Politecnico di Torino

Politecnico di Torino

Laurea Magistrale in Ingegneria Meccanica
A.a. 2020/2021
Sessione di Laurea Luglio 2021

Sviluppo di un algoritmo di Unsupervised Learning per l'ottimizzazione del design di veicoli ibridi elettrici

Relatori:
Daniela Anna Misul
Claudio Maino
Alessandro Di Mauro
Alessandro Falai

Candidato:
Angelo Borneo

Politecnico
di Torino

Abstract

Negli ultimi anni si è assistito sempre più ad un interesse verso le tematiche ambientali, con un occhio particolare alle emissioni inquinanti. Per questo motivo, nell'ambito dell'industria automobilistica, lo sviluppo tecnologico è stato spinto in gran parte dalle politiche attuate per la riduzione delle emissioni di CO2 ed in quest'ottica una valida soluzione è l'utilizzo di veicoli ibridi elettrici (HEVs: Hybrid Electric Vehicles) che portano ad una riduzione di emissioni di CO2 TTW (Tank To Wheel), letteralmente dal serbatoio alla ruota. Un'accurata progettazione risulta fondamentale, ecco perché la scelta dei parametri di design quali la cilindrata del motore a combustione interna, piuttosto che la potenza della macchina elettrica o i rapporti di trasmissione risulta importante al punto da sviluppare strumenti in grado di ottimizzarli.
L'obiettivo di questo progetto è proprio quello di inserirsi in questo contesto di progettazione ottimizzata di veicoli ibridi, cercando di ottenere contemporaneamente risultati confrontabili con quelli ottenibili con algoritmi di tipo deterministico, come la Dynamic Programming che funge da benchmark per la valutazione delle prestazioni del tool sviluppato, ed una leggerezza del programma che richieda meno tempo e potenza di calcolo.
Le tecnologie utilizzate sono algoritmi di clustering, appartenenti alla categoria dell'unsupervised learning, e le Reti Neurali Profonde (Deep Learning) che simulano i meccanismi di apprendimento del cervello umano: varie connessioni tra diversi strati di neuroni che garantiscono il flusso di nozioni e la possibilità e capacità della rete di adattarsi e imparare. Dati i risultati ottenuti con una sola pipeline di reti neurali su tutti i dati a disposizione, per avvicinarsi maggiormente alle performance ottimali è stata studiata la distribuzione interna del dataset per verificare la presenza di una zona che potesse mettere in difficoltà la rete.
Si individuano così due zone con una diversa distribuzione dei dati ed effettivamente una delle due presenta prestazioni peggiori. Per questo motivo si crea una pipeline per ciascuna delle due zone. Queste pipeline lavorano in parallelo, entrambe allenate sull'intero dataset ma con una calibrazione degli iper-parametri diversa, e poi testate sulla rispettiva zona di dataset. Entrambe sono composte da due reti neurali: una per valutare se il veicolo sarà in grado di completare il ciclo di guida prescelto (classificatore di fattibilità), e l'altra che in caso positivo prevede le emissioni TTW di CO2 (regressore).
I risultati sono promettenti in quanto risultano migliori di quelli ottenuti con una sola pipeline.
Per questo progetto è stato utilizzato Spyder Anaconda che è un ambiente di sviluppo integrato open-source per la programmazione in linguaggio Python. Le librerie utilizzate sono Keras e Tensorflow.

Sommario

Indice delle figure

Indice delle tabelle

1 Introduzione

1.1 Motivazioni

Negli ultimi anni l'interesse nei confronti di argomenti quali preservare l'ambiente e ridurre l'inquinamento è in continua crescita. Oltre all'interesse dell'opinione pubblica, anche i governi nazionali e le istituzioni internazionali, con attenzioni in continua crescita, stabiliscono regolamentazioni e limiti sempre più stringenti [1].
Uno dei settori che è certamente tra i più coinvolti è quello automobilistico, per il quale i cambiamenti e le innovazioni sono sicuramente guidati in gran parte dalle restrizioni imposte, portando ad un aumento delle soluzioni sostenibili ("green"). Infatti i veicoli dotati di motore a combustione interna (ICE) emettono inquinanti (CO_2, NOx, CO e idrocarburi incombusti) per produrre la potenza necessaria, a causa della combustione dei derivati del petrolio, che è, come ben noto, una fonte di energia non rinnovabile.
Queste sono le principali motivazioni che spiegano la necessità di investire in sviluppo e ricerca di nuove tecnologie, come ad esempio i veicoli elettrici a batteria(BEV: Battery Electric Vehicles), i veicoli ibridi elettrici (HEV: Hybrid Electric Vehicles) o ancora i veicoli elettrici a celle di combustibile. Ecco perché una delle tecnologie più promettenti è quella della trazione elettrica [2].

1.2 Soluzioni tecniche

Nei veicoli elettrici a batteria l'unica fonte di energia è appunto il pacco batteria, e la capacità di trazione è ottenuta quindi tramite una macchina elettrica. Il principale vantaggio di questa soluzione è l'abbattimento delle emissioni di CO_2 in fase di utilizzo. Infatti le emissioni "tank to wheel" (letteralmente dal serbatoio alle ruote) sono totalmente annullate, e qualora le fonti di energia utilizzate nel processo produttivo fossero rinnovabili le emissioni totali (well to wheel) potrebbero veramente essere notevolmente ridotte. I principali svantaggi che questa soluzione porta con se sono il peso e la gestione dell'ingombro del pacco batteria, così come il basso range di autonomia. Quest'ultimo problema assume anche un peso maggiore a cause dei lunghi tempi di ricarica necessari, ed emerge soprattutto per viaggi lunghi e quindi extraurbani. Notevole è il miglioramento a cui si assiste negli ultimi anni proprio nell'ottica dell'incremento dell'autonomia e della velocità di ricarica, anche se, soprattutto per quest'ultimo problema, da risolvere non solo tramite lo sviluppo di soluzioni per aumentare la velocità di carica, ma anche diffondendo sul territorio le infrastrutture per il rifornimento, le differenze tra regioni e nazioni sono ancora importanti.
Per quanto riguarda i veicoli ibridi invece le fonti di energia sono sia la batteria che il combustibile per il motore a combustione interna. Il sistema è più complicato dovendo integrare due tecnologie, ma questa soluzione permette di aumentare il range di autonomia rispetto ai veicoli a batteria, e di ridurre le emissioni di CO_2 e il consumo di carburante rispetto ai veicoli con motore a combustione interna convenzionali [3].

1.3 Architetture dei veicoli ibridi

Una prima distinzione nel campo dei veicoli ibridi consiste nella possibilità di caricare la batteria del veicolo collegandola alla rete elettrica (plug-in HEV) o meno (la batteria viene ricaricata tramite il motore a combustione interna)[4]. Le differenze principali sono che i PHEVs hanno batterie più grandi degli altri, un range di guida in modalità puramente elettrica maggiore e quindi possono ridurre ancora di più le emissioni. Un'altra differenza è la strategia di controllo, infatti mentre per i veicoli plug-in si porta la batteria al termine della carica per poi collegarla per la ricarica, per i veicoli non plug-in si cerca di mantenere lo stato di carica della batteria ad un livello medio-alto. Uno dei problemi principali dei PHEVs è il costo dovuto alle batterie, che potrebbe non essere del tutto ammortizzato dal risparmio di costo del combustibile.
I veicoli ibridi possono essere ulteriormente classificati in base all'architettura: in serie, in parallelo e una combinazione delle due, e la scelta di una di queste architetture dipende dalla missione di guida e da una valutazione dei costi e delle performance [4].
La peculiarità dell'architettura in serie è che le ruote sono guidate solamente dal motore elettrico, che deve quindi essere in grado di fornire tutta la potenza richiesta. L'ibridizzazione avviene al livello della fonte di energia: il motore a combustione interna è collegato al motore elettrico, con la funzione di produrre energia elettrica per caricare la batteria o per alimentare il motore elettrico. Inoltre l'ICE non può operare contemporaneamente in alimentazione e carica.
L'architettura in parallelo, che è molto usata, è caratterizzata dalla possibilità di fornire potenza alle ruote motrici sia dalla macchina elettrica, la quale può essere installata sia sull'assale anteriore che posteriore, che dal motore a combustione interna allo stesso tempo. La configurazione in parallelo include due linee di potenza separate che convergono in un dispositivo di accoppiamento che può accoppiare o le coppie o le velocità dei due percorsi, il che complica la strategia di controllo rispetto a quella dell'architettura in serie. Quando il motore elettrico non è sufficientemente grande non può esistere una modalità di trazione puramente elettrica, presente invece quando il motore elettrico è sufficientemente grande, così che sia il motore elettrico che il termico possono guidare il veicolo sia insieme che separatamente. Il ruolo della macchina elettrica è quello di assistere quella termica, permettendole di lavorare in condizioni di alta efficienza e di recuperare energia attraverso la frenata rigenerativa. La strategia di controllo punta a ridurre il consumo di carburante e le emissioni inquinanti.
L'architettura complessa (serie e parallelo) punta a combinare i vantaggi delle due architetture e ad evitare gli svantaggi.
Un'ulteriore classificazione dei veicoli ibridi è possibile sulla base della posizione del motore elettrico:

