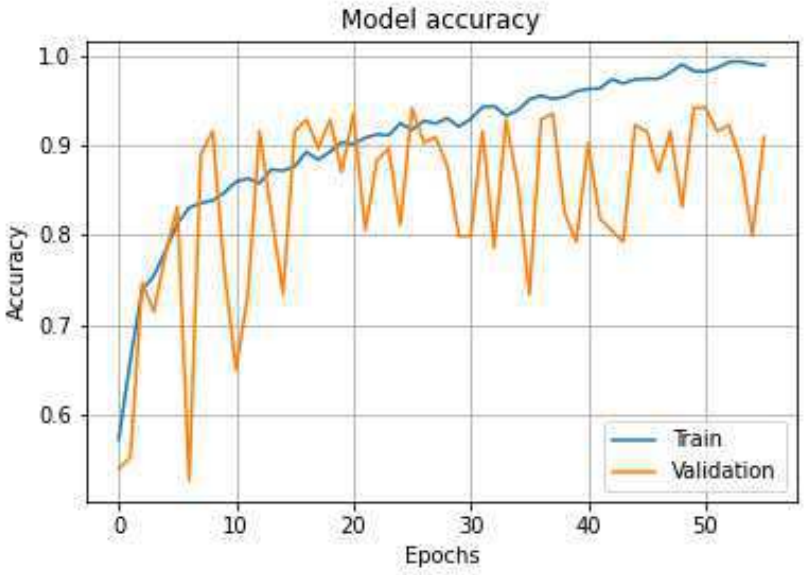

Figura 2.38: Train e validation accuracy spettrogramma

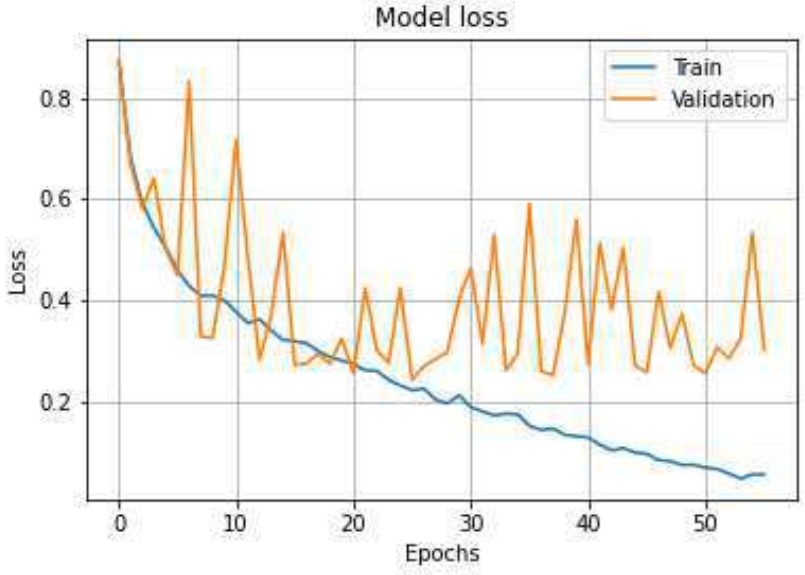

Figura 2.39: Train e validation loss spettrogramma

Il numero di epoche impiegato per la convergenza è simile ai casi precedenti, ma

l'andamento delle accuracy e delle loss è molto più frastagliato.

2.6.4 Scalogramma

L'ultimo degli esperimenti 2D utilizza uno scalogramma per rappresentare le serie temporali come immagini.
Uno scalogramma è la rappresentazione grafica dell'intensità di un segnale in funzione del tempo e della scala, una pseudo frequenza ottenuta tramite la trasformata Wavelet continua. Il tempo viene visualizzato sull'asse orizzontale, mentre la scala sull'asse verticale.
La trasformata Wavelet continua è una rappresentazione di un segnale mediante l'uso di una forma d'onda oscillante di lunghezza finita o a decadimento rapido (nota come wavelet madre). Questa forma d'onda è scalata e traslata per adattarsi al segnale in ingresso (da cui l'ottenimento di un segnale complesso in funzione del tempo τ e della scala s) (Equazione 2.11):

$$\mathcal{CWT}(\tau, s) = \frac{1}{\sqrt{s}} \int_{-\infty}^{+\infty} f(t)\Psi_\sigma(\frac{t-\tau}{s})\, dt \tag{2.11}$$

Ψ_σ è la wavelet madre. É stata utilizzata la wavelet di Morlet, una wavelet composta da una parte complessa esponenziale (portante) moltiplicata per una finestra gaussiana (inviluppo) (Equazione 2.12):

$$\Psi_\sigma(t) = c_\sigma \pi^{-\frac{1}{4}} e^{-\frac{1}{2}t^2}(e^{i\sigma t} - \kappa_\sigma) \tag{2.12}$$

É stata utilizzata la libreria SsqueezePy [11] per l'implementazione della trasformata Wavelet continua.
Dopo l'esecuzione delle trasformata Wavelet continua è stato applicato il modulo della trasformata punto per punto, per ottenere così una trasformata reale.
Di seguito una parte di codice utilizzata per il caricamento del dataset e per la realizzazione dello scalogramma:

```
def get_dataset(path):
  entry_list = []
  files_list = glob.glob(os.path.join(path, "*.csv"))
  # Load every csv inside path as a numpy matrix, preprocess and create a list
  for filename in tqdm(files_list):
    ds_entry = pd.read_csv(filename, index_col=None, header=0)

    loaded_entry = ds_entry.values[1:3000,4:7]
    # Subsample 1 every 10 samples
    ds_entry_processed = loaded_entry[::10].copy()

    W_x, scales_x = cwt(ds_entry_processed[:,0].astype(float), 'morlet')
    W_y, scales_y = cwt(ds_entry_processed[:,1].astype(float), 'morlet')
    W_z, scales_z = cwt(ds_entry_processed[:,2].astype(float), 'morlet')

    d_x = np.absolute(W_x)
    d_y = np.absolute(W_y)
    d_z = np.absolute(W_z)

    d_all = np.stack([d_x, d_y, d_z], axis=2)
    entry_list.append(d_all)
  ds_array = np.stack(entry_list)

  return ds_array
```

I dati di input di dimensioni 222x300x3 sono stati utilizzati con la rete neurale 2D CNN Microbox.
In Figura 2.40 si può vedere degli esempi di scalogramma appartenenti alla classe vera e alla classe falsa.

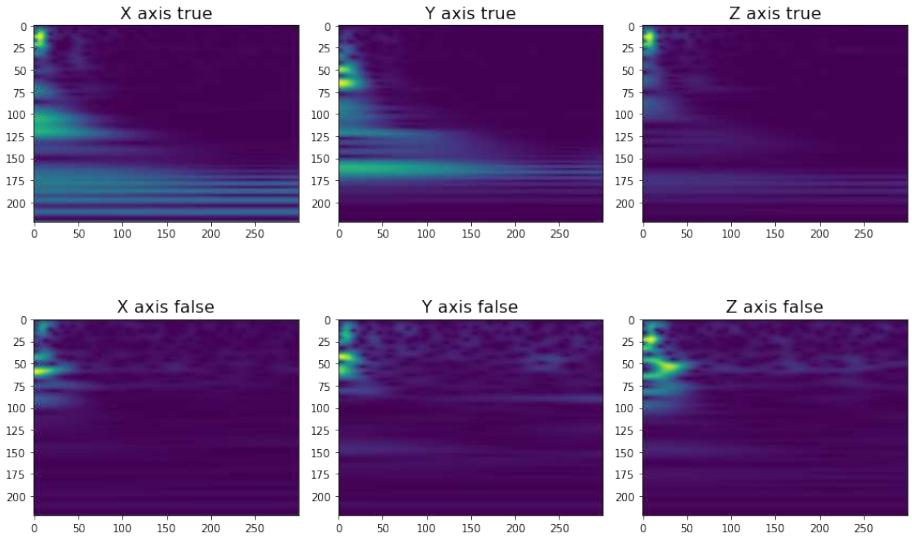

Figura 2.40: Scalogramma, classe vera e classe falsa

In Figure 2.41 e 2.42 si possono vedere le accuracy di train e di validation e le loss di train e di validation.

