# Classificazione di ECG con tecniche di Deep Learning

Andrea de Donato (Student ID: 536795) and Paolo Di Simone (Student ID: 584638)

*Università di Roma Tre*
*Corso di Deep Learning*
*Roma, Italia*

1 Marzo, 2023

## I Introduzione

L'elettrocardiogramma, o ECG, é un test diagnostico, di tipo strumentale, che registra e riporta graficamente il ritmo e l'attività elettrica del cuore. Lo strumento usato per l'elettrocardiogramma è chiamato elettrocardiografo. Il medico a cui spetta, di solito, l'interpretazione di un tracciato elettrocardiografico è un cardiologo, ossia un medico specializzato in cardiologia.

## II Workflow del progetto

Il progetto é stato sviluppato attraverso i seguenti punti:

1. **Ricerca riferimenti**: abbiamo cercato dei paper di riferimento circa la classificazione degli ECG attraverso tecniche di Deep Learning;
2. **Ricerca dataset**: abbiamo cercato vari dataset che fornissero gli ECG sotto forma di sequenze temporali o immagini;
3. **Analisi del dataset scelto**: abbiamo scelto come dataset il *MIT-BIH arrhythmia database*, visto che è stato utilizzato in tutti i paper di riferimento. Questo dataset lo abbiamo analizzato in termini qualitativi e quantitativi, quindi abbiamo analizzato la quantità di dati disponibili, la distribuzione delle label, lo sbilanciamento delle label disponibili e i tipi di segnali disponibili;
4. **Pre-processing**: abbiamo effettuato un pre-processing dei segnali disponibili, in modo da estrarre sequenze di segnale di lunghezza predefinita che contenessero una sola etichetta;
5. **Addestramento**: abbiamo addestrato vari modelli di Deep Learning, inclusi quelli trattati nei paper di riferimento, e abbiamo valutato le loro prestazioni;
6. **Confronto**: abbiamo confrontato le prestazioni dei vari modelli.

## III Dataset

Il dataset utilizzato è il MIT-BIH arrhythmia database che presenta 48 ECG di 47 persone, in particolare:

- 25 uomini fra 32 e 89 anni
- 22 donne fra 23 e 89 anni

Ogni record ha una frequenza di campionamento di 360 Hz e una durata di circa 30 minuti. In figura 1 possiamo vedere un estratto di 10 secondi di uno dei 48 file.

Tutti i battiti di ogni record sono stati etichettati da esperti nel campo ECG.

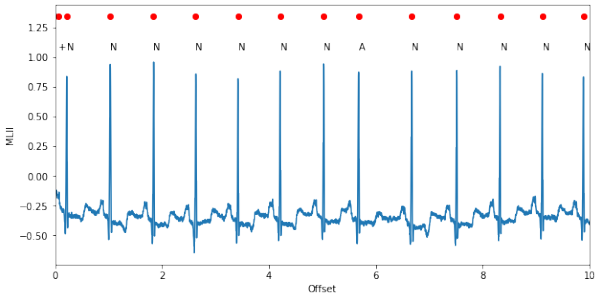


Figure 1. Estratto ECG

In figura 2 possiamo osservare come sono distribuite le label all'interno del dataset (l'asse delle ordinate è in scala logaritmica). Si noti il forte sbilanciamento fra il numero di etichette N e le altre.

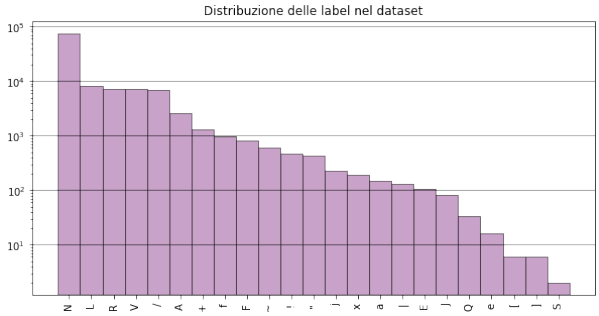


Figure 2. Distribuzione label nel dataset

## IV Pre-processing

In questa fase siamo andati a suddividere ogni ECG in battiti, in modo che:

- Ogni battito fosse un segnale di dimensione fissa 300 (meno di un secondo di segnale);
- Ogni battito avesse una ed una sola etichetta;
- L'etichetta fosse al centro del battito.

Alla fine di questo processo abbiamo ottenuto un dataset con 104.222 segnali (battiti) etichettati.