- P0, la macchina elettrica è collegata al motore termico tramite una cinghia sull'azionamento accessorio;
- P1, macchina elettrica collegata all'albero motore dell'ICE;
- P2, macchina elettrica posizionata tra l'ICE e la trasmissione;
- P3, macchina elettrica posizionata tra la trasmissione e il differenziale;
- P4, macchina elettrica posizionata sull'assale posteriore, mentre l'ICE è collegato con l'assale anteriore.

1.4 Logiche di controllo

Al fine di far decidere all'unità di controllo che scelta effettuare in un particolare momento, esistono diverse tipologie di strategie di controllo [3]. Con l'obbiettivo di sfruttare al meglio le potenzialità di un veicolo ibrido, è fondamentale una buona strategia di controllo che assicuri le migliori performance in termini di consumo di carburante e/o di emissioni inquinanti. In generale le logiche di controllo prendono come input i parametri disponibili sullo stato del veicolo (ad esempio la velocità, la pendenza della strada, la richiesta di potenza dal guidatore) e restituisce in output le decisioni, riguardanti ad esempio la marcia da selezionare, la divisione di potenza tra macchina termica ed elettrica, la necessità di lavorare in modalità di ricarica della batteria ecc. Tre categorie di logiche di controllo sono state sviluppate: metodi di ottimizzazione globale, metodi di ottimizzazione statici e metodi euristici [3]. Questi ultimi di solito forniscono una soluzione approssimata, basata sull' occorrenza di eventi, ma sono poco dispendiose dal punto di vista computazionale. Alcuni esempi sono: "Fuzzy Logics", Reti Neurali e strategie basate sull'individuazione di una regola. Il concetto degli ottimizzatori statici è invece quello di un'ottimizzazione istantanea, senza considerare l'intera missione, e per questo motivo non assicurano la soluzione ottimale.
Entrambi i tipi di logiche di controllo garantiscono però una leggerezza dal punto di vista computazionale che si traduce in velocità di calcolo della strategia, rendendole implementabili a bordo.
Infine gli ottimizzatori globali sono quelli che forniscono il miglior risultato e gli unici che raggiungono effettivamente l'ottimo. Questi sono ad esempio *Algoritmi Genetici* e *Dynamic Programming* la quale è di fondamentale importanza per questo progetto in quanto ne rappresenta il punto di partenza. Gli *Algoritmi Genetici* partono da una generica popolazione, costituita da diverse combinazioni di parametri controllati, e arrivano ad isolare un singolo individuo che rappresenta la soluzione ottimale. La *Dynamic Programming* si basa su un processo decisionale ottenuto tramite una logica di propagazione *backward-forward*. Per descrivere tutti i possibili stati del sistema (e il percorso che si ottiene collegandoli) viene implementata una griglia di controllo. L'obiettivo è quello di passare dallo stato iniziale a quello finale seguendo il percorso più breve, quindi l'ottimizzazione risiede nella scelta degli stati intermedi da attraversare.
Inoltre gli ottimizzatori globali vengono usati come benchmark per gli altri due tipi menzionati.

1.5 Contributo e strumenti di questo progetto

Questo progetto è la continuazione di un progetto esistente [5], [6], il cui obiettivo complessivo è quello di fornire uno strumento utile per la fase di progettazione dei veicoli ibridi. Il progetto di partenza è costituito da tre reti neurali in serie: la prima ad essere stata sviluppata è un regressore, in grado di prevedere le emissioni di CO2 di un insieme di dati composto da layout "feasible" e cioè capaci di portare a termine il ciclo guida. In seguito è stata sviluppata una seconda rete da inserire a monte rispetto alla prima, permettendo la classificazione tra i layout feasible, da far passare alla rete successiva, e quelli unfeasible, da scartare. Infine una terza rete è stata interposta tra le due. Si tratta di un ulteriore classificatore che funge da ulteriore filtro, se desiderato, per passare al regressore i soli layout feasible che rispettino dei vincoli prestazionali (classificatore di ammissibilità).
Il principale contributo di questo progetto è stato quello di fornire dei mezzi per effettuare uno studio a priori del dataset da dare in pasto alla pipeline di reti, con l'obiettivo di verificare l'eventuale presenza di una zona che potesse mettere in difficoltà la pipeline, ed in caso ottimizzare l'architettura della stessa o modificarla per migliorare le prestazioni del tool.
Infatti non studiando a priori la composizione dei dati e fornendoli direttamente alla pipeline di reti neurali è impossibile stabilire in che parte le prestazioni siano influenzate dallo strumento stesso o dai dati forniti. Risulta quindi importante inserire uno step di pre-processing prima di lanciare gli step di apprendimento della rete.
Il tool esistente usa algoritmi di Deep Learning per raggiungere i propri obiettivi, per questo motivo e per mantenere la compatibilità tra i progetti, anche in questo progetto saranno usati gli stessi mezzi. Inoltre anche algoritmi di Clustering sono stati utilizzati per il conseguimento degli obiettivi. Nella maggior parte dei casi le tecniche di Deep Learning fanno riferimento a problemi di tipo Supervised, e cioè per i quali sono note le soluzioni e hanno scopi di tipo predittivo (classificazione o regressione). In generale esistono algoritmi di Supervised ed Unsupervised Learning e per la seconda categoria, a differenza di quanto appena detto per la prima, le soluzioni non sono date ed è l'algoritmo stesso che permette di trovare relazioni tra i dati. Le applicazioni di questi algoritmi hanno invece uno scopo descrittivo e vengono principalmente usati per il clustering e per la rilevazione di anomalie.
L'apprendimento della rete si basa sui risultati ottenuti con la Dynamic Programming, sicuramente più accurati e affidabili ma allo stesso tempo più dispendiosi in termini di tempo e potenza di calcolo necessaria, al contrario di un algoritmo di Deep Learning che se ben ottimizzato è in grado di fornire risultati abbastanza buoni ma in molto meno tempo.
La trattazione è organizzata in questo modo: una prima parte è dedicata alle tecniche utilizzate nel presente e nel precedente progetto, così da avere una base teorica per la trattazione e la presentazione dei risultati. Segue una parte di descrizione del dataset e del modello, per passare poi allo studio del dataset e delle analisi che scaturiscono e i relativi risultati.
Per validare l'utilità di questo step senza allungare troppo i tempi di simulazione necessari sono state utilizzate le sole reti di classificazione di fattibilità e quella di regressione, "spegnendo" il filtro di ammissibilità.

2 Teoria Deep Learning - Reti Neurali

2.1 Perché Deep Learning?

Il Deep Learning è un sottogruppo del Machine Learning, che a sua volta costituisce una sottoclasse della più generale Intelligenza Artificiale[7]. La principale differenza tra il Deep Learning e il Machine Learning in generale è la maggiore complessità della struttura che caratterizza il Deep Learning e la sua elevata capacità di apprendere da un grande numero di dati.