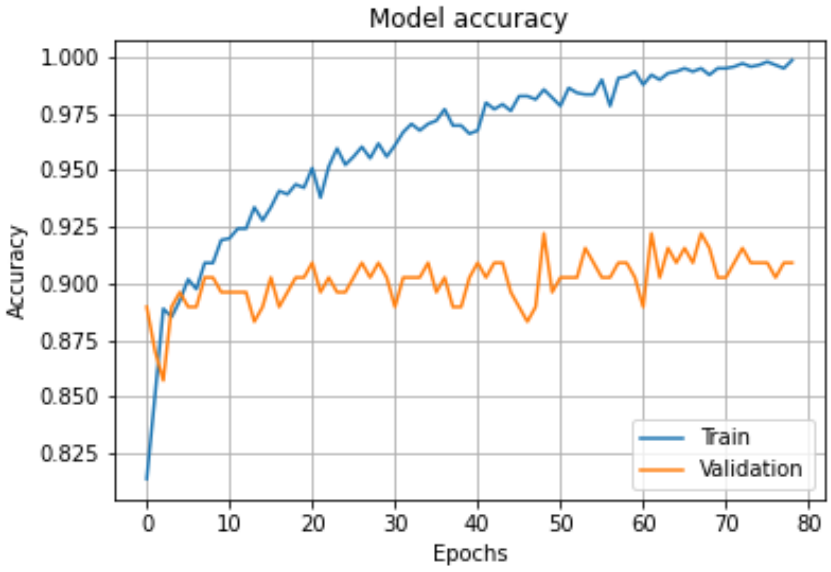

Figura 2.41: Train e validation accuracy scalogramma

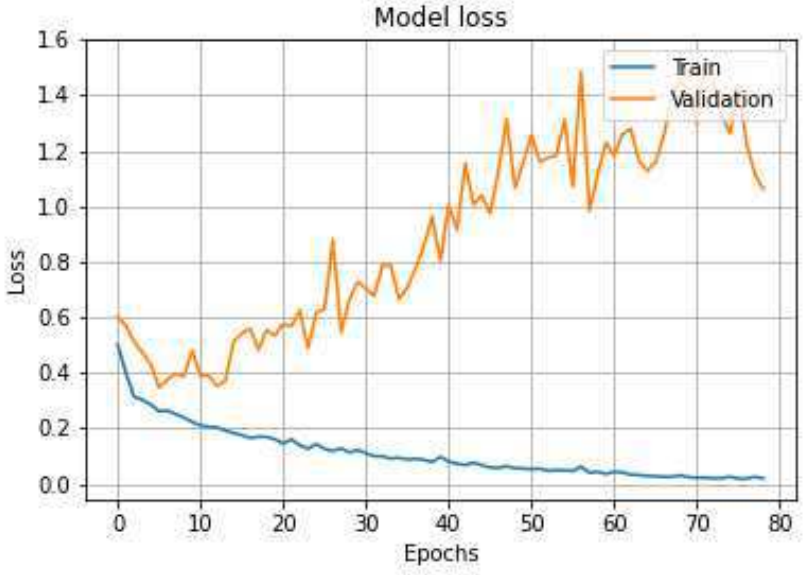

Figura 2.42: Train e validation loss scalogramma

Il numero di epoche impiegato per la convergenza è leggermente più alto dei casi precedenti ma comunque contenuto, e si ha un andamento delle loss e delle accuracy abbastanza frastagliato.

Capitolo 3

Fase di test e confronto dei risultati

In questo capitolo verranno mostrati i risultati migliori ottenuti dagli esperimenti elencati nel capitolo precedente, e verrà poi effettuato un confronto tra i modelli ottenuti.
Di seguito una parte di codice utilizzata per l'esecuzione del test, per il calcolo delle metriche, e per il plot della matrice di conclusione e il salvataggio dei risultati su disco:

```
model = keras.models.load_model(results_path + '/' + model_name + '.h5')
model.summary()
results = model.evaluate(X_test, y_test)

print('\nTest Loss, Test Accuracy, Test AUC:', results)

# Generate confusion matrix and compute metrics
y_pred = model.predict(X_test)
y_pred = y_pred.argmax(axis=1)
conf_mat = confusion_matrix(y_test.argmax(axis=1), y_pred)
tn, fp, fn, tp = conf_mat.ravel()
precision = precision_score(y_test.argmax(axis=1), y_pred)
recall = recall_score(y_test.argmax(axis=1), y_pred)
f1_score = f1_score(y_test.argmax(axis=1), y_pred)

print('\nPrecision: ' + str(precision))
print('Recall: ' + str(recall))
print('F1-score: ' + str(f1_score))
print('\nTrue Negatives: ' + str(tn))
print('False Positives: ' + str(fp))
print('False Negatives: ' + str(fn))
print('True Positives: ' + str(tp) + '\n')

fig = plt.figure()
sns.heatmap(conf_mat, annot=True, fmt="d",cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion matrix')
fig.savefig(results_path + '/' + model_name + '_confusion_matrix.png')
plt.show()

data = {'a': [results[0], results[1], results[2], precision, recall,
             f1_score, tn, fp, fn, tp]}
df = pd.DataFrame(data=data, index=['Test Loss', 'Test Accuracy',
                                    'Test AUC', 'Precision', 'Recall',
                                    'F1-score', 'True Negatives',
                                    'False Positives', 'False Negatives',
                                    'True Positives'])
df.to_csv(results_path + '/' + model_name + '_results.csv', header=False)
```

3.1 Metriche

Di seguito verranno brevemente elencate le metriche utilizzate per la valutazione dei modelli ottenuti.

Verranno rappresentati i veri positivi, i veri negativi, i falsi positivi e i falsi negativi, rispettivamente, con TP, TN, FP, FN.

- Accuracy: la metrica più intuitiva, è semplicemente un rapporto tra il numero di campioni classificati correttamente e il numero di campioni totali (Formula 3.1):

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.1}$$

- Precision: è la percentuale delle previsioni positive corrette sul totale delle previsioni positive del modello (Formula 3.2):

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

- Recall: è la percentuale delle previsioni positive corrette sul totale delle istanze positive (Formula 3.3):

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

- F1-score: è la media armonica delle metriche Precision e Recall (Formula 3.4):

$$F1 - Score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{3.4}$$

- AUC (Area Under the Curve): è un parametro dato dall'area sottesa dalla curva ROC (Receiver Operating Characteristic), uno schema grafico per classificatore binario. Più la curva abbraccia l'angolo in alto a sinistra (sintomo di buona classificazione), più l'AUC è maggiore.

3.2 1D nel dominio del tempo

In questa sezione verranno esposti i risultati relativi agli esperimenti con serie monodimensionali nel dominio del tempo.

3.2.1 Microbox

I risultati ottenuti con l'utilizzo del dataset Microbox e il numero di parametri del modello sono esposti in Tabella 3.1.

Accuracy	0.9038
Precision	0.9333
Recall	0.8704
F1-score	0.9008
AUC	0.9324
Numero di parametri	15727

Tabella 3.1: Risultati e numero di parametri del modello Microbox 1D nel dominio del tempo

In Figura 3.1 si può vedere la matrice di confusione relativa.

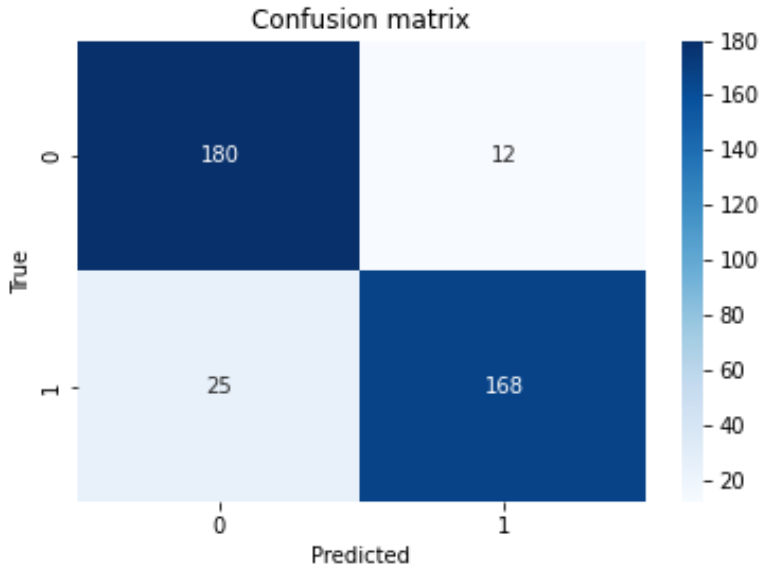

Figura 3.1: Matrice di confusione del modello Microbox 1D nel dominio del tempo

3.2.2 Infobox

I risultati ottenuti con l'utilizzo del dataset Infobox e il numero di parametri del modello sono esposti in Tabella 3.2.

Accuracy	0.9213
Precision	0.9520
Recall	0.8910
F1-score	0.9205
AUC	0.9654
Numero di parametri	114812

Tabella 3.2: Risultati e numero di parametri del modello Infobox 1D nel dominio del tempo

In Figura 3.2 si può vedere la matrice di confusione relativa.

