Indice

Elenco delle figure

1

Introduzione

1.1 Contesto generale

In questo lavoro di tesi verranno descritti alcuni concetti chiave nel design e nello sviluppo di modelli basati su approcci guidati dall'*Intelligenza Artificiale* nel contesto medico, con l'obbiettivo di aumentare l'efficienza e l'affidabilità di specifici sistemi di ventilazione meccanica.

La ricerca di nuovi approcci nell'ambito della respirazione polmonare è stata sicuramente spinta e stimolata dalla forte necessità legata al contesto pandemico che stiamo vivendo dal Marzo 2020 e che sta toccando, in maniera più o meno marcata, ogni parte del globo.

Lo scoppio della pandemia da COVID-19 ha generato una grande pressione su interi reparti di terapia intensiva (*ICU - Intensive Care Unit*) e, più in generale, su tutte le strutture ospedaliere.

Oltre ad una forte limitazione sulla disponibilità di posti letto e di aree destinate a trattamenti *ICU*, la diffusione del COVID-19 ha determinato, in tutti i Paesi del mondo colpiti dalla pandemia, una scarsità di ventilatori rispetto al numero di pazienti che ha richiesto/richiede l'utilizzo di questo supporto respiratorio, essendo che la catena produttiva non è stata in grado di reggere a questa ondata di richieste di supporti utilizzabili per la ventilazione.

Questo contesto ha creato un estremo bisogno di dare forma a nuove tipologie di respiratori polmonari semplici ma tecnicamente validi e funzionali (per ovvi motivi), adatti per una produzione di massa su larga scala e in un breve periodo di tempo, per far fronte così al rischio che il numero di pazienti risulti essere maggiore della quantità di ventilatori disponibili negli ospedali e nelle strutture affini.

1.2 Contesto dell'applicazione

Per le motivazioni riportate nella sezione 1.1, si è assistito alla volontà e al desiderio di molti protagonisti di mettersi in gioco dando vita a nuovi progetti e a diverse iniziative: tra queste si colloca MVM (vedi sezione 1.3).

Al progetto MVM hanno partecipato diverse realtà sia istituzionali che private, sia italiane che estere: anche l'Università degli Studi di Bergamo, con il gruppo di ricerca guidato dal prof. Angelo Gargantini, sta partecipando attivamente allo sviluppo del software di gestione del respiratore stesso.

Nello specifico, il punto che verrà trattato in questo elaborato, sarà quello relativo all'analisi della possibilità di rendere adattivo il ciclo di respirazione di MVM alle necessità e abilità del paziente sottoposto alla *ventilazione meccanica* utilizzando tecniche, poi ampiamente discusse e presentate, basate sull'*Intelligenza Artificiale*.

L'obbiettivo è quindi quello di poter ottenere dei modelli in grado di rilevare automaticamente lo sforzo di inspirazione (il cosiddetto *trigger inspiratorio*) e di espirazione del paziente, per far si che il ciclo di controllo di MVM sia in grado di adattare il proprio supporto alla respirazione in maniera *real-time* a quelle che sono le esigenze del paziente, modificando i parametri caratterizzanti il controllo del respiro.

Questo approccio adattivo risulta essere importante per evitare problemi di asincronismo tra le capacità di respirazione del paziente e il supporto fornito dal respiratore: questo mismatch può portare infatti ad una condizione definita in gergo medico di *PVA* (*Patient Ventilator Asynchrony*), la quale potrebbe causare danni ai polmoni e un'aumento della mortalità per i pazienti sottoposti a ICU [5].

L'analisi di questi asincronismi tra ciclo di respirazione del paziente e ciclo di respirazione del ventilatore è solitamente realizzata tramite un'ispezione visiva del medico/infermiere sulle forme d'onda di flusso e pressione: questa azione risulta però essere eccessivamente dispendiosa in termini di tempo, con una forte soggettività sulle analisi effettuate e sulle decisioni prese.

Da qui si capisce come, la presenza di modelli in grado di lavorare in maniera continua, risulti essere un punto cruciale in un contesto delicato come quello della ventilazione meccanica.

1.3 MVM: Mechanical Ventilator Milano

Mechanical Ventilator Milano (*MVM*) è un respiratore polmonare ideato circa un anno fa, nel marzo 2020, su idea e iniziativa di un gruppo di ingegneri e fisici guidati dal professor *Cristiano Galbiati*: questo ventilatore è stato pensato per essere industrializzato e prodotto sin da subito così da far fronte, nella maniera più rapida e versatile possibile, all'avanzamento dell'epidemia da COVID-19 [4].

La velocità di produzione e, soprattutto, di replicazione del progetto è legata al fatto che esso basa il suo funzionamento su componenti facilmente reperibili *"off-the-shelf"* nel mercato e a basso costo, per arrivare così ad ottenere un ventilatore affidabile e facile da utilizzare [7].

MVM riprende la struttura e le specifiche di funzionamento del ventilatore *Manley*[17] utilizzando però un design completamente differente, in cui tutte le parti meccaniche mobili sono state sostituite con componenti elettro-meccanici: questo ha permesso così di ottenere un controllo migliore dei parametri respiratori ed un incremento della robustezza e dell'affidabilità sulle operazioni di ventilazione a lungo termine.

Nello specifico MVM è stato progettato per gestire la respirazione meccanica del paziente secondo due approcci:

- PCV (*Pressure Control Ventilation*) (vedi sezione 1.4.3);
- PSV (*Pressure Support Ventilation*) (vedi sezione 1.4.4).

La scelta di supportare solamente modalità gestite in pressione (evitando quindi di considerare anche approcci basati sul controllo del volume di aria inspirato) è legata al fatto che, le modalità *pressure based*, risultano essere più indicate e appropriate per pazienti affetti da COVID-19 dato che la somministrazione di una ventilazione con una pressione eccessiva potrebbe causare un danno ancora maggiore ai polmoni, già gravemente compromessi [4].

Si vuole quindi far si che la macchina sia in grado di supportare il paziente anche nella decisione di quando fare un respiro, rendendo quindi autonomo il paziente stesso per quanto riguarda l'inizio dell'atto respiratorio [21].

1.4 La respirazione e la ventilazione meccanica: cenni

Il presente elaborato non entrerà nello specifico del funzionamento dell'apparato respiratorio: in questa sezione si cercherà comunque di esporre alcuni concetti di base inerenti al funzionamento della respirazione e della ventilazione meccanica.

1.4.1 La respirazione

La respirazione è un meccanismo naturale: è un riflesso talmente spontaneo che, per la maggior parte del tempo, non ci accorgiamo neanche che lo stiamo compiendo.

La respirazione consiste nell'assunzione di ossigeno e nell'eliminazione di anidride carbonica in due fasi:

- Inspirazione;
- Espirazione.

Nella prima delle due fasi l'aria, dopo aver attraversato le vie aeree superiori e i bronchi, penetra negli alveoli polmonari i quali vanno a trattenere le particelle di ossigeno; nella seconda invece i polmoni espellono l'aria presente al loro interno, rilasciando nello specifico l'anidride carbonica.

Il ritmo della respirazione è automatico ma, i muscoli coinvolti, sono volontari e ogni loro contrazione e successivo rilassamento sono stimolati da impulsi nervosi.

1.4.2 La ventilazione meccanica

La ventilazione meccanica rappresenta una forma di terapia strumentale che, attraverso un macchinario esterno, supporta il paziente affetto da insufficienza respiratoria grave, permettendogli di ventilare adeguatamente le cave polmonari e di mantenere scambi gassosi nella norma fra polmoni e ambiente [24], andando a sostituirsi così a quel meccanismo di contrazione volontaria dei muscoli, venuto meno nel paziente.

Quando la ventilazione viene eseguita correttamente, secondo quelle che sono le condizioni e le esigenze del paziente, essa può permettere ai polmoni di riprendersi evitando appunto di essere responsabili dello sforzo respiratorio il quale sarà sobbarcato dal ventilatore, che si farà carico della maggior parte del lavoro per far respirare il paziente.

Quando invece viene eseguita in maniera errata, la ventilazione meccanica può generare condizioni cliniche avverse come:

- Scomfort del paziente durante la respirazione;
- Prolungamento della necessità di sedazione;
- Nei casi più drastici, diminuzione della probabilità di sopravvivenza.

Da qui si comprende come, la necessità di fornire una ventilazione il più possibile su misura e in grado di adattarsi alle necessità del paziente, sia un concetto estremamente prioritario.

In precedenza è stato introdotto il fatto che, il ventilatore MVM, basa la gestione della respirazione su un controllo in pressione; più in generale i cicli di respirazione possono essere gestiti in due modalità di controllo principali (vedi figura 1.1):

- *Volume Control*: la variabile controllata è quella relativa al *tidal volume*, ovvero alla quantità di volume totale di aria con la quale i polmoni vengono ad essere riempiti, la quale verrà mantenuta costante.

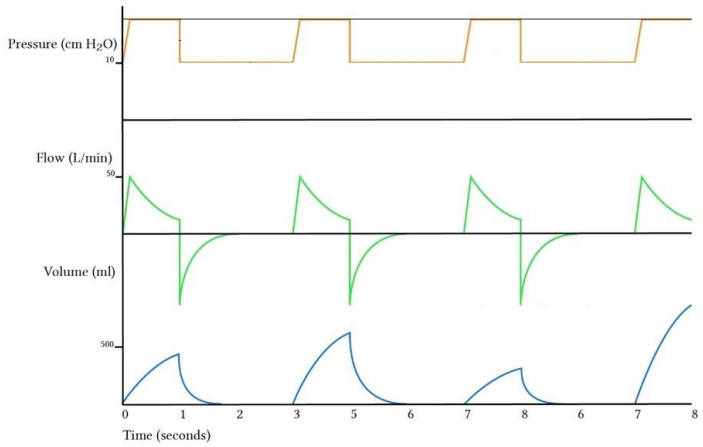
  Come conseguenza del controllo in volume, si ottiene un flusso che si muove con andamento piatto (essendo che flusso e volume sono direttamente legati da un operazione di integrazione);

- *Pressure Control*: qui invece il respiratore va a controllare la pressione inspiratoria, che viene mantenuta costante durante tutta la fase di inspirazione a prescindere dal volume corrente che sarà poi sviluppato negli alveoli polmonari del paziente (il quale dipende a sua volta da molti fattori fisiologici).

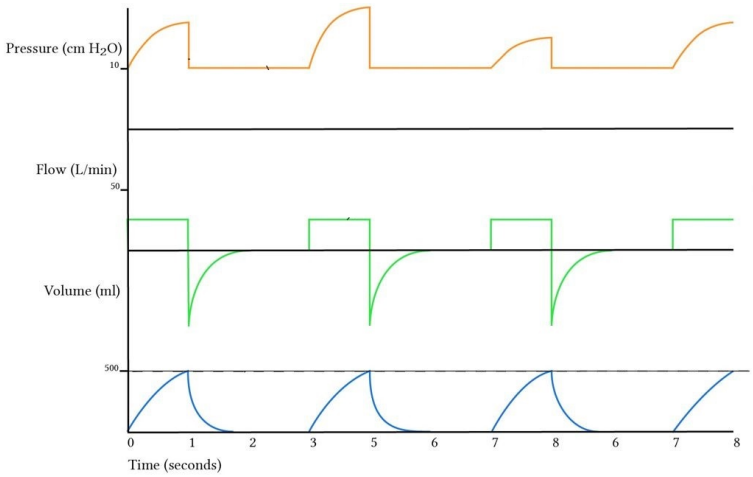
  Come sottolineato nella suddivisione fatta nella sezione 1.3, il respiratore meccanico MVM è dotato di due modalità di respirazione, basate entrambe su un approccio *pressure control* per motivi come maggior comfort e minor rischio di danneggiamento degli alveoli polmonari, che derivano dall'utilizzo di questa tipologia di ventilazione.

Proviamo ora ad approfondire queste due modalità di ventilazione, per riuscire così ad avere una conoscenza quantomeno sufficiente per comprendere i risultati ed essere in grado di effettuare delle analisi grafiche sulle curve di flusso e di pressione con cui andremo a lavorare in fase di studio delle tecniche di monitoraggio che verranno poi proposte.

In particolare, indipendentemente da quella che è la variabile tramite la quale viene effettuato il controllo della respirazione, si possono definire due modalità di interazione paziente-ventilatore, che differiscono sostanzialmente da chi è il responsabile della decisione di due parametri caratterizzanti la respirazione, ovvero il *tempo inspiratorio* (T_i) e il *Frequency Rate* (FR), ovvero la frequenza con cui si ripete un ciclo respiratorio [3].



(a) *Esempio di respirazione con controllo in pressione (curve di pressione con andamento costante)*



(b) *Esempio di respirazione con controllo in volume (curva di volume con crescita costante da cui ne deriva curva di flusso con andamento costante)*

Figura 1.1: Esempi di curve di ventilazione

Possiamo avere quindi due tipologie di interazioni:

- *Controllata*: la ventilazione spontanea del paziente è completamente sostituita da una respirazione forzata da parte del ventilatore, il quale va a realizzare gli atti inspiratori automaticamente ad intervalli fissi stabiliti dall'operatore tramite parametri specifici;

- *Assistita*: in questa modalità il paziente, che respira costantemente in maniera autonoma, viene assistito durante l'atto inspiratorio da un aumento di pressione erogato dal ventilatore. Per ovvie ragioni di sicurezza, in tutte le modalità di ventilazione assistita è solitamente possibile impostare un numero minimo di atti inspiratori spontanei al di sotto del quale vengono erogati atti controllati

(ventilazione di backup) in modo da garantire una frequenza respiratoria minima indipendentemente dal drive respiratorio del paziente.

Andiamo ora ad approfondire i due aspetti relativi alla modalità controllata e assistita presentati qua sopra, applicandoli al contesto della respirazione con controllo in pressione.

1.4.3 PCV

La *Pressure Control Ventilation* rappresenta una modalità di ventilazione forzata, in cui è il ventilatore a determinare il tempo di inspirazione senza che vi sia la partecipazione del paziente: è per questo motivo che tale modalità viene utilizzata nella fase più acuta dell'insufficienza respiratoria, quando il paziente è profondamente sedato e paralizzato.

In questo contesto l'operatore imposta una pressione di picco inspiratorio (PIP) e il ventilatore insuffla aria fino al raggiungimento di tale valore: raggiunto questo limite il ventilatore interrompe l'insufflazione mantenendo questo valore costante e aprendo la valvola che consente la fuoriuscita dell'aria dopo un determinato lasso di tempo, dando poi il via alla fase espiratoria.

Oltre a questo parametro principale l'operatore ne va ad impostare anche altri, tra cui:

- livello di pressione positiva finale;
- la lunghezza, in termini di tempo, della fase inspiratoria;
- il numero di respiri al minuto (FR).

Come si vede in figura 1.2 la pressione raggiunge un valore massimo, impostato dall'operatore, dal quale non si discosta: in questo modo si è in grado di garantire una protezione da sovra pressioni ai pazienti sottoposti a ventilazione assistita.

1.4.4 PSV

La modalità definita invece *Pressure Support Ventilation*, chiamata anche *patient-triggered*, viene utilizzata quando il paziente respira spontaneamente (essendo che ogni atto respiratorio è iniziato da lui) ma che non è ancora pronto per essere estubato.

Qui il ventilatore applica una pressione costante di supporto nelle vie aeree, dal momento in cui viene rilevato un inizio di inspirazione per tutta la durata di questa fase, che si sincronizza con lo sforzo inspiratorio del paziente.

Il trigger inspiratorio viene associato ad una diminuzione della pressione, individuato come un avvallamento nella serie temporale rappresentante l'andamento della pressione. Il funzionamento infatti è il seguente: durante l'espirazione la valvola inspiratoria del ventilatore è chiusa; nel momento in cui il paziente cerca di iniziare

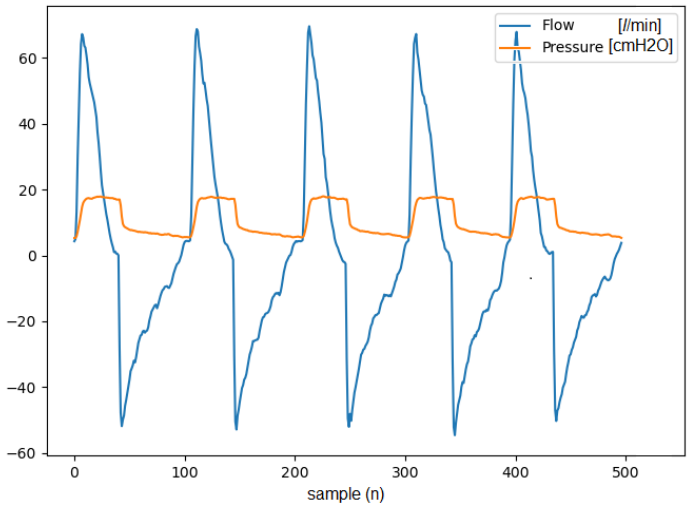


Figura 1.2: Altro esempio di forme d'onda di flusso e pressione nella modalità PCV

l'inspirazione successiva, si genera una pressione negativa nel circuito del ventilatore. Infatti, come descritto dalla legge di Boyle, essendo che il prodotto di pressione e volume è costante, se aumenta il volume dei polmoni (= inspirazione) ma non vi entra nuovo gas (valvola inspiratoria chiusa) la pressione diminuisce.

Questa diminuzione corrisponde all'evento di trigger inspiratorio.

La pressione di supporto offerta dal ventilatore va poi ad arrestarsi quando il flusso inspiratorio del paziente scende a un livello percentuale impostato dall'operatore da un valore soglia assoluto (solitamente sotto il 30% del picco) o, più generalmente, quando il flusso misurato inverte il proprio segno, riportando il ciclo respiratorio alla sua baseline.

Nel caso in cui il paziente non vada a triggerare alcun inizio di respirazione entro una certa finestra temporale (definita come *apnea-trigger time window*), MVM modifica il controllo da PSV a PCV, così da garantire una frequenza respiratoria minima anche in caso di problemi di respirazione e attivando allo stesso tempo un allarme.

1.5 Tecnologie e tools utilizzati

- Python 3.6;
- Keras;
- Google Colab;
- Libreria open source *ventMAP* [11];
- Tools per la visualizzazione del design della rete neurale *Net2Vis* [2];
- Computazione di train svolta su server @Arera w/ Ubuntu 18.04.5 LTS.

2

Scopo e contesto del progetto

2.1 Obiettivo del progetto

Come già introdotto nella sezione 1.1 l'obbiettivo di questo elaborato è quello di approfondire la possibilità di utilizzare reti convoluzionali in grado di estrarre informazioni utilizzabili dal ciclo di controllo del respiratore per rendere adattiva la ventilazione partendo dalle forme d'onda di flusso e pressione.

Questo obbiettivo è stato raggiunto step by step, capendo inizialmente la disponibilità di datasets su cui poter lavorare e, successivamente, cercando di implementare differenti scelte per capire quale potrebbe essere il modello più adatto per andare ad identificare gli istanti di inizio inspirazione ed espirazione: questi istanti individuati dal modello convoluzionale, nell'idea attuale del progetto, verranno poi ad essere utilizzati come supporto alle tempistiche del ciclo di respirazione di MVM, per capire quindi la presenza o meno di asincronismi e per adattare di conseguenza il ciclo stesso.

Si è quindi delineato un problema di *multivariate time series* in un contesto di *multi class classification* che è stato approcciato ed analizzato secondo due diversi strade, che verranno poi ampiamente documentate nelle sezioni successive e la cui visione d'insieme è presentata in figura 3.8.

2.2 Stato attuale MVM

La respirazione meccanica, soprattutto quella invasiva, comporta forti rischi di infezione, danneggiamento dei polmoni, intossicazione da ossigeno e anche rischi legati ad errori degli operatori.

Per far fronte agli elevati standard di sicurezza richiesti dalle organizzazioni mediche, attualmente MVM presenta un sistema integrato di allarmi molto sofisticato basato sullo standard EN 60601 - 1-8:2007, il quale garantisce che siano rispettati e soddisfatti vincoli sul ciclo di respirazione e che vengano ad assere sollevati allarmi in caso di anomalie.

Nel corso degli sviluppi, il progetto MVM ha inoltre ottenuto diverse certificazioni (vedi [19]) tra cui:

- L'organizzazione *Health Canada*, nel settembre 2020, ha autorizzato Vexos per la produzione di 10000 unità del ventilatore MVM destinate alla vendita e all'importazione sul territorio Canadese;
- MVM è stato classificato dalla U.S. FDA (*U.S. Food and Drug Administration*), all'interno del proprio sistema regolatorio a 3 classi;
- MVM ha ricevuto l'autorizzazione per l'utilizzo d'emergenza (EUA).

Per quanto concerne invece lo stato attuale del sistema di *trigger detection* di cui MVM è dotato, si ha che attualmente il respiratore utilizza un approccio analitico legato ad un'analisi matematica delle features delle forme d'onda.

Si è infatti andato ad implementare, via software, un calcolo derivativo [1] utile appunto per rendere il respiratore in grado di rilevare le presenza di avvallamenti nella curva di pressione corrispondenti al trigger di inizio respirazione.

3

CNN applicate alla respirazione

Generalmente, quando si parla di CNN (*Convolutional Neural Networks*), si fa riferimento a modelli che lavorano in un contesto di classificazione di dati 2D (tipicamente immagini): ci sono però anche altre due tipologie di reti convoluzionali che possono essere utilizzate, ovvero quelle mono-dimensionali (1D) e quelle tri-dimensionali (3D), a seconda appunto del tipo di informazione che il modello è in grado di elaborare.

Come tutti i modelli nell'ambito del Machine Learning, anche le reti neurali si prefiggono l'obbiettivo di *imparare* una funzione ignota f (in uno spazio delle ipotesi H) fornito un certo insieme di dati D: questa funzione sarà poi in grado di fornire supporto per una corretta gestione del problema di classificazione che si sta cercando di risolvere.

In particolare, il modello complessivo, riceverà in input un vettore di dati e andrà a trasformarlo attraverso una serie di layers nascosti: ognuno di essi presenta una struttura tale da fornire un comportamento non lineare per mezzo di specifiche funzioni di attivazione.