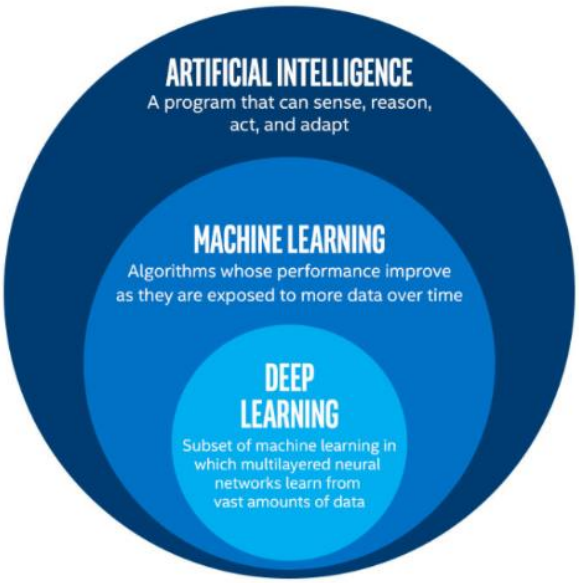


Figura 1: Gerarchia dell'Intelligenza Artificiale, Machine Learning, Deep Learning

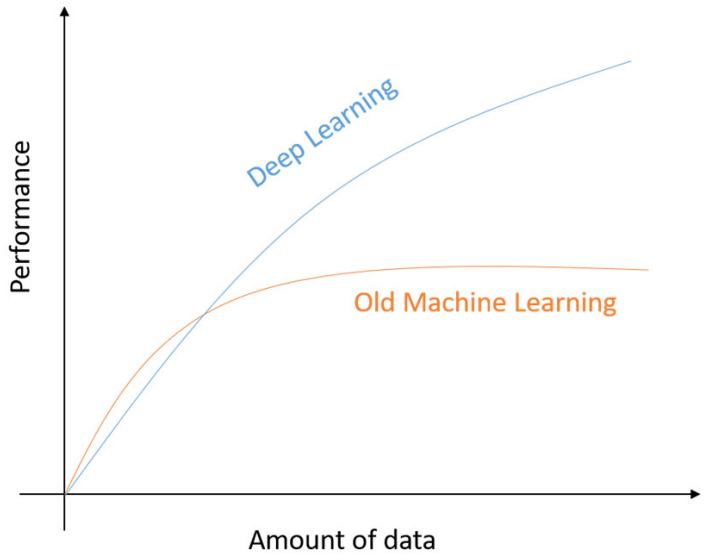


Figura 2:Performance all'aumentare del numero di dati

La comparsa di questa tecnologia risale al 1958, anno in cui Frank Rosenblatt presentò "Perceptron", un algoritmo che, dopo aver imparato le principali caratteristiche di uomini e donne fosse in grado di classificare un insieme di persone nelle due categorie di uomini e donne appunto. A causa delle limitazioni hardware l'algoritmo non ebbe successo, ma dal 1998, anno in cui si assiste alla comparsa della prima rete neurale convoluzionale (alla base di tutti gli attuali algoritmi di riconoscimento delle immagini), il successo e l'ascesa della tecnologia sono stati rapidissimi.
Al giorno d'oggi la presenza di intelligenza artificiale nelle aziende sfocia in ogni settore e crescono di anno in anno gli investimenti, le pubblicazioni scientifiche, gli incontri (ecc) riguardanti l'intelligenza artificiale.

2.2 Reti Neurali

Il Deep Learning, basato sull'apprendimento tramite reti neurali artificiali, simula il cervello umano caratterizzando l'apprendimento secondo connessioni tra diversi strati, detti *layer*, con un apprendimento sempre più profondo che porta ad estrarre dai dati delle caratteristiche ("features") di livello via via superiore. I suddetti *layer* sono costituiti da unità o nodi, ciascuno dei quali simula il funzionamento di un neurone, ricevendo uno o più input e, a seguito di un'elaborazione, fornendo un output.
Il neurone è composto da un nucleo e dai prolungamenti cellulari, i dendriti che prendono il segnale in ingresso e l'assone che trasporta l'output:

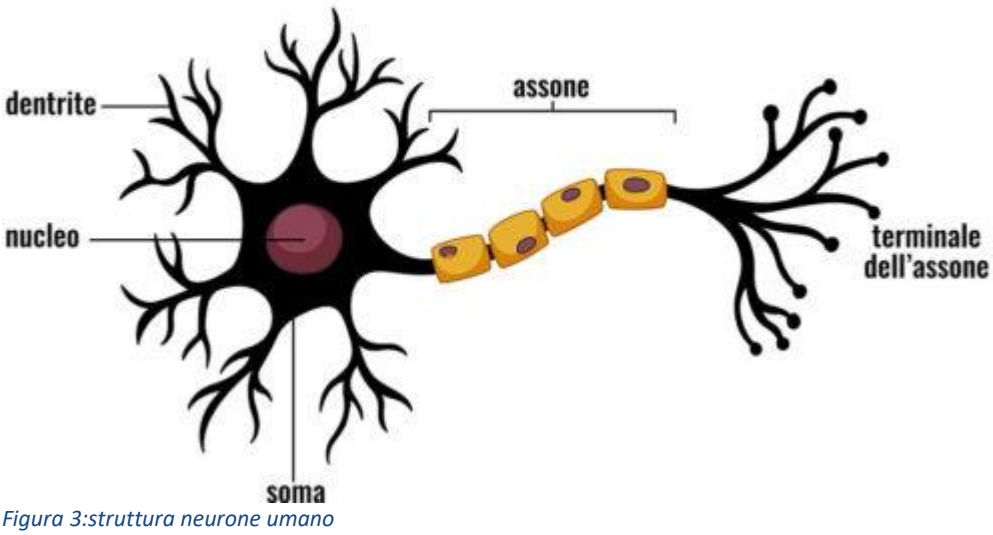


Figura 3:struttura neurone umano

Per un neurone artificiale lo schema è lo stesso: input e output svolgono rispettivamente le funzioni di dendrite e assone per l'unità di attivazione, che è l'analogo del nucleo:

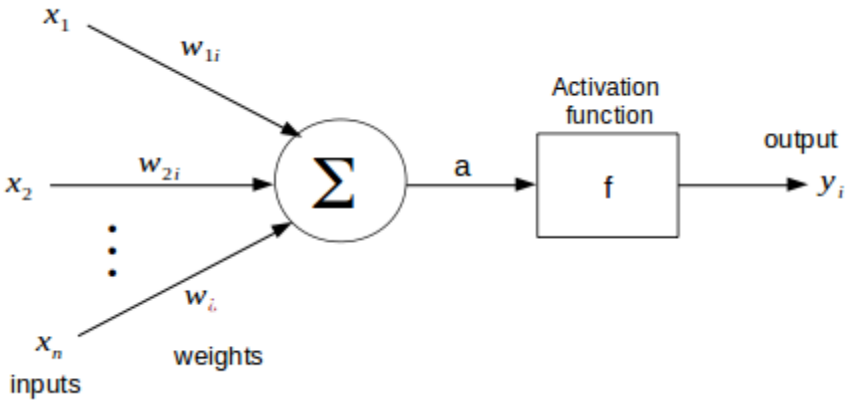


Figura 4: Struttura neurone artificiale

Facendo riferimento a quanto rappresentato in figura gli input sono le x che vengono moltiplicate per i pesi che sono i parametri della rete (indicati con le w) e, una volta sommati, si applica la funzione di attivazione per ottenere l'output.
Quanto appena descritto però rappresenta come, a partire dall'input, la rete e quindi ogni layer ed ogni unità, fornisce il proprio output, fase nota come "Forward Propagation". La fase di apprendimento invece avviene grazie alla "Backward Propagation" durante la quale i parametri e quindi la struttura della rete sono in grado di aggiornarsi attraverso quello che viene definito come allenamento. Per questa fase di allenamento non vengono usati tutti i dati a disposizione: questi vengono divisi in ulteriori dataset in quanto oltre all'allenamento sono fondamentali altri due step che sono la validazione e il testing su esempi che la rete non ha visto prima, ma dei quali devono essere note le etichette per valutare le prestazioni. Per la formazione dei dataset di validazione e di test è necessaria una scelta accurata del numero di dati da destinare a ciascun dataset. Quando i dati a disposizione non sono tantissimi la frazione di dati da usare in fase di test non è inferiore allo 0.1, quindi il restante 90% dei dati viene usato per la fase di apprendimento e validazione. Quando il numero di dati è elevatissimo si può arrivare ad utilizzare anche il 99% dei dati per la fase di allenamento e solo l'1% per il test.
Le strutture della rete neurale profonda possono essere molto diverse tra loro in base al loro scopo e ai dati a disposizione [8]. Tuttavia mostrano sempre alcune caratteristiche comuni. Per lo scopo di questo progetto, è sufficiente descrivere il flusso di lavoro di base dei modelli predittivi. Questi hanno bisogno di una fase di allenamento su un dataset che contiene anche le labels esatte, cioè le soluzioni, degli esempi. Le due fasi nominate in precedenza, e cioè la propagazione in avanti e indietro, avvengono proprio nella fase di allenamento.
Come detto, una rete neurale è costituita da più layers con tanti neuroni ciascuno. La soluzione comunemente adottata è del tipo MIMO (Multi Input Multi Output) e cioè un layout totalmente connesso, caratterizzato dalle seguenti parti:

- input layer, è il primo dei layer della rete;
- output layer, è l'ultimo layer e, in base alla funzione della rete può avere una o più unità;
- hidden layers, sono tutti i layer compresi tra il primo e l'ultimo.