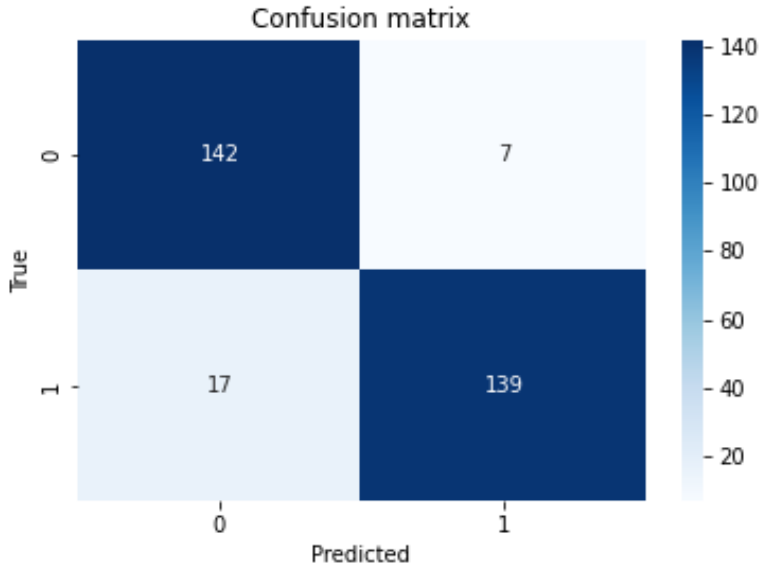

Figura 3.2: Matrice di confusione del modello Infobox 1D nel dominio del tempo

3.3 1D nel dominio della frequenza

In questa sezione verranno esposti i risultati relativi agli esperimenti con serie monodimensionali nel dominio della frequenza.

3.3.1 Microbox

I risultati ottenuti con l'utilizzo del dataset Microbox e il numero di parametri del modello sono esposti in Tabella 3.3.

Accuracy	0.8909
Precision	0.9252
Recall	0.8473
F1-score	0.8846
AUC	0.9410
Numero di parametri	15727

Tabella 3.3: Risultati e numero di parametri del modello Microbox 1D nel dominio della frequenza

In Figura 3.3 si può vedere la matrice di confusione relativa.

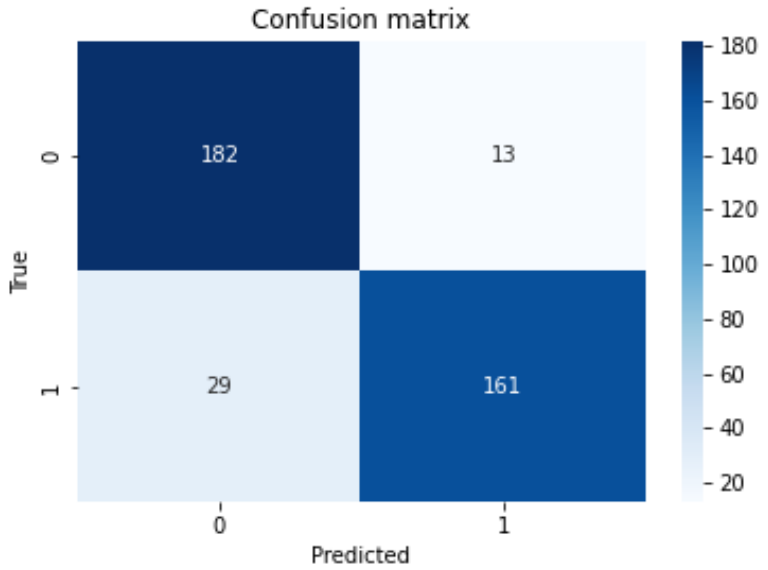

Figura 3.3: Matrice di confusione del modello Microbox 1D nel dominio della frequenza

3.3.2 Infobox

I risultati ottenuti con l'utilizzo del dataset Infobox e il numero di parametri del modello sono esposti in Tabella 3.4.

Accuracy	0.9180
Precision	0.9300
Recall	0.8986
F1-score	0.9140
AUC	0.9510
Numero di parametri	114812

Tabella 3.4: Risultati e numero di parametri del modello Infobox 1D nel dominio della frequenza

In Figura 3.4 si può vedere la matrice di confusione relativa.

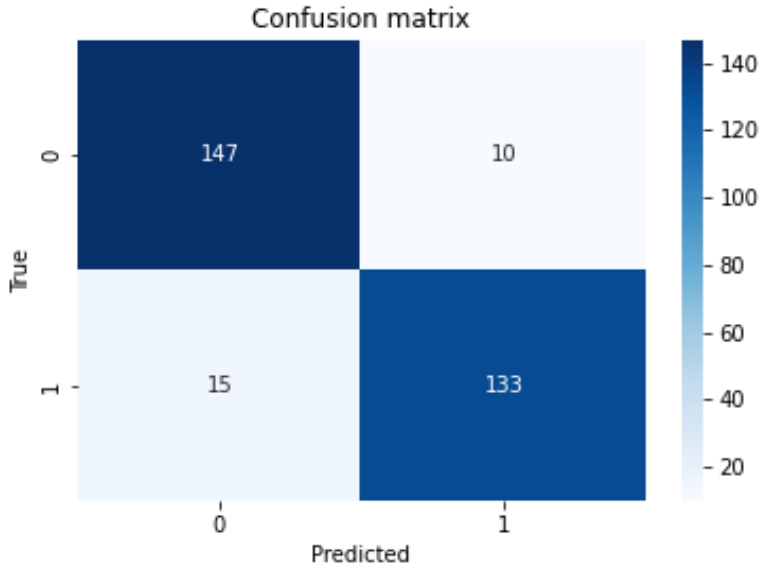

Figura 3.4: Matrice di confusione del modello Infobox 1D nel dominio della frequenza

3.4 1D con norma nel dominio del tempo

In questa sezione verranno esposti i risultati relativi agli esperimenti con serie monodimensionali e con l'applicazione della norma nel dominio del tempo.

3.4.1 Microbox

I risultati ottenuti con l'utilizzo del dataset Microbox e il numero di parametri del modello sono esposti in Tabella 3.5.

Accuracy	0.8571
Precision	0.8631
Recall	0.8497
F1-score	0.8563
AUC	0.9299
Numero di parametri	13727

Tabella 3.5: Risultati e numero di parametri del modello Microbox 1D nel dominio del tempo con applicazione della norma

In Figura 3.5 si può vedere la matrice di confusione relativa.

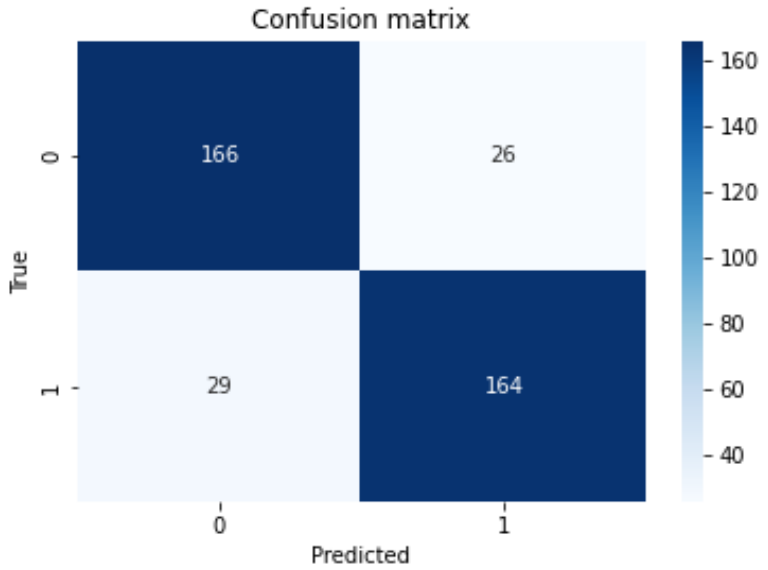

Figura 3.5: Matrice di confusione del modello Microbox 1D nel dominio del tempo con applicazione della norma

3.4.2 Infobox

I risultati ottenuti con l'utilizzo del dataset Infobox e il numero di parametri del modello sono esposti in Tabella 3.6.

Accuracy	0.9147
Precision	0.9316
Recall	0.9090
F1-score	0.9202
AUC	0.9576
Numero di parametri	112612

Tabella 3.6: Risultati e numero di parametri del modello Infobox 1D nel dominio del tempo con applicazione della norma

In Figura 3.6 si può vedere la matrice di confusione relativa.