In questo modo si è in grado di ottenere un'unica e complessa funzione tra tutte quelle possibili nello spazio delle ipotesi a cui si faceva riferimento in precedenza: la complessità di tale funzione ottenuta dipende in particolare dal numero di layers che vengono ad essere accumulati uno affianco all'altro.

Questi layer della rete vengono attraversati in due direzioni opposte durante la fase di train, andando prima in una direzione e poi in quella contraria. Esse rappresentano due step sequenziali, che permettono di aggiornare i pesi della rete e di calibrare quello che è l'output fornito dal modello in modo da minimizzare una funzione di loss prescelta: si può vedere quindi come sia una tecnica utile per risolvere sostanzialmente un problema di ottimizzazione.

Le due fasi a cui ci si riferisce sono:

- *Feed-forward*: questo step consiste nell'introdurre i dati nella rete, calcolando poi le varie combinazioni lineari che permettono di comporre i logits a cui poi sarà applicata una funzione di attivazione non lineare (l'applicazione di una

funzione lineare non consente infatti di ottenere l'estrazione delle features che si vogliono isolare). Ripetendo questo step per ogni layer si può andare a calcolare, in maniera diretta, il valore in uscita dalla rete.

Il suo nominativo deriva dal fatto che il flusso di calcolo avviene in maniera naturale nella direzione "in avanti" (ovvero *forward*), partendo dall'input, attraversando la rete e giungendo all'output;

- *Back-propagation*: andando ad utilizzare il concetto matematico di derivata, la quale gode della proprietà di decomposizione, siamo in grado di procedere verso un'ottimizzazione della funzione di costo (o di loss) definita.

  La domanda che guida questa seconda fase, riguarda il fatto di chiedersi di "quanto cambierà l'errore totale in uscita se si vanno a modificare i pesi interni della rete di una certa quantità δ?", che altro non fa che coinvolgere il concetto di derivata delle funzione di costo.

  Tramite questo algoritmo l'errore, espresso dalla funzione di costo, viene propagato all'indietro dal layer di uscita a quello di input per mezzo di una sequenza di operazioni matematiche basate sul concetto di *chain rule* applicabile alle operazioni di derivazione.

  L'andamento derivativo, ovvero l'espressione di quanto varia il valore dell'uscita al variare dei pesi della rete, esprime un comportamento prettamente locale: è per questo motivo che è preferibile (anzi, necessario) limitare l'update dei pesi a valori estremamente piccoli tramite l'utilizzo di un *learning rate* all'interno dell'equazione di update come riportato qualitativamente nella seguente equazione.

  $$\theta = \theta - \alpha \frac{\partial}{\partial \theta} J(\theta)$$

  Questo comporterà la necessità di avere un maggior numero di iterazioni prima di giungere a convergenza ma, allo stesso tempo, consente di evitare comportamenti non desiderati del train del modello per via delle non linearità nelle funzioni di attivazione.

Questi concetti di feed-forward e, soprattutto, di back-propagation si basano su una corposa parte di concetti matematici: essi però non devono essere codificati direttamente, per il fatto che *Keras* offre una forte astrazione di questi passaggi rendendo la loro fruizione molto più snella e veloce.

E' per tale motivo che, in questa sede, non sono stati presi in considerazione tai aspetti, estremamente dettagliati, che guidano il train del modello stesso.

3.1 Classificazione VS Segmentazione

L'utilizzo di tecniche relative al Deep Learning ha permesso di fare enormi passi avanti negli ultimi anni, specialmente nel contesto della *computer vision*: data l'enorme diffusione, possiamo infatti andare ad individuare diversi livelli di granularità in cui la *computer vision* stessa ha acquisito sempre maggiori abilità nello studio e nell'analisi dei dati.

Per presentare quello che è stato l'approccio seguito in questo elaborato e come sono stati utilizzati i modelli convoluzionali, partiamo con il presentare alcuni contesti principali in cui le reti neurali trovano maggiore utilizzo (in questo frangente si farà riferimento, per semplicità, al contesto in cui l'input fornito alla rete neurale siano immagini, ovvero informazioni 3D):

- **Classificazione**: si tratta dell'approccio classico nel quale trovano ampio utilizzo le CNN. Qui, fornita un'immagine in ingresso, ci si aspetta che la rete neurale sia in grado di assegnare a tale immagine un'etichetta, ovvero una classe di appartenenza. Il concetto fondamentale è che viene assunto per vero il fatto che, nell'immagine, sia presente una e una sola entità sottoposta a classificazione;

- **Classificazione con localizzazione**: in questo contesto, oltre alla classificazione dell'elemento nell'immagine, la rete neurale è in grado di andare a delimitare e localizzare tale oggetto tramite un riquadro (definito anche *bounding box*), il quale però continua ad essere l'unico presente nella scena;

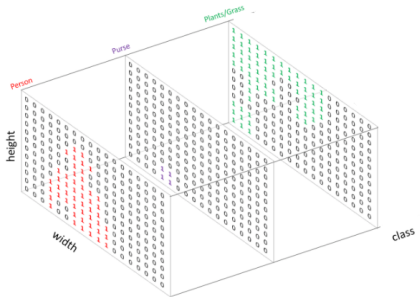
- **Object detection**: qui si estende il concetto di localizzazione in cui, oltre a definire dove si trova un certo oggetto nell'immagine, si va anche a delineare di quale oggetto si tratta. Vengono perciò considerate valide anche immagini contenenti istanze multiple di oggetti che si intende andare a classificare;

- **Semantic Segmentation**: questo approccio è definito anche come *dense prediction*, essendo che si va a realizzare una classificazione di ogni singolo pixel. Uno dei punti da mettere in evidenza è che, se all'algoritmo viene somministrata una scena con tre auto, esso sarà in grado di definire, in maniera più o meno precisa, l'insieme di pixel che concorrono a rappresentare l'entità "auto".

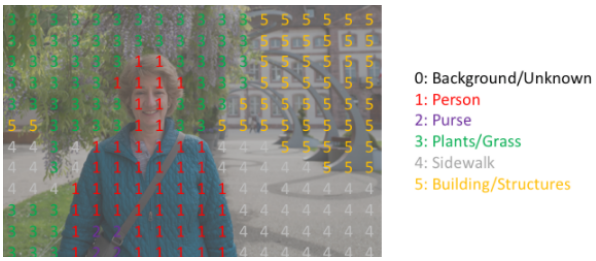
  Esso però andrà ad etichettarle tutte e tre come auto e senza distinzione, a differenza dei contesti precedenti, non riuscendo a distinguerle tra di loro (vedi esempio in figura 3.1);

- **Instance Segmentation**: rappresenta lo stesso approccio presentato per la semantic segmentation con l'unica differenza che, oltre a classificare ogni singolo pixel, la rete neurale è in grado di differenziare un'istanza di un oggetto da un'altra istanza dello stesso oggetto.

(a) *Immagine originale*



(b) *Per ogni possibile classe viene creato un canale in uscita nel formato one-hot encoding*



(c) *Sovrapposizione dei label assegnati ad ogni pixel*

Figura 3.1: Esempio di *semantic segmentation* applicato nel campo della *computer vision* [10]

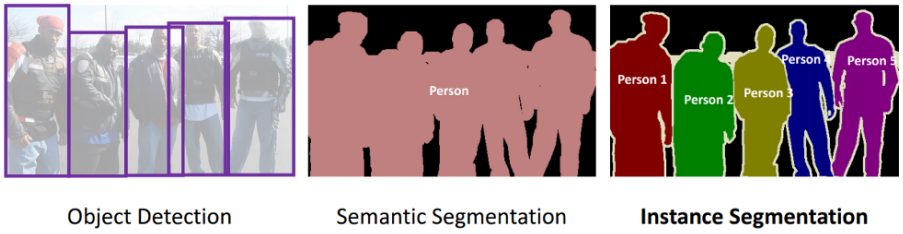


Figura 3.2: Comparazione del livello di granularità tra i vari contesti della classificazione

In figura 3.2 sono riportate delle esemplificazioni di alcuni dei principali approcci presentati: in particolare, nei contesti di *semantic segmentation* e *instance segmentation*, è chiaro come l'uscita non sarà semplicemente una singola etichetta o un insieme di coordinate dei vertici delle bounding box, ma sarà un'immagine (tipicamente con le stesse dimensioni dell'immagine di input) in cui ogni pixel risulta essere classificato con una specifica classe tra quelle disponibili per quel determinato problema.

Nello specifico, in questo elaborato, andremo a concentrarci sul contesto della *semantic segmentation* in cui, invece di classificare ogni singolo pixel come nell'ambito delle immagini, ci porremo il problema di andare a classificare ogni singolo istante lungo l'asse temporale.

3.2 CNN e i suoi blocchi fondamentali

Come introdotto nei capitoli precedenti, una rete convoluzionale può essere vista come un insieme di layers sequenziali, ognuno dei quali implementa un comportamento non lineare che permette di estrarre, in maniera automatica, le features maggiormente rilevanti dell'informazione fornita in ingresso.

I blocchi fondamentali, i quali vanno a definire il comportamento della rete neurale, hanno una struttura variabile e che può essere fortemente customizzata a seconda delle esigenze: ognuno di loro infatti può presentare o meno dei parametri che devono essere appresi dalla CNN e avere oppure non avere degli iper-parametri che è necessario impostare in maniera arbitraria in fase di design del modello, a seconda del contesto del problema.

Andiamo di seguito a presentare i due blocchi fondamentali.

3.2.1 Layer convoluzionale

Come si può intuire dal nome, il blocco convoluzionale è uno degli elementi fondamentali nel contesto delle CNN.

Nello specifico questo layer corrisponde all'implementazione dell'operatore matematico di convoluzione tra funzioni, molto utilizzato anche nell'ambito della *signal processing*: esso permette, fornite in ingresso due funzioni (f e g), di ottenere una terza funzione ($f * g$), la quale esprime come la forma di una funzione risulti essere modificata dall'altra.

Nel contesto delle reti neurali si va quindi ad effettuare una convoluzione tra quella che è l'informazione in ingresso rispetto ai valori del filtro stesso, i quali non sono altro che i parametri che la rete andrà ad apprendere durante la fase di train.

Si tratta perciò di applicare una convoluzione sulla matrice di input, che significa semplicemente andare ad effettuare una moltiplicazione termine per termine, sommando in seguito i singoli prodotti: il risultato che si ottiene, facendo muovere il filtro in tutte le dimensioni disponibili, è ancora una matrice (figura 3.3) contenente le features estratte dall'input.

In particolare, possiamo andare a riassumere quelle che sono le dimensionalità coinvolte nel contesto della convoluzione:

- CNN 1D: il filtro si muove in una sola direzione. L'input e l'output della rete sono bidimensionali (serie temporali);

- CNN 2D: nonché la tipologia di convoluzione più utilizzata, in cui il filtro ha 2 direzioni possibili per il movimento dato che l'informazione con cui si ha a che fare è tridimensionale (immagini);

- CNN 3D: si ha che il kernel si muove in 3 direzioni, per il fatto che i dati in input e output risultano essere 4D (MRI - *Magnetic Resonance Imaging*, CT Scans - *Computer Tomography*).

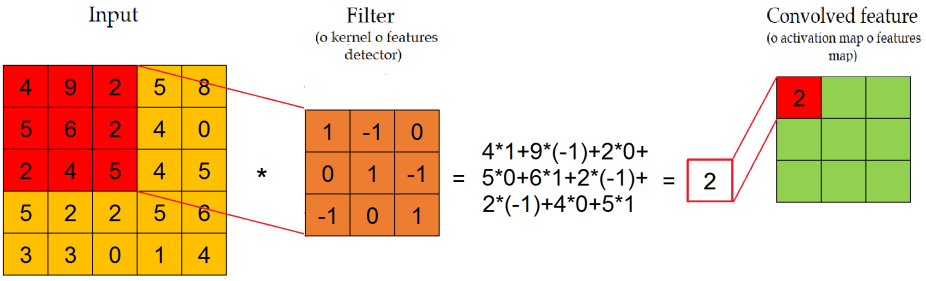


Figura 3.3: Esempio di operazione di convoluzione applicato a dati 2D [20]

Nel gergo tecnico delle CNN la matrice convoluta sull'immagine (o più in generale sui dati in ingresso al layer) è chiamata *filtro*, *kernel* o *feature detector* (proprio per il fatto che il suo comportamento è quello di un "agente" in grado di mettere in evidenza le features principali dell'input stesso).

La matrice che invece si ottiene dall'operazione di moltiplicazione tra il filtro e l'area dell'immagine (che varia durante la convoluzione) prende il nome di *convolved feature*, *activation map* o *feature map*: viene spontanea la deduzione che, filtri differenti, portino ad ottenere feature maps diverse a parità di ingresso.

Si può facilmente intuire come la dimensione del kernel risulti essere un iper-parametro che permette di andare a definire quella che è l'area ricettiva di ogni filtro stesso: maggiore sarà questa dimensione più grande sarà l'area sottoposta ad una singola convoluzione ma minore sarà la granularità e il dettaglio dell'informazione estratta.

Ci sono altri iper-parametri che consentono di definire le dimensioni delle feature maps in uscita dal layer convoluzionale e che devono essere scelti in fase di definizione del design della rete stessa. Essi sono (vedi figura 3.4):

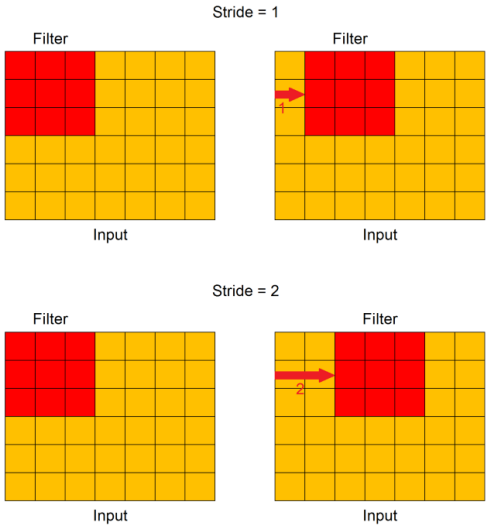
- *Numero di filtri*: indica quanti filtri sono utilizzati nella fase di convoluzione. Questo parametro permette di definire quante sono le features che si desidera estrarre: è ovvio che, maggiore sarà il numero di filtri, più numerose saranno le features estratte dal layer, essendo che in uscita si ha un volume la cui profondità coincide con il numero di filtri scelto;

- *Stride*: è il numero di pixel, o più in generale di unità, di cui il filtro va a spostarsi ad ogni iterazione. Avere perciò uno stride settato a valori elevati comporta il fatto di ottenere feature maps più piccole in uscita;

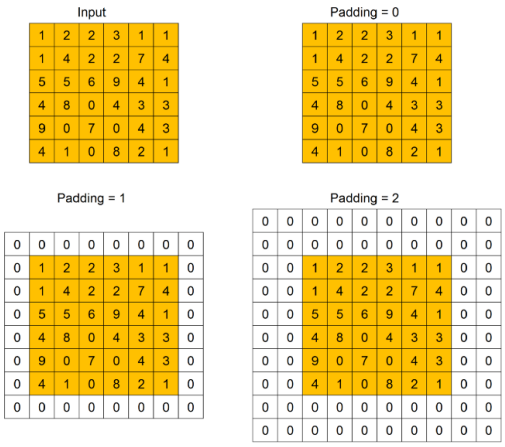
- *Zero padding*: questo parametro invece indica quante righe e colonne extra risulta essere necessario aggiungere fuori dai contorni del vettore di input. Esso è

utilizzato perché consente di evitare la diminuzione delle dimensioni dell'output rispetto all'input per evitare che, dopo parecchi layers convoluzionali, la feature map diventi effettivamente molto piccola: in questo modo, aggiungendo un contorno di "pad", non modifichiamo il contenuto informativo ma preserviamo la dimensione del dato. Nel caso in cui la matrice prodotta dalla convoluzione sia dimensionalmente uguale a quella in input si parla di padding *same*; invece, se le dimensioni vengono ridotte, per via del naturale funzionamento della convoluzione, si parla di padding *valid*.

(a) *Numerosità dei filtri*



(b) *Stride*



(c) *Padding*

Figura 3.4: Principali iper-parametri impostabili per i layer convoluzionali [20]

Un ulteriore iper-parametro utilizzabile nell'ambito della convoluzione è quello definito come *dilation rate*, il quale permette di settare lo spazio presente tra i valori del kernel.

Per esempio, un filtro di dimensioni 3×3 con un dilation rate pari a 2 avrà una stessa area ricettiva di un filtro 5×5 utilizzando però solamente 9 parametri: da questo si può dedurre come esso sia un settaggio particolarmente utilizzato nel contesto della segmentazione real-time, perché permette di diminuire il peso computazionale, garantendo comunque buone prestazione di convoluzione.

Questo approccio definisce perciò una differente tipologia di convoluzione, la quale permette al kernel di estrarre features da un campo di azione più largo mantenendo allo stesso tempo invariato il costo computazionale, essendo che non si ha un corrispettivo aumento dei parametri che devono essere appresi.

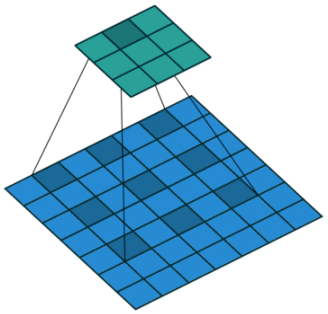


Figura 3.5: Esempio di dilated convolution con kernel di dimensione 3×3, senza padding e con un coefficiente di dilation pari a 2

3.2.2 Pooling

L'operazione di pooling permette di ottenere una significativa riduzione delle dimensioni della feature map, riducendo il numero di parametri che il modello deve apprendere e limitando la possibilità di incorrere in overfitting sui dati (per la troppa complessità del modello, il quale potrebbe tendere ad adattarsi il più possibile ai dati usati per il train, non permettendo il necessario processo di generalizzazione).

Il pooling consiste sostanzialmente nell'andare ad applicare una sliding window di dimensione fissa sulla feature map in ingresso, facendola scorre lungo tutte le sue dimensioni: essendo che, come vedremo successivamente, l'operazione di pooling è fissa (in base a quella che è la scelta della tipologia in fase di design della rete), si ha che esso rappresenta un layer il quale non necessita di parametri che devono essere appresi, come si evince dalla figura 3.6.

C1_encoder_4 (BatchNormaliza	(None, 16, 1, 16)	64
encoder_5 (MaxPooling2D)	(None, 8, 1, 16)	0
encoder_6 (Conv2D)	(None, 8, 1, 32)	2592
encoder_7 (BatchNormalizatio	(None, 8, 1, 32)	128

Figura 3.6: Numero di parametri richiesti dal layer di MaxPooling (ultima colonna)

Ci sono principalmente due tipologie di pooling:

- *Max Pooling*: per ogni sliding window, il risultato dell'operazione di pooling corrisponde al valore massimo permettendo così anche una reiezione del rumore, essendo che vengono mantenute solamente le informazioni più importanti (vedi figura 3.7);
- *Average Pooling*: in questo caso invece, il risultato corrisponde al valore medio tra i valori compresi nella finestra.

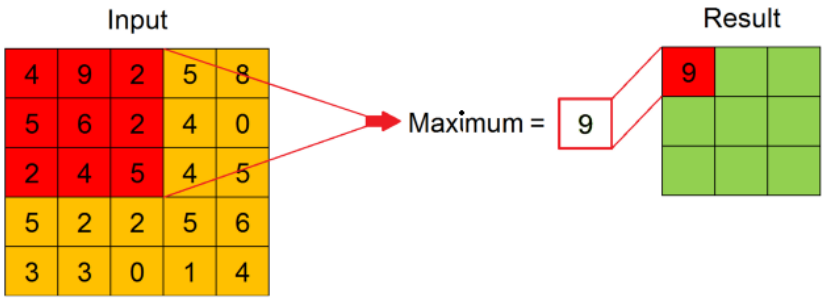


Figura 3.7: Esempio di applicazione di Max Pooling a dati 2D con $pool\_size = (3, 3)$ [20]

Come sottolineato per i blocchi convoluzionali, anche per le operazioni di max e average pooling si va a definire quella che è la grandezza della finestra sulla quale deve essere eseguita l'operazione di massimo o media.

Questo parametro, solitamente definito *pool_size*, caratterizza l'aggressività della riduzione dimensionale che il layer mette in atto: maggiore sarà questo valore più veloce sarà la riduzione dimensionale che verrà operata attraverso i vari layer, ma meno precisa risulterà essere la rappresentazione dell'informazione di ingresso.

3.3 CNN e serie temporali

Dopo aver introdotto, in maniera generica e ad alto livello i concetti base per l'utilizzo e l'applicazione delle reti convoluzionali, andiamo ora a restringere il campo d'azione per l'applicazione di questi modelli neurali all'ambito della respirazione polmonare, ovvero per la classificazione delle serie temporali.

In particolare possiamo definire le serie temporali come una sequenza ordinata di valori reali, appartenenti nello specifico a due macro categorie a seconda della dimensionalità dei dati:

- Serie temporali *uni-variate*, se sono rappresentate da un semplice lista ordinata di valori, in cui si ha un'unica variabile osservata lungo tutto l'asse temporale;
- Serie temporali *multi-variate*, se invece sono composte da m serie temporali uni-variate, che formano così una sequenza multipla di serie temporali.

3.3.1 Estrazione di features

Il problema dell'utilizzo e del trattamento delle serie temporali ha acquisito, negli ultimi anni, un'importanza sempre maggiore in diversi settori, essendo che sono aumentate a dismisura le fonti in grado di generare moli costanti e continue di dati temporali in diversi contesti, come per esempio in ambiti health-care, cyber security, finanza, marketing etc.

Queste aree sono diventate di conseguenza molto interessanti per l'applicazione di algoritmi in grado di elaborare informazioni provenienti dalle serie temporali stesse.

In particolare, ognuno di questi ambiti richiede un modo di affrontare il problema che può seguire principalmente due filoni:

- Un primo approccio più classico e rivisitato in letteratura, è caratterizzato da due stadi principali di elaborazione:
  - Una fase in cui si vanno ad utilizzare o degli algoritmi (per esempio algoritmi di *dynamic time warping*) per misurare la differenza tra le serie temporali che devono essere classificate, o approcci statistici/analitici per rappresentare la serie temporale come un insieme di features (come per esempio l'analisi spettrale del segnale o altre features di stampo statistico);
  - In una seconda fase invece si vanno ad utilizzare algoritmi per classificare i propri dati, come per esempio KNN (*k-nearest-neighbors*) o SVM (*Space Vector Machine*), i quali richiedono l'estrazione di un insieme di features, più o meno complesse, in una fase antecedente alla vera e propria classificazione.