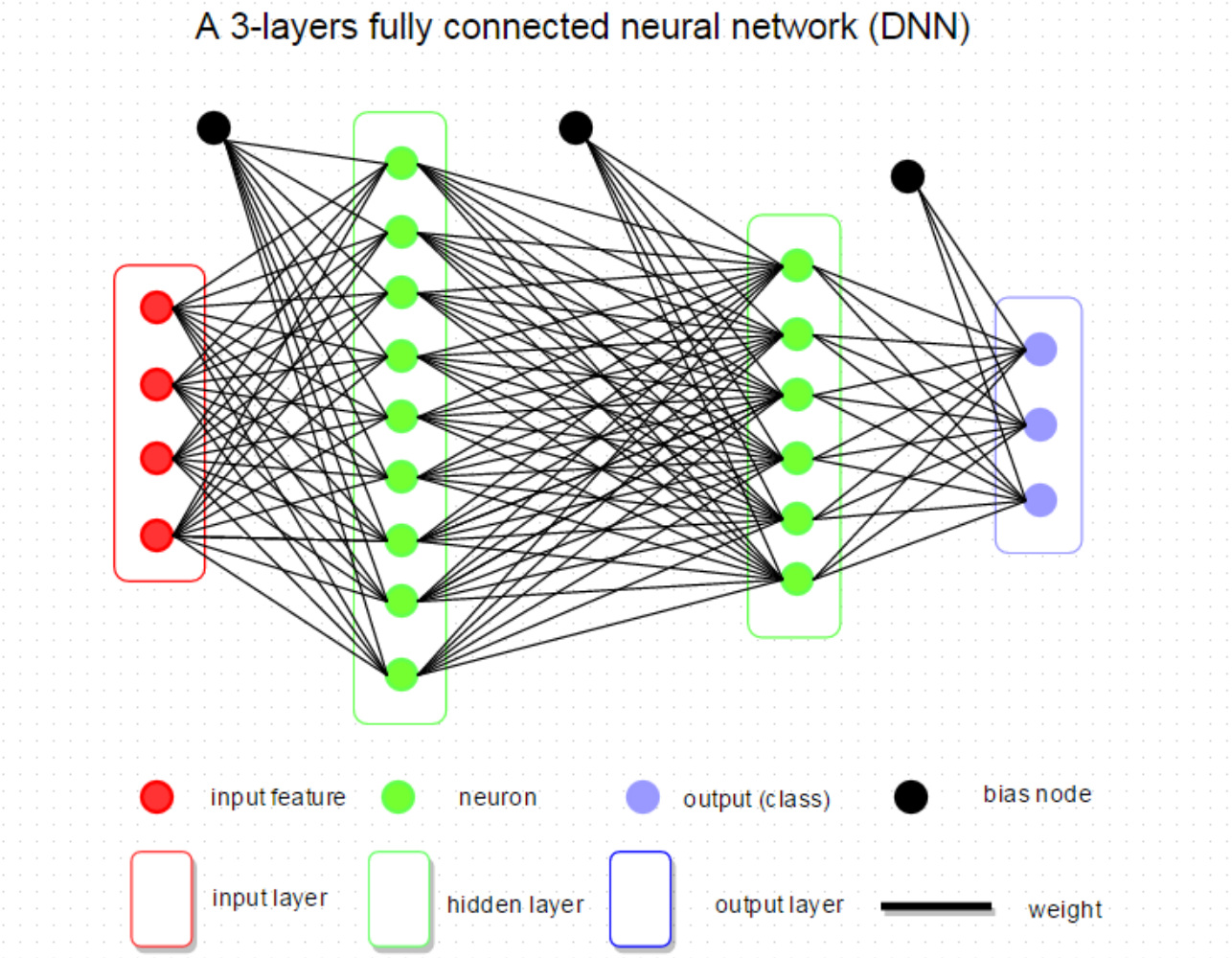
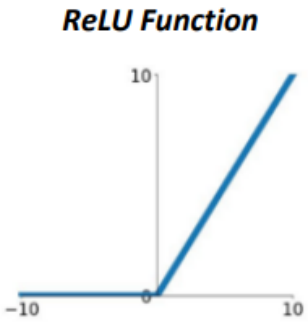


Figura 5: Struttura Rete Neurale

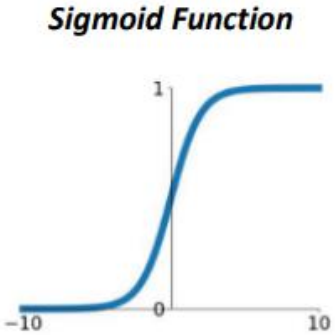
Come si può ben vedere in figura in ogni layer oltre alle unità appena descritte sono presenti delle unità dette di bias, che non sono come le altre unità in quanto non ricevono un input al quale applicano una funzione di attivazione per fornire un output, ma vanno a costituire una componente dell'input delle altre unità. I bias e i pesi sono i parametri della rete.
Come accennato in precedenza per ottenere l'output vengono applicate delle funzioni di attivazione. Ne esistono diverse, la funzione sigmoidea e la tangente iperbolica sono caratterizzate da un'immagine limitata: (0 , 1) la prima (-1 , 1) la seconda, il che mostra la potenzialità di queste funzioni qualora impiegate come funzioni di attivazione dell'output layer di una rete di classificazione, rendendo agevole la determinazione dell'appartenenza ad una classe piuttosto che ad un'altra. I problemi di queste funzioni riguardano invece la capacità di dare output diversi a fronte di input diversi ma ad esempio molto elevati o molto piccoli, inibendo l'apprendimento se usate nei layer interni della rete. Per gli hidden layer sono infatti sempre più usate funzioni di attivazione a rampa, ma non con una sola pendenza in tutto il range di applicazione.
Quella usata in questo progetto è la funzione ReLU[9], che sta per Rectified Linear Unit, la cui formulazione è:



$$\sigma(x) = \begin{cases} 0 & for\ x < 0 \\ x & for\ x > 0 \end{cases}$$

Figura 6: Rectified Linear Unit

Per l'output layer dei classificatori la funzione utilizzata è la funzione sigmoidea:



$$\sigma(x) = \frac{1}{1 + e^{x}}$$

Figura 7: Funzione Sigmoidea

2.3 Propagazione in avanti

Come accennato in precedenza la fase di propagazione in avanti è quella che permette di ottenere l'output della rete. Dopo aver descritto brevemente il flusso logico si procede in questo paragrafo alla descrizione analitica:

$$x_k = \sum_{j=1}^{m} \left(w_{k,j} \cdot z_j\right) + b_k$$

$$z_k = g(x_k)$$

Come già detto in precedenza le *x* rappresentano gli input (del *k*-esimo nodo del layer *i*-esimo che potrebbe essere indicato come secondo pedice in una notazione ancor più dettagliata); le *w* indicano i pesi che correlano il nodo *k* del layer *i* con il nodo *j* del layer precedente *(i-1)* il cui numero di nodi è indicato con la lettera *m*; il termine b_k rappresenta il bias; con le *z* si indicano gli output dei nodi, in particolare nella formula dell'input z_j è l'output del *j*-esimo nodo del layer *i-1*, invece z_k è l'output del *k*-esimo nodo del layer *i*; infine la $g(\,)$ indica la funzione di attivazione. Scrivendo queste equazioni per tutti i nodi di un layer ciò che si ottiene è un sistema di equazioni che può essere ridotto in forma matriciale:

$$\begin{bmatrix} x_{1,i} \\ x_{2,i} \\ . \\ . \\ . \\ x_{n,i} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & & w_{1,m} \\ w_{2,1} & w_{2,2} & & w_{2,m} \\ . & . & & . \\ . & . & \dots & . \\ . & . & & . \\ w_{n,1} & w_{n,2} & & w_{n,m} \end{bmatrix} * \begin{bmatrix} z_{1,i-1} \\ z_{2,i-1} \\ . \\ . \\ . \\ z_{m,i-1} \end{bmatrix} + \begin{bmatrix} b_{1,i} \\ b_{2,i} \\ . \\ . \\ . \\ b_{n,i} \end{bmatrix}$$

Che scritta con la notazione vettoriale diventa:

$$\bar{x}_i = \left(\boldsymbol{w} \cdot \bar{z}_{i-1} + \bar{b}\right)$$

Ed applicando la funzione di attivazione si ottiene:

$$\bar{z}_i = g\left(\boldsymbol{w} \cdot \bar{z}_{i-1} + \bar{b}\right)$$

Come si può notare dall'espressione matriciale estesa la matrice dei pesi $\boldsymbol{w}$ è una matrice di dimensione n*m, dove *n* è il numero di unità del layer *i* ed *m* è il numero di unità del layer *i-1*, che è quindi la dimensione del vettore $\bar{z}_{i-1}$.
L'importanza di un'espressione vettoriale non è la sola comodità di scrittura, ma è un modo per aumentare l'efficienza del codice, riducendo notevolmente i tempi di calcolo.