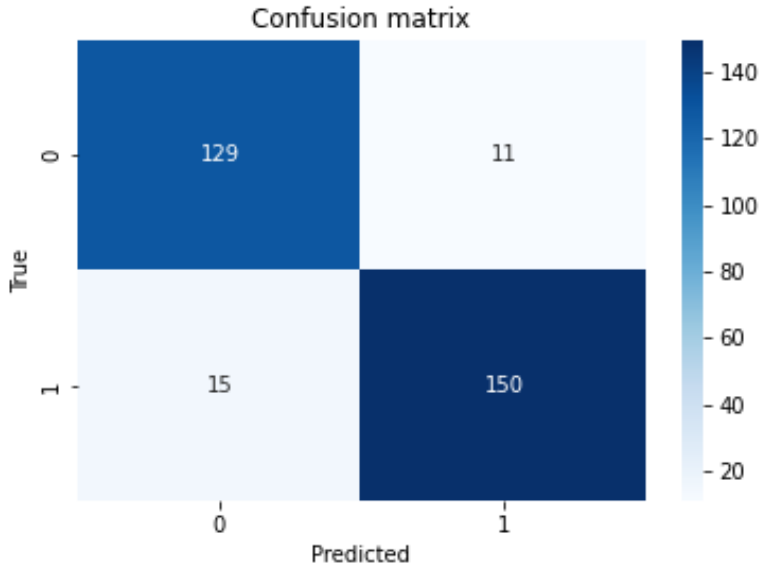

Figura 3.6: Matrice di confusione del modello Infobox 1D nel dominio del tempo con applicazione della norma

3.5 1D con norma nel dominio della frequenza

In questa sezione verranno esposti i risultati relativi agli esperimenti con serie monodimensionali e con l'applicazione della norma nel dominio della frequenza.

3.5.1 Microbox

I risultati ottenuti con l'utilizzo del dataset Microbox e il numero di parametri del modello sono esposti in Tabella 3.7.

Accuracy	0.8363
Precision	0.8603
Recall	0.8020
F1-score	0.8301
AUC	0.9224
Numero di parametri	13727

Tabella 3.7: Risultati e numero di parametri del modello Microbox 1D nel dominio della frequenza con applicazione della norma

In Figura 3.7 si può vedere la matrice di confusione relativa.

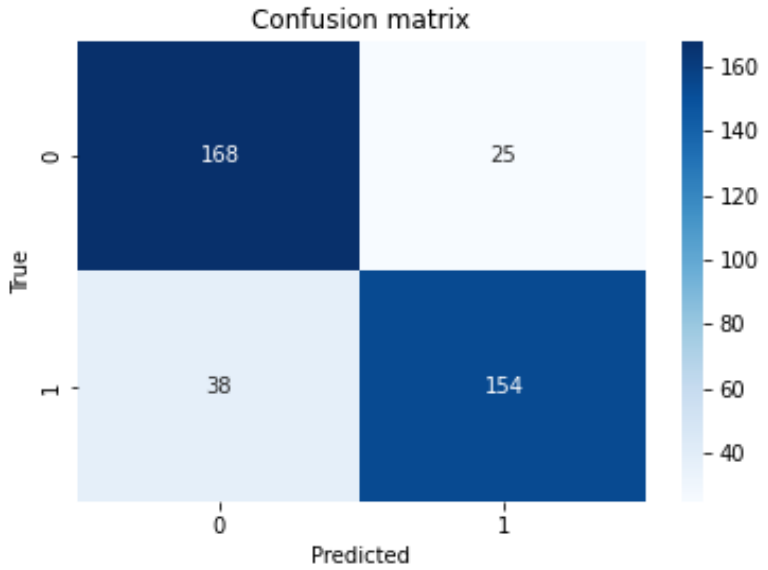

Figura 3.7: Matrice di confusione del modello Microbox 1D nel dominio della frequenza con applicazione della norma

3.5.2 Infobox

I risultati ottenuti con l'utilizzo del dataset Infobox e il numero di parametri del modello sono esposti in Tabella 3.8.

Accuracy	0.8983
Precision	0.8857
Recall	0.8920
F1-score	0.8888
AUC	0.9384
Numero di parametri	112612

Tabella 3.8: Risultati e numero di parametri del modello Infobox 1D nel dominio della frequenza con applicazione della norma

In Figura 3.8 si può vedere la matrice di confusione relativa.

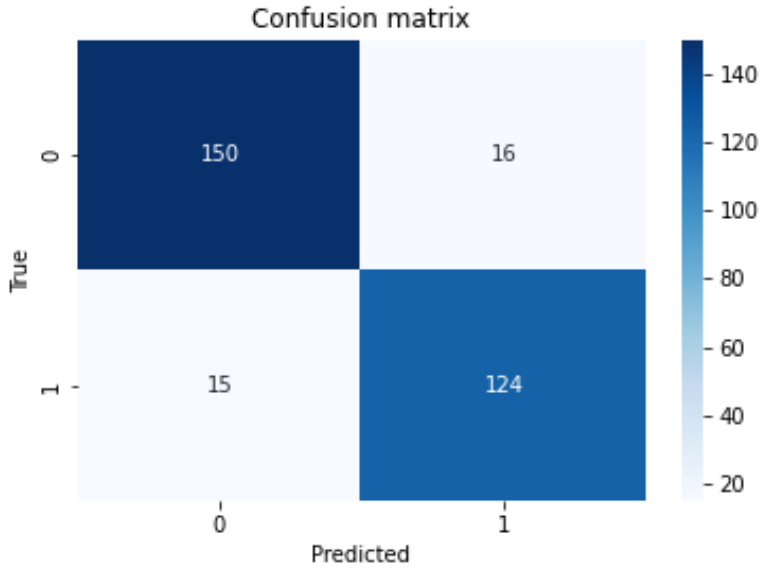

Figura 3.8: Matrice di confusione del modello Infobox 1D nel dominio della frequenza con applicazione della norma

3.6 1D con rotazioni nel dominio della frequenza

In questa sezione verranno esposti i risultati relativi agli esperimenti con serie monodimensionali ruotate nel dominio della frequenza.

3.6.1 Microbox

I risultati ottenuti con l'utilizzo del dataset Microbox e il numero di parametri del modello sono esposti in Tabella 3.9.

Accuracy	0.8989
Precision	0.9277
Recall	0.8652
F1-score	0.8954
AUC	0.9514
Numero di parametri	15727

Tabella 3.9: Risultati e numero di parametri del modello Microbox 1D con rotazioni nel dominio della frequenza

In Figura 3.9 si può vedere la matrice di confusione relativa.

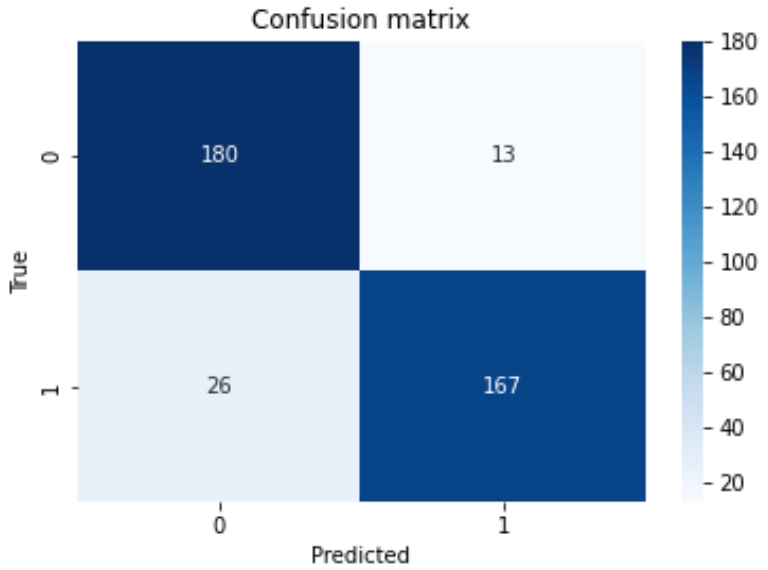

Figura 3.9: Matrice di confusione del modello Microbox 1D con rotazioni nel dominio della frequenza

3.6.2 Infobox

I risultati ottenuti con l'utilizzo del dataset Infobox e il numero di parametri del modello sono esposti in Tabella 3.10.

Accuracy	0.9150
Precision	0.9568
Recall	0.8692
F1-score	0.9109
AUC	0.9372
Numero di parametri	114812

Tabella 3.10: Risultati e numero di parametri del modello Infobox 1D con rotazioni nel dominio della frequenza

In Figura 3.10 si può vedere la matrice di confusione relativa.