    La difficoltà in questa modalità sta nella parte di *feature engineering*, ovvero la fase in cui si selezionano e si definiscono le features utili e necessarie per

approcciare il problema: questo richiede la presenza e il supporto di esperti dell'area a cui il problema fa riferimento;

- Un secondo modo di approcciare il problema consiste nell'utilizzare le reti convoluzionali (CNN) le quali, dal canto loro, eliminano la necessità di estrarre queste features manualmente essendo che sono in grado di creare una rappresentazione informativa dei dati in maniera automatica tramite quella che è l'operazione base di questi modelli, ovvero la convoluzione (vedi sezione 3.2).

  Questo concetto ha reso quindi le CNN estremamente duttili e in grado di garantire elevate performances anche in diversi contesti e con differenti fonti di dati: si può quindi dedurre come, tramite questo processo di *features selection*, il modello sia in grado di capire "cosa", ovvero quale informazione è presente nell'input, perdendo però il concetto di "dove" questa informazione si trova.

Possiamo quindi riassumere come i metodi di approccio più classici, afferenti alla sfera del Machine Learning, seguano un approccio mirato alla *features selection*; dall'altro lato, una modellazione del problema tramite CNN consente invece un approccio tramite una soluzione mirata alla *features extraction* in cui è il modello stesso, grazie alle sue componenti non lineari, ad apprendere come e quali sono le features da considerare.

In particolare lo studio di questi modelli non lineari ha dimostrato che l'utilizzo delle CNNs per la classificazione delle serie temporali ha apportato notevoli vantaggi rispetto ai metodi più "classici", come per esempio maggiore robustezza al rumore e maggiore capacità di estrazione di informazioni indipendenti dal tempo.

L'utilizzo di reti convoluzionali nell'ambito della classificazione e, più nello specifico, della segmentazione, è un problema molto ben affrontato dalla community nell'ambito dell'*image processing* e della *computer vision*: nello specifico, come già messo in evidenza in precedenza, andare a *segmentare un immagine* (nell'accezione propria dell'ambito dell'AI) significa andare a classificare ogni singolo pixel dell'immagine assegnando quindi una specifica classe ad ognuno.

Quello che si vuole creare è quindi una classificazione per ogni singolo pixel: esemplificativa di questo concetto è la figura 3.1, in cui si vede come, la classificazione che si desidera ottenere in uscita avrà le stesse dimensioni (in termini di altezza e larghezza) dell'immagine in ingresso mentre presenterà una profondità variabile a seconda del numero di elementi che si vogliono classificare (nell'esempio, essendo 5 le entità che si vogliono classificare, avremo una profondità pari a 5, come si può vedere sempre in figura 3.1).

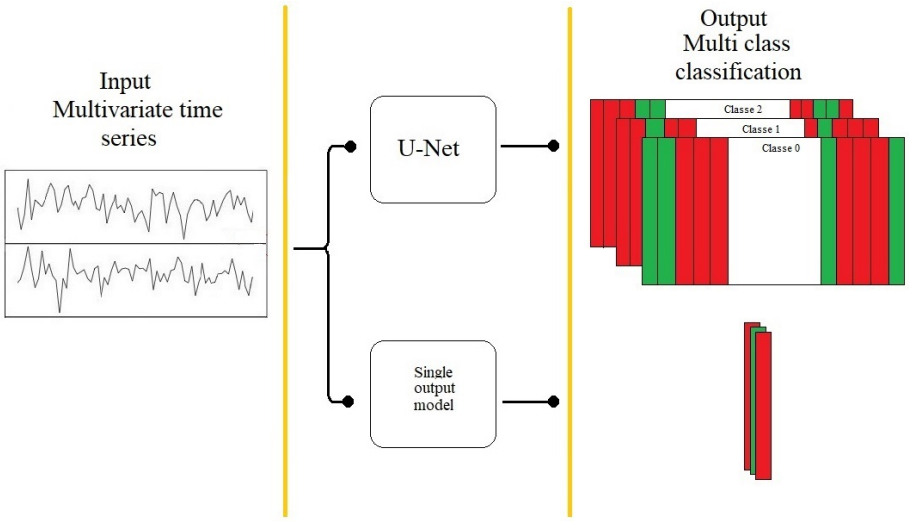


Figura 3.8: Framework di convoluzione applicato nel contesto di *multivariate time series* e *multi class classification* per MVM

3.4 CNN nel contesto MVM

In questo elaborato, come riportato in figura 3.8, andremo quindi a trasportare il concetto della semantic segmentation e della classificazione più in generale, dalle immagini alle serie temporali: nello stesso modo abbiamo pensato di utilizzare un modello per segmentare le waveforms di flusso e pressione, assegnando così ad ogni istante una classificazione (vedi figura 3.9), che potrà essere del tipo

- Inizio inspirazione;
- Inizio espirazione;
- Background: questa classe è necessaria per fornire un gruppo di appartenenza a tutti gli istanti che non sono classificati ne come inizio inspirazione ne come inizio espirazione, risultando essere complementare alle prime due.

Per questa prima tipologia di approccio, basato sul concetto di segmentazione, la scelta del modello da poter utilizzare sarebbe potuta ricadere su una rete composta da un insieme di layer convoluzionali con padding *same* così da mantenere, attraverso i vari layer, la dimensione dell'input ed ottenendo l'obbiettivo della segmentazione, ovvero quello di estrarre features riportandole poi sulla stessa dimensionalità dell'ingresso.

Questo approccio risulterebbe essere però troppo oneroso a livello computazionale ed è per questo che è molto più diffuso l'utilizzo di un approccio con design ad "U", con percorsi di encoder e di decoder.

Nel secondo tipo di modello implementato invece, siamo andati a considerare il problema dell'individuazione degli istanti di tempo di inizio inspirazione ed espirazione come un problema più classico di classificazione in cui, ad essere classificato, non è stata

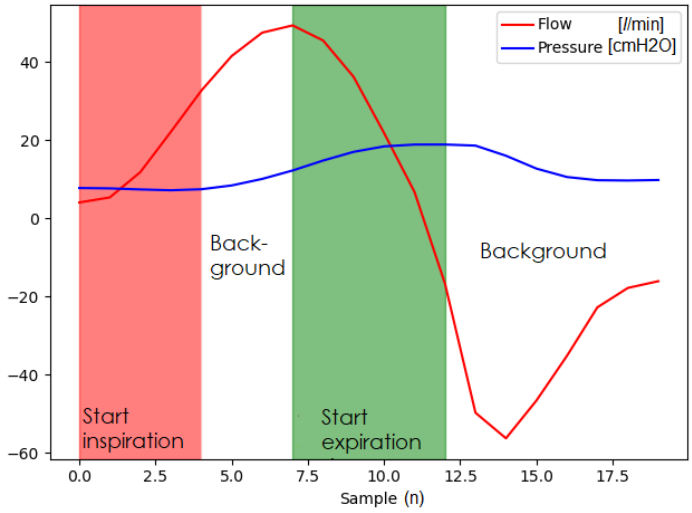


Figura 3.9: Esempio di applicazione del concetto di segmentazione ai segnali di flusso e di pressione

l'intera serie temporale fornita in ingresso ma solamente l'ultimo istante (ovvero quello più recente), così da avere un funzionamento maggiormente orientato ad un contesto di classificazione real-time dell'istante esattamente precedente a quello attuale in cui avviene la lettura dei valori di pressione e di flusso.

Riassumendo sono stati due gli approcci principali seguiti:

- modello U-NET (sezione 5);
- modello categorico a singola uscita (sezione 6).

Nel contesto MVM si ha quindi a che fare con un problema

- *multivariate time series*, poichè in ingresso abbiamo la presenza di più variabili, ovvero le curve di flusso e di pressione;
- *multi class/ single label problem*, essendo che si hanno, in uscita, più etichette possibili per ogni istante, nello specifico tre, ognuna mutuamente esclusiva con le altre due.

In particolare le etichette per ogni istante dovranno essere mutuamente esclusive tra di loro per il fatto che, nel nostro contesto, ad ogni istante corrisponde solamente una tipologia di classificazione: questo si differenzia dal contesto *multi-label*, in cui si hanno sempre più possibili classi in uscita ma con la possibilità di avere però un overlapping tra le classi stesse [8].

3.5 Configurazione dei modelli

Per entrambe le strade seguite, il cui design verrà poi ad essere presentato successivamente, sono quattro i fattori chiave da considerare e analizzare; tre di questi devono essere definiti e impostati durante lo step di compilazione del modello, come si vede nei listati 3.2 e 3.3. Essi sono:

- *callbacks*;
- *optimizer*;
- *loss function*;
- *metriche*.

3.5.1 Callbacks

L'utilizzo di callbacks è un elemento estremamente utile durante la fase di train del modello: esse permettono di compiere diverse azioni a vari livelli di granularità durante il train, in maniera tale da ottenere e salvare delle informazioni intermedie che potrebbero essere utili per ulteriori analisi o anche come salvataggi di sicurezza della computazione svolta fino a quel punto, nel caso in cui ci dovessero essere dei problemi nel completamento del training.

Nel nostro elaborato siamo andati a far uso di alcune callabacks standard, riportate anche nel listato 3.1, quali:

- *Early Stopping*: questa tecnica permette di ridurre la possibilità di portare il modello a overfitting, terminando il processo di apprendimento nel momento in cui la curva di train risulti appiattirsi o dopo un certo numero di epoche consecutive senza un incremento dell'apprendimento;
- *ReduceLROnPlateau*: tramite questa API siamo in grado di andare a diminuire il parametro di *learning rate* nel momento in cui il modello raggiunge un punto definito di *plateau*. Questa diminuzione (vedi figura 3.10) permette ottenere un'accelerazione del train verso il minimo, unitamente ad una evoluzione delle metriche utilizzate;

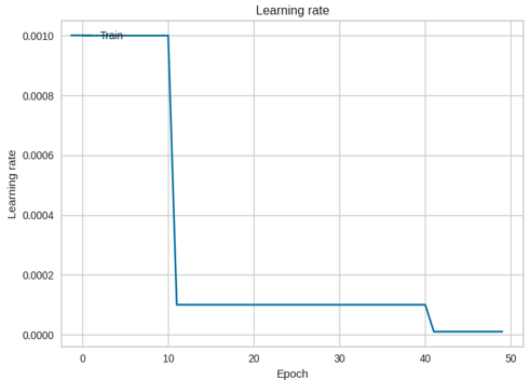


Figura 3.10: Riduzione del *learning rate* durante la fase di train

- *Model Checkpoint*: questa callback permette di salvare il modello in un checkpoint file (con formato hdf5 o h5) dopo ogni epoca, così da assicurarsi un backup in caso di problemi;
- *Tensorboard*: essa permette di salvare, dopo ogni batch, le metriche, i grafi e altre diverse informazioni in un formato molto utile e ben formattato per la visualizzazione, tramite una dashboard dedicata.

```
early = EarlyStopping(monitor='val_loss' if useCCE else 'val_dice_coef'
    , min_delta=1e-4, patience=patience, verbose=1, restore_best_weights
    =True)

reduce = ReduceLROnPlateau(monitor='val_loss' if useCCE else '
    val_dice_coef', factor=0.1, patience=5)

checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_accuracy' if
     useCCE else 'val_dice_coef', verbose=1, save_best_only=True,
    save_weights_only=True, mode='max')

logdir = os.path.join("definitive_logs", datetime.now().strftime("%H_%
    M_%d_%m_%Y"))
tensorboard_callback = TensorBoard(logdir, histogram_freq=1,
    write_graph=True, update_freq=1)
```

Listing 3.1: Callbacks utilizzate durante la fase di train

3.5.2 Optimizer

La definizione degli optimizer consiste nella scelta della tipologia di algoritmo che caratterizza la modalità con cui vengono aggiornati gli attributi della rete neurale durante la fase di train, con l'obbiettivo di andare a scovare i minimi della funzione di loss in maniera iterativa durante il training: in base a come avviene questa ricerca si possono definire diversi approcci.

Nel nostro caso abbiamo basato la scelta su quelle che risultano essere le indicazioni più consistenti e diffuse in letteratura.

Infatti, molti lavori di ricerca definiscono e utilizzano, come approccio standard, l'optimizer *Adam*: esso è considerato il più veloce e il più efficiente ad arrivare a convergenza, ed è per questo che l'abbiamo applicato in fase di compilazione dei modelli (come si vede nei listati 3.2 e 3.3).

3.5.3 Loss function

```
9  def compile_with_cross_entropy(modelMVM):
10   optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
       epsilon=1e-08)
11   modelMVM.compile(loss=keras.losses.categorical_crossentropy,
       optimizer=optimizer, metrics=['accuracy'])
```

Listing 3.2: Compilazione del modello convoluzionale tramite CE

```
12 def compile_with_dice_loss(modelMVM):
13   optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
       epsilon=1e-08)
14   modelMVM.compile(optimizer=optimizer, loss=dice_loss, metrics=[
       dice_coef])
```

Listing 3.3: Compilazione del modello convoluzionale tramite Dice

Nell'approcciare un problema tramite modelli convoluzionali e, più in generale nel contesto del Machine Learning, risulta essere estremamente importante definire quella che è la funzione di loss che si vuole andare a minimizzare durante la fase di train tramite l'algoritmo definito dalla funzione di optimizer (vedi sezione 3.5.2).

La funzione che cerchiamo di minimizzare/massimizzare è genericamente chiamata *objective function* e viene spesso definita appunto *loss function* oppure *error function*, qualora rappresenti un'entità che deve essere minimizzata.

Essa permette di confrontare i dati forniti in uscita dalla rete convoluzionale, ovvero la predizione realizzata dal modello, con quello che è l'output desiderato, ovvero le etichette (*ground truth*) definite a priori e che si vuole far apprendere al modello stesso.

La scelta della funzione di loss è un problema cruciale, essendo che è molto importante definire e calibrare tale funzione a seconda di quello che è il contesto del problema in cui il modello verrà introdotto: per questo motivo ci sono alcuni consigli e linee guida da seguire e qualche dettaglio da mettere in evidenza.

Softmax

In prima battuta è importante sottolineare come la scelta della funzione di loss sia direttamente legata a quella che è la funzione di attivazione utilizzata nel layer di output, la quale ricopre un ruolo fondamentale essendo che va a mappare la probabilità di ogni istante di appartenere ad una delle classi definite per il problema: è per questo che essa è solitamente differente dalla funzione di attivazione utilizzata per gli *hidden layer*.

Se ne deduce quindi come funzione di attivazione finale e loss function siano due elementi estremamente interconnessi: nella definizione del nostro problema, facente parte della sfera della *multi-class classification*, l'ultimo layer della rete convoluzionale utilizza come funzione di attivazione la *softmax*.

Questa funzione va a generalizzare la funzione logistica (detta anche *sigmoind function*) la quale prende in ingresso un numero reale x e restituisce in uscita un altro numero reale compreso tra 0 e 1.

La funzione softmax dal canto suo invece, prende come input un vettore x di valori reali e ritorna un vettore z, sempre composto da numeri reali, in cui ogni elemento appartiene all'intervallo $(0, 1]$ e in cui tutti gli elementi sommano a 1, dato che ogni elemento scalare di z rappresenta una probabilità.

Per completezza andiamo ad esplicitare la formula matematica che caratterizza quello che è il comportamento della funzione in questione: in particolare si può notare che viene applicata la funzione esponenziale, utilizzando come esponenti i j valori che compongono l'input del layer finale; il prodotto che si ottiene viene poi ad essere semplicemente normalizzato, andando a dividere per la somma di tutti questi termini, come riportato nella formula seguente.

$$Sofmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Quanto presentato finora risulta essere completamente celato ed implementato all'interno delle sofisticate librerie di backend di *Keras*: per questo motivo non dobbiamo occuparci di implementare direttamente questa funzione ma possiamo sfruttarle direttamente in fase di stup del problema (vedi codice 3.4).

```
15 model.add(Conv2D(filters=num_of_classes, kernel_size=(1, 2), activation
       ='softmax', name=...))
```

Listing 3.4: Setup della funzione di attivazione *softmax* nel layer di output

Quanto riportato nel listato 3.4 nasconde un ragionamento leggermente più complesso: in questo caso infatti la funzione softmax non viene applicata ad un vettore (ovvero al logits), ma bensì ad una matrice di features.

Trattandosi infatti di un problema di segmentazione applicato alle serie temporali, in uscita avremo una matrice in cui ogni riga corrisponde allo score assegnato dalla

rete ad ogni classe per quel determinato istante. Per precisione di notazione quindi, come riportato anche in [23], la funzione di attivazione softmax diventa:

$$p_k(\mathbf{x}) = \frac{e^{a_k(\mathbf{x})}}{\sum_k' 1^K e^{a_{k'}(\mathbf{x})}}$$

dove:

- $a_k(\mathbf{x})$ rappresenta la features map fornita del penultimo layer della rete convoluzionale;
- $\mathbf{x}$ indica l'insieme della feature map per un determinato istante: nel nostro caso si tratta di un vettore con 3 dimensioni;
- K indica il numero di classi (nel nostro caso 3);

Categorical Cross entropy

La prima tipologia di funzione di loss che siamo andati ad utilizzare è stata la classica funzione di loss utilizzata nei problemi di classificazione *multi-class*: la *categorical cross entropy*, la quale ha origini dal contesto della teoria dell'informazione.

Nello specifico questa funzione di loss opera andando a misurare il grado di similarità tra il vettore delle probabilità fornito dal layer di attivazione basato sulla Softmax e quello che è il vettore di ground truth: uno dei modi più semplici per calcolare quella che è la similarità tra due vettori, consiste nell'andare a realizzare il prodotto vettoriale tra i vettori stessi.

La funzione di cross entropy invece, come si vede in figura 3.11, va a combinare ogni singola probabilità con la relativa probabilità nel vettore "verità": da evidenziare come, per gli elementi appartenenti al vettore ground truth, essendo codificati in modalità *one-hot encoded*, non si applica l'operazione di logaritmo, essendo che andrebbe a causare accessi problematici per valori pari a 0, in cui il logaritmo tende a $-\infty$.

La *categorical cross entropy*, che dunque risulta essere la loss function utilizzata di default per i problemi di *multi-class classification*, permette di misurare facilmente quanto due distribuzione di probabilità discrete siano distanti una dall'altra tramite la seguente equazione:

$$Loss = -\sum_{classi} y_{true} log(y_{pred})$$

Questa sommatoria, che si sviluppa su tutte le classi disponibili, verrà calcolata per ogni singolo istante andando poi a produrne la media come valore finale in uscita: sarà poi questo errore medio che guiderà il train del modello.

Come già presentato per la softmax, anche in questo caso il framework di *Keras* offre una propria implementazione interna di questa tipologia di loss: per noi è

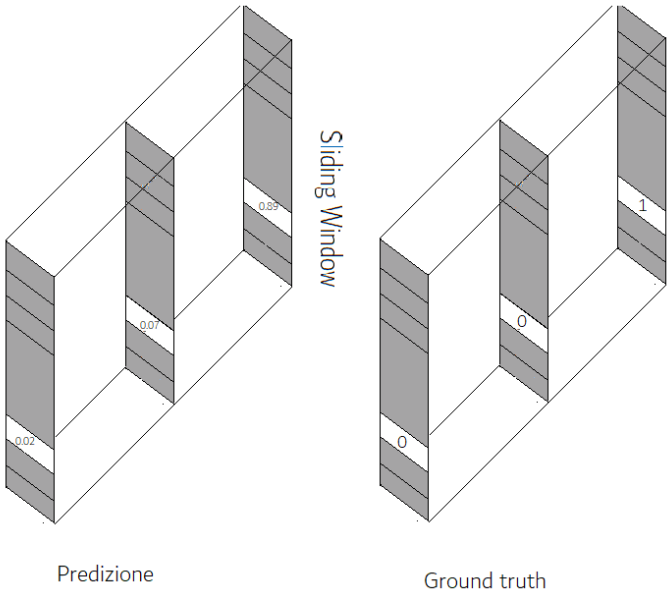


Figura 3.11: Categorical cross entropy

quindi sufficiente definire di voler utilizzare la loss *categorical cross entropy* in fase di compilazione del modello, come si può evincere dal listato 3.3.

Essendo però che la funzione di loss *categorical cross entropy* valuta singolarmente ogni predizione e poi va a realizzare una media lungo tutte le predizioni, si sta sostanzialmente andando a realizzare un livello di learning eguale per ogni istante: questo può essere un problema nel momento in cui si ha a che fare con uno sbilanciamento di classi, in cui è presente una classe prevalente rispetto alle altre, le quali risultano invece essere minoritarie.

Dice Loss

Per ottenere quindi una funzione di loss che non risulti essere fortemente condizionata dalla distribuzione delle classi, ci si può basare su quello che è il concetto espresso dal *Dice Coefficient*, il quale non è altro che la misura dell'overlap tra due samples (come espresso nella sezione 3.5.4).

Infatti il dato trattato in questo contesto risulta essere, per sua natura, fortemente sbilanciato: va da sè che, nello studio del design della CNN, sia necessario prendere in considerazione il problema del *class imbalance* (vedi sezione 4.5).

Come riportato sempre nella sezione 3.5.4, ottenere un *Dice Coefficient* pari a 1 sta a significare che si ha una completa sovrapposizione tra predizione e valori reali: per massimizzare quindi l'overlap tra i due campioni, ed essendo che ci si prefigge l'obbiettivo di ottener una funzione di loss da minimizzare, definiamo un parametro di questo tipo:

$$\text{Dice Loss} = 1 - Dice$$

```
16 def dice_loss(y_true, y_predict):
17   return 1 - dice_coef(y_true, y_predict)
```

Listing 3.5: Implementazione di Dice Loss

3.5.4 Metriche

Nel train di modelli convoluzionali le metriche sono utilizzate solamente per "giudicare" le performance del modello, senza influenzare direttamente quelle che sono le scelte relative al tuning dei pesi della rete stessa lungo il training.