2.4 Propagazione all'indietro

Questa è la fase che permette alla rete di apprendere, di modificare i propri parametri al fine di ottenere un output che sia il più vicino possibile alla soluzione. Come il nome stesso dice, in questa fase si parte dalla fine per arrivare all'inizio. In particolare si procede calcolando l'errore tra la soluzione effettiva e la previsione della rete, ed è per questo motivo che la fase di propagazione in avanti deve precedere quella all'indietro, con un'inizializzazione random dei parametri. L'errore in questione tra soluzione effettiva e previsione è più in generale la funzione di costo (cost function) o una loss function, in particolare quest'ultima è l'errore calcolato per ciascun esempio, invece la cost function combina in un solo valore le loss function di tutti gli esempi utilizzati in un singolo passaggio. A seconda dell'obiettivo della rete esistono diverse loss functions.
In generale le espressioni per queste funzioni possono essere:

$$Loss = L(\hat{y}_i\,, y_i)$$

$$Cost = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}_i\,, y_i)$$

Dove $\hat{y}_i$ rappresenta la previsione e y_i la soluzione reale.
L'obiettivo è quello di minimizzare questa cost function andando a trovare una combinazione ottimale di tutti i parametri. Per questo si può esprimere questa funzione come funzione dei parametri:

$$Cost = J(w\,, b)$$

Per il raggiungimento del minimo di questa funzione esistono diverse procedure, la maggior parte delle quali si basano sul concetto di "gradient descent"[10]. La direzione nella quale muoversi per minimizzare l'errore è scelta calcolando il gradiente della funzione stessa: $\frac{\partial}{\partial w_{i,j}^{l}} J(w)$.

Una volta calcolato il gradiente questo viene usato per aggiornare il valore dei parametri:

$$\begin{cases} w_{i+1} = w_i - \alpha \dfrac{\partial J(w,b)}{\partial w} \\ b_{i+1} = b_i - \alpha \dfrac{\partial J(w,b)}{\partial b} \end{cases}$$

È importante sottolineare che il parametro α è fondamentale per garantire un buon apprendimento. Esso infatti, noto come *Learning Rate* (rapporto di apprendimento appunto), stabilisce la grandezza del passo da compiere nella direzione individuata dal gradiente.
La procedura si ripete iterativamente andando ad aggiornare i parametri muovendosi nella direzione indicata.

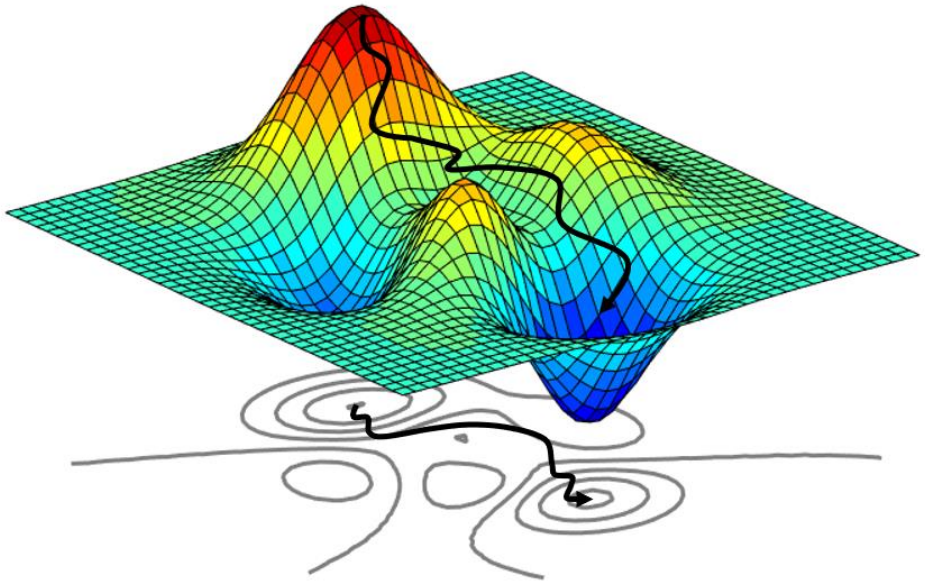


Figura 8: Convergenza Gradient Descent

2.5 Validazione

Uno step importante da affiancare alla fase di training e che preceda quella di testing è la fase di validazione. È un modo per valutare la rete dopo ogni epoca monitorandone le prestazioni su una serie di esempi su cui la rete non è addestrata [11]. Questa pratica consente all'operatore di vedere se l'apprendimento procede nel modo giusto. Anche il dataset di validazione è composto sia dagli esempi che dalle etichette, ma per la fase di validazione non è prevista alcuna fase di apprendimento e quindi di backward propagation.

2.6 Loss functions

Come accennato in precedenza esistono diverse funzioni in base all'obiettivo finale [12]. Quelle presenti in questo progetto sono due, una per il classificatore ed una per il regressore.
Per il classificatore si usa la binary cross entropy loss, che è molto usata nei problemi di classificazione binaria. La formulazione matematica è la seguente:

$$J = -\frac{1}{N} \cdot \sum_{i=1}^{N} y_i \, log \, \hat{y}_i + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

Dove y_i sono le etichette reali degli esempi e sono contenute nel dataset. I valori sono discreti: 0 o 1. Invece le $\hat{y}_i$ sono le previsioni della rete che assumono valori continui compresi tra 0 e 1. Grazie al termine che moltiplica ciascun logaritmo solo una delle due parti della somma sarà attivata, infatti con y_i = 1 sarà attiva solo la prima e con $y_i = 0$ solo la seconda.
Per il regressore invece la funzione di costo utilizzata è quella della radice dell'errore quadratico medio, la cui espressione è:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

Dove i significati dei simboli sono gli stessi del caso precedente.

2.7 Parametri ed Iper-parametri

Come detto in precedenza l'obiettivo principale è di minimizzare la funzione di costo. A tal proposito per trovare la migliore combinazione dei pesi o parametri è fondamentale trovare la combinazione migliore degli altri parametri fondamentali di una rete neurale, quali il rapporto di apprendimento, il numero di layers, il numero di unità che compongono i layer stessi, ecc che vengono chiamati iper-parametri. La scelta di questi avviene all'interno di un iperspazio tramite una strategia di ottimizzazione. Nel progetto esistente, e quindi per coerenza nel presente, gli iper-parametri sono i seguenti:

- Learning rate: come già detto è fondamentale per la dimensione del passo da fare nella fase di aggiornamento dei pesi. Se troppo grande la soluzione può non convergere al minimo o addirittura divergere, al contrario se troppo piccolo la convergenza potrebbe essere troppo lenta.