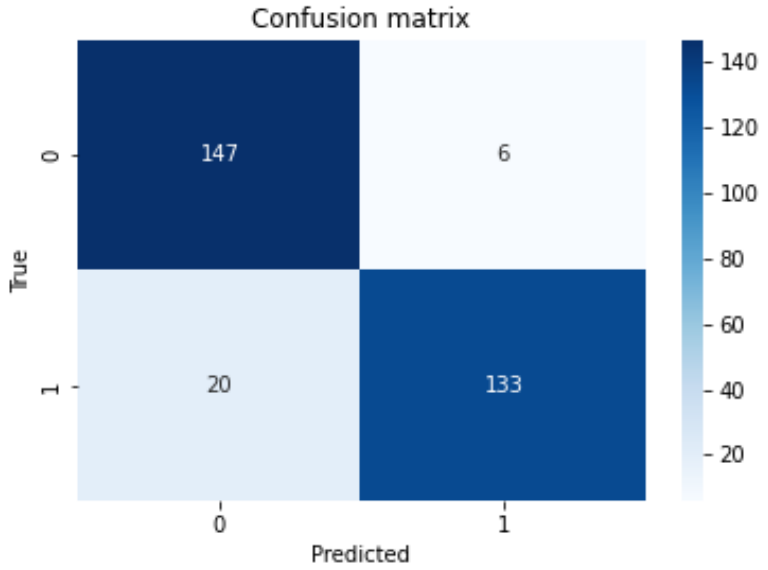

Figura 3.10: Matrice di confusione del modello Infobox 1D con rotazioni nel dominio della frequenza

3.7 Esperimenti con trasformazioni 2D delle serie temporali

In questa sezione verranno esposti i risultati relativi agli esperimenti con l'applicazione di tecniche per la trasformazione delle serie monodimensionali in immagini.

3.7.1 Recurrence plot

I risultati ottenuti con l'applicazione del recurrence plot e il numero di parametri del modello sono esposti in Tabella 3.11.

Accuracy	0.8779
Precision	0.8750
Recall	0.8883
F1-score	0.8816
AUC	0.9372
Numero di parametri	307074

Tabella 3.11: Risultati e numero di parametri del modello Microbox 2D recurrence plot

In Figura 3.11 si può vedere la matrice di confusione relativa.

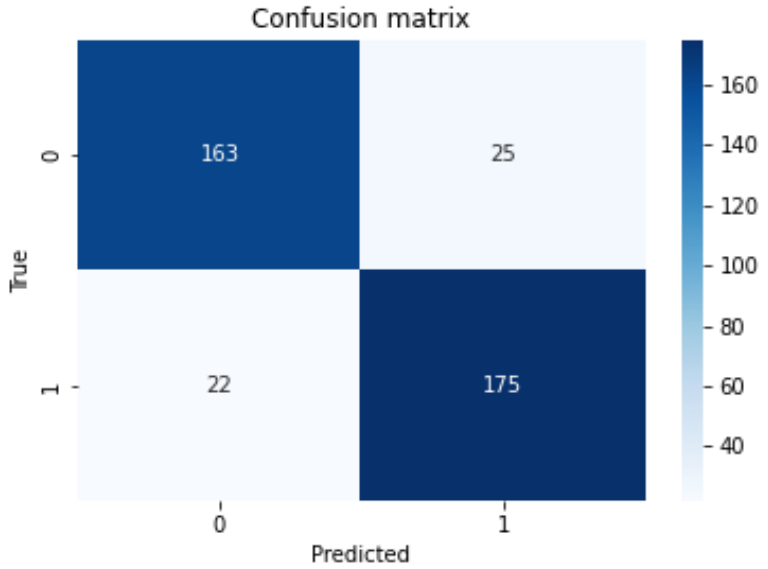

Figura 3.11: Matrice di confusione del modello Microbox 2D recurrence plot

3.7.2 Recurrence plot accoppiato

I risultati ottenuti con l'applicazione del recurrence plot accoppiato e il numero di parametri del modello sono esposti in Tabella 3.12.

Accuracy	0.8701
Precision	0.9047
Recall	0.8172
F1-score	0.8587
AUC	0.9166
Numero di parametri	307074

Tabella 3.12: Risultati e numero di parametri del modello Microbox 2D recurrence plot accoppiato

In Figura 3.12 si può vedere la matrice di confusione relativa.

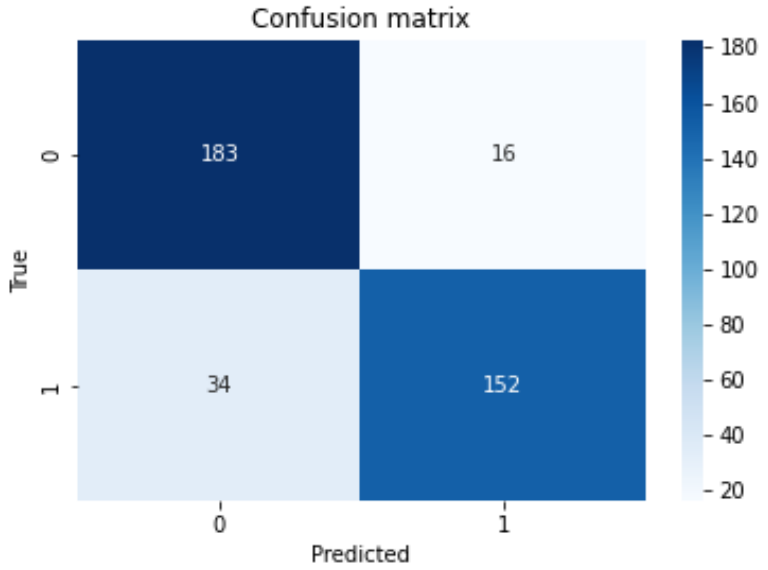

Figura 3.12: Matrice di confusione del modello Microbox 2D recurrence plot accoppiato

3.7.3 Spettrogramma

I risultati ottenuti con l'applicazione dello spettrogramma e il numero di parametri del modello sono esposti in Tabella 3.13.

Accuracy	0.8857
Precision	0.8769
Recall	0.8952
F1-score	0.8860
AUC	0.9470
Numero di parametri	143234

Tabella 3.13: Risultati e numero di parametri del modello Microbox 2D spettrogramma

In Figura 3.13 si può vedere la matrice di confusione relativa.

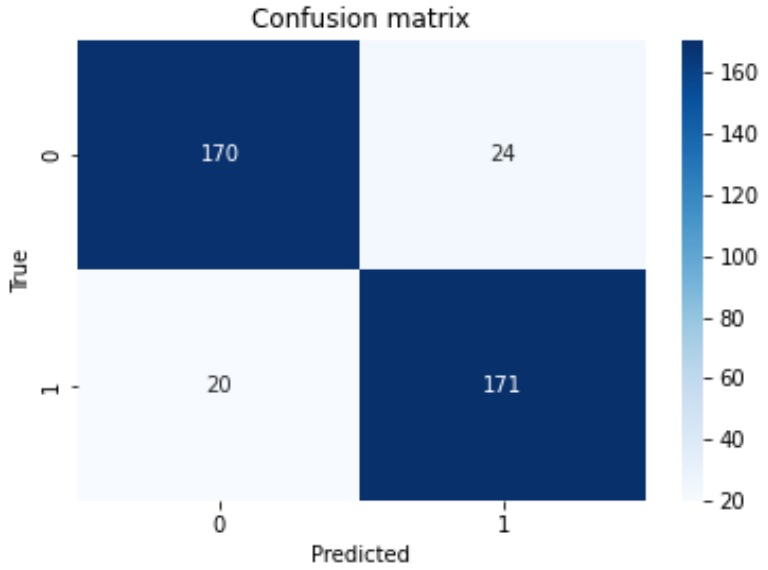

Figura 3.13: Matrice di confusione del modello Microbox 2D spettrogramma

3.7.4 Scalogramma

I risultati ottenuti con l'applicazione dello scalogramma e il numero di parametri del modello sono esposti in Tabella 3.14.

Accuracy	0.9116
Precision	0.9447
Recall	0.8769
F1-score	0.9095
AUC	0.9675
Numero di parametri	2748290

Tabella 3.14: Risultati e numero di parametri del modello Microbox 2D scalogramma

In Figura 3.14 si può vedere la matrice di confusione relativa.