Nel nostro approccio abbiamo utilizzato due metriche differenti, essendo che, come riportato nella sezione 3.5.3, siamo andati ad utilizzare due differenti funzioni di errore per le quali sono necessarie metriche adeguate, ovvero:

- *Accuracy* per la loss CCE: essa rappresenta la percentuale di istanti che sono classificati correttamente;
- *Dice coefficient* per la Dice loss: questo coefficiente permette di definire la "similarità" tra due campioni.

  In particolare il *Dice Coefficient* è molto simile alla funzione di errore *Intersection Over Union* (IoU): esso ha infatti valori compresi nel range tra 0 e 1, dove il valore 1 denota un perfetto overlap tra i due samples, mentre il valore 0 indica una completa assenza di sovrapposizione tra i due insieme.

  Questo coefficiente può essere calcolato tramite l'equazione riportata in figura 3.12: si può notare come al numeratore sia presente il quantitativo di elementi in comune tra i due set, mentre al denominatore viene riportato il numero totale di elementi che compongono i due insieme A e B. Così facendo, ovvero dividendo per il totale del numero degli elementi che compongono i due insiemi, il coefficiente risulta essere normalizzato, evitando problematiche legate allo sbilanciamento delle classi.

  Possiamo quindi applicare questo coefficiente al nostro contesto, come schematizzato in figura 3.13, tramite la formula matematica seguente:

$$\frac{2\sum_{istanti} y_{true} y_{pred}}{\sum_{istanti} y_{true}^2 + \sum_{istanti} y_{pred}^2}$$

  Si può notare che il numeratore incrementa solamente nel momento in cui valore predetto e valore reale coincidono (entrambi pari a 1), proprio per il fatto che i valori reali sono codificati in *one-hot encoded* e quindi, quando la probabilità reale è 0, si avrà un prodotto che genererà un risultato nullo.

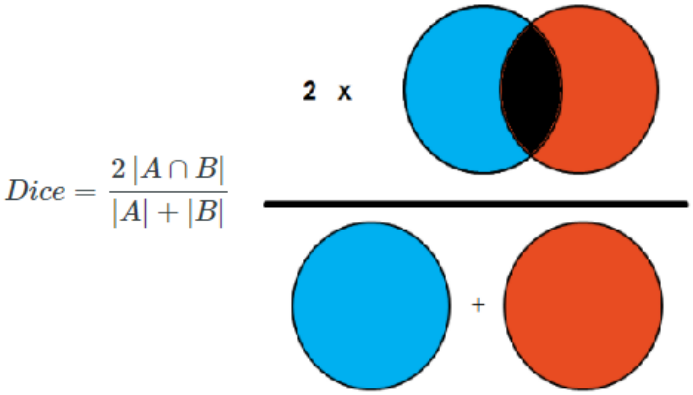


Figura 3.12: Calcolo e rappresentazione di Dice Coefficient

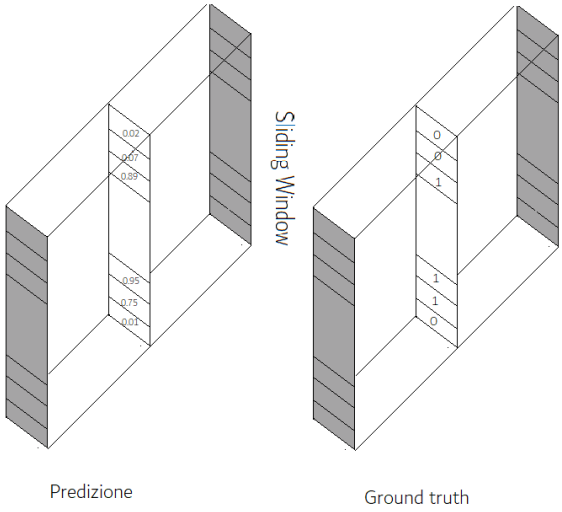


Figura 3.13: Esempio di approccio tramite Dice loss

Come sappiamo *Keras* rappresenta una libreria che offre un alto livello di astrazione nella compilazione e composizione di modelli utili per il ML: in particolare, per quanto concerne tutta la parte di modellizzazione e calcolo matematico, si affida ad alcuni blocchi di backend quali *Tensorflow* e *Theano*. Nel listato 3.6 andiamo ad importare tali backend per effettuare delle operazioni su tensori: nello specifico sono stati utilizzati:

- Flatten: rappresenta la riduzione delle dimensioni dei vettori di predizione e di ground truth.

  I vettori in input, per il modello relativo al contesto della semantic segmentation, avranno dimensionalità pari a $(16, 1, 3)$ mentre, successivamente alla fase di flattening, si otterrà una dimensionalità di $(48, 1)$;

  Si può capire quindi come, dopo l'operazione di flattening, il confronto tra la predizione e i valori reali avvenga tra ogni singolo valore, fornendo quindi una

metrica in grado di avere uno sguardo d'insieme su tutto il blocco;

- Sum: va a fornire la somma di tutti gli elementi del vettore passato come parametro.

```
from keras import backend as K

def dice_coef(y_true, y_pred, smooth=1):
  y_true_f = K.flatten(y_true)
  y_pred_f = K.flatten(y_pred)
  intersection = K.sum(y_true_f * y_pred_f)
  return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(
    y_pred_f) + smooth)
```

Listing 3.6: Implementazione custom di Dice Coefficient

4

Operazioni preliminari

4.1 Selezione del dataset

Per passare da un approccio analitico, come quello attualmente utilizzato su MVM per rilevare il trigger di inspirazione, ad un approccio basato su *IA* è necessario avere a disposizione un dataset completo di forme d'onda con cui andare ad allenare la rete neurale: questa parte di ricerca e composizione è stata una tra le più corpose dell'intero elaborato.

Nella comunità scientifica infatti risultano essere diffusi alcuni studi relativi all'analisi e alla classificazione delle forme d'onda relative alla respirazione: nella maggior parte dei casi però tali studi vanno ad utilizzare datasets non pubblici, per motivi legati a standard di privacy a cui ci si deve attenere nel momento in cui si trattano dati sensibili del paziente, come lo sono questi relativi alla respirazione assistita.

Essendo il nostro approccio è esplorativo e senza ausilio di strutture cliniche esterne su cui far riferimento, ci siamo dovuti affidare a datasets già esistenti e liberamente fruibili.

Un primo approccio tentato è stato quello di analizzare uno dei più grandi dataset medici disponibili online: MIMIC [13].

Esso rappresenta una base di dati pubblica e liberamente fruibile, popolata e mantenuta dal *MIT Lab for Computational Physiology*: al suo interno sono presenti dati anonimizzati relativi a informazioni demografiche, parametri vitali, terapie mediche etc, di circa 60000 pazienti ammessi in ICU.

Dopo aver ottenuto l'accesso ad una porzione di questo dataset, sottoposta ad accesso ristretto con limitazioni sull'utilizzo e sulla divulgazione dei dati contenuti, siamo andati a verificare la presenza o meno di informazioni legate alle curve di flusso e di pressione dei pazienti sottoposti ad ICU: questa indagine però non ha prodotto i risultati sperati, essendo che il dataset in questione risulta essere privo delle informazioni che stavamo cercando.

La scelta, più o meno forzata da quelle che erano le nostre necessità e le disponibilità in letteratura, è ricaduta perciò su altri due dataset [22] e [27].

4.2 Datasets overview

Nel nostro approccio siamo andati a considerare i datasets utilizzandoli secondo una scala temporale relativa: non abbiamo infatti tenuto conto dell'informazione assoluta del tempo, rappresentata dall'orario di inizio e di fine respiro, ma abbiamo considerato i samples come una semplice sequenza continua di campioni.

A maggior ragione, questa informazione temporale non è stata considerata per il fatto che, le forme d'onda di flusso e pressione componenti i datasets in questione sono state campionate ad una frequenza costante di 50 Hz, rendendo quindi ininfluente per i nostri obbiettivi la presenza dell'informazione del tempo assoluto.

Entrando nel dettaglio del dateset [27], esso è stato collezionato da 103 soggetti sottoposti a terapia intensiva all'interno della University of California Davis Medical Center (UCDMC) secondo l'internal review board (IRB) 647002, tramite il ventilatore Purittan Bennet 840 (PB-840): essi consistono nella forme d'onda della respirazione meccanica (vedi sezione 1.4) relative a flusso e pressione, campionate ad una frequenza di 50 Hz appunto.

In particolare questo dataset contiene anche le etichette relative alla tipologia di ventilazione per ogni singolo respiro, intenso come un insieme di samples da un'istante *BS* (Breath Start) ad un'istante *BE* (Breath End): questo lavoro di *labelling* è stato svolto manualmente da tre esperti medici, senza un confronto diretto in fase di etichettatura, così da ottenere un processo di *labelling* il più veritiero possibile e sollevando dubbi solamente nel momento in cui ci fossero state discrepanze sulla tipologia di classe scelta da uno dei tre componenti del team.

Nello specifico il gruppo di lavoro è andato a verificare, per ogni singolo ciclo di respirazione, la presenza di una delle seguenti cinque modalità di ventilazione:

- Volume Control (VC);
- Pressure Control (PC);
- Pressure Support (PS);
- Continuos Positive Airway Pressure (CPAP);
- Proportional Assist Ventilation (PVA).

Questa informazione non risulta essere direttamente utilizzabile nel nostro approccio, essendo che non fornisce dati relativi all'inizio dell'inspirazione e della espirazione, ma ci fornisce un'informazione aggiuntiva sul dataset.

Dall'altro lato, il dataset [22] che ci è stato fornito su gentile concessione dal gruppo di lavoro facente riferimento al professor *Jason Yates Adams*, il quale opera sempre presso l'University of California: il suo gruppo infatti è l'autore anche del progetto di ricerca che ha utilizzato come base il dataset precedentemente presentato [27].

Questa seconda base di dati invece, nello studio di ricerca orginale, è stata utilizzata per lo studio e la classificazione delle tipologie di PVA: i dati etichettati mettono quindi a disposizione una serie di features (temporali, volumetriche, di pressione etc) che definiscono le caratteristiche della forma d'onda unitamente anche alla classificazione di 4 differenti PVA e di 4 *clinical artifacts*, ovvero situazioni che si possono manifestare durante la respirazione meccanica e che dipendono dalla tecnica utilizzata e non dalla naturale decorrenza dell'azione respiratoria.

Per essere comprese appieno, le classificazioni presentate nel lavoro di ricerca originale richiedono alcune conoscenze prettamente di stampo medico; in questo elaborato, essendo che tali etichette non vengono ad essere direttamente utilizzate per semplicità, bypassiamo questa analisi e ci limitiamo a presentare le classi prese in considerazione, le quali sono:

- Le seguenti PVA
  - Flow Asynchrony (FA);
  - Breath Stacking Asynchrony (BSA);
  - Double Trigger (DBL);
  - Asynchrony Not Otherwise Specified (aNos);
- I seguenti *clinical artifacts*:
  - Cough (CO);
  - Suction (SU);
  - Multi Trigger (MT);
  - Waveform Not Otherwise Specified (wNos);

4.3 Composizione e utilizzo dei datasets

La mole totale di dati componenti i datasets [22] e [27] è stata utilizzata in due modi differenti:

- in un primo approccio siamo andati ad unire i due datasets in unico set di dati, trascurando le informazioni relative alle classi di appartenenza di ogni singolo respiro e considerando solamente i valori grezzi relativi al flusso respiratorio [l/min] ed alla pressione rilevata nelle vie aeree [$cmH2O$].

Esso conterrà quindi una grande diversità di tipologie di respirazioni e di funzionamenti, permettendo così ai modelli di apprendere un maggior numero di casistiche.

Di seguito riportiamo alcune informazioni relative alla numerosità dei campioni appartenenti al dataset totale composto in questo primo approccio:

  - 176 files in formato .csv;
  - 333173 respiri;
  - 42456667 campioni di flusso e di pressione;
  - circa 10 giorni totali di dati relativi alla respirazione, infatti: $42456667 * \frac{1}{50}\frac{s}{sample} \simeq 849133s \simeq 235h \simeq 10gg$

- in un secondo approccio, essendo che il ventilatore MVM lavora in modalità *Pressure Based* (vedi sezione 1.3), abbiamo provato ad isolare dal dataset [22] solamente quei respiri catalogati dal team dell'*University of California Davis Medical Center* come in modalità *Pressure Control* (PC) o *Pressure Support* (PS).

  In sostanza, tra tutti i samples a disposizione, è stato fatto un controllo incrociato con i *gold_std* files (ovvero i files contenenti le etichette) per capire così quali e quanti respiri fossero stati catalogati nella modalità di nostro interesse: abbiamo quindi filtrato il dataset completo, andando poi a eseguire lo stesso processo di *pre-processing* e di formattazione dei dati applicato nel primo caso.

  Per completezza riportiamo alcune informazioni numeriche anche per questo "sotto" dataset ricavato da quello iniziale, evidenziando come la numerosità sia minore rispetto al precedente, ma comunque abbastanza consistente:

  - 151078 respiri;
  - 21532777 campioni di flusso e di pressione.

I dati e le relative numerosità presentate qui sopra, sono riferiti ai semplici dati grezzi: è da tener presente quindi che, dopo l'applicazione dell'algoritmo di sliding window (sezione 4.4.3), si andrà ad ottenere un incremento della numerosità degli istanti totali per la presenza di sovrapposizione (overlap) tra campioni sequenziali.

Nella figura 4.1 è riportato uno *scatter plot* della distribuzione dei campioni, rappresentati sul piano flusso-pressione: in particolare i punti di colore *rosso* rappresentano istanti classificati come *inizio inspirazione*, quelli *verdi* consistono negli istanti di *inizio espirazione* mentre le entità blu rappresentano istanti di *background*.

Da notare come si siano delineate quattro macro zone: iniziamo con il mettere in evidenza come, per i punti di inizio inspirazione (rossi), il flusso sia sempre positivo proprio per il fatto che l'aria entra sempre nello stesso verso nel momento in cui si da il via ad un respiro.

L'inizio dell'espirazione invece, per i campioni presi in considerazione, sembra iniziare sempre ad una pressione positiva: da notare come la concentrazione sia forte a cavallo del punto in cui il flusso cambia segno, proprio per il fatto che l'indice *x0_index*

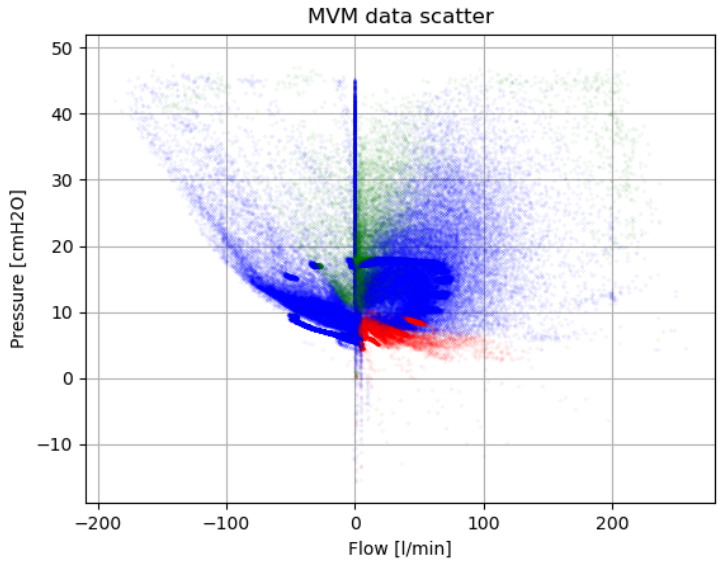


Figura 4.1: Scatter plot di una parte dei dati con l'informazione della relativa classe di appartenenza

(figura 4.3) corrisponde, nella maggior parte delle situazioni, all'istante in cui il flusso attraversa l'asse delle ascisse, cambiando quindi segno.

4.4 Pre processing

Dopo aver individuato su quali dati basare il train dei modelli, si è proceduto con il pre processing dei dati, il quale è consistito prevalentemente in queste due fasi:

- Etichettatura degli istanti;
- Ricomposizione del dataset e salvataggio in strutture dati facilmente fruibili dal modello durante la fase di train.

4.4.1 Labelling

L'operazione di etichettatura dei dati è di fondamentale importanza per il fatto che, i modelli che poi verranno ad essere allenati, andranno a definire i propri parametri interni in modo tale da andare a rendere il minore possibile l'errore di predizione che viene valutato confrontando la predizione in uscita del modello e il valore reale definito tramite il processo di labelling.

Nello specifico è stato etichettato ogni singolo istante temporale, andando a ricreare delle serie temporali di pari lunghezza di quelle in ingresso: questo approccio è tipico del contesto della *semantic segmentation*, in cui ad ogni pixel (delle immagini) viene fatto corrispondere una determinata classe.

Nel nostro caso, come già riportato, avremo a disposizione tre classi differenti:

- Classe "0": (*inizio inspirazione*);
- Classe "1": (*inizio espirazione*);
- Classe "2": nessuno degli altri (*background*).

Nello specifico la classificazione di questi istanti è stata tradotta in una sequenza di *0* e *1* secondo l'approccio *one-hot encoded*: questa modalità di definizione delle classi di appartenenza è tipico nell'ambito della classificazione e permette di andare a lavorare con CNN che presentano in output valori espressi in termini probabilistici.

Per presentare come è stata realizzata la procedura di labelling tramite il codice 4.1 e che ha prodotto i risultati mostrati in tabella 4.1, è necessario esplicitare il ragionamento seguito per assegnare una certa classe piuttosto che un'altra, introducendo anche gli strumenti utilizzati.

Dati (X)		**Label (Y)**		
Flusso	**Pressione**	**Inizio inspirazione**	**Inizio espirazione**	**Background**
1.5	9.71	1	0	0
6.31	9.55	1	0	0
25.44	9.54	1	0	0
52.98	9.9	1	0	0
73.48	11.16	1	0	0
87.64	13.45	0	0	1
95.24	16.34	0	0	1

Tabella 4.1: Esempio di classificazione *one hot encoded* applicata ad ogni singolo istante

La natura degli istanti di inizio inspirazione ed espirazione è strettamente legata all'andamento delle forme d'onda di flusso, pressione e (qualora fosse integrata) volume: il riconoscimento di questi punti temporali riguarda però un ambito molto specializzato, in cui è necessario un occhio estremamente esperto e con molta esperienza nel settore per riconoscere velocemente questi particolari istanti ispezionando visivamente l'andamento delle curve.

Per facilitare questa operazione, essendo appunto che non abbiamo avuto a supporto un team di medici specializzati per dedurre visivamente questi istanti direttamente dalle forme d'onda, siamo andati ad utilizzare la libreria open source *ventMAP* [11] [22].

La libreria sopra citata è stata ideata e progettata per andare a lavorare con le informazioni relative alla respirazione composti da una sequenza di dati grezzi compresi tra due istanti di BS e BE, con l'aggiunta di un'informazione di timestamp all'inizio di ogni singolo ciclo di respiro e la cui formattazione generale è riportata in figura 4.2.

```
<breath1 start datetime stamp>
BS, S:<vent BN 1>
<breath info>
BE
<breath2 start datetime stamp>
BS, S:<vent BN 2>
```

(a) *Formattazione delle forme d'onda di input*

Flusso	Pressione
BS	
0.80	3.41
1.50	3.23
7.07	2.86
...	...
-1.99	5.90
-1.97	5.90
-2.08	5.98
BE	
BS	
0.81	3.20
1.32	3.08
6.06	2.94

(b) *Esempio di formattazione, secondo lo schema 4.2a, dei dati grezzi relativi alle curve di flusso e pressione*

Figura 4.2: Formato dei dati ed esemplificazione con dati grezzi

La libreria *ventMAP* permette di estrarre numerose informazioni relative alle curve d'onda che gli vengono passate in ingresso.

Tra queste, due sono risultate fondamentali per l'individuazione degli istanti di inizio inspirazione ed espirazione:

- **l'inizio di ogni respiro**: essendo che il dataset [22] contiene già un'informazione temporale su inizio e fine del ciclo di respirazione (vedi figura 4.2a e tabella 4.2b), l'inizio dell'inspirazione è stata fatta coincidere con l'inizio della respirazione, ovvero con l'istante BS;

- **l'inizio dell'espirazione**: per l'espirazione il discorso è leggermente più complesso. Nel funzionamento della respirazione (vedi sezione 1.4), l'inizio dell'atto espiratorio può essere fatto coincidere con istanti aventi caratteristiche diverse.

  Nel nostro approccio abbiamo utilizzato l'indice fornito direttamente dalla libreria *ventMAP*, denominato *x0_index* (vedi figura 4.3), il quale rappresenta l'istante in cui la curva di flusso cambia segno ovvero quando il flusso respiratorio inverte la propria direzione (che corrisponde, in termini grafici, ad un attraversamento dell'asse delle ascisse). Tuttavia, ci sono anche cicli di respirazione dove la curva di flusso non attraversa l'asse delle x: per questo motivo, sempre all'interno della libreria *ventMAP*, sono state incorporate delle *euristiche*, definite da esperti del settore, che permettono di migliorare il processo di definizione della fine dell'inspirazione e del conseguente inizio della fase di espirazione.

  Sicuramente, un approccio più completo potrebbe essere quello di andare a considerare l'intera curva di flusso relativa ad un atto respiratorio determinandone

il massimo, per poi così andare a determinare l'inizio dell'atto espiratorio nel punto in cui si ha un discostamento dal massimo, in termini percentuali, di un valore arbitrario.

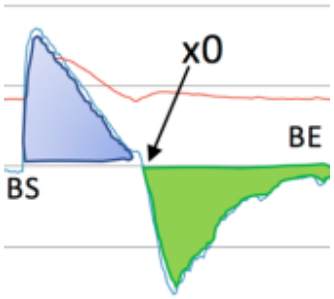


Figura 4.3: Istante di passaggio da fase di inspirazione a fase di espirazione

Per tener conto anche di eventuali errori nella classificazione fornita dalla libreria *ventMAP* e per garantire al modello convoluzionale una maggiore possibilità di apprendimento e una maggiore robustezza si è deciso, su suggerimento anche di [18], di andare a considerare non solamente un timestamp unico corrispondente agli istanti di inizio inspirazione ed espirazione, ma bensì una finestra di 5 istanti.