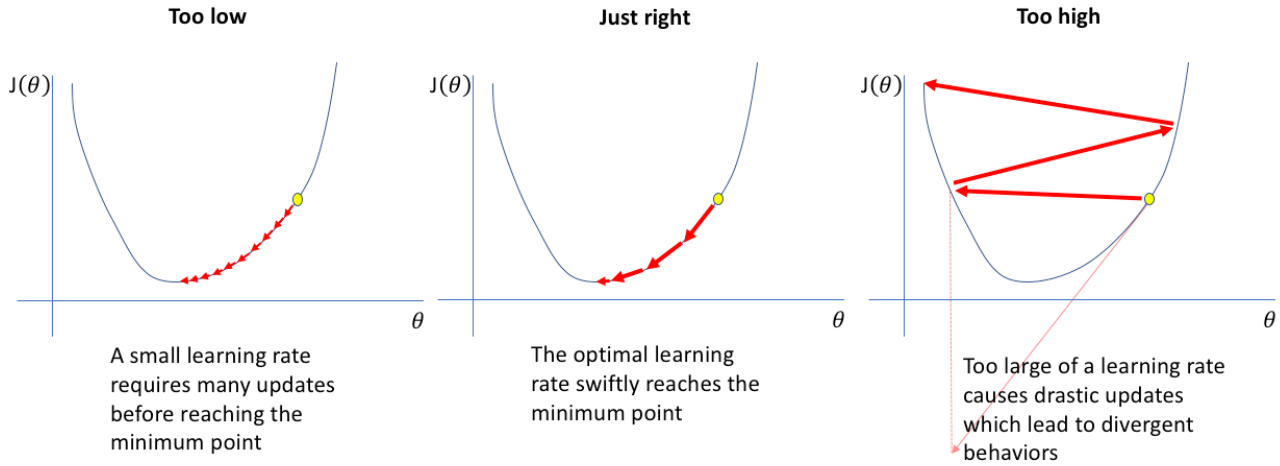


Figura 9: Effetto del tasso di apprendimento

- Numero di layer e di unità negli stessi: questi costituiscono la struttura stessa della rete. La complessità del modello da rappresentare è ciò che determina quanti layer e unità saranno necessari. All'aumentare della profondità della rete aumentano anche le prestazioni, incrementando chiaramente il tempo di calcolo.
- Dropout: serve per contrastare il problema dell'overfitting. In generale esistono due tipi di problemi che sono l'overfitting e l'underfitting. Il primo consiste in un fitting eccessivo dei dati di training che non permette poi una generalizzazione sufficiente a garantire delle performance valide in fase di test, il secondo è il problema opposto, infatti i dati di allenamento non vengono fittati a sufficienza, introducendo errori elevati sia in training che in validation. Tra i due l'underfitting è meno comune e più facile da risolvere (aumentare la complessità della rete o il tempo di allenamento), l'overfitting invece richiede un intervento apposito. Uno dei possibili interventi è appunto il dropout, che permette di disattivare con una certa probabilità le unità dei layer. Le unità disattivate variano in maniera casuale ad ogni iterazione, così da cercare di spingere la rete a generalizzare il più possibile le informazioni presenti nei dati.

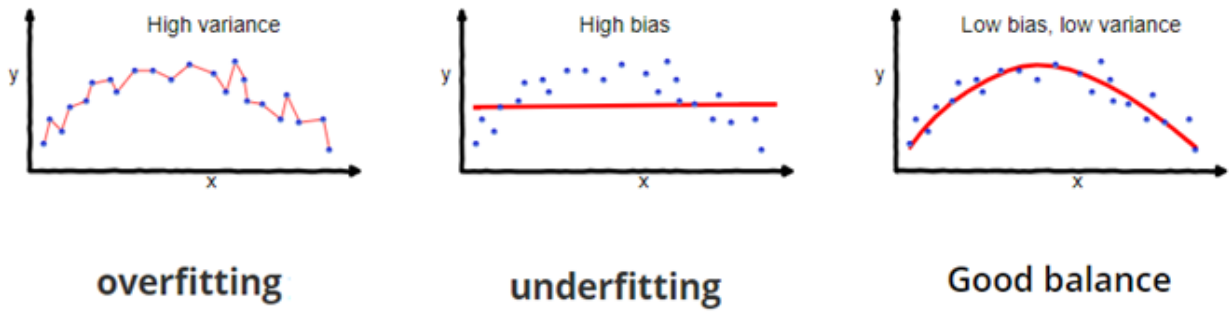


Figura 10: Overfitting e underfitting

- Regolarizzazione L2: è un altro approccio per cercare di eliminare l'overfitting. Questo, infatti, può essere causato dal dare un peso troppo elevato ad alcune unità e per evitarlo si inserisce un termine nella funzione di costo che è l'iperparametro in questione moltiplicato per la sommatoria dei quadrati dei pesi, così che minimizzando la funzione di costo i parametri non possano essere troppo elevati.
- Numero di epoche: può essere un iperparametro ma in questo progetto è un valore fisso. Si definisce come epoca il numero di volte che l'algoritmo passa attraverso il set di allenamento. Di solito è un valore elevato in quanto aumentando il numero di epoche migliora il fitting dei dati. A questo parametro è collegato anche un altro parametro: quando il dataset è molto grande invece di allenare la rete su tutto il dataset ad ogni passaggio, si decide di dividere il dataset in "batches" e allenare il modello per una batch

alla volta. In questo modo per un minor numero di epoche la rete avrà svolto più step di apprendimento, il quale risulterà più veloce.

- Batch size: con questo iper-parametro si definisce la dimensione della batch, cioè il numero di esempi processati in un singolo passaggio della fase di allenamento.

2.7.1 Batch size

A seconda della grandezza della batch si possono presentare 3 situazioni:

- Full batch gradient descent: la batch è l'intero dataset. Questa condizione implica che ad ogni iterazione si compia un'epoca della fase di allenamento. Il problema, soprattutto quando il dataset è molto numeroso, è il tempo necessario per processare tutti i dati contemporaneamente, però è allo stesso tempo l'unica soluzione che, se non sono presenti minimi locali (che data la numerosità di dimensioni sono davvero poco comuni), garantisca la convergenza al minimo. La curva relativa all'errore è sempre decrescente.
- Mini batch gradient descent: sono processati un numero di esempi minore della numerosità di esempi totali per ogni iterazione. Chiaramente in questo caso un'iterazione non corrisponde ad un'epoca. L'idea è che con un numero di esempi minore del numero totale di esempi disponibili si possa comunque garantire un aggiornamento abbastanza buono dei parametri. Questo risulta vero ed inoltre il processo risulta anche più rapido rispetto al caso full-batch. A differenza di quest'ultimo però non è garantita la convergenza al minimo globale e non è detto che ad ogni iterazione la funzione di costo diminuisca. Allo stesso tempo però il trend globale della funzione di costo è quello di decrescita.

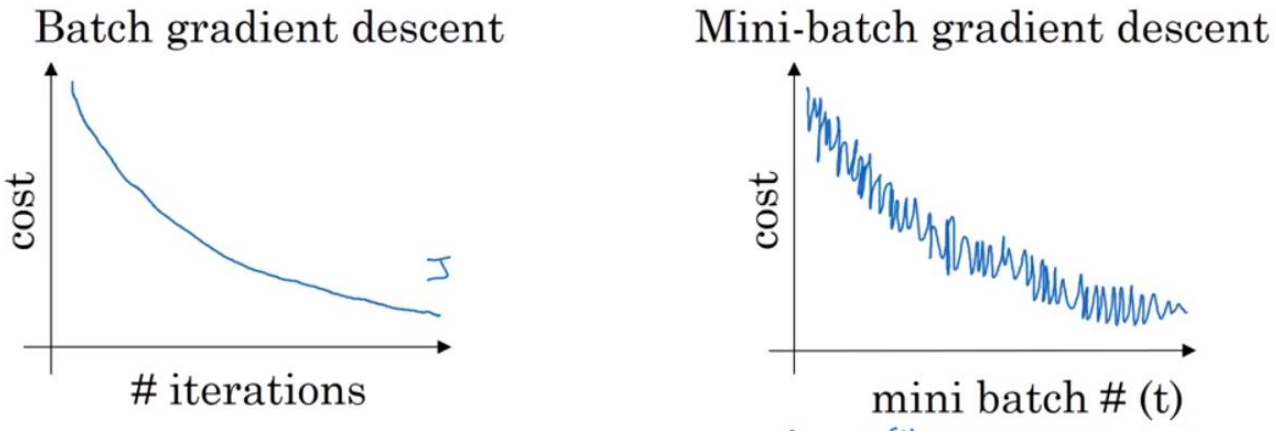


Figura 11:Full-batch e Mini-batch Gradient Descent

- Stochastic gradient descent: in questo caso ad ogni iterazione viene processato un solo esempio. Ciò che ne consegue è una continua variazione della funzione di costo, un aumento del tempo di calcolo e di solito performance e apprendimento peggiori.