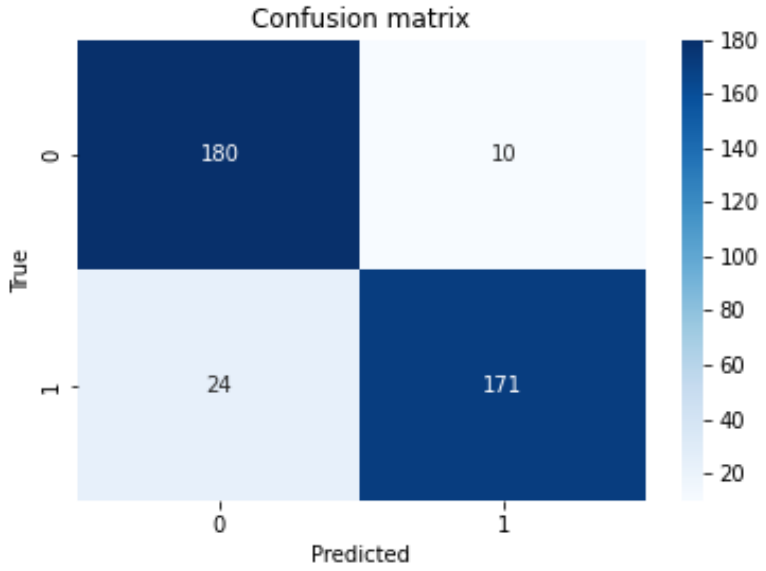

Figura 3.14: Matrice di confusione del modello Microbox 2D scalogramma

3.8 Confronto dei risultati

Le metriche più importanti da considerare per la nostra applicazione sono l'accuracy e la recall, perchè avere un basso numero di falsi negativi consente un'analisi di un maggior numero di casi.
Infatti, un eventuale falso negativo può essere riconosciuto ed escluso a posteriori, poichè viene classificato come vero, e quindi viene analizzato successivamente. Tuttavia, un falso negativo non viene mai analizzato dopo la classificazione, perciò si perde un caso positivo.

Può essere utile cambiare la soglia di classificazione, ad un valore minore del 50%, per poter minimizzare i falsi negativi.
In Tabella 3.15 si possono vedere i valori di accuracy e di recall per ogni modello a confronto, insieme con il numero di parametri.

Dataset	Tecnica	Numero di parametri	Accuracy	Recall
Microbox	1D tempo	15727	0.9038	0.8704
	1D frequenza	15727	0.8909	0.8473
	1D tempo con norma	13727	0.8571	0.8497
	1D frequenza con norma	13727	0.8363	0.8020
	1D frequenza con rotazioni	15727	0.8989	0.8652
	2D recurrence plot	307074	0.8779	0.8883
	2D recurrence plot accoppiato	307074	0.8701	0.8172
	2D spettrogramma	143234	0.8857	0.8952
	2D scalogramma	2748290	0.9116	0.8769
Infobox	1D tempo	114812	0.9213	0.8910
	1D frequenza	114812	0.9180	0.8986
	1D tempo con norma	112612	0.9147	0.9090
	1D frequenza con norma	112612	0.8983	0.8920
	1D frequenza con rotazioni	114812	0.9150	0.8692

Tabella 3.15: Risultati dei vari modelli a confronto

Osservando la tabella, si vede che, per il dataset Microbox, il migliore risultato in assoluto è quello del caso 2D scalogramma. Tuttavia è anche il modello con più parametri, più di 2 milioni, contro il secondo modello migliore, il caso 1D nel tempo, con 15 mila. Si osserva anche che i modelli 2D recurrence plot accoppiato e il 2D spettrogramma riescono ad ottenere una recall maggiore, pur avendo meno accuracy.
Considerando fondamentale la velocità di classificazione, il modello 1D nel tempo risulta il migliore. Se altrimenti si vuole considerare più importante il risultato, il modello 2D scalogramma risulta il migliore, e pone una base per lo sviluppo della metodologia per poter ottenere risultati ancora migliori.
Per quanto riguarda i metodi che risolvono il problema delle rotazioni del sistema di riferimento, il metodo 1D nella frequenza con l'uso delle rotazioni risulta di gran lunga il migliore, ottenendo quasi lo stesso risultato del caso 1D nel tempo.
Per il dataset Infobox invece, il modello migliore risulta l'1D nel tempo.
Riguardo i metodi che risolvono il problema delle rotazioni del sistema di riferimento, il metodo 1D nella frequenza con l'uso delle rotazioni risulta il migliore assieme al caso 1D nel tempo con l'uso della norma. I risultati di questi due metodi sono molto simili al caso 1D nel tempo. Il caso con la norma inoltre ha ottenuto la miglior recall di tutti, considerevolmente migliore del caso con rotazioni, e con meno parametri totali.

Capitolo 4

Applicazioni per STM32

In questo capitolo verranno descritte due applicazioni realizzate per microcontrollore STM32 che permettono la classificazione direttamente sul dispositivo, portando così la fase di processamento e di classificazione direttamente in automobile. Si può cosi utilizzare il lato server del sistema solo per la ricezione della risposta finale, con numerosi vantaggi in termini di consumo.

4.1 Dispositivo STM32F429I-DISC1

Il dispositivo STM32 per il quale sono state sviluppate le applicazioni è la scheda STM32F429I-DISC1 [12], visibile in Figura 4.1.

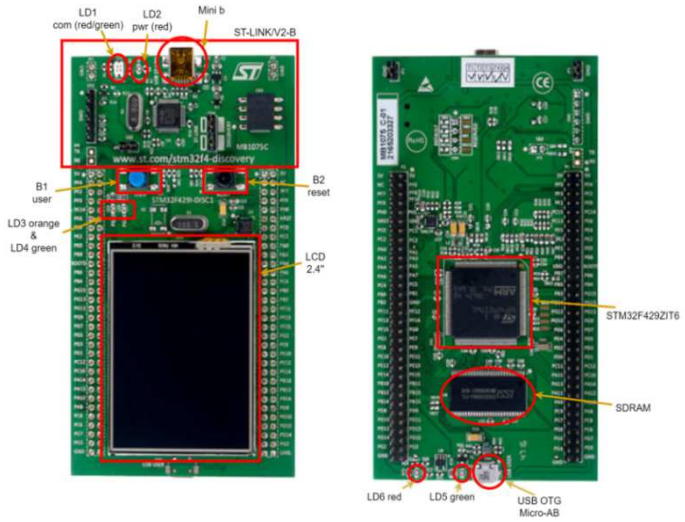

Figura 4.1: Scheda STM32F429I-DISC1

4.1.1 Specifiche

Le caratteristiche del dispositivo sono visibili in Tabella 4.1.

Microcontrollore	STM32F429ZIT6
Memoria flash	2Mbytes
RAM	256Kbytes
Display	2.4" QVGA TFT LCD
Input/Output	Connettore Micro-AB-USB
Bottoni push	1x User, 1x Reset
SDRAM	64Mbit
Programmer e debugger embedded	ST-LINK/V2-B
Giroscopio	ST-MEMS I3G4250D
Alimentazione	Mini-USB 3.3/5V

Tabella 4.1: Specifiche scheda STM32F429I-DISC1

La scheda viene alimentata attraverso il connettore Mini-USB da 3.3/5V. I dati da classificare vengono passati alla scheda tramite il connettore Micro-AB-USB. Una caratteristica peculiare di questo dispositivo è il display integrato, che è utilizzato per la visualizzazione dei risultati.

4.1.2 Programmazione

Per programmare la scheda tramite il programmer embedded ST-LINK/V2-B è necessario:

- Assicurarsi che i jumper CN4 e JP3 siano inseriti
- Alimentare la scheda tramite il connettore USB Mini-B
- Verificare che i led LD2 (PWR) e LD1 (COM) siano accesi

In Figura 4.2 è visibile un dettaglio sui jumper CN4 descritti sopra, mentre in Figura 4.3 è visibile uno schema dei collegamenti della scheda.