In particolare sono stati presi i 5 istanti successivi a quello definito come inizio inspirazione e i 5 istanti precedenti a quello definito invece come inizio espirazione.

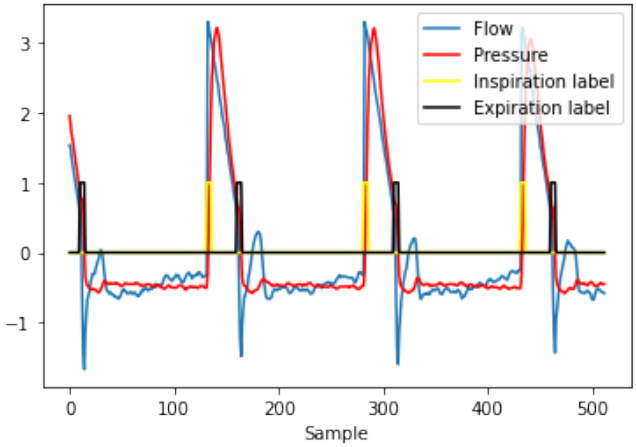


Figura 4.4: Esempio di labelling riportato sulle curve standardizzate di flusso e pressione

```
start_inspiration = np.zeros(len(flow))
start_expiration = np.zeros(len(flow))
background = np.ones(len(flow))

# Inspiration start
start_inspiration[0] = 1
start_inspiration[1] = 1
start_inspiration[2] = 1
start_inspiration[3] = 1
start_inspiration[4] = 1
# Expiration start
start_expiration[x0_index - 4] = 1
start_expiration[x0_index - 3] = 1
start_expiration[x0_index - 2] = 1
start_expiration[x0_index - 1] = 1
start_expiration[x0_index] = 1
# Background
background[0] = 0
background[1] = 0
background[2] = 0
background[3] = 0
background[4] = 0
background[x0_index - 4] = 0
background[x0_index - 3] = 0
background[x0_index - 2] = 0
background[x0_index - 1] = 0
background[x0_index] = 0
```

Listing 4.1: Assegnazione classe di appartenenza ai relativi istanti

Per ottenere un'etichettatura di tipo *one-hot encoded* abbiamo utilizzato il codice 4.1, in cui si può vedere come inizialmente vengano creati 3 vettori composti da una sequenza di *0* (grazie al comando *np.zeros*) o di *1* (tramite il comando *np.ones*) per ognuno delle 3 classi che andiamo a gestire.

Successivamente viene settata a *1* la probabilità di inizio inspirazione ad inizio del vettore della classe relativa e per i 5 istanti successivi, mentre per l'inizio espirazione consideriamo l'indice *x0_index* e i 5 istanti che lo precedono. Il vettore di background invece è stato inizializzato con tutti i valori pari a 1, tranne poi andare a convertire a 0 i valori negli indici in cui si ha inspirazione ed espirazione, in maniera da ottenere classi mutuamente esclusive tra di loro.

Dal processo di labelling abbiamo quindi ottenuto un dataset con le seguenti numerosità per quanto riguarda le entità relative alla classificazione *ground_truth* di ogni istante:

- Numero totale di inizio inspirazione: 333173;
- Numero totale di inizio espirazione: 333173;

- Numero totale di istanti classificati come "classe 0": $333173 * 5 = 1665865$;
- Numero totale di istanti classificati come "classe 1": $333173 * 5 = 1665865$;
- Numero totale di istanti classificati come "classe 2": 42456667.

4.4.2 Etichette sovrapposte

Nel processo di labelling dei dati, essendo che non solamente un'istante viene ad essere catalogato come inizio inspirazione o espirazione ma bensì 5 istanti contigui ricevono la stessa classificazione (vedi sezione 4.4.1 e listato 4.1), si possono creare delle situazioni di inconsistenza: esse possono essere dovute per esempio a situazioni in cui il respiro risulti affetto da alcuni eventi, come tosse per esempio, che fanno si che istanti di inizio espirazione vengano identificati in prossimità dell'inizio dell atto inspiratorio.

Nello specifico si potrebbe infatti erroneamente ottenere una sovrapposizione delle classi assegnate in fase di labelling, essendo che gli istanti di inizio inspirazione sono assegnati andando verso destra nel tempo mentre quelli di inizio espirazione sono assegnati andando verso sinistra nella scala temporale: questo potrebbe quindi portare il modello ad apprendere la possibilità di avere due eventi nello stesso istante, riducendone la robustezza.

Per questo motivo tutti quei respiri (circa 1700 in totale) che hanno presentato, durante l'operazione di etichettatura, una sovrapposizione degli istanti (come in figura 4.5), non sono stati presi in considerazione ed eliminati dal dataset finale.

Per fare ciò, come si vede nel listato 4.2 siamo andati a verificare che l'indice *x0_index* (vedi figura 4.3) ottenuto dalla libreria *ventMAP* e i 5 istanti che vanno a precedere questo punto, catalogati tutti come inizio espirazione, non risultassero essere sovrapposti con gli istanti iniziali, quelli a cui invece è assegnata la classe di inizio inspirazione.

```
52 if x0_index - 4 == 0 or x0_index - 4 == 1 or x0_index - 4 == 2 or
       x0_index - 4 == 3 or x0_index - 4 == 4:
53   overlap += 1
54   print("Overlap number " + str(overlap) + " in file " + file + "
       during breath number " + str(BN))
55   continue
```

Listing 4.2: Controllo di non sovrapposizione durante la fase di *labelling*

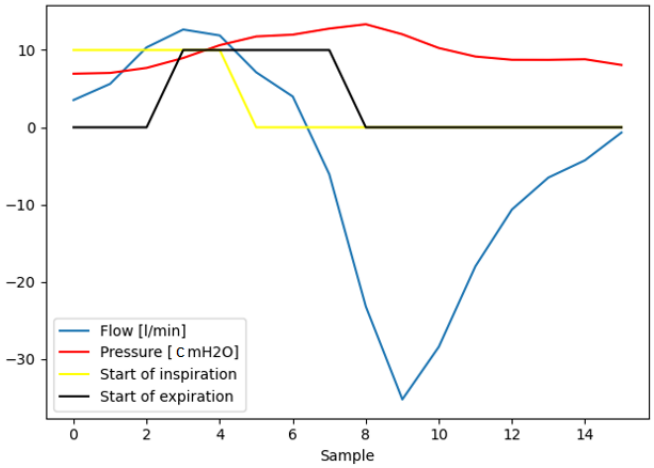


Figura 4.5: Esempio di respiro con overlap tra label di inizio inspirazione ed espirazione

4.4.3 Composizione in blocchi tramite sliding window

Una volta definito il labelling per le informazioni relative a flusso e pressione, siamo andati a formattare le curve e le etichette in un formato che poi sarebbe stato di facile utilizzo nella successiva fase di train.

Il risultato ottenuto è sostanzialmente un volume di dati la cui struttura qualitativa è riportata in figura 4.6, dove ogni strato rappresenta un singolo blocco che la CNN poi sarà in grado di ricevere in ingresso durante la fase di train.

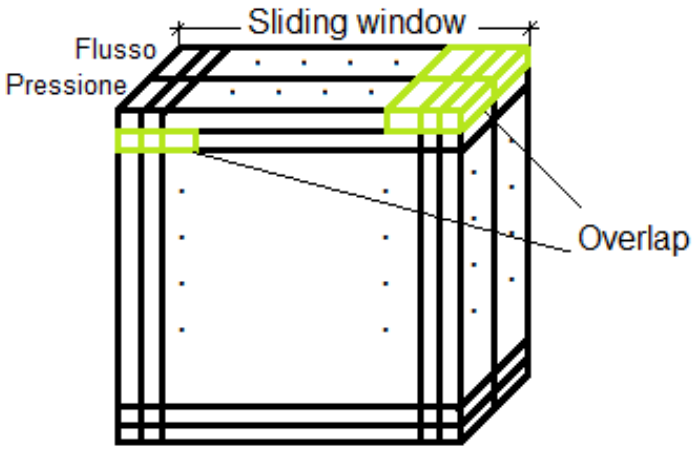


Figura 4.6: Rappresentazione qualitativa della dimensionalità dei dati salvati nei file con estensione .npy

Sliding window

Fornita la sequenza di dati grezzi in ingresso, l'applicazione dell'algoritmo di sliding window (vedi algoritmo 1) ha consentito di andare a raggruppare tali dati in una sequenza di vettori andando così ad eseguire un re-framing della dimensionalità iniziale e rendendo così adatto il dataset ad essere fornito in ingresso ai modelli convoluzionali.

L'algoritmo 1 permette inoltre di introdurre anche un concetto di sovrapposizione (*overlap*) dei campioni: essa consente infatti di aumentare la numerosità dei dati utilizzati per allenare i modelli, essendo che una parte dei dati risulta essere duplicata in due blocchi sequenziali.

Questo significa che, per due sample consecutivi, gli ultimi n istanti del primo blocco e i primi n istanti del secondo blocco saranno gli stessi.

Algorithm 1: iterazioni per l'applicazione della sliding window

Data: Parametri dell'algoritmo:

- **T** = serie temporale originale;
- **L** = lunghezza della finestra;
- **s** = step di scorrimento verso destra della finestra.

```
i = 0;
n = 0; // numero totale finale di finestre
F = [ ]; // risultato finale della frammentazione in windows
while i + L <= T.count do
    F[n]=T[i...(i+L-1)];
    i = i + s;
    n = n + 1;
end
return F;
```

Nello specifico abbiamo utilizzato due overlap differenti, per i due approcci presentati rispettivamente nelle sezioni 5 e 6:

- Overlap pari a 8: in questo caso la sovrapposizione tra due campioni consecutivi è del 50%. Così facendo, essendo che l'output del modello U-Net segmenta e classifica tutti gli istanti passati in ingresso, abbiamo rielaborato le osservazioni in ingresso con una sliding window pari a 16 ed un overlap di 8 istanti;
- Overlap pari a 15: in questo approccio invece, c'è un overlap maggiore tra campioni consecutivi utilizzati per allenare i modelli. Nello specifico è stata fatta questa scelta, per conformità con il design del modello *single output*, in cui la predizione in uscita corrisponde alle probabilità di appartenenza, ad una delle tre classi definite, dell'ultimo istante campionato.

  Applicando quindi un overlap così marcato è come se il modello imparasse a riconoscere istanti temporalmente successivi, presi singolarmente uno per volta.

  In questo approccio si è quindi usata una sliding window pari a 16 ma con overlap di 15: in pratica, ad ogni step successivo, la rete vede 15 istanti già visti ed uno nuovo, il quale sarà proprio l'istante oggetto della classificazione in uscita dalla rete stessa.

Se ne può dedurre come, la numerosità dei campioni prodotti con questo overlap estremamente aggressivo, risulti essere decisamente più elevata rispetto al primo approccio, a parità di quantità di dati grezzi forniti in ingresso.

Per salvare i dati così formattati abbiamo utilizzato il formato *.npy*, il quale permette uno store dei dati compatto e facile da spostare garantendo anche il mantenimento della formattazione e della dimensionalità dei dati stessi (vedi codice 4.3).

```
56 if not os.path.exists(path_result):
57   with open(path_result, 'wb') as f:
58     np.save(f, X_reshaped)
59     np.save(f, Y_reshaped)
60 else:
61   with open(path_result, 'ab') as f:
62     np.save(f, X_reshaped)
63     np.save(f, Y_reshaped)
```

Listing 4.3: Salvataggio dei dati nel formato *.npy*

4.5 Class Imbalance

Nel contesto della ventilazione meccanica i dati vengono collezionati con frequenza costante e in maniera continua: per esempio, con un campionamento che avviene ad una frequenza di 50 *Hz* avremo che in un secondo vengo ad essere raccolti 50 samples. Da questo semplice ragionamento possiamo dedurre che, per acquisizioni giornaliere, in cui quindi si campiona con continuità l'andamento della respirazione del paziente, si avranno a disposizione un grande quantitativo di dati.

Come riporta [25], un uomo adulto compie in media tra i 12 e i 18 respiri al minuto (RR = *Respiratory Rate* = 12-15): ponendoci nel caso peggiore per il contesto della *class imbalance* si avrà che, nella durata media di un respiro di 5 secondi (RR = 12), gli istanti che si cercano, ovvero quelli di inizio inspirazione ed inizio espirazione, risultano essere presenti con numerosità decisamente minore rispetto ai restanti istanti.

Essendo che i dati utilizzati sono stati campionati appunto con una frequenza di 50 *Hz*, avremo un numero di campioni in 5 secondi pari a circa 250: di questi 250 samples, 10 saranno considerati, in fase di labelling dei dati, come istanti di inizio inspirazione ed espirazione, i restanti 240 invece rappresenteranno istanti catalogati come background.

Come si può vedere in figura 4.7, la somma delle percentuali di ogni singola classe raggiunge quota a 100: questo conferma la bontà del processo di *labelling*, in cui ad ogni istante temporale è stato assegnato una e una sola classe tra le tre disponibili.

Questa analisi è stata effettuata sul data set originale senza aver applicato l'algoritmo di *sliding window*, il quale, per ovvie ragioni, va ad aumentare la numerosità

del dataset modificandone anche la numerosità delle etichette ma mantenendo simili le proporzioni tra le classi.

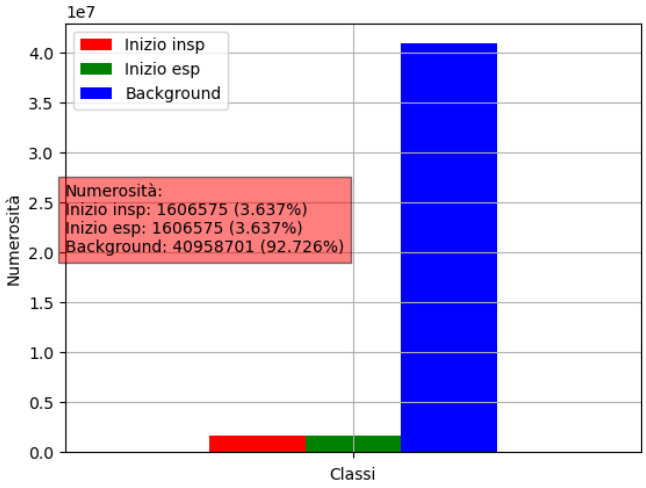


Figura 4.7: Numerosità dei samples per ogni categoria con cui sono stati etichettati ogni singolo istante utilizzato in fase di train e di test della rete

4.6 Normalizzazione dei dati

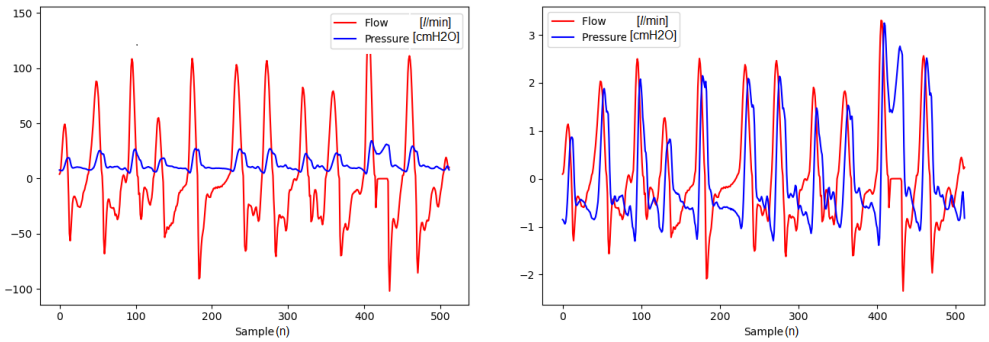
Le reti neurali convoluzionali imparano come deve essere mappato l'input in un output attraverso dei campioni presenti in un training set e un'inizializzazione dei pesi del modello con piccoli valori: questa inizializzazione, utile per prevenire situazione di stallo durante il train, richiede uno *scaling* dell'input essendo che grossi valori potrebbero portare alla divergenza del gradiente con il quale vengono aggiornati i pesi della rete e quindi ad un train errato del modello.

```
64 flow_mean = int(np.mean(X[:, 0]) * 100) / 100
65 pressure_mean = int(np.mean(X[:, 1]) * 100) / 100
66 flow_std = int(np.std(X[:, 0]) * 100) / 100
67 pressure_std = int(np.std(X[:, 1]) * 100) / 100
68 temp_block_X[:, 0] = (temp_block_X[:, 0] - flow_mean) / flow_std
69 temp_block_X[:, 1] = (temp_block_X[:, 1] - pressure_mean) /
       pressure_std
```

Listing 4.4: Normalizzazione delle curve di flusso e di pressione

L'operazione di normalizzazione dei dati è sempre buona norma che venga applicata al dataset fornito in input al modello e consiste nel rescaling della distribuzione dei valori in maniera tale da rendere la media dei dati pari a 0 e la deviazione standard pari a 1, tramite la seguente equazione:

$$x' = \frac{x - x_{mean}}{\sigma}$$



(a) *Curve di flusso e pressione non normalizzate*

(b) *Curve di flusso e pressione normalizzate*

Figura 4.8: Esempio di normalizzazione

4.7 Split dei dati

```
70 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size
       =0.2, random_state=123)
```

Listing 4.5: Divisione del dataset in train set e test set

Nel percorso di approccio e soluzione del problema nel contesto dei modelli convoluzionali, dopo aver processato i dati è importante andare a partizionarli.

Per fornire al modello un buon grado di generalizzazione, si è soliti quindi andare a riordinare in maniera random e dividere i dati in *train set* e *test set* utilizzando le funzionalità messe a disposizione dal framework *Keras* tramite la funzione *train_test_set_split*: in particolare è stato utilizzato il classico approccio che prevede una partizione 80-20, ovvero l'80% dei dati viene introdotto nel set di train e il restante 20% viene fatto confluire nel set di test, utilizzato per verificare la bontà del modello allenato.

La fetta relativa ai dati di training, viene ad essere ulteriormente spezzata da *Keras* il quale, in fase di training del modello e tramite il parametro *validation_split* (vedi codice 4.6), permette di impostare la percentuale di dati del dataset di train passato in input che deve essere utilizzata nella fase di validation.

```
71 history = modelMVM.fit(X_train, Y_train, epochs=epochs, verbose=True,
       validation_split=0.2, batch_size=batch_size,
72 callbacks=[reduce, early, tensorboard_callback, checkpoint])
```

Listing 4.6: Train del modello utilizzando uno split automatico per il validation set

5

MVM U-Net: Encoder - Decoder

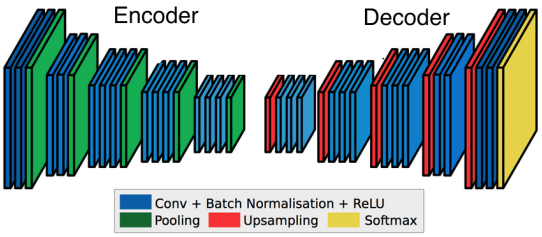


Figura 5.1: Design generale di modelli CNN con sezioni di Encoder e Decoder [10]

Il modello proposto in questa sezione rappresenta una CNN *fully connected* con approccio definito *encoder-decoder*, ispirata alla popolare architettura U-Net proposta originariamente per la segmentazione delle immagini nello studio di ricerca [23].

Questo modello basa le sue funzionalità su un design definito spesso ad "U" (per la rappresentazione molto simile alla lettera U), in cui sono utilizzati tutti i blocchi basilari presentati nella sezione 3.2: si tratta infatti, anche in questo modello particolare, di una sequenza di layer convoluzionali e di max pooling.

L'architettura nello specifico contiene due sezioni: un percorso di riduzione dei dati, in cui le informazioni vengono contratte per estrarre le principali informazioni dall'ingresso, e un percorso di espansione in cui l'informazione estratta viene riportata alla stessa dimensionalità dell'input (vedi figura 5.1).

E' importante evidenziare come, in questa modellizzazione, siano stati utilizzati layer convoluzionali 2D: questo significa che, alle forme d'onda di flusso e pressione, che normalmente sono vettori 2D, è stata aggiunta una terza dimensione così da renderle, per dimensionalità, sostanzialmente equipollenti a delle immagini in bianco e nero (vedi equazione seguente e codice 5.1).

$$\textit{Input shape} = \textbf{timesteps} \times \textbf{n\_features} \times \textbf{n\_channels} = (16, 2, 1)$$

```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
X_test=X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
Y_train=Y_train.reshape(Y_train.shape[0],Y_train.shape[1],1,Y_train.shape[2])
Y_test=Y_test.reshape(Y_test.shape[0], Y_test.shape[1], 1, Y_test.shape[2])
```

Listing 5.1: Aggiunta di una dimensione fake sia ai dati di train che ai dati di test

Questa scelta è stata presa per preservare una maggiore aderenza ai modelli presenti in letteratura, i quali sono principalmente diffusi e applicabili in contesti relativi alla segmentazione delle immagini (che possono essere viste appunto come vettori 3D dove la terza dimensione rappresenta l'informazione relativa al colore).

In particolar modo abbiamo deciso di realizzare due modelli comparativi di complessità differenti, andando a modificare il parametro *depth* che corrisponde al numero di blocchi che vengono ad essere composti e impilati uno sopra l'altro (vedi listato 5.2):

- un primo modello, più semplice, con una profondità pari a 2 e composto da $\simeq 70$ *mila* parametri;
- un secondo modello, più completo e, di conseguenza, con un maggior numero di parametri da addestrare basato su quella che è la struttura e la profondità presentata nell'approccio [23] in cui vengono ad essere composti 4 livelli di profondità: esso rappresenta invece un modello formato da $\simeq 1$ *milione* e 100 *mila* parametri.