2.7.2 Ricerca degli iperparametri

Il modo in cui gli iperparametri vengono cercati nell'iperspazio determina sia le performance che il dispendio computazionale e quindi di tempo. Tra le numerose tecniche presenti quella utilizzata è la "Random Search". Essa infatti, confrontata con la "Grid Search", presenta performance migliori oltre ad essere semplice da implementare e da capire. Questa in particolare risulta più efficiente in iperspazi grandi[13], che vengono divisi in sottospazi nei quali la distribuzione delle combinazioni random è più uniforme, facilitando la ricerca della soluzione ottimale.

2.8 Adam optimizer

L'ottimizzatore utilizzato in questo progetto è Adam [14] (Adaptive Moment Optimizer) la cui principale differenza con il mini-batch gradient descent è che il rapporto di apprendimento non è costante durante l'allenamento.
Questo ottimizzatore combina i vantaggi di due altri ottimizzatori dei quali non è obiettivo di questa tesi spiegare il funzionamento, che sono:

- Adagrad: usa un tasso di apprendimento per ciascun parametro, il che migliora le prestazioni su problemi con gradienti sparsi;
- RMSProp: usa tassi di apprendimento per ciascun parametro adattati in base alla media di grandezze recenti dei gradienti per il peso in questione (ad es. quanto velocemente sta cambiando).

2.9 k-fold cross validation

Nei paragrafi precedenti è stata sottolineata l'importanza di una fase di validazione, che permetta di valutare prima della fase di test come il modello si comporti su dati non utilizzati in fase di allenamento. La procedura utilizzata è la seguente:
il dataset viene diviso in k parti (k-fold). A questo punto il modello viene allenato su k-1 delle parti in cui è stato diviso e sulla restante viene validato. Dopo aver valutato le performaces con la configurazione ottenuta, questa viene cambiata, andando a cambiare la parte di dataset che non viene usata in training ma in validation, fino ad utilizzare ciascuna delle parti come set di validazione una volta, così da provare tutte le configurazioni. Dopo di che si confrontano tutte le combinazioni e si sceglie la migliore.

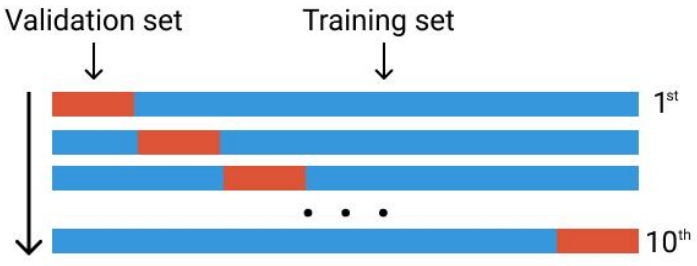


Figura 12:k-fold cross validation

2.10 Permutation feature importance

È una tecnica appartenente al concetto dell'Explainable AI [15](Intelligenza Artificiale spiegabile), utilizzata per rendere gli algoritmi del Deep Learning più comprensibili. Questa viene applicata ad un modello allenato per verificare quanto le previsioni del modello dipendono da ciascuna feature. Ciò che si fa è mescolare in modo casuale le voci di una feature per K volte e verificare quanto varia uno score scelto rispetto al valore di riferimento che chiaramente consiste nel valore ottenuto senza mescolare nessuna feature.
L'espressione matematica è la seguente:

$$i_j = s - \frac{1}{K} \cdot \sum_{k=1}^{K} s_{k,j}$$

Dove *i* è l'importanza della j-esima feature, K è il numero di iterazioni ed s è lo score.
In questo progetto lo score di riferimento è il *Mattews Correlation Coefficient*.

3 Teoria sul Clustering

Il clustering è una tecnica del Machine Learning che consiste nel raggruppamento dei punti di un dataset. In particolare la maggior parte delle applicazioni che riguardano il clustering, come per il caso di questo progetto, sono tecniche di Unsupervised Learning, e si sfruttano algoritmi per trovare all'interno dei dati delle relazioni che siano gli algoritmi stessi ad intuire.
Queste tecniche sono usate in settori quali la segmentazione del mercato, analisi dei social networks, o ancora in settori come l'astronomia, ma in generale vengono utilizzati in molti settori e per l'analisi dei dati.
Gli algoritmi a disposizione, così come i problemi da affrontare, sono diversi. A seconda del tipo di problema ci possono essere algoritmi più o meno appropriati. Non è scopo di questa tesi descriverli tutti, ma quelli trattati sono quelli che verranno riportati di seguito[16].
Questi sono:

- K-means
- MeanShift
- DBSCAN

3.1 K-means

L'algoritmo K-means raggruppa i dati cercando di separare i campioni in n gruppi di uguale varianza, riducendo al minimo un criterio noto come inerzia o somma dei quadrati all'interno del cluster. Questo algoritmo richiede che sia specificato il numero di cluster. Si adatta bene a un gran numero di campioni ed è stato utilizzato in una vasta gamma di aree di applicazione in molti campi diversi.
L'algoritmo k-means divide un insieme N di campioni X in K cluster disgiunti C, ciascuno descritto dalla media dei campioni nel cluster, definita come "centroide"; in generale non sono punti dei campioni X sebbene vivano nello stesso spazio. Il K-means mira a scegliere i centroidi che minimizzano l'inerzia o il criterio della somma dei quadrati all'interno del cluster:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\|x_i - \mu_j\|^2 \right)$$

L'inerzia può essere definita come una misura di quanto siano coerenti internamente i cluster. L'inerzia non è una metrica normalizzata: si sa solo che valori inferiori sono migliori e zero è ottimale.
Due problemi di questo metodo possono essere la convessità dei cluster o la distanza euclidea per uno spazio di dimensioni molto elevate. Per questa analisi non sono problemi rilevanti.
(L'esecuzione di un algoritmo di riduzione delle dimensioni come l'analisi delle componenti principali (PCA) prima del clustering può ridurre il secondo problema e accelerare i calcoli).

3.1.1 Scelta del numero di cluster

La scelta del numero di cluster rappresenta una delle difficoltà di questo algoritmo. Per individuare la soluzione ottimale si ricorre al metodo del gomito "Elbow method" che consiste nel rappresentare la funzione obiettivo (da minimizzare) in relazione al numero di cluster, ed individuare, se presente, il valore per il quale la diminuzione della funzione obiettivo diventa meno rapida[17]. In questa applicazione la funzione obiettivo è rappresentata dall'inerzia, come descritto in precedenza. Un esempio di curva a gomito è quella riportata nella figura seguente:

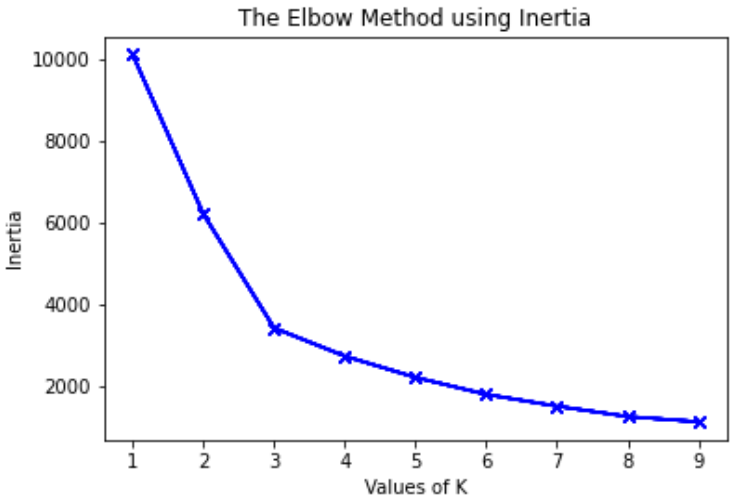


Figura 13:Elbow Method

Risulta abbastanza evidente che 3 cluster risulta essere una scelta particolarmente appropriata. Chiaramente il valore di inerzia continua a diminuire all'aumentare del numero di cluster in quanto tanto più essi sono piccoli tanto minore sarà la distanza tra gli elementi del cluster e il centroide, ma questo non vuol dire che il risultato sia più rappresentativo del problema.