Le etichette che sono state utilizzate (con la relativa occorrenza all'interno del dataset), e poi prese in considerazione nel progetto, sono:

- N, normal beat: 71611
- L, left bundle branch block beat: 7943
- R, right bundle branch block beat: 7206
- V, premature ventricular contraction: 5411
- F, fusion of ventricular and normal beat: 756
- A, atrial premature contraction: 2140
- S, premature or ectopic supraventricular beat: 2
- /, paced beat: 6957
- Q, unclassifiable beat: 25

Nelle figure 3, 4, 5 e 6 sono riportati degli esempi di battiti etichettati.

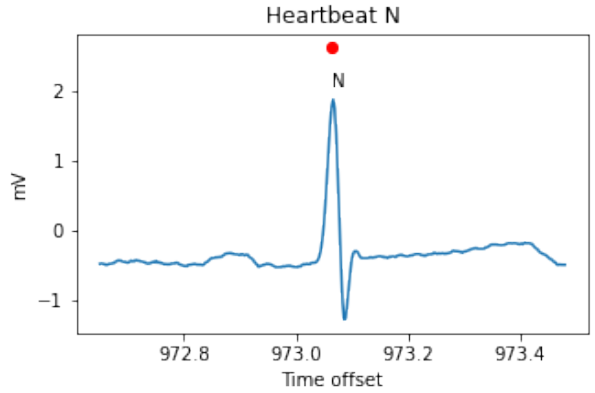


Figure 3. Label N

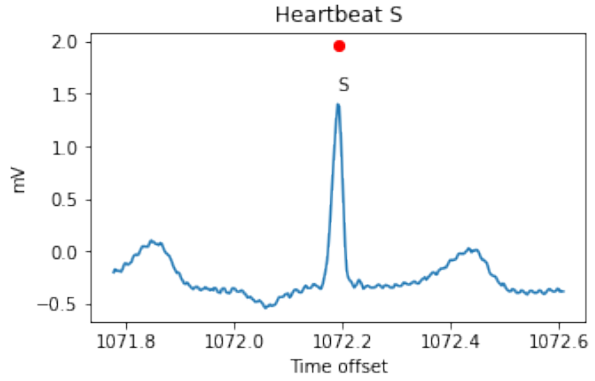


Figure 4. Label S

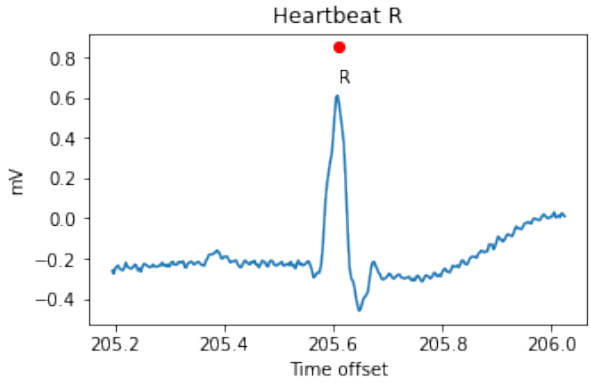


Figure 5. Label R

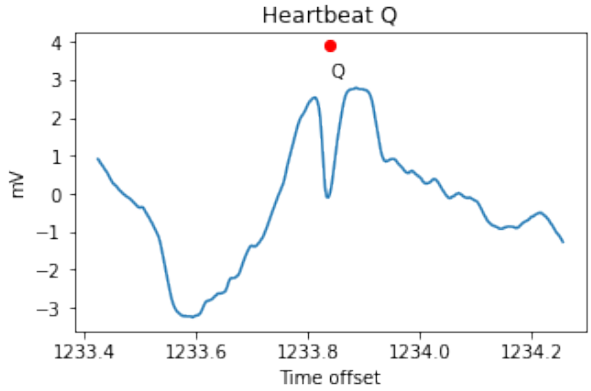


Figure 6. Label Q

Il segnale dell'ECG è stato diviso in heartbeat sulla base dei fiducial point (FP) che sono stati determinati attraverso l'algoritmo di Pan-Tompkins.
La classificazione è stata fatta attraverso una CNN che lavora il segnale iniziale attraverso tre pipeline.

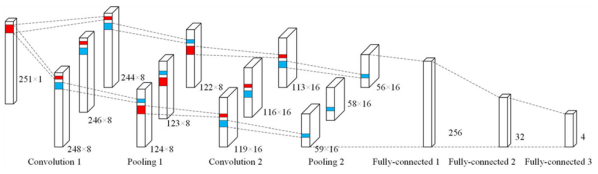


Figure 7. Architettura primo articolo

In particolare, abbiamo un input di 251x1 (heartbeat). Il primo layer convoluzionale (Convolution 1) è composto da 3 pipline che hanno 8 filtri 1D di dimensione 4, 6 e 8 rispettivamente e stride 1. Poi abbiamo un layer di max pooling (Pooling 1) con dimensione 2 e stride 2. Il terzo layer è convoluzionale (Convolution 2) con 16 (sul paper c'è scritto 24, ma nell'immagine 16) filtri 1D di dimensione 6, 8 e 10 rispettivamente, con stride 1. Poi abbiamo di nuovo un layer di max pooling (Pooling 2) con dimensione 2 e stride 2. Successivamente le tre pipeline vengono combinate (non so come, forse

## V Analisi paper

### 1 Primo Articolo

Il dataset utilizzato è il *MIT-BIH arrhythmia database*. Il paper si occupa di classificare i battiti cardiaci rilevati attraverso elettrocardiogramma in N, S, V, F e Q.

Il lavoro è stato svolto in tre fasi: preprocessing, divisione del dataset e classificazione.

solo rappresentazione flatten). Infine, abbiamo una MLP di 3 layer con rispettivamente 256, 32 e 4 nodi. Per tutti i layer, CNN e FC, è stata utilizzata RELU. Dopo l'ultimo layer FC abbiamo una softmax. Le 4 classi sono N, S, V e F (non classifica Q a quanto pare). Si utilizza batch size di 64, learning rate 0.01 e l'addestramento è fatto su 100 epoche.

### 2 Secondo Articolo

Il dataset utilizzato è *MIT-BIH Arrhythmia Database*. Ogni record contiene due grafici: MLII e uno tra V1, V2, V4 o V5. Nella posizione di ogni R-peak (picco) è stata fatta un'annotazione (label). Nel paper vengono usati solo i grafici MLII. In particolare, questi vengono segmentati in heartbeat prendendo sample lunghi 300 punti che sono centrati sull'R-peak (150 punti prima e 149 dopo). Questa segmentazione permette di estrarre 107 679 segmenti etichettati N, S, V, A, Q, F. Il lavoro è stato svolto in due step: preprocessing e classificazione.

Dal momento che il segnale è rumoroso (rumore causato da interferenze mioelettriche, interferenze di power line e baseline drift), attraverso il preprocessing, questo rumore è stato rimosso. Il segnale sporco è stato pulito attraverso dei filtri. Il segnale sporco è stato decomposto da Daubechies wavelet 6 in sei livelli e le i coefficienti dei livelli 3 e 6 sono stati mantenuti e usati per la ricostruzione del segnale. Dopo il denoising il segnale è stato segmentato come precedentemente descritto.

L'architettura è la seguente:

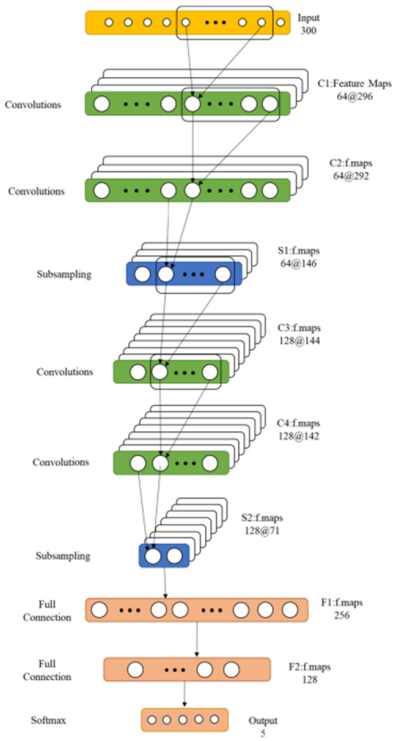


Figure 8. Architettura primo articolo

Ci sono 9 layer:

- 4 layer convoluzionali C1, C2, C3 e C4 che hanno filtri rispettivamente di dimensione 5, 5, 3 e 3 (stride 1) (vedere immagine per numero filtri);
- 2 layer di max pooling con size e stride 2;
- 3 layer fully connected 256, 128, 5;
- 1 layer di softmax

Tutti i layer hanno funzione di attivazione ReLU. La loss è una cross entropy. Il batch size è 256. Lo split è 90/10, il 90 di training è splittato a sua volta in 90/10 (per la validation). Ottimizzatore Adam, con beta1 = 0.9 e beta2 = 0.999. Dropout utilizzato nei layer C2, C4, F1, F2.

### 3 Terzo Articolo

Il dataset utilizzato è *PTB* che contiene una quantità di dati ECG frammentati in singoli battiti cardiaci. La dimensione totale del dataset raggiunge i 19904 campioni. Sappiamo che le CNN richiedono una grande quantità di dati per avere un'elevata precisione; tuttavia, utilizzando ogni battito dello stesso paziente come un nuovo dato non sempre aiuta ad incrementare l'accuratezza, ma in realtà può andare a creare degli errori. Il dataset è composto da undici classi differenti. Per progettare e addestrare una architettura CNN di Deep Learning viene usato Matlab. In questa rete CNN, l'optimizer che viene usato è Adam. Questo optimizer è costituito da due coefficienti: gradiente (B1) e i suoi valori al quadrato (B2). Questi coefficienti sono fissati rispettivamente a 0,9 e 0,99. Il numero di iterazioni per l'epoca e il numero di epoche sono rispettivamente 128 e 22. Nella maggior parte dei casi, il layer RELU viene utilizzato per memorizzare i dati dell'immagine. Poiché la rete CNN proposta ha segnali negativi e positivi, RELU converte tutti i valori negativi in zero e può causare la perdita di dati indesiderata. Pertanto, il layer Leaky RELU è usato per ottenere i risultati desiderati. In questo layer qualsiasi numero negativo, minore di zero, è moltiplicato per una certa costante x.

- Convolutional Layer 2D
- Batch Normalization
- Leaky Relu
- MaxPooling Layer

Tale blocco è ripetuto tre volte e infine si hanno come ultimi layer della rete un FC, Softmax e un output per la classificazione. I risultati ottenuti sulle differenti classi in termini di accuracy sono nel peggior caso il 96,5% fino ad arrivare su 3 classi di 11 ad un massimo del 100%.

## VI Addestramento di modelli

### 1 Primo modello: Paper 1

Nel primo modello si utilizzano i layer convolutivi 1d, necessari per andare a ridurre (comprimere) la dimensione temporale, e MaxPooling layer 1d che consentono di andare ad effettuare un downsampling e quindi ridurre il numero di parametri man mano che si procede verso il layer di output. Il MaxPooling layer ha solitamente i seguenti iperparametri:

- pool size = 2
- stride = 2

I layer convolutivi sono disposti in coppie alternate dai MaxPooling layer. Sono presenti due coppie di layer convulutivi 1d. La prima coppia (di cui il primo riceve l'input) ha 64 filtri con un kernel size pari a 5 e lo stride pari ad 1. La seconda coppia di layer convolutivi ha 128 filtri e un kernel size pari a 3, sempre con stride pari a 1. Il tipo di padding utilizzato per tutti i layer convolutivi all'interno del modello è 'valid', quindi non viene applicato alcun padding. Dopo i layer convolutivi è presente un layer che "flattenizza" il risultato dei layer convolutivi precedenti per essere utilizzato in input ai layer Dense. Al termine della rete sono presenti 3 layer Dense (FC) con rispettivamente 256, 128 e 5 nodi. L'ultimo Dense layer ha 5 nodi poichè corrispondono al numero di classi che devono essere predette.

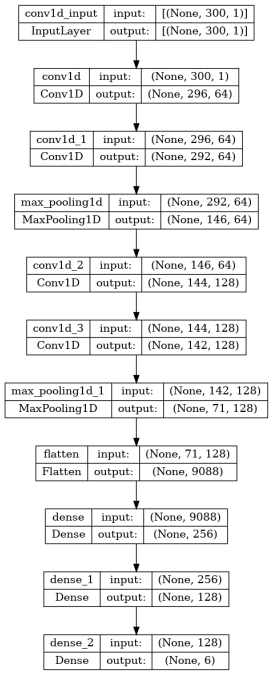


Figure 9. Architettura Modello 1

Nel modello è stato applicato un optimizer Adam. L'optimizer Adam consente di usufruire di diversi benefici tra cui:

- Consente una convergenza più rapida durante l'addestramento
- Aiuta a diminuire il rumore durante la fase di addestramento
- E' in grado di adattarsi ai cambiamenti dei gradienti durante l'addestramento
- Ha pochi parametri da regolare e si adatta automaticamente ai dati

L'addestramento è stato effettuato su una epoca su un batch size di 32 esempi, che corrispondono al numero di esempi che vengono utilizzati in una singola iterazione di addestramento, utilizzando la loss 'binary crossentropy'. Le prestazioni sono state valutate su un apposito validation set. I risultati ottenuti a seguito dell'addestramento sono:

- Loss = 0.0191
- Accuracy = 0.9844

In figura 11 è riportata la matrice di confusione del modello addestrato.

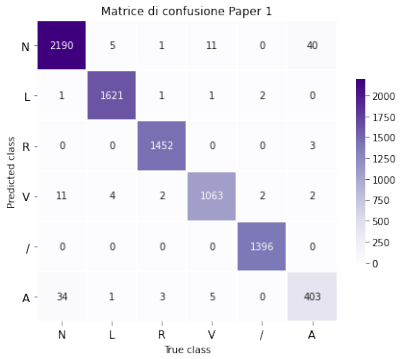


Figure 10. Matrice confusione Modello 1

## 2 Secondo modello: Paper 2

Il secondo modello è inspirato a funzionamento dell'inception module di GoogleNet, dove l'input è fornito contemporaneamente a tre pipeline di processamento distinte. Le tre pipeline sono composte da Convolutional 1D layer e da MaxPooling layer in maniera alternata. Ciò che distingue le diverse pipeline sono le dimensioni del kernel (filtro), infatti l'input arriva in tutte le pipeline ad un Conv1D layer con 8 filtri (feature maps) ma le dimensioni del kernel sono rispettivamente 4, 6 e 8. Le tre pipeline sono composte da 4 elementi: il primo e terzo elemento corrispondono a Conv1D layer, mentre il secondo e quarto elemento corrispondono a MaxPooling layer. Gli output delle tre pipeline vengono concatenati e successivamente vengono "flattenizzati", ovvero disposti su una sola dimensione. La parte finale della rete è composta da 3 Dense layer con rispettivamente 256, 128 e 5 nodi. L'ultimo Dense layer è composto da 5 nodi che corrispondono al numero di classi di cui fare la predizione.

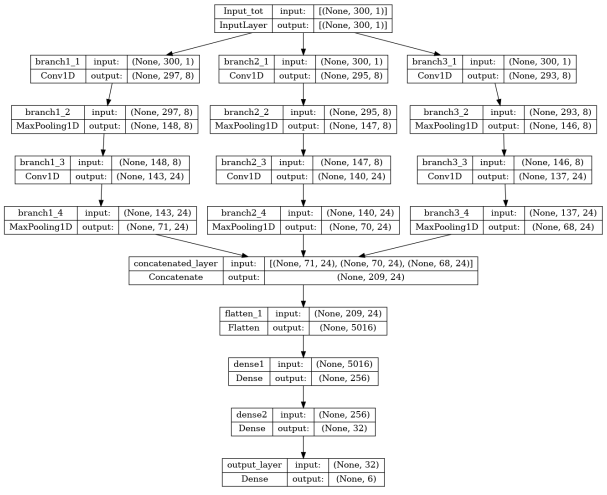


Figure 11. Architettura Modello 2

Nel modello è stato applicato un optimizer Adam e l'addestramento è stato effettuato su una epoca su un batch size di 32 esempi, che corrispondono al numero di esempi che vengono utilizzati in una singola iterazione di addestramento, utilizzando la loss 'binary crossentropy'. Le prestazioni sono state valutate su un apposito validation set.

I risultati ottenuti a seguito dell'addestramento sono:

- Loss = 0.0164
- Accuracy = 0.9856

In figura 12 è riportata la matrice di confusione del modello addestrato.

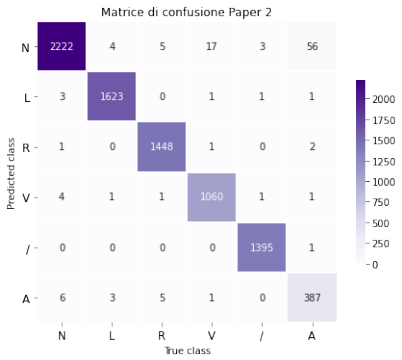


Figure 12. Matrice confusione Modello 2

## 3 Terzo modello: Wavenet

Il terzo modello consiste nell'implementazione di una wavenet, La wavenet si occupa di andare ad elaborare sequenze di segnali con diverse lunghezze e consiste in una Fully convolutional neural network, dove i Conv layer hanno diversi dilatetion factors che permettono al suo campo ricettivo di crescere in maniera esponenziale e coprire milioni di timestamp. Dopo il training, ad ogni step durante il sampling un valore viene mostrato da una distribuzione di probabilità calcolata dalla rete. Questo valore è successivamente utilizzato in input come feedback per la nuova predizione al successivo step. La campionatura (sampling) viene effettuata attreverso una funzione di attivazione softmax.

L'implementazione nel nostro caso prevede l'utilizzo di una serie di Conv1D layer, dove il dilatation factor aumenta man mano che si procede verso l'output della rete. Nel caso specifico abbiamo 4 Conv1D layer con 20 kernel ognuno e kernel size pari a 2. In questi 4 layer abbiamo che il dilatation aumenta dal layer di input ai layer più distanti dall'input rispettivamente da 2, 4, 8 e 16. Questi 4 layer utilizzano la Relu come funzione di attivazione. Dopo questi 4 layer, abbiamo un ulteriore Conv1d layer tradizionale con 10 kernel e kernel size pari a 1. Prima di arrivare al Dense layer per la predizione finale, l'output del layer precedente (Conv1D layer con 10 kernel) viene "flattenizzato" quindi ottenendo i valori su un'unica dimensione. Infine, il Dense layer è composto da 5 nodi (pari al numero di classi di cui è necessatio effettuare la predizione) e implementato attraverso la funzione di attivazione Softmax.

Nel modello è stato applicato un optimizer Adam e l'addestramento è stato effettuato su un'epoca su un batch size di 32 esempi, che corrispondono al numero di esempi che vengono utilizzati in una singola iterazione di addestramento, utilizzando la loss 'binary crossentropy'. Le prestazioni sono state valutate su un apposito validation set.

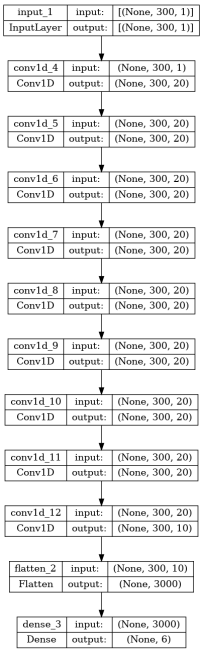


Figure 13. Architettura Modello 3

I risultati ottenuti a seguito dell'addestramento sono:

- Loss = 0.0354
- Accuracy = 0.9677

In figura 14 è riportata la matrice di confusione del modello addestrato.

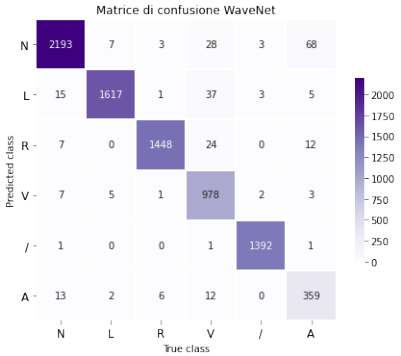


Figure 14. Matrice confusione Modello 3

## 4 Quarto modello: LSTM 1

Il quarto modello consiste nell'utilizzo di LSTM in maniera sequenziale. L'input del modello arriva ad un layer LSTM con un numero di unità pari a 128 con il parametro 'return sequence' = True, quest'ultimo consente di fornire in output un sequenza della stessa lunghezza dell'input e non un vettore. Il layer LSTM è susseguito da un Dropout layer con iperparametro

pari a 0.2. Sono presenti due layer LSTM alternati in questo modo attraverso due dropout con iperparametri identici. Dopo il secondo layer di Dropout si ha un Dense layer con funzione di attivazione Softmax e numero di nodi pari al numero di classi da predire (ovvero 5 nodi).

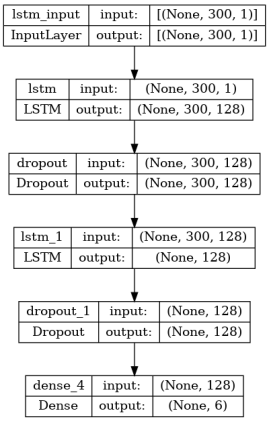


Figure 15. Architettura Modello 4

Nel modello è stato applicato un optimizer Adam e l'addestramento è stato effettuato su una epoca su un batch size di 32 esempi, che corrispondono al numero di esempi che vengono utilizzati in una singola iterazione di addestramento, utilizzando la loss 'binary crossentropy'. Le prestazioni sono state valutate su un apposito validation set.

I risultati ottenuti a seguito dell'addestramento sono:

- Loss = 0.1029
- Accuracy = 0.9033

In figura 16 è riportata la matrice di confusione del modello addestrato.

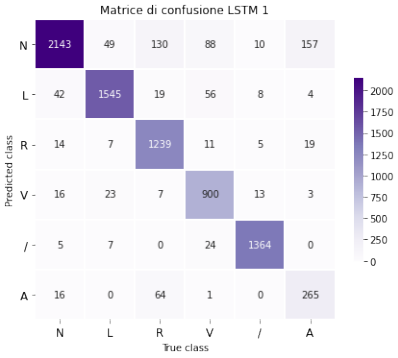


Figure 16. Matrice confusione Modello 4

## 5 Quinto Modello: LSTM 2

Il quinto modello ha un'architettura composta da layer LSTM con funzione di attivazione Swish. La funzione Swish si ottiene moltiplicando X per la funzione Sigmoide. La funzione Swish è proposta dal team Brain di Google. I loro esperimenti mostrano che Swish tende a funzionare più velocemente di Relu nei modelli profondi su diversi set di dati impegnativi. Il contro di tale funzione di attivazione è l'essere computazionalmente costosa. I vantaggi sono che non causa la fuga del gradiente ed è leggermente più veloce rispetto a Relu. Il grafico della funzione di attivazione Swish è riportato in figura 17.

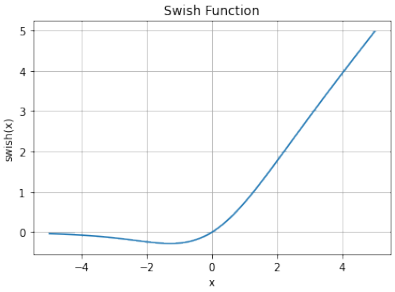


Figure 17. Grafico funzione Swish

L'architettura del modello è composta dalla seguente struttura ripetuta 3 volte:

- Convolutional 1D layer
- Batch normalization
- Activation Swish function

In ogni ripetizione di questa struttura cambia il numero di nodi presenti all'interno del Conv 1D con rispettivamente 32, 64 e 128 nodi. Al termine della terza ripetizione di questa struttura è presente un MaxPooling 1D layer per effettuare un downsampling. Dopo il MaxPooling layer, sono presenti in serie, due layer LSTM con 128 unità. Dopo LSTM è presente un GlobalAveragePooling1D con l'idea di generare una mappa delle caratteristiche per ogni categoria corrispondente dell'attività di classificazione nell'ultimo layer MPLConv. Invece di aggiungere layer Fully Connected (FC) sopra le feature map, prendiamo la media di ogni feature map e il vettore risultante viene immesso direttamente nel layer softmax. Quindi come ultimo layer dopo il GlobalAveragePooling 1D, si ha un Dense layer con numero di nodi pari al numero di classi da predire che sono 5 e funzione di attivazione Softmax che genera un distribuzione di probabilità sulle classi candidate.

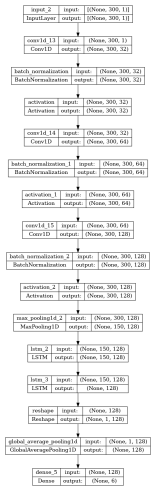

Figure 18. Architettura Modello 5

Nel modello è stato applicato un optimizer Adam e l'addestramento è stato effettuato su una epoca su un batch size di 32 esempi, che corrispondono al numero di esempi che vengono utilizzati in una singola iterazione di addestramento, utilizzando la loss 'binary crossentropy'. Le prestazioni sono state valutate su un apposito validation set.

I risultati ottenuti a seguito dell'addestramento sono:

- Loss = 0.0521
- Accuracy = 0.9477

In figura 19 è riportata la matrice di confusione del modello addestrato.

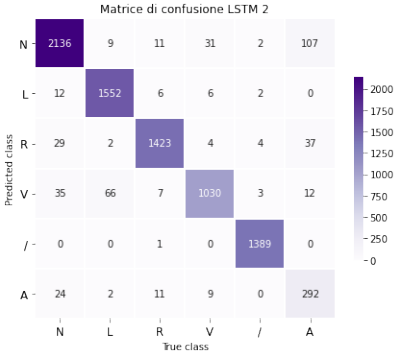


Figure 19. Matrice confusione Modello 5

## 6 Sesto Modello: DBN

Il sesto modello si basa sull'approccio DBN. Le DBN (Deep Belief Networks) sono composte da più strati di unità nascoste, che sono utilizzati per l'estrazione di caratteristiche del segnale. Possono essere utilizzate per la classificazione delle aritmie e per la predizione della mortalità a breve termine. L'architettura implementata in questo caso specifico consiste in Dense layer alternati con Dropout layer. Il primo layer è un Dense layer con 256 nodi, il secondo Dense layer ha 128 e il terzo Dense layer ha 64 nodi e infine il Dense layer di output ha un numero di nodi pari al numero di classi da predire. La funzione di attivazione utilizzata in tutti i layer ad eccezione dell'ultimo (output) è la Relu, mentre per l'ultimo Dense layer di output si ha funzione di attivazione Softmax. I Dropout layer si trovano sempre nel mezzo di due Dense layer e hanno il valore dell'iperparametro pari a 0.2.

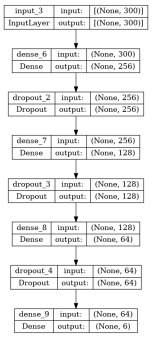


Figure 20. Architettura Modello 6

Nel modello è stato applicato un optimizer Adam con learning rate pari a 0.001 e l'addestramento è stato effettuato su 2 epoche su un batch size di 32 esempi, che corrispondono al numero di esempi che vengono utilizzati in una singola iterazione di addestramento, utilizzando la loss 'binary crossentropy'. Le prestazioni sono state valutate su un apposito validation set.

I risultati ottenuti a seguito dell'addestramento sono:

- Loss = 0.0221
- Accuracy = 0.9804

In figura 21 è riportata la matrice di confusione del modello addestrato.

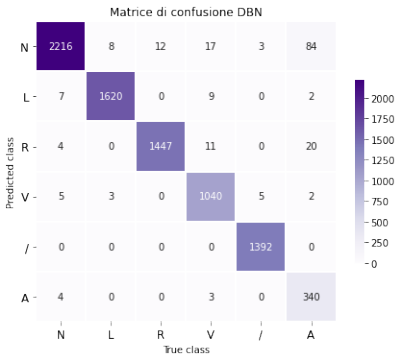


Figure 21. Matrice confusione Modello 6

## 7 Settimo Modello: Autoencoder

Il settimo modello si basa sull'approccio Autoencoder. L'approccio autoencoder è composto da un encoder che si occupa di codificare l'input e da un decoder che si occupa di ricostruire la versione originale dei dati. L'autoencoder è composto da 3 strati interni speculari oltre allo strato di input e di output. Il primo e l'ultimo strato intermedio sono Dense layer con 100 nodi e funzione di attivazione Relu, mentre il secondo strato intermedio è composto da 50 nodi e funzione di attivazione Relu. La prima cosa che viene fatta è codificare l'input attraverso l'encoder. Si utilizza la rappresentazione codificata come input per la classificazione.

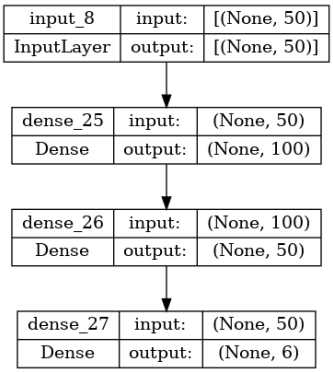


Figure 22. Architettura Modello 7

Nel modello è stato applicato un optimizer Adam e l'addestramento è stato effettuato su 3 epoche su un batch size

di 64 esempi, che corrispondono al numero di esempi che vengono utilizzati in una singola iterazione di addestramento, utilizzando la loss 'categorical crossentropy'. Le prestazioni sono state valutate su un apposito validation set.

I risultati ottenuti a seguito dell'addestramento sono:

- Loss = 0.0898
- Accuracy = 0.9763

## VII Conclusioni

L'obiettivo di questo progetto è quello di valutare varie alternative per la classificazione di ECG attraverso tecniche di Deep Learning rispetto allo stato dell'arte attuale e di individuare, ove possibile, dei modelli con delle prestazioni migliori rispetto a quest'ultimi. Attraverso un'attenta analisi dei modelli migliori in questo ambito abbiamo notato che i modelli che si comportano meglio sono:

- Modello paper 1
- Modello paper 2
- DBN
- Autoencoder

Questi due approcci permettono di ottenere risultati paragonabili con i risultati dei modelli presenti nei paper di riferimento.