  Questa forte differenza del numero di parametri è dovuta al fatto che, man mano che la profondità della rete aumenta, cresce anche la numerosità dei filtri di un fattore $\times 2$: è per questo motivo che, in questo secondo modello, avendo una profondità maggiore che consente di arrivare ad avere fino a 256 filtri, si ottengono un numero più elevato di parametri.

```
def create_model(depth, input_shape, activation, num_of_classes,
    n_crops, kernel_size, kernel_init, kernel_reg, padding, dil, filters
    , pools):
    modelMVM = Sequential()
    modelMVM.add(InputLayer(input_shape=input_shape, name="Input"))
    residual_cons, end_filter_value = create_encoder(modelMVM, depth=
    depth, [...], name="encoder", name_prefix="")
    filters = end_filter_value
    create_decoder(modelMVM, residual_cons, depth=depth, [...], name="
    decoder", name_prefix="")
    create_dense_modeling(modelMVM, num_of_classes, filters, activation
    , name_prefix="")
    return modelMVM
```

Listing 5.2: Design, espresso ad alto livello, della rete convoluzionale U-Net

```
n_periods = X_train.shape[1]
input_dims = X_train.shape[2]
n_channels = 1 # Like RGB for images (for RGB this number is equal to
    3)
input_shape = (n_periods, input_dims, n_channels)
num_of_classes = 3
n_crops = 0
kernel_size = 5
filters = 16
depth = 2
pool_size = 2
pools = [pool_size] * depth
dilation_rate = (2, 1)
activation = "relu"
kernel_init = "he_normal"
kernel_reg = l2(0.0005)  # Typical value
padding = "same"
checkpoint_path = "./checkpoint_MVM/" + datetime.now().strftime("%H_%M_
    %d_%m_%Y") + "/cp-{epoch:04d}.ckpt"
use_CCE = False
epochs = 50
batch = 100
```

Listing 5.3: Iper parametri utilizzati per il modello in figura 6.1

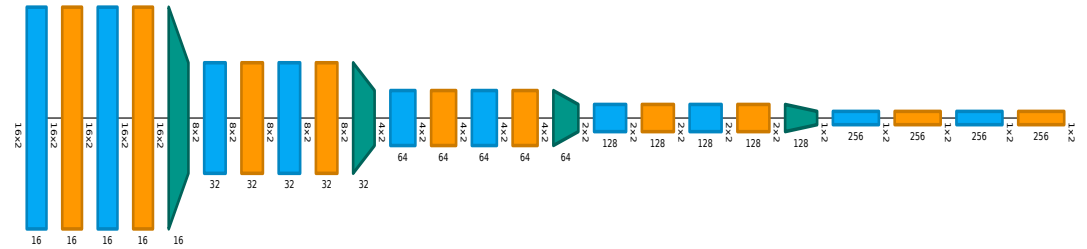
5.1 Struttura

Dando uno sguardo d'insieme al modello, possiamo evidenziare come esso consista in due parti principali:

- Encoder;
- Decoder e output finale.

5.1.1 Encoder

Esso consiste in una sequenza di layer convoluzionali e di pooling: la quantità di blocchi appartenente a tale sequenza è definita tramite il parametro *depth* il quale, come già messo in evidenza, consente di andare a definire il numero di blocchi che vengono ad essere posti in sequenza e quindi la complessità del modello.



(a) *Design*

(b) *Legenda dei layer*

Figura 5.2: Design della parte di encoder con profondità = 4

Come si può notare da una visione di insieme della figura 5.2, il path di encoder permette di andare a diminuire la dimensionalità dell'input ottenendo layer con una risoluzione più bassa ma contenenti informazioni più di alto livello e che risultano essere fondamentali per definire la classificazione dell'input.

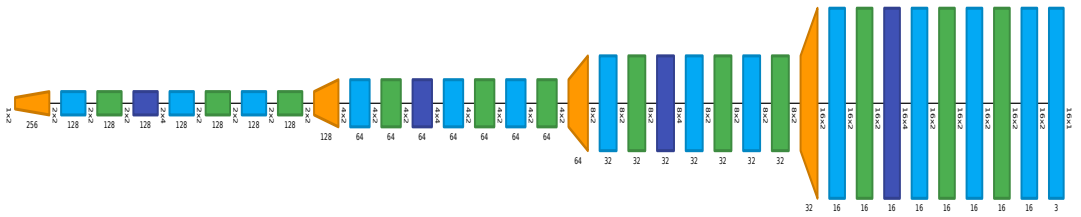
Sempre in figura 5.2 si può vedere come ogni blocco sia composto da una sequenza formata dai seguenti layers:

- Conv2D;
- BatchNormalization;
- Conv2D;
- BatchNormalization;
- MaxPoooling2D.

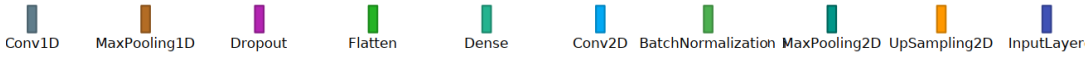
Si può notare come la dimensionalità diminuisca man mano che aumenta la profondità: si parte con un input che ha dimensione $(16, 2, 1)$ e, applicando 4 blocchi *MaxPolling2D* con $pool\_size = (2, 1)$, si ottiene ad ogni step un dimezzamento della dimensione del dato, passando da una 16×2 a 1×2: da evidenziare come la dimensione del pooling sia 2D ma, nonostante questo, la seconda dimensione dell'operazione di pooling è pari proprio ad 1, per il fatto che la terza dimensione del dato è pari all'unità e non ha contenuto informativo.

Allo stesso tempo si ha un aumento della profondità dovuto al numero di filtri applicati, i quali vengono raddoppiati ad ogni livello di profondità: partendo da un valore di 16 filtri iniziali, si giunge ad averne 256 nel punto di valle della nostra rete ad "U".

5.1.2 Decoder e output



(a) *Design*



(b) *Legenda dei layer*

Figura 5.3: Design della parte di decoder con profondità = 4

Per ovvi motivi di simmetria, essendo che ci si pone l'obbiettivo di ottenere in uscita la stessa dimensionalità dell'ingresso, il percorso di decoder risulta avere le stesse caratteristiche della parte di encoder sia in termini di dimensionalità degli iperparametri sia per la tipologia di blocchi, i quali saranno ancora composti da

- Conv2D;
- BatchNormalization;
- Conv2D;
- BatchNormalization;
- Upsampling2D.

Nella parte di decoder ci sono principalmente due aspetti che bisogna mettere in evidenza:

- UpSampling: rappresenta il complementare del concetto di features reduction. Essendo infatti che nel contesto della semantic segmentation dobbiamo considerare di mantenere sia il concetto di "quale" features è presente e di "dove" essa si trova, è necessario aumentare quella che è la dimensionalità dei layers.

  Nello specifico questo comando permette di andare a ripetere le righe e le colonne del dato sul quale è applicato il comando per un numero di volte definito dal parametro *size* il quale nel nostro caso sarà pari a (2,1).

- Concatenamento di features dal layer simmetrico di encoder: questo è un concetto importante perché, se andassimo ad utilizzare solamente il path di encoder seguito da quello di decoder, si avrebbe una forte perdita di informazioni a basso livello ovvero di features non rilevanti (che sono tralasciate dal percorso di encoder), ma che risultano essere necessarie per andare a riprodurre l'informazione fornita in input ed ottenere una classificazione corretta ed accurata.

Per ottenere queste features a basso livello, si permette ai layer di decoder di accedere ad alcune informazioni intermedie prodotte dai layers dell'encoder, ottenendo così il comportamento definito come di *skip connections*.

Queste informazioni intermedie dell'encoder vengono concatenate, con un'operazione di *crop and match*, agli input dei corrispettivi layer intermedi del decoder nella direzione della dimensione dei filtri: in questo modo si fornisce la possibilità di completare la ricostruzione dell'informazione.

L'output del path di decoder può quindi essere considerato come la confidenza che ogni istante ha di appartenere alle tre classi definite nel problema: si nota infatti in figura 5.3 che gli ultimi due layer sono rappresentati da layer di tipo convoluzionali, la cui dimensionalità finale dei filtri risulta essere pari al numero delle classi che vengono ad essere mappate nel problema, ovvero 3.

Il layer di uscita rappresenta, come già introdotto in precedenza, un punto importante del modello, il quale permette di indicizzare quelle che sono le features apprese dai layers nascosti: in particolare esso utilizza una funzione di attivazione di tipo *softmax* (vedi sezione 3.5.3), la quale produce in uscita un vettore che somma a 1 e dove, il valore più grande, indica probabilità che l'input ha di appartenere ad una specifica classe.

6

MVM single output model

In questo secondo approccio invece non è più stata seguita l'idea di utilizzare un concetto di *semantic segmentation*, ma si è andati verso un approccio più standard nel contesto della classificazione: in questo caso il modello convoluzionale presenta in uscita solamente una singola predizione composta a sua volta da tre probabilità rappresentanti le possibilità che, l'ultimo istante della sequenza passata in ingresso alla rete neurale appartenga ad una delle 3 classi prese in considerazione.

Oltre ad aver modificato la conformazione dell'output, in questo secondo approccio siamo andati a considerare una dimensionalità dell'input più comune e standard per i problemi relativi alla classificazione delle serie temporali, in cui esse vengono trattate come entità 2D con dimensionalità pari a

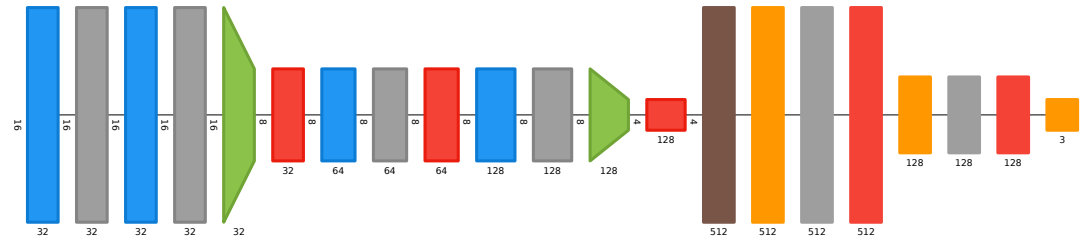
$$\textit{Input shape} = \textbf{timesteps} \times \textbf{n\_features} = (16, 2)$$

dove n_features rappresenta il numero di variabili che compongono la serie temporale multivariata (nel nostro caso questa dimensione sarà 2, per la presenza delle informazioni relative a flusso e pressione). Da questo ne consegue anche che i blocchi convoluzionali e di pooling utilizzati avranno dimensionalità 1D (*Conv1D*, *MaxPooling1D* etc..).

Essendo le dimensionalità minori, anche il comportamento dei blocchi fondamentali risulta essere più semplice: in particolare abbiamo che il filtro convoluzionale (kernel) si muove nell'unica direzione che ha a disposizione, andando dall'inizio della serie temporale verso la fine della stessa.

Come presentato nella sezione 3.2 nel campo delle immagini, anche in questo caso gli elementi del kernel vengono moltiplicati per il corrispondente elemento nella serie temporale e successivamente vengono sommati tra di loro: al risultato verrà poi applicata una funzione di attivazione non lineare.

Il risultato di questa operazione di convoluzione sarà ancora una serie temporale, la cui lunghezza dipende dalla grandezza del filtro (nel nostro caso la lunghezza rimarrà



(a) *Design*

(b) *Legenda dei layer*

Figura 6.1: Design del modello categorico con singola predizione in uscita

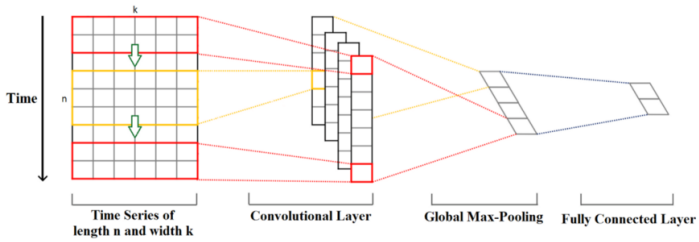


Figura 6.2: Approccio generico per la classificazione delle serie temporali

costante, essendo che viene applicata una convoluzione con padding di tipo *same*) e la cui numerosità coincide con la numerosità dei filtri (vedi figura 6.2)

Nel listato 6.1 è riportato il codice per la composizione del modello a singola uscita: da evidenziare come l'ultimo layer corrisponda ad un blocco di tipo *Dense* che utilizza sempre una funzione di attivazione *softmax* e con un numero di classi pari a 3 (come definito nella sezione degli iper-parametri 6.2).

La dimensionalità dell'output è visibile anche in figura 6.1 in cui si può notare come l'uscita sia composta da 3 elementi.

```
105 def create_categorical_model(input_shape, activation, num_of_classes,
        kernel_size, kernel_init, kernel_reg, padding, dilation, filters,
        pool_size):
106
107   modelMVM = Sequential()
108   modelMVM.add(Conv1D(32, kernel_size=kernel_size, activation=
        activation, padding=padding, dilation_rate=dilation,
        kernel_initializer=kernel_init, kernel_regularizer=kernel_reg,
        input_shape=input_shape))
109   modelMVM.add(BatchNormalization())
110
111   modelMVM.add(Conv1D(32, kernel_size=kernel_size, activation=
        activation, padding=padding, dilation_rate=dilation,
```

```
    kernel_initializer=kernel_init, kernel_regularizer=kernel_reg))
112 modelMVM.add(BatchNormalization())
113 modelMVM.add(MaxPooling1D(pool_size=2))
114 modelMVM.add(Dropout(0.25))
115
116 modelMVM.add(Conv1D(64, kernel_size=kernel_size, activation=
    activation, padding=padding, dilation_rate=dilation,
    kernel_initializer=kernel_init, kernel_regularizer=kernel_reg))
117 modelMVM.add(BatchNormalization())
118 modelMVM.add(Dropout(0.25))
119
120 modelMVM.add(Conv1D(128, kernel_size=kernel_size, activation=
    activation, padding=padding, dilation_rate=dilation,
121 kernel_initializer=kernel_init, kernel_regularizer=kernel_reg))
122 modelMVM.add(BatchNormalization())
123 modelMVM.add(MaxPooling1D(pool_size=2))
124 modelMVM.add(Dropout(0.25))
125
126 modelMVM.add(Flatten())
127
128 modelMVM.add(Dense(512, activation=activation))
129 modelMVM.add(BatchNormalization())
130 modelMVM.add(Dropout(0.5))
131
132 modelMVM.add(Dense(128, activation=activation))
133 modelMVM.add(BatchNormalization())
134 modelMVM.add(Dropout(0.5))
135
136 modelMVM.add(Dense(num_of_classes, activation='softmax'))
137
138 return modelMVM
```

Listing 6.1: Design della rete tramite comando Sequential fornito dal framework *Keras*

```
139 n_periods = X_train.shape[1]
140 input_dims = X_train.shape[2]
141 num_of_classes = 3
142 kernel_size = 3
143 filters = 16
144 pool_size = 2
145 dilation_rate = 2
146 activation = "relu"
147 kernel_init = "he_normal"
148 kernel_reg = l2(0.0005)  # Typical value
149 padding = "same"
150 input_shape = (n_periods, input_dims)
151 checkpoint_path = "./checkpoint_MVM/" + datetime.now().strftime("%H_%M_%d_%n_%Y") + "/cp-{
    epoch:04d}.ckpt"
152 use_CCE = True
153 epochs = 50
154 batch = 100
```

Listing 6.2: Iper parametri utilizzati per il modello in figura 6.1

7

Risultati

Gli step di pre processing presentati nei capitoli precedenti, unitamente alle possibili scelte per la definizione della loss function e del design del modello convoluzionale, sono stati presi in considerazione per andare eseguire poi un approccio comparativo tra le varie opzioni così da scegliere il modello il quale risulta essere in grado di fornire maggiore affidabilità, con un basso margine di errore, nell'individuazione degli istanti di inizio inspirazione, espirazione e di background (come descritto nella sezione 3.4).

In particolar modo durante le prove realizzate, siamo andati a mantenere costanti i valori assegnati agli iper-parametri, modificando solamente la funzione di loss utilizzata e il design della CNN.

Inoltre abbiamo tenuto conto anche di un'ulteriore differenziazione possibile: essendo che, come già messo in evidenza, il respiratore MVM lavora solamente in modalità *PC* e *PS* (vedi sezione 1.3), abbiamo seguito due strade:

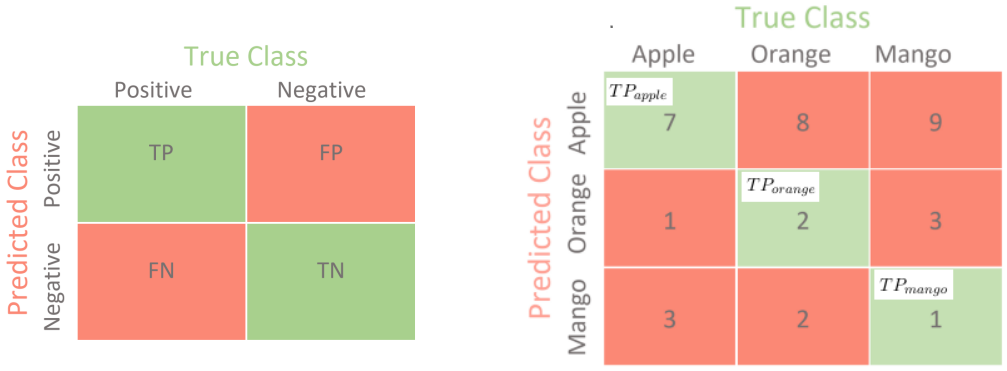
- Un train dei modelli utilizzando tutti i dati disponibili nei datasets utilizzati;
- Un train, per la tipologia modelli che ha dato maggiore garanzia di accuratezza nella prima fase, tramite l'utilizzo solamente di dati relativi a respirazioni di tipo PC/PS.

7.1 Metriche di confronto

Per confrontare la bontà dei modelli siamo andati ad utilizzare alcune metriche in grado di fornire una valutazione consistente dell'affidabilità del modello stesso, evitando contesti in cui la presenza di *class imbalance* potesse fornire una misurazione distorta del funzionamento del modello stesso.

Il panorama in letteratura per quanto riguarda le possibili metriche utilizzabili è molto vasto: la loro scelta dipende essenzialmente dalla tipologia del modello e dal contesto in cui esso risulta essere applicato.

In particolare sono state prese in considerazione le seguenti metriche, alcune delle quali rappresentano una rivisitazione di coefficienti spesso utilizzati per la compa-



(a) *Confusion matrix per problemi di classificazione binaria*

(b) *Confusion matrix per problemi con classi multiple con rappresentazione del concetto di TP per ogni classe*

Figura 7.1: Esempi di *confusion matrix*

razione tra modelli nel contesto della classificazione e, il cui utilizzo, è fortemente consigliato per il contesto di problemi con classi sbilanciate.

Nello specifico esse derivano dal concetto di *confusion matrix*, riportato nella figura 7.1, sia per problemi di classificazione binaria sia per contesti come il nostro di classificazione multi classe.

Come primo step andiamo ad introdurre il concetto generale espresso dalle metriche utilizzate per confrontare i modelli facendo riferimento alla figura 7.1a:

- *Precision*: generalmente la precision rappresenta l'abilità del modello di andare ad identificare solo le istanze corrette di ogni classe e viene espresso come $\frac{TP}{TP+FN} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$
- *Recall*: la recall standard indica la capacità del modello di trovare tutte le istanze corrette per ogni classe, secondo la formula $\frac{TP}{TP+FP} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$
- *F1-Score*: rappresenta una media tra *precision* e *recall* il quale, nel momento in cui risulta essere pari a 1, indica un'elevata precision e, allo stesso tempo, un'elevata recall ed è ottenibile come $\frac{2\times\text{Recall}\times\text{Precision}}{\text{Recall+Precision}}$.

Per esempio, nella confusion matrix in figura 7.1b, per la classe "apple" avremo i seguenti valori di classificazione:

- TP_{apple} = elemento sulla diagonale = 7;
- TN_{apple} = somma degli elementi che non sono stati classificati correttamente come "apple" (matrice escludendo riga e colonna "apple") = 2 + 3 + 2 + 1 = 8;
- FP_{apple} = tutti gli elementi che sono stati classificati come "apple", quando in realtà sono o "orange" o "mango" = 8 + 9 = 17;

- FN_{apple}" = oggetti classificati come "orange" o "mango", quando in realtà sono "apple" = 7 + 3 = 4.

Queste tre metriche sono conformi però con problemi di classificazione binaria, in cui le classi disponibili sono solamente due: quando si va a lavorare con classi multiple, come nel nostro contesto, intervengono i concetti di

- *macro-average*: consiste nel calcolare le metriche presentate precedentemente in maniera separata per ogni classe, andando poi a comporle come valore medio;
- *micro-average*: in questo caso, contrariamente all'approccio *macro*, si va a calcolare una metrica fornita dall'aggregazione delle varie classi. Questo approccio è spesso utilizzato per dataset che soffrono di sbilanciamento delle classi, essendo che si va a prendere in considerazione la numerosità di ogni classe;
- *weighted-average*: sfruttando la numerosità di ogni classe, esso va a comporre lo score di ognuna delle classi moltiplicandolo per la relativa numerosità.

Per mantenere il più pulita possibile la comparazione tra i modelli, è stato scelto di effettuare un confronto solamente tramite l'uso dei parametri di *micro F1-score*, che altro non è che la composizione di *micro precision* e *micro recall* calcolati considerando i totali TP (True Positive), FP (False Positive) e FN (False Negative) di ogni classe e combinandoli poi insieme.

Oltre a questi primi tipici parametri, rivisti per essere applicati al contesto di multi-classificazione, siamo andati a considerare altre due metriche molto utili per problemi che soffrono di sbilanciamento della numerosità delle classi:

- *Coefficiente Kappa di Cohen*: esso è un coefficiente statistico che rappresenta il grado di accuratezza e affidabilità di una classificazione statistica, ed è una delle migliori metriche per valutare classificatori multi-classe su set di dati sbilanciati.

  Le metriche tradizionali, basate sul rapporto di classificazione, sono infatti sbilanciate verso la classe di maggioranza e presuppongono una distribuzione identica delle classi effettive.

  Al contrario il coefficiente di Cohen è un indicatore di concordanza che misura la vicinanza delle classi predette alle classi reali, comparandole ad un classificatore casuale.

  L'output è normalizzato tra 0 e 1 e pertanto può essere confrontato direttamente nell'attività di classificazione: più il valore risulta essere vicino ad uno, più elevata sarà la qualità della classificazione fornita dal modello.

- *Curve ROC* (Receiver Operating Characteristc curve): esse rappresentano un altro strumento per confrontare la bontà dei modelli tramite un'ispezione visiva, dando così una valida alternativa allo studio e all'analisi di diverse confu-

sion matrix, ottenibili impostando differenti valori di threshold per i modelli in questione.