3.1.2 Analisi delle componenti principali

L'Analisi delle componenti principali (Principal Component Analysis, da cui PCA) è una tecnica del machine learning utilizzata per determinare le componenti principali ed usarle per effettuare un cambio di base dei dati. [18]
Spesso viene usata per ridurre le dimensioni dei dati, proiettando i punti di uno spazio di n dimensioni lungo k vettori, dove k è il numero di dimensioni che si vuole ottenere.
Questi k vettori vengono individuati minimizzando l'errore compiuto nel proiettare i punti su di essi.
L'obiettivo di questa riduzione delle dimensioni è spesso quello di visualizzare i dati di uno spazio multidimensionale. Chiaramente essendo una scelta dettata dal rendere rappresentabili i risultati ottenuti la scelta di k ricade, in queste applicazioni, nel range 1-3.

3.2 Meanshift

Anche questo è un algoritmo basato sui centroidi, che funziona aggiornando i candidati ad essere i centroidi (media dei punti) di una data regione. Questi candidati vengono quindi filtrati in una fase di post-elaborazione per eliminare quelli molto vicini (quasi duplicati) e formare l'insieme finale di centroidi. Questo è un algoritmo che imposta automaticamente il numero di cluster, facendo affidamento su un parametro: la larghezza di banda, che determina la dimensione della regione in cui cercare. Questo parametro può essere impostato manualmente o essere stimato tramite un'apposita funzione se la larghezza di banda non è nota.
Tra i 3 algoritmi questo è stato valutato come il meno adatto per un'analisi del dataset.

3.3 DBSCAN

L'ultimo algoritmo analizzato è il DBSCAN. Questo vede i cluster come aree ad alta densità separate da aree a bassa densità. A causa di questa visione piuttosto generica, i cluster trovati da DBSCAN possono avere qualsiasi forma.
La componente centrale del DBSCAN è il concetto di core samples, che sono campioni che si trovano in aree ad alta densità. Un cluster è quindi un insieme di core samples, uno vicino all'altro, e un insieme di campioni che sono vicini a un core sample (ma non sono essi stessi core samples).
Ci sono due parametri per l'algoritmo: numero minimo di esempi (min_samples) e la distanza entro la quale debba essere presente un altro esempio (eps), che definiscono formalmente cosa si intende con denso. Min_samples maggiori o eps inferiori indicano una maggiore densità necessaria per formare un cluster.
Più formalmente, si definisce un campione principale come un campione nel set di dati in modo tale che esistano min_samples altri campioni entro una distanza di eps, che sono definiti come vicini del campione principale.
Qualsiasi campione principale fa parte di un cluster, per definizione. Qualsiasi campione che non è un campione principale ed è almeno a una distanza eps da qualsiasi campione principale, è considerato un valore anomalo dall'algoritmo.
Chiaramente anche questo algoritmo presenta vantaggi e svantaggi.[19]

3.3.1 Vantaggi

- DBSCAN non richiede di specificare a priori il numero di cluster nei dati, al contrario di k-means.
- Può trovare cluster di forma arbitraria.
- Ha una nozione di rumore ed è robusto per gli outlier.
- Richiede solo due parametri ed è per lo più insensibile all'ordinamento dei punti nel database.
- I parametri min_samples e ε possono essere impostati direttamente da un esperto del dominio, se i dati sono ben compresi.

3.3.2 Svantaggi

- DBSCAN non è del tutto deterministico: i punti di bordo raggiungibili da più di un cluster possono essere associati ad uno o all'altro cluster, a seconda dell'ordine in cui vengono elaborati i dati. Nella maggioranza delle situazioni questa situazione non si verifica, inoltre il problema non affligge né i punti principali né gli outlier e ha scarso impatto sul risultato del clustering.
- La qualità di DBSCAN dipende dalla misura della distanza utilizzata nella funzione regionQuery(P,ε). La metrica di distanza più comune è la distanza euclidea. Soprattutto per i dati ad alta dimensionalità, questa metrica può avere grosse difficoltà, rendendo difficile trovare un valore appropriato per ε. Questo effetto, però, è presente in tutti gli algoritmi basati sulla distanza euclidea.
- DBSCAN non può raggruppare bene i set di dati con grandi differenze di densità, poiché la scelta di una sola combinazione min_samples-ε non può essere appropriata per tutti i cluster, problema verificatosi nel presente progetto.
- Se i dati e la scala non sono ben compresi, la scelta di una soglia di distanza significativa ε può essere difficile.

3.3.3 Scelta dei parametri

Per quanto riguarda la scelta dei parametri saranno sicuramente necessari più tentativi ed esperienza per ottenere i valori più opportuni.

- Per la scelta di min_samples si usa una regola di partenza che dice $\min_{\text{samples}} \geq D + 1$ dove con D si indica il numero di dimensioni nel dataset (numero di features). In generale scegliere min_samples =1 implicherebbe che per definizione ogni punto sarebbe core sample, il che non avrebbe senso. La scelta deve ricadere su valori ≥ 3, e per dati con rumore, è di solito meglio scegliere un valore di min_samples più elevato.
- Per la scelta di ε: Il valore di ε può essere scelto utilizzando un grafico k-distance (per un insieme di n oggetti P in uno spazio metrico è un grafico orientato con P che è l'insieme di vertici e con un bordo orientato da p a q ogni volta che q è un vicino più vicino di p), tracciando la distanza dal vicino più vicino k = min_samples-1 ordinato dal valore più grande a quello più piccolo. Buoni valori di ε sono quelli corrispondenti ad un "gomito" in questo grafico. Se è scelto troppo piccolo, una grande parte dei dati non sarà raggruppata; mentre per un valore troppo alto, i cluster si fonderanno e la maggior parte degli oggetti sarà nello stesso cluster. In generale, sono preferibili valori piccoli di ε.

3.4 Confronto performance kmeans – DBSCAN

Per avere un'idea più chiara della differenza di performance a parità di dataset tra k-means e DBSCAN si fa riferimento alla seguente immagine:

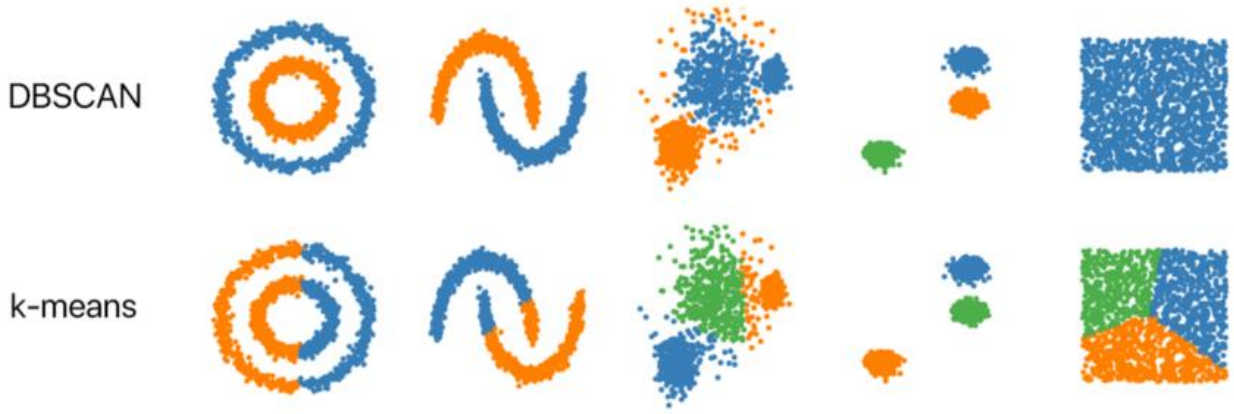


Figura 14: Performance k-means e DBSCAN su dataset uguali

Come si può notare le performance presentano proprio quelle differenze concettuali espresse, con il k-means che divide il dataset in regioni di spazio (o piano), e non è quindi in grado di individuare zone di forma arbitraria come invece il DBSCAN. Allo stesso tempo in un dataset con una distribuzione uniforme di punti il DBSCAN trova un solo macro-cluster. Questo è il caso che si presenta nell'ultima delle 4 situazioni illustrate ed è simile a quello che si verifica in una clusterizzazione bidimensionale del dataset utilizzato in questo progetto.

4 Descrizione del Dataset

4.1 Acquisizione dei dati

Nel primo capitolo si è già detto che i dati che vengono utilizzati per allenare, validare e testare la rete sono risultati della Dynamic Programming (DP). Come già detto infatti l'obiettivo è quello di ottenere un tool che possa aiutare o sostituire la DP in fase di progettazione, e