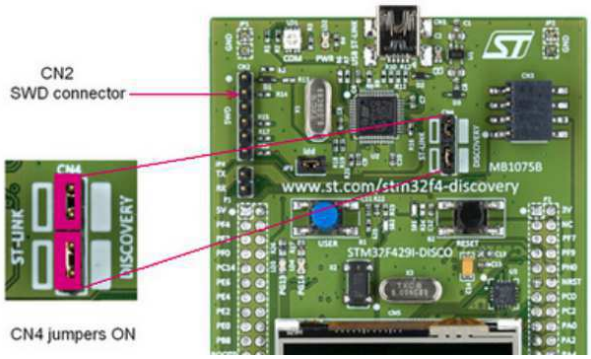

Figura 4.2: Dettaglio sui jumper della scheda STM32F429I-DISC1

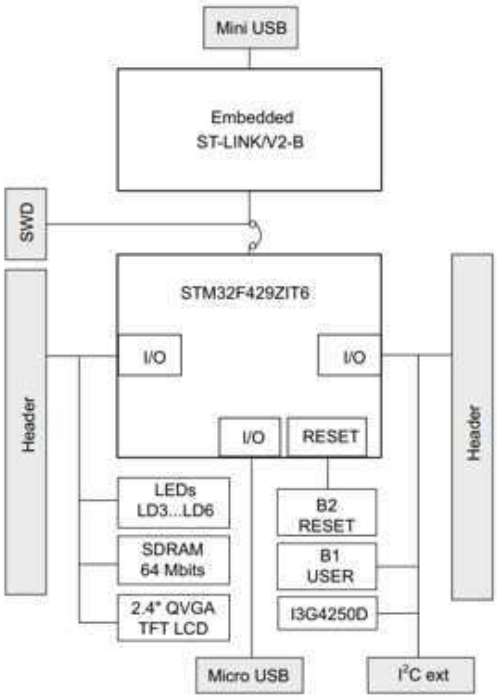

Figura 4.3: Schema dei collegamenti della scheda STM32F429I-DISC1

Per programmare le applicazioni, è stata utilizzata l'IDE di sviluppo di STM32, ovvero STM32CubeIDE [13], e il suo modulo di espansione specifico per l'implementazione su microcontrollore di intelligenze artificiali, X-CUBE-AI [14]. STM32CubeIDE è una piattaforma di sviluppo C/C++ con configurazione delle periferiche, generazione di codice, compilazione di codice e funzionalità di debug per microcontrollori e microprocessori STM32. Si basa sul framework Eclipse e sulla toolchain GCC per lo sviluppo e GDB per il debug. STM32CubeIDE integra le funzionalità di configurazione e creazione di progetti da STM32CubeMX. X-CUBE-AI è un pacchetto di espansione di STM32Cube che estende le capacità di STM32CubeMX con la conversione automatica di algoritmi di intelligenza artificiale pre-addestrati, inclusi i modelli di rete neurale e di machine learning.

4.2 Funzionalità

Le applicazioni sviluppate consentono di classificare serie temporali, passate al dispositivo tramite una chiavetta USB, e di visualizzare a schermo i risultati. Le due applicazioni svolgono esattamente lo stesso compito, una è stata creata con l'utilizzo di un modello 1D Microbox nel tempo, e l'altra utilizzando un modello 1D Infobox nel tempo.

4.2.1 USB drive

Le serie temporali devono essere salvate in un drive USB con la seguente struttura (Figura 4.4):

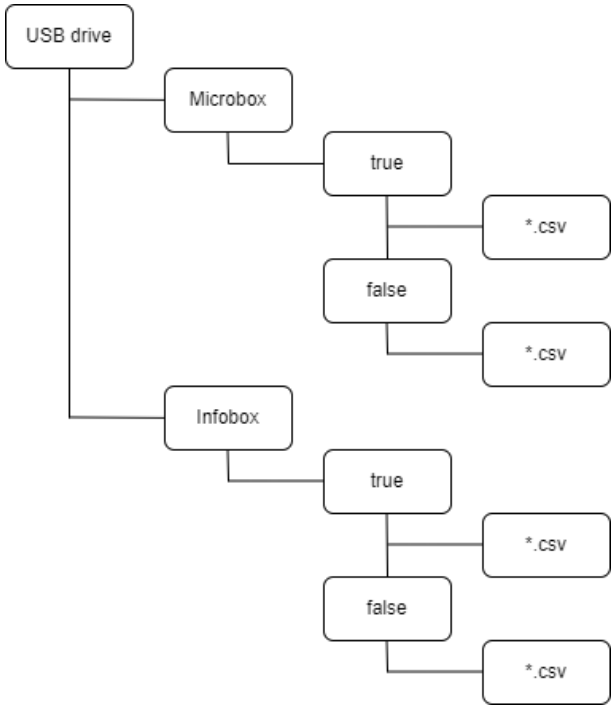

Figura 4.4: Struttura del drive USB

4.2.2 Classificazione

Al collegamento con l'alimentazione della scheda vengono brevemente visualizzate a schermo delle informazioni (Figura 4.5), e in seguito le serie temporali vengono classificate. Viene tenuto il conto delle serie e quello dei veri positivi, veri negativi, falsi positivi, e falsi negativi.
Per ogni serie viene calcolata l'accuracy totale e il tempo di inferenza (Figura 4.6).

Figura 4.5: Informazioni iniziali (applicazione Infobox)

Figura 4.6: Processo di classificazione (applicazione Infobox)

4.2.3 Attesa dell'USB drive o drive disconnesso

Se il drive non è connesso o se è stato scollegato prima della fine della classificazione, viene visualizzato un messaggio a schermo (Figura 4.7).

Figura 4.7: Attesa del drive USB

4.2.4 Risultati finali

Finita la classificazione, vengono mostrati i risultati finali (Figura 4.8). Vengono mostrate le seguenti metriche:

- Accuracy
- Tempo di inferenza totale
- Veri negativi
- Falsi positivi
- Falsi negativi
- Veri positivi
- Precision
- Recall
- F1-score

Figura 4.8: Risultati (applicazione Infobox)

4.3 Implementazione

Tramite STM32CubeIDE è stato generato il codice C necessario per la gestione delle periferiche, in particolare per la gestione della periferica USB per il drive USB, il codice per l'uso della libreria FatFs [15] per l'apertura e la lettura dei file csv presenti all'interno, e il codice per generare le reti neurali.

4.3.1 Definizione parametri

Vengono definiti i seguenti parametri, necessari per la lettura dei file csv. I parametri sono differenti per le applicazioni Microbox e Infobox. Verrà mostrato il codice relativo all'applicazione Infobox.
Di seguito la definizione dei parametri:

```
// Set params
// Number of series to test
#define N_SAMPLES 305
// Length of each series
#define SERIES_LENGTH 1200
// Axes of each series
#define CHANNELS 2
// Bytes to read for each series
#define BTR 32871
/* USER CODE END Private defines */
```

Il corretto formato dei file csv viene generato quando si salva il set di test utilizzando i notebook relativi agli esperimenti, così come il parametro BTR, che si riferisce ai bytes da leggere per csv.
I parametri N_SAMPLES, SERIES_LENGTH, e CHANNELS si riferiscono, rispettivamente, al numero di elementi nel set di test e alla forma dell'input, nel caso 1D Infobox nel tempo 1200x2.

4.3.2 Gestione dello stato dell'applicazione

Attraverso un handler, viene gestito lo stato dell'applicazione. Ci sono tre possibili stati:

- Application Ready: quando il drive USB è correttamente connesso, l'applicazione inizia la fase di classificazione.
- Application Idle: quando il drive USB non è stato ancora connesso al dispositivo, viene semplicemente mostrato a schermo un messaggio.
- Application Disconnect: quando il drive USB vine disconnesso durante il processo di classificazione, viene visualizzato un messaggio e vengono azzerati i veri positivi, i veri negativi, i falsi positivi, e i falsi negativi per permettere il riavvio della classificazione quando l'USB verrà riconesso.

Di seguito si vede il codice che gestisce lo stato dell'applicazione, eccetto per l'Application Ready, che verrà approfondita nei prossimi paragrafi.

```
/* Infinite loop */
while (1)
{
  MX_USB_HOST_Process();
  /* USER CODE BEGIN 3 */
  // Handler of application state
  switch(Appli_state){
    case APPLICATION_READY:
    ...
    break;
    case APPLICATION_IDLE:
      sprintf(buf, "IDLE");
      // Application is ready but there's no USB connected
      BSP_LCD_DisplayStringAtLine(0,(uint8_t*)"Waiting for USB");
    break;
    case APPLICATION_DISCONNECT:
      sprintf(buf, "DISCONNECT");
      // If test is not finished while USB is disconnected,
      // reset variables
      if (done == 0)
      {
        y_true = 0;
        sample = 0;
        tn = 0;
        fp = 0;
        fn = 0;
        tp = 0;
        for (int i=2;i<10;i++)
        {
          BSP_LCD_ClearStringLine(i);
        }
        BSP_LCD_DisplayStringAtLine(0,(uint8_t*)"Waiting for USB");
      }
    break;
    default:
    break;
  }
}
/* USER CODE END 3 */
```

4.3.3 Application Ready: lettura dei file csv

Per ogni classe, vengono aperti e letti ad uno ad uno i file csv contenti le serie temporali:

```
sprintf(buf, "Test: %d/%d    ", sample, N_SAMPLES);
BSP_LCD_DisplayStringAtLine(0,(uint8_t*)buf);
// For all the folders
for (int folders=0;folders<2;folders++)
{
  sprintf(folder,"/infobox/%s", classes_folders[folders]);
  // If folder found, check for file, open it, read it, and close it
  if (f_chdir(folder)==FR_OK)
  {
    res = f_findfirst(&dir, &fno, "","*.csv");
    while ((res == FR_OK) && (fno.fname[0]))
    {
      sprintf(buf, "Open file %s", fno.fname);
      if (f_open(&file, fno.fname, FA_READ) != FR_OK)
      {
        sprintf(buf, "Open file error");
      }
      f_read(&file, &csv_full, BTR, &br);
      if (br!=BTR)
      {
        sprintf(buf, "Read file error");
      }
      f_close (&file);
      ...
    }
    f_closedir(&dir);
    sprintf(buf, "Finished %d",res);
  }
}
```

In seguito, ogni csv viene diviso in token contenenti ciascuno un valore delle serie temporali, che vengono poi inseriti in maniera ordinata in un array (prima tutta la serie relativa all'asse x, poi tutta la serie relativa all'asse y):

```
// Divide the file read into tokens, first lines, then values
int j = 0;
char* token = strtok(csv_full, "\r\n");
while( token != NULL )
{
  csv_line[j]=token;
  j ++;
  token = strtok(NULL, "\r\n");
}
int k = 0;
for (int i = 1;i<SERIES_LENGTH + 1;i++)
{
  char* token2 = strtok(csv_line[i], ",");
  while( token2 != NULL )
  {
    X_test_str[k]=token2;
    k ++;
    token2 = strtok(NULL, ",");
  }
}
```

4.3.4 Application Ready: classificazione e calcolo delle metriche

I dati acquisiti vengono poi convertiti opportunamente e inviati alla rete neurali per la classificazione. Viene anche calcolato il tempo di inferenza:

```
// Convert values to double
for (int i = 0;i<SERIES_LENGTH*CHANNELS;i++)
{
  X_test[i] = strtod(X_test_str[i], NULL);
}
// Fill input buffer for neural network
for (uint32_t i = 0; i < AI_NETWORK_IN_1_SIZE; i++)
{
  ((ai_float *)in_data)[i] = (ai_float)X_test[i];
}
// Perform inference
Tinf1 = HAL_GetTick();
nbatch = ai_network_run(network, &ai_input[0], &ai_output[0]);
if (nbatch != 1)
{
  sprintf(buf, "Error: could not run inference");
}
Tinf2 = HAL_GetTick();
// Compute inference time
nn_inference_time = ((Tinf2>Tinf1)?(Tinf2-Tinf1):((1<<24)-Tinf1+Tinf2));
tot_inference_time = tot_inference_time + nn_inference_time;
```

Viene poi letto l'output della rete neurale, calcolata la predizione, aggiornati i veri positivi, falsi positivi, veri negativi, e falsi negativi, calcolata l'accuratezza e stampate a schermo le informazioni:

```
// Read output of neural network
y_prob[0] = ((float *)out_data)[0];
y_prob[1] = ((float *)out_data)[1];
// Make prediction
if (y_prob[0] > y_prob[1])
{
  y_pred = 0;
}
else
{
  y_pred = 1;
}
// Count tn, fp, fn, tp
if ((y_pred == y_true) && (y_true == 0))
{
  tn ++;
}
else if ((y_pred == y_true) && (y_true == 1))
{
  tp ++;
}
  else if ((y_pred != y_true) && (y_true == 0))
{
  fp ++;
}
else
{
  fn ++;
}
// Print metrics and inference time (milliseconds)
sample ++;
sprintf(buf, "Test: %d/%d    ", sample, N_SAMPLES);
BSP_LCD_DisplayStringAtLine(0,(uint8_t*)buf);
sprintf(buf, "Accuracy: %f", (float)(tn + tp)/(float)(tn + tp + fn + fp));
BSP_LCD_DisplayStringAtLine(2,(uint8_t*)buf);
sprintf(buf, "Inf. time: %ldms", nn_inference_time);
BSP_LCD_DisplayStringAtLine(4,(uint8_t*)buf);
sprintf(buf, "True negatives: %d", tn);
BSP_LCD_DisplayStringAtLine(6,(uint8_t*)buf);
sprintf(buf, "False positives: %d", fp);
BSP_LCD_DisplayStringAtLine(7,(uint8_t*)buf);
sprintf(buf, "False negatives: %d", fn);
BSP_LCD_DisplayStringAtLine(8,(uint8_t*)buf);
sprintf(buf, "True positives: %d", tp);
BSP_LCD_DisplayStringAtLine(9,(uint8_t*)buf);
res = f_findnext(&dir,&fno);
```

Finita la classificazione, vengono calcolate e visualizzate a schermo anche precision, recall, e F1-score, insieme con il tempo totale:

```
// Compute precision, recall, F1-score
done = 1;
BSP_LCD_DisplayStringAtLine(0,(uint8_t*)"Test done      ");
sprintf(buf, "Tot. time: %.3fs", (float)(tot_inference_time)/1000);
BSP_LCD_DisplayStringAtLine(4,(uint8_t*)buf);
sprintf(buf, "Precision: %f", (float)tp/(float)(tp+fp));
BSP_LCD_DisplayStringAtLine(11,(uint8_t*)buf);
sprintf(buf, "Recall: %f", (float)tp/(float)(tp+fn));
BSP_LCD_DisplayStringAtLine(12,(uint8_t*)buf);
sprintf(buf, "F1-score: %f",
2*(float)tp/(float)(tp+fp)*(float)tp/(float)(tp+fn)/
((float)tp/(float)(tp+fp)+(float)tp/(float)(tp+fn)));
BSP_LCD_DisplayStringAtLine(13,(uint8_t*)buf);
```

Conclusione

In questo lavoro di tesi si è studiato il problema della classificazione automatica di incidenti stradali, proponendo delle soluzioni basate sull'uso di reti neurali convoluzionali.
Ci si è posti il problema delle rotazioni del sistema di riferimento, nel caso in cui si voglia poter campionare le accelerazioni tramite un dispositivo non solidale all'automobile, come ad esempio uno smartphone.
Si sono proposte soluzioni basate su data augmentation ruotando le serie con l'applicazione della trasformata di Fourier, e soluzioni basate sull'uso della norma.
Si sono poi confrontati i risultati ottenuti, considerando importanti sopratutto l'accuracy dei modelli, la recall, e il numero di parametri.
Infine, è stata descritta l'implementazione di applicazioni per microcontrollore STM32 per la classificazione direttamente in automobile, con numerosi vantaggi, specialmente in termini di consumo.
Questo lavoro pone delle basi per poter migliorare la classificazione. Sviluppando l'approccio 2D, in particolare la tecnica dello scalogramma, si possono ottenere risultati ancora migliori.
I metodi per l'indipendenza del sistema di riferimento pongono le basi per lo sviluppo di applicazioni che funzionino direttamente su smartphone, unendo anche il concetto dell'edge AI direttamente su un telefono.
Uno dei primi passi possibili è quello di testare i modelli addestrati con l'uso delle rotazioni o della norma con un dataset proveniente direttamente da smartphone, per potersi concentrare così su altri problemi che possono sorgere, come ad esempio la rumorosità dei diversi sensori degli smartphone, o le vibrazioni che possono spostare uno smartphone in fase di campionamento.

Bibliografia

[1] Fairconnect. fairconnect.it, 2022.

[2] Nico Osimani. Supervised learning algorithms for car accident classification. github.com/NicoOsimani/Supervised-learning-algorithms-for-car-accident-classification, 2022.

[3] Introduction to 1d convolutional neural networks in keras for time sequences. blog.goodaudience.com, 2018.

[4] Google Brain team. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] Google colab. colab.research.google.com, 2022.

[6] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[7] Travis E. Oliphant et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[9] Samplerate. pypi.org/project/samplerate, 2022.

[10] Eric Jones Travis Oliphant, Pearu Peterson et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[11] OverLordGoldDragon. Ssqueezepy. github.com/OverLordGoldDragon/ssqueezepy, 2022.

[12] Stm32f429i-disc1. st.com/resource/en/user_manual/um1670-discovery-kit-with-stm32f429zi-mcu-stmicroelectronics.pdf, 2020.

[13] Stm32cubeide. st.com/en/development-tools/stm32cubeide.html, 2022.

[14] X-cube-ai. st.com/en/development-tools/stm32cubemx.html, 2019.

[15] Fatfs. elm-chan.org/fsw/ff/00index_e.html, 2022.

Ringraziamenti

Ringrazio il Dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche, il relatore Prof. Aldo Franco Dragoni e i membri dell'AIRTLab, Nicola Falcionelli, Paolo Sernani, Selene Tomassini, Paolo Contardo e Sara Abbonzio.
Ringrazio i membri dell'azienda FairConnect per la collaborazione, Gianluigi Brasili, Gabriele Saputelli, Fabio Faraone, Roberto Merlino e Luigi Marinucci.
Infine, ringrazio Manuel Verlengia la cui tesi di laurea ha posto le basi per lo sviluppo del mio lavoro.