Queste curve sono infatti ottenute plottando i diversi valori di TPR e FPR, facendo variare alcune threshold del modello ed ottenendo così una curva di trade-off tra le seguenti due grandezze:

- TPR (True Positive Rate): indica la capacità del modello di classificare elementi correttamente e corrisponde al concetto di *recall* presentato precedentemente;
- FPR (False Positive Rate): esso invece indica la probabilità del modello di classificare erroneamente, ovvero di lanciare un falso allarme.
  Esso può essere ottenuto come $\frac{FP}{FP+TN} = \frac{\text{False Positives}}{\text{False Positives+True Negatives}}$.

Cosa va quindi a significare l'andamento della curva diagonale di colore nero (visibile anche in figura 7.3)?

Prendiamo per esempio le coordinate $(TPR = 1, FPR = 1)$: questo punto, per ogni classe considerata, indica che stiamo correttamente classificando i dati della classe presa in considerazione ma, allo stesso tempo, stiamo sbagliando tutti i samples appartenenti all'altra classe essendo che si tratta appunto di un contesto binario, dato che viene considerata ogni classe presa singolarmente.

Tutti i punti sulla linea diagonale, indicano quindi un contesto in cui $TPR = FPR$: questo significa che la porzione di dati *correttamente* classificati come appartenenti a quella classe è pari alla porzione di dati *non correttamente* classificati come appartenenti alla classe complementare.

Sulla diagonale di queste curve, verrà ad essere visualizzata la classificazione del modello random: se le curve che otteniamo si trovano al di sotto della diagonale avremo a che fare con modelli pessimi; se invece la curva risulta trovarsi nella metà superiore, il modello sarà in grado di dimostrare di avere acquisito conoscenza utile per la classificazione durante il train sui dati, proprio perché l'obbiettivo è quello di diminuire il più possibile il valore FPR mantenendo alto il valore della sua controparte TPR.

Questa bontà è indicata anche dal valore AUC (*Area Under Curve*) che indica l'area sottesa dalla curva ed è rappresentato da valori compresi tra 0 e 1 e che fornisce un valore sul quale basare la comparazione tra vari modelli.

Essendo che esse sono definite solamente per problemi di classificazione binaria, è necessario andare a realizzare un leggero adattamento per i problemi di *multi-class classification*: si arriva così a definire una curva per ogni singola classe, indicando perciò il comportamento di ogni classe rispetto alle altre (come se fosse un confronto Round Robin), come si può evincere dal listato 7.1.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from itertools import cycle
plt.style.use('ggplot')

def ROC_curves(Y_test, Y_predicted, model_name, n_classes=3):
  # Plotting and estimation of FPR, TPR
  fpr = dict()
  tpr = dict()
  roc_auc = dict()
  for i in range(n_classes):
  fpr[i], tpr[i], _ = roc_curve(Y_test[:, i], Y_predicted[:, i])
  roc_auc[i] = auc(fpr[i], tpr[i])
  colors = cycle(['blue', 'red', 'green'])
  for i, color in zip(range(n_classes), colors):
  plt.plot(fpr[i], tpr[i], color=color, lw=1.5,
  label='ROC curve of class {0} (area = {1:0.2f})' ''.format(i + 1,
    roc_auc[i]))
  plt.plot([0, 1], [0, 1], 'k-', lw=1.5)
  plt.xlim([-0.05, 1.0])
  plt.ylim([0.0, 1.05])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title(model_name + ' - Receiver operating characteristic for
    multi-class data')
  plt.legend(loc="lower right")
  plt.show()
```

Listing 7.1: Funzione per la definizione delle curve di ROC tramite framework sklearn.metrics

7.2 Analisi dei risultati

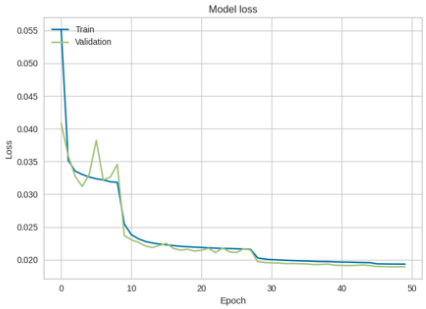
Analizzando le metriche riportate nella tabella 7.1, si può notare come il modello in grado di dare una maggiore accuratezza risulti essere quello che basa il proprio funzionamento sul concetto di *semantic segmentation* e con depth pari a 2: in particolare, nonostante il modello risulti essere più semplice rispetto al suo corrispettivo con profondità pari a 4, esso ha fornito una maggiore garanzia di accuratezza nel riconoscere gli istanti caratterizzanti il problema.

Come si evince dalla figura 7.2, la rete performa bene sia sui dati di train che, soprattutto, sul set di validazione: inoltre, in figura 7.3, si ha una conferma di quanto riportato graficamente dall'andamento dell'errore e dell'accuracy in fase di train.

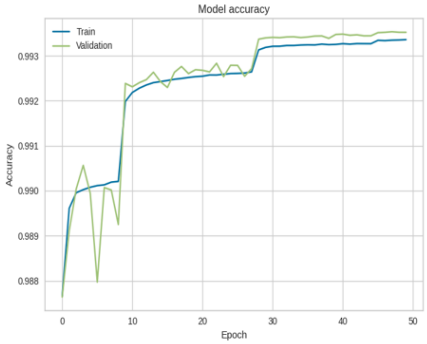
Infatti si può notare come le curve di ROC si trovino ampiamente sopra il livello corrispondente alla performance del modello random, ovvero la linea diagonale: a questa prima analisi visiva, si aggiunge un'ulteriore conferma legata ai valori di AUC che risultano essere estremamente positivi per tutte e tre le classi.

Design	Funzione di Loss	Descrizione modello	Metriche						
			Micro Precision	Micro Recall	Micro F1-Score	Cohen Kappa	AUC (Area Under Curve)		
							Inizio Insp.	Inizio Esp.	BG
U-NET	Categorical Cross Entropy	U-Net depth = 2 All Datas	0.99	0.99	0.99	0.959	0.99	0.97	0.98
		U-Net depth = 2 Only PC-PS	0.99	0.99	0.99	0.949	0.96	0.94	0.96
		U-Net depth = 4	0.99	0.99	0.99	0.946	0.94	0.89	0.92
	DICE	U-Net depth = 2	0.98	0.98	0.96	0.891	0.97	0.96	0.97
		U-Net depth = 4	0.94	0.95	0.97	0.883	0.94	0.92	0.93
Single Output Model	Categorical Cross Entropy	All Datas	0.98	0.98	0.98	0.865	0.94	0.89	0.92
		Only PC-PS	0.97	0.97	0.97	0.783	0.93	0.87	0.89
	DICE	-	0.98	0.98	0.98	0.804	0.96	0.80	0.86

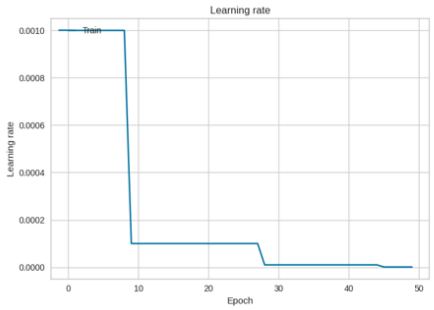
Tabella 7.1: Riassunto delle metriche rilevate per i diversi modelli



(a) *Loss*



(b) *Accuracy*



(c) *Modifica automatica del Learning Rate durante il training*

Figura 7.2: Grafici relativi al train del modello di CNN U-Net con profondità = 2

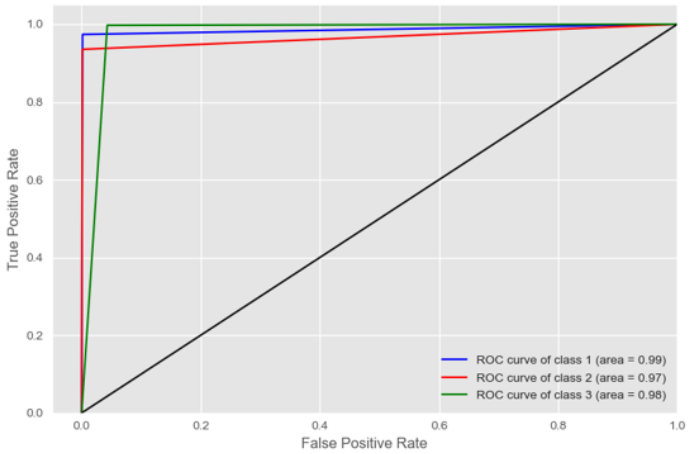


Figura 7.3: Curve di ROC e valori AUC per modello U-Net con *CCE* e profondità = 2

7.3 Analisi delle predizioni

In questa sezione andiamo invece a riportare alcune analisi grafiche di quello che è il comportamento predittivo della rete: in particolare, in prima battuta siamo andati a testare la bontà della predizione sulla parte di test del dataset utilizzato durante il train avendo così a disposizione anche la curva *ground_truth* (curva blu in figura 7.4).

In un secondo momento invece abbiamo preso in considerazione un batch di dati fornitoci direttamente dal gruppo di lavoro MVM: in particolare per questi dati non risultano essere a disposizione le etichette relative alla tipologia di istanti; per alcune di esse sono stati forniti alcuni parametri temporali che hanno permesso di effettuare dei ragionamenti incrociati sulla qualità della classificazione. Per altri dati ancora, non sono state fornite informazioni temporali: la bontà della classificazione quindi sarà valutata in maniera qualitativa, analizzando la sequenzialità di eventi inspiratori ed espiratori.

7.3.1 Predizione da dati di test

Nella composizione 7.4, abbiamo realizzato l'analisi della predizione del modello sui dati che non sono stati somministrati durante il train, ovvero sul dataset di test: si può notare come la capacità predittiva del modello sia buona essendo che solamente pochi istanti vengono misclassificati.

Questo va a confermare la bontà delle metriche riportate in tabella 7.1.

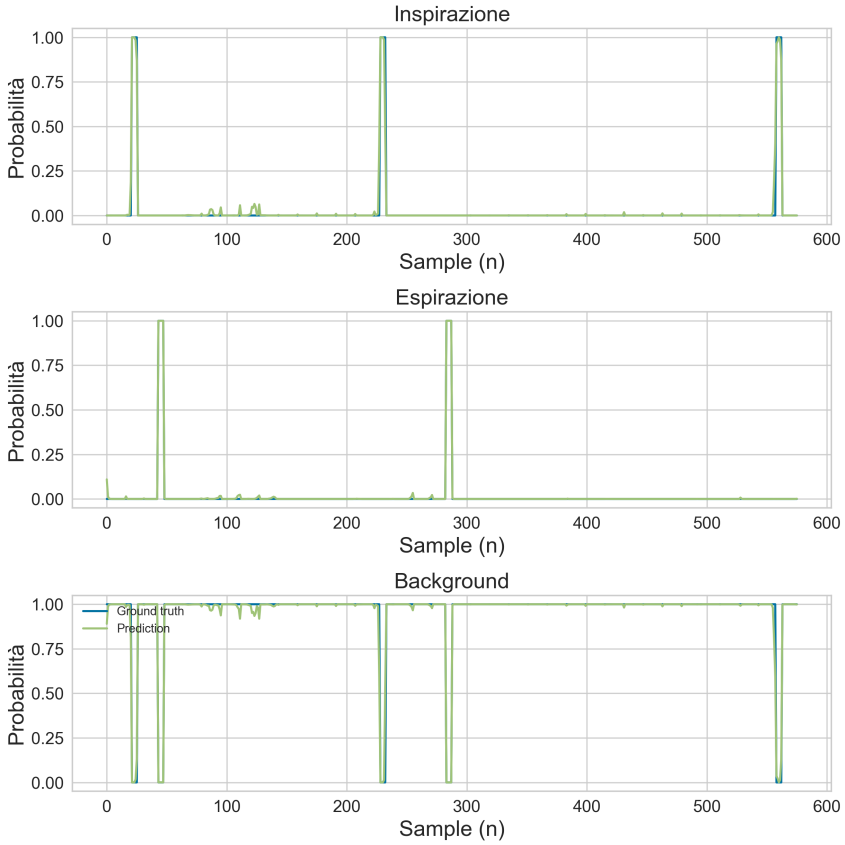
7.3.2 Predizione in situazione di apnea

Nella figura 7.5, siamo invece andati a testare il comportamento del modello in caso di apnea del paziente: questa situazione corrisponde ad una condizione di assenza di trigger di inspirazione e di espirazione da parte del paziente, essendo che le relative forme d'onda risultano essere completamente azzerate.

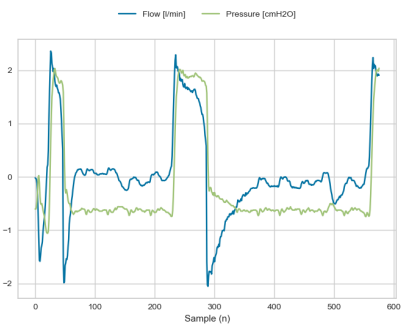
Per rendere più realistico il test non abbiamo considerato delle curve di flusso e pressione completamente nulle: questa analisi risulta essere relativamente banale essendo che, nel momento in cui si vanno ad inserire dei valori nulli in input al modello, i vari layer vanno a compiere operazioni di combinazione lineare tramite moltiplicazione vettoriale su valori nulli, ottenendo così un'uscita completamente nulla.

Abbiamo deciso quindi, come test eventualmente da approfondire in sviluppi futuri, di analizzare il comportamento della rete quando in ingresso gli vengono passati segnali affetti da rumore.

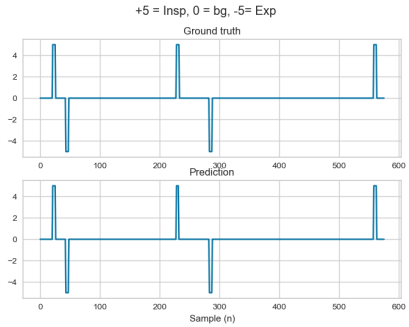
In questo modo si intende infatti riprodurre il più possibile una condizione di funzionamento reale: le misurazioni di flusso e pressione infatti non risultano essere mai perfettamente pari a 0, per la presenza di errore sia in fase di misurazione che di acquisizione.



(a) *Probabilità predette per ognuna delle tre classi considerate*



(b) *Curve di flusso e pressione oggetto di classificazione*



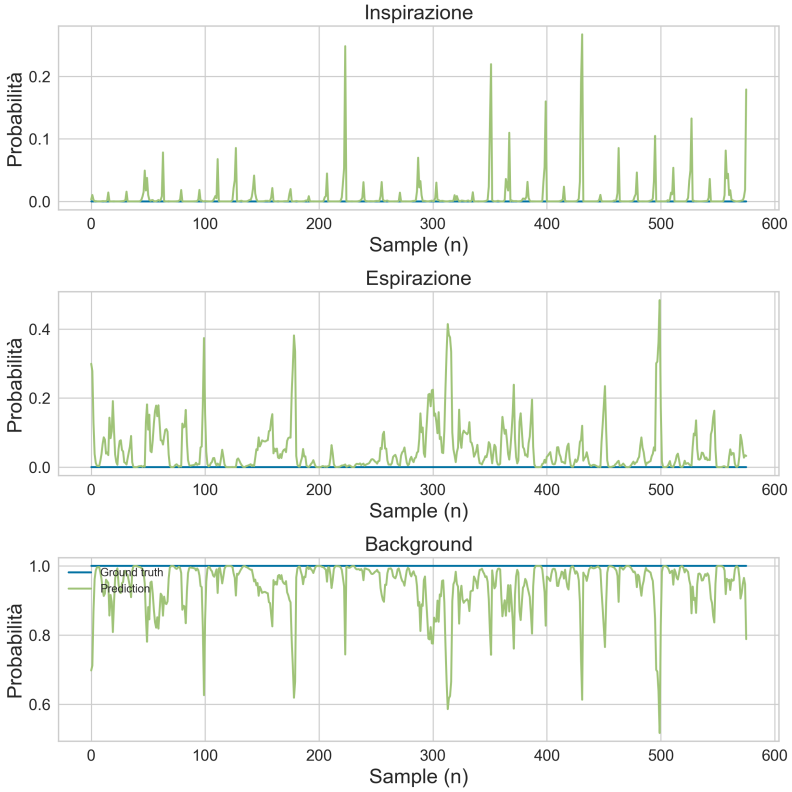
(c) *Predizione in uscita*

Figura 7.4: Rappresentazione di input - probabilità - output utilizzando i dati di test

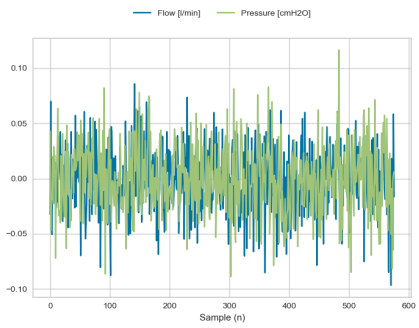
Come si vede nel listato 7.2 al segnale completamente pari a zero, ottenuto tramite le funzionalità di numpy *np.zeros*, siamo andati ad aggiungere un rumore gaussiano con media nulla e varianza ridotta essendo che, i segnali che la rete prende in input, risultano essere normalizzati e, di conseguenza, lo sarà anche l'eventuale noise che avrà perciò una *magnitude* minore.

La risposta del modello testato in questa condizione di apnea è ottima, essendo che sono molto bassi i picchi di probabilità che compaiono nella predizione degli istanti di

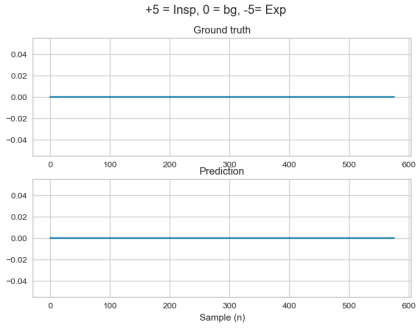
inizio inspirazione e espirazione, mentre invece la probabilità di classificazione come background è prossima all'unità.



(a) *Probabilità predette nel caso di apnea con noise di misurazione*



(b) *Simulazione di curve di flusso e pressione in un contesto di apnea con aggiunta di noise di misurazione*



(c) *Predizione in uscita*

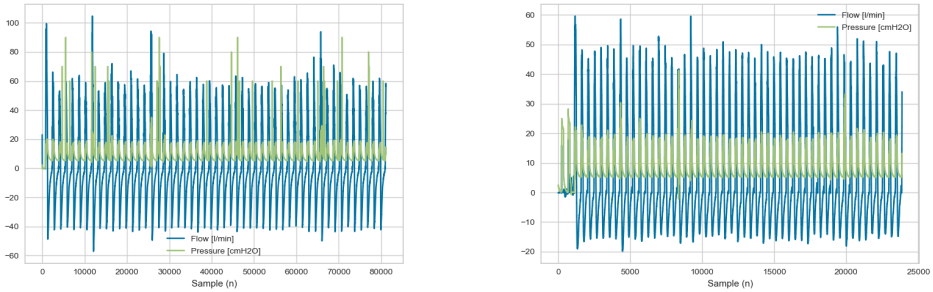
Figura 7.5: Rappresentazione di input - probabilità - output in un contesto di apnea del paziente con misurazioni affette da rumore

```
input_shape = (input_i.shape[0], input_i.shape[1], input_i.shape[2])
zeros = np.zeros(input_shape)
noise = np.random.normal(0, 0.03, input_shape)
input = zereos + noise
```

Listing 7.2: Creazione di rumore e aggiunta al segnale in ingresso

7.3.3 Predizione da dati MVM

Come ultimo step, per cercare ulteriori conferme della bontà del modello, siamo andati ad utilizzare un pull di dati acquisiti direttamente da MVM e la cui visione di insieme è riportata in figura 7.6.



(a) *Primo gruppo di dati MVM* (b) *Secondo gruppo di dati MVM*

Figura 7.6: Dati acquisiti da MVM

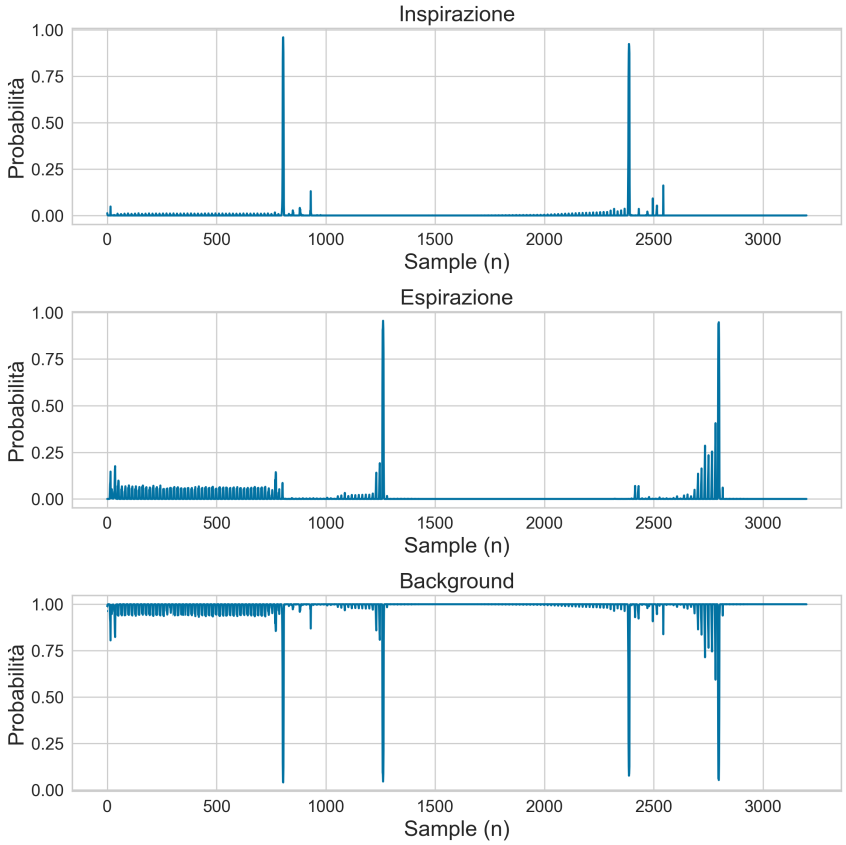
In figura 7.7, sono riportati i risultati ottenuti dall'analisi di una parte di forme d'onda campionate da MVM.

La bontà della classificazione in questo caso è stata verificata tramite un controllo sulla classe predetta: nello specifico è stato analizzato il fatto che la predizione delle classi fosse alternata, ovvero che ad un evento di inizio inspirazione segua, dopo alcuni istanti classificati come background, un evento di espirazione garantendo così la corretta predizione dell'alternanza dei cicli respiratori e delle relative fasi.

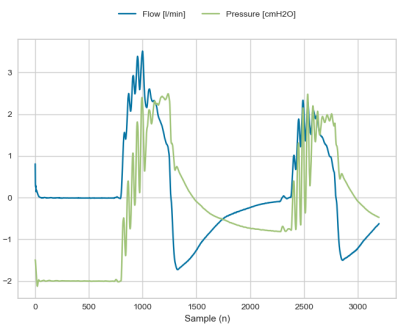
Questo approccio si è reso necessario per due motivazioni:

- Non è a disposizione la classificazione diretta dei singoli respiri, ovvero le indicazioni di quando inizia l'inspirazione e l'espirazione;
- Non sono inoltre note le informazioni relative a
  - Frequenza di campionamento;
  - Frequency Rate (FR);
  - Ratio I:E.

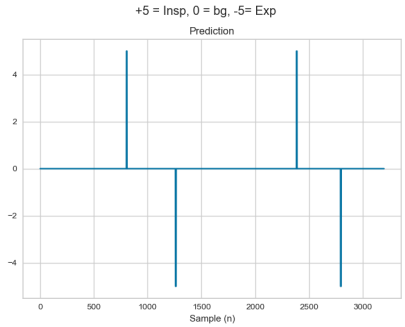
  Essi avrebbero permesso di fare un ragionamento sulla quantità di campioni appartenenti alla fase di inizio inspirazione e il numero, sempre di campioni, facenti parte della fase di inizio espirazione, avendo così un modo più analitico per verificare la bontà di tali predizioni.



(a) *Probabilità predette per ognuna delle tre classi considerate*



(b) *Curve di flusso e pressione oggetto di classificazione*



(c) *Predizione in uscita*

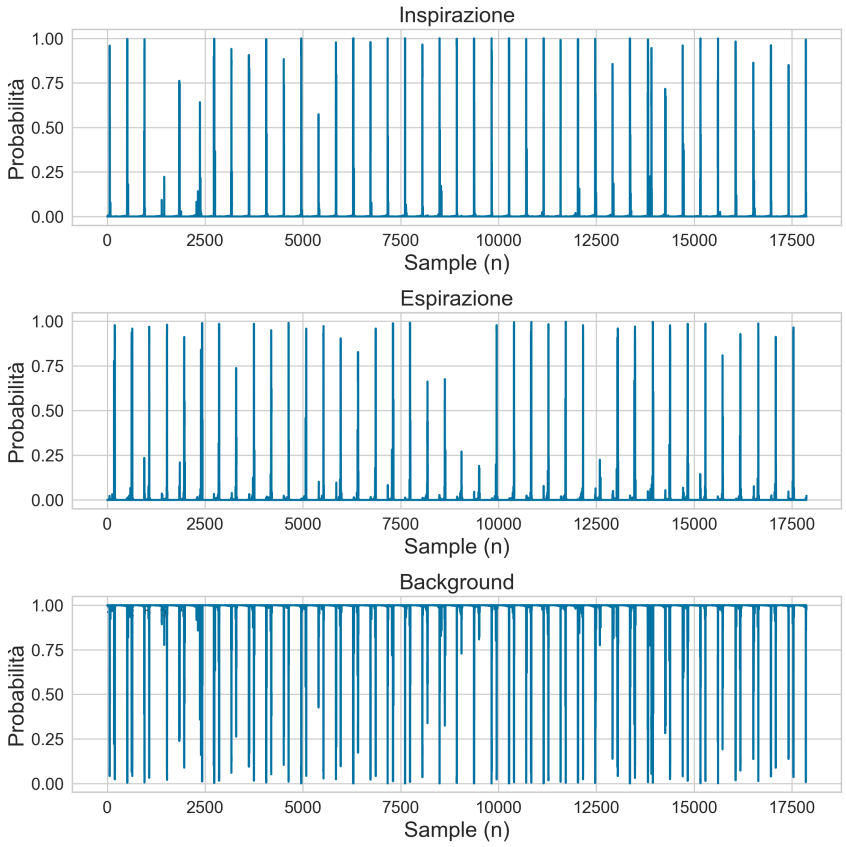
Figura 7.7: Rappresentazione di input - probabilità - output utilizzando una parte del primo insieme di dati MVM

In figura 7.8 invece riportiamo i risultati dello stesso procedimento applicato al secondo gruppo di dati: si nota come i risultati siano anche qui buoni. In particolare si vuole sottolineare come, per questo set di misurazioni e a differenza del primo set di dati, siano emerse tre tipologie di errori:

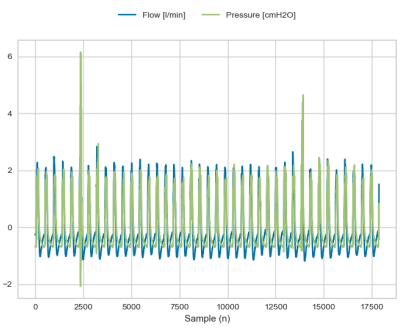
- mancato inizio inspirazione;
- mancato inzio espirazione;

- doppio trigger.

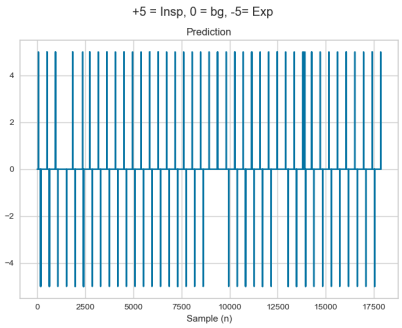
Questo ci fa comunque capire come il modello, in determinate situazioni, risulti avere qualche punto debole, che potrà essere migliorato e limato in futuro, sia dal modello stesso sia dal ciclo di controllo del ventilatore MVM.



(a) *Probabilità predette per ognuna delle tre classi considerate*



(b) *Curve di flusso e pressione oggetto di classificazione*



(c) *Predizione in uscita*

Figura 7.8: Rappresentazione di input - probabilità - output utilizzando il secondo insieme di dati MVM

8

Conclusioni e sviluppi futuri

I risultati ci hanno suggerito il fatto che sia possibile creare dei modelli convoluzionali per la detection di specifici istanti nel contesto delle forme d'onda dei ventilatori meccanici, anche in caso di presenza di altre tipologie di ventilazione, non gestite direttamente nel contesto MVM.

Si è visto come, il modello basato sulla semantic segmentation, sia in grado di fornire una maggiore garanzia sulla precisione e sulla robustezza della classificazione: in particolare è emerso come, un'elevata complessità del modello, non abbia portato ad una corrispondente crescita delle performances.

Sicuramente il modello *single output model* rappresenta una valida alternativa, anche per la semplicità del design e per la facilità di interpretazione della classificazione fornita in output.

E' vero anche che, le tematiche affrontate in questo elaborato, rappresentano solamente un primo approccio al problema, precursore di ulteriori sviluppi in cui potrebbero essere approfonditi ed analizzati alcuni punti che cerchiamo di tracciare di seguito:

- **Creazione di un asset per MVM**: comporre un dataset custom, acquisendo e processando i dati provenienti dai pazienti realmente trattati con il respiratore MVM. Per fare questo è necessario rispettare forti requisiti di privacy e di anonimizzazione del dato;

- **Etichettatura manuale**: tramite l'ausilio di esperti in ambito medico del team MVM, si potrebbe andare a *labellare* i dati provenienti da MVM, ispezionando visivamente le forme d'onda di flusso e pressione.

  Ci sono molti tools (come [26]) utilizzati nell'ambito della segmentazione delle immagini, che potrebbero consentire di segmentare facilmente il dataset;

- **Modifica della dimensionalità dei dati**: questo può essere ottenuto aumentando la sliding window con cui sono stati elaborati i dati, dando così la

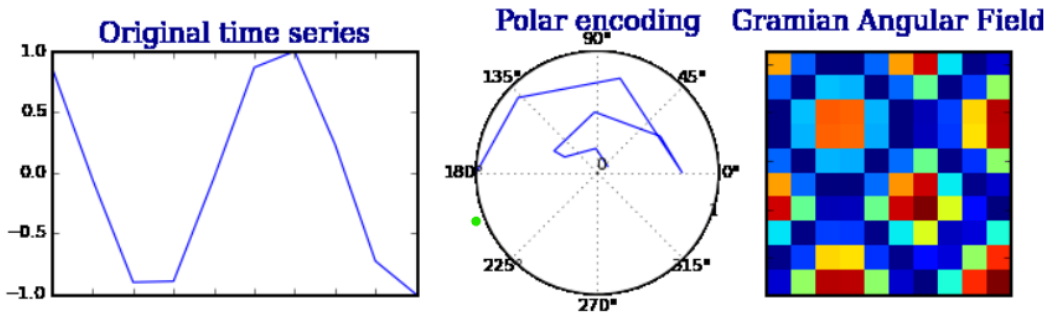


Figura 8.1: Esempio di encoding di serie temporali tramite immagini

possibilità al modello stesso di apprendere maggiori relazioni tra vari istanti su una scala temporale più lunga;

- **Pre-trained model**: si potrebbero utilizzare dei modelli già allenati (concetto di *transfer learning*);
- **Introduzione CNN nel ciclo di controllo**: gestire quella che è l'informazione fornita in uscita dai modelli analizzati, andando ad inserirla all'interno del contesto più ampio del ciclo di controllo MVM;
- **Trasformata di Garmian**: approfondire la possibilità di andare a trasformare le serie temporali in immagini tramite metodi analitici molto diffusi nella community scientifica (come, per esempio, la *trasformata angolare di Garmian*), potendo poi utilizzare metodi molto robusti di classificazione e/o di segmentazione sulle immagini (vedi figura 8.1).

9

Appendice

	ID	Tipo di layer	Out dim	Kernel	Filtri	Activation	Pad
	1	Input	16x2x1	-	-	-	-
ENCODER	2	Conv/BN	16x2x16	(5,1)	16	ReLU	same
	3	Conv/BN*	16x2x16	(5,1)	16	ReLU	same
	4	Max Pool	8x2x16	(2,1)	-	-	valid
	5	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	6	Conv/BN*	8x2x32	(5,1)	32	ReLU	same
	7	Max Pool	4x2x32	(2,1)	-	-	valid
	8	Conv/BN	4x2x64	(5,1)	64	ReLU	same
	9	Conv/BN*	4x2x64	(5,1)	64	ReLU	same
	10	Max Pool	2x2x64	(2,1)	-	-	valid
	11	Conv/BN	2x2x128	(5,1)	128	ReLU	same
	12	Conv/BN*	2x2x128	(5,1)	128	ReLU	same
	13	Max Pool	1x2x128	(2,1)	-	-	valid
	14	Conv/BN	1x2x256	(5,1)	256	ReLU	same
	15	Conv/BN	1x2x256	(5,1)	256	ReLU	same
DECODER	16	UpSampling	2x2x256	(2,1)	-	-	-
	17	Conv/BN	2x2x128	(5,1)	128	ReLU	same
	18	Crop-Concat(12, 17)	-	-	-	-	-
	19	Conv/BN	2x2x128	(5,1)	128	ReLU	same
	20	Conv/BN	2x2x128	(5,1)	128	ReLU	same
	21	UpSampling	4x2x128	(2,1)	-	-	-
	22	Conv/BN	4x2x64	(5,1)	64	ReLU	same
	23	Crop-Concat(9, 22)	-	-	-	-	-
	24	Conv/BN	4x2x64	(5,1)	64	ReLU	same
	25	Conv/BN	4x2x64	(5,1)	64	ReLU	same
	26	UpSampling	8x2x64	(2,1)	-	-	-
	27	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	28	Crop-Concat(6, 27)	-	-	-	-	-
	29	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	30	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	31	UpSampling	16x2x32	(2,1)	-	-	-
	32	Conv/BN	16x2x16	(5,1)	16	ReLU	same
	33	Crop-Concat(3, 32)	-	-	-	-	-
	34	Conv/BN	16x2x16	(5,1)	16	ReLU	same
	35	Conv/BN	16x2x16	(5,1)	16	ReLU	same
OUT	36	Conv	16x2x256	(5,1)	16	ReLU	same
	37	Conv	16x1x3	(1,2)	3	Softmax	same
Trainable params: 1.100.387							

Tabella 9.1: Modello U-Net completo con profondità pari a 4 (BN = \textit{batch normalization})

	ID	Tipo di layer	Out dim	Kernel	Filtri	Activation	Pad
	1	Input	16x2x1	-	-	-	-
ENC	2	Conv/BN	16x2x16	(5,1)	16	ReLU	same
	3	Conv/BN*	16x2x16	(5,1)	16	ReLU	same
	4	Max Pool	8x2x16	(2,1)	-	-	valid
	5	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	6	Conv/BN*	8x2x32	(5,1)	32	ReLU	same
	7	Max Pool	4x2x32	(2,1)	-	-	valid
	14	Conv/BN	4x2x64	(5,1)	64	ReLU	same
	15	Conv/BN	4x2x64	(5,1)	64	ReLU	same
DEC	16	UpSampling	8x2x64	(2,1)	-	-	-
	17	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	18	Crop-Concat(6, 17)	-	-	-	-	-
	19	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	20	Conv/BN	8x2x32	(5,1)	32	ReLU	same
	21	UpSampling	16x2x32	(2,1)	-	-	-
	22	Conv/BN	16x2x16	(5,1)	16	ReLU	same
	23	Crop-Concat(3, 22)	-	-	-	-	-
	24	Conv/BN	16x2x16	(5,1)	16	ReLU	same
	25	Conv/BN	16x2x16	(5,1)	16	ReLU	same
OUT	36	Conv	16x2x64	(5,1)	16	ReLU	same
	37	Conv	16x1x3	(1,2)	3	Softmax	same
Trainable params: 67.939							

Tabella 9.2: Modello U-Net semplificato con profondità pari a 2 (BN = \textit{batch normalization})

ID	Tipo di layer	Out dim	Kernel	Filtri	Activation	Pad
1	Input	16x2	-	-	-	-
2	Conv/BN	16x32	3	32	ReLU	same
3	Conv/BN	16x32	3	32	ReLU	same
4	Max Pool	8x32	2	-	-	-
5	Dropout	8x32	0.25	-	-	-
6	Conv/BN	8x64	3	64	ReLU	same
7	Dropout	8x64	0.25	-	-	-
8	Conv/BN	8x128	3	128	-	-
9	Max Pool	4x128	2	-	-	-
10	Dropout	4x128	0.25	-	-	-
11	Flatten	512	-	-	-	-
12	Dense	512		512	ReLU	-
13	BN	512	-	-	-	-
14	Dropout	512	0.5	-	-	-
15	Dense	512	-	128	ReLU	-
16	BN	128	-	-	-	-
17	Dropout	128	0.5	-	-	-
18	Dense	3	-	3	ReLU	-
Trainable params: 364.739						

Tabella 9.3: Modello con singola uscita

10

Ringraziamenti

Sono stati 5 anni, 1825 giorni, 39 420 000 respiri di *UniBG*, tutti racchiusi in questo elaborato finale, e per cui è doveroso fare dei ringraziamenti.

Innanzitutto, un grazie al Prof. Angelo Gargantini, a Silvia e ad Andrea, per avermi seguito e supportato durante tutto il percorso di stesura di questo elaborato. *Grazie!*

Non posso che andare poi a ringraziare i miei genitori per avermi permesso di portare a termine i miei studi, non facendomi mai mancare nulla. *Grazie!*

Anche a tutti i miei Amici, quelli che sono stati, e saranno, compagni di mille avventure in grado di rendere più movimentati questi 5 anni. *Grazie!*

Un ringraziamento anche a Fabio, Daniele, Stefano e Daniele, amici prima che colleghi, per avermi accolto nel loro team e per avermi fatto crescere come persona prima ancora che come Ingegnere. *Grazie!*

Grazie anche a tutti i miei compagni di squadra, con cui ho avuto modo di condividere tanti momenti felici, dentro e fuori dal campo di pallavolo. *Grazie!*

Michele

Bibliografia

Report

[4] C. Galbiati et al. *Mechanical Ventilator Milano (MVM): A Novel Mechanical Ventilator Designed for Mass Production in Response to the COVID-19 Pandemic*. Rapp. tecn.

[7] Cristiano Galbiati. *Cristiano Galbiati - Princeton/INFN/GSSI*. Rapp. tecn. 2020. URL: https://www.youtube.com/watch?v=yd6pLr5was8.

[21] Oscar Putignano. *Oscar Putignano - UniMIB*. Rapp. tecn. 2020. URL: https://www.youtube.com/channel/UC-CibX7cUZ6Tf_p5OAD1ZoQ/videos?view=0&sort=dd&shelf_id=1.

Articoli

[2] Bäuerle Alex e Ropinski Timo. "Net2Vis: Transforming Deep Convolutional Networks into Publication-Ready Visualizations". In: *arXiv preprint arXiv:1902.04394* (2019).

[5] Lluìs Blanch et al. "Asynchronies during mechanical ventilation are associated with mortality". In: *Intensive Care Medicine 41.4* (2015), pp. 633–641.

[6] Perselev et al. "U-Time: A Fully Convolutional Network for Time Series Segmentation Applied to Sleep Staging". In: *arXiv* (2019). DOI: arXiv:1910.11162.

[13] Johnson AEW Pollard TJ Shen L Lehman L Feng M Ghassemi M Moody B Szolovits P Celi LA e Mark RG. "MIMIC-III, a freely accessible critical care database." In: *Scientific reports* 1.1 (2016), p. 1. DOI: 10.1038/sdata.2016.35.

[17] R. W. Manley. "A new mechanical ventilator". In: *Association of anaesthetists* (1961). DOI: https://doi.org/10.1111/j.1365-2044.1961.tb13830.x.

[18] T.H.G.F Bakkes R.J.H. Montree1 M. Mischi F. Mojoli e S. Turco. "A machine learning method for automatic detection and classification of patient-ventilator asynchrony". In: *Scientific reports* 1.1 (2020), p. 4.

[22] Adams Jason Y Lieng Monica K Kuhn Brooks T Rehm Greg B Guo Edward C Taylor Sandra L Delplanque Jean-Pierre Anderson Nicholas R. "Development and validation of a multi-algorithm analytic platform to detect off-target mechanical ventilation". In: *Scientific reports* 7.1 (2017), p. 14980.

[23] O. Ronneberger, P.Fischer e T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: LNCS 9351 (2015). (available on arXiv:1505.04597 [cs.CV]), pp. 234–241. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[27] Rehm Gregory B Kuhn Brooks T Nguyen Jimmy Anderson Nicholas R Chuah Chen Nee Adams Jason Yeates. "Improving Mechanical Ventilator Clinical Decision Support Systems with a Machine Learning Classifier for Determining Ventilator Mode." In: *Studies in health technology and informatics* 264 (2019), pp. 318–322.

Libri

[3] Roberto Cosentini Steano Aliberti Anna Maria Brambilla. *ABC della ventilazione meccanica non invasiva in urgenza*. 2ª ed. Milano MI: McGrawHill, 2010.

Siti web

[1] *A Numerical Second Derivative from Three Points*. 2019. URL: Availableat:\url{https://mathformeremortals.wordpress.com/2013/01/12/a-numerical-second-derivative-from-three-points/}(20/02/2021).

[8] Harsha Goonewardana. *Evaluating Multi-Class Classifiers*. 2019. URL: Availableat:\url{https://medium.com/apprentice-journal/evaluating-multi-class-classifiers-12b2946e755b#:~:text=Cohen's%20Kappa,the%20actual%20and%20predicted%20classes}(10/01/2021).

[9] Harsha Goonewardana. *Evaluating Multi-Class Classifiers*. 2019. URL: Availableat:\url{https://medium.com/apprentice-journal/evaluating-multi-class-classifiers-12b2946e755b#:~:text=Cohens%20Kappa,the%20actual%20and%20predicted%20classes}(14/03/2021).

[10] Divam Gupta. *A Beginner guide to Deep Learning based Semantic Segmentation using Keras*. 2019. URL: Availableat:\url{https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html}(21/11/2020).

[11] hahncity. *ventMAP - Open Source Multi-Purpose Ventilator Analytics Library*. Available at: https://github.com/hahnicity/ventMAP (08/03/2021).

[12] Arun Kumar. *Semantic Image Segmentation using Fully Convolutional Networks*. 2020. URL: Availableat : \url{https : //towardsdatascience . com/ semantic - image - segmentation - using - fully - convolutional - networks - bf0189fa3eb8}(14/03/2021).

[14] Harshall Lamba. *Understanding Semantic Segmentation with UNET*. Available at: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47 (25/02/2021).

[15] Z Little. *Conv1D, Conv2D and Conv3D*. Available at: https : //xzz201920 . medium.com/conv1d-conv2d-and-conv3d-8a59182c4d6 (23/12/2020).

[16] Serafeim Loukas. *ROC Curve explained using a COVID-19 hypothetical example: Binary and Multi-Class Classification tutorial*. Available at: https : // towardsdatascience . com / roc - curve - explained - using - a - covid - 19 - hypothetical-example-binary-multi-class-classification-bab188ea869c (17/03/2021).

[19] MVM. *Home MVM*. 2020. URL: Availableat : \url{http : //mvm . care / it / home-it/}(11/03/2021).

[20] Marco Del Pra. *Time Series Classification with Deep Learning*. Available at: https : //towardsdatascience . com/time - series - classification - with - deep-learning-d238f0147d6f (15/12/2020).

[24] Chiara Vannin. *Le caratteristiche della ventilazione meccanica*. Available at: https://www.nurse24.it/studenti/risorse-studenti/le-caratteristiche-della-ventilazione-meccanica.html7 (15/02/2021).

[25] Wikipedia. *Respiratory rate*. 2021. URL: Availableat : \url{https : / / en . wikipedia.org/wiki/Respiratory_rate\#}(15/03/2021).

[26] wkentaro. *labelme*. Available at: https : //github . com / wkentaro / labelmeP (15/03/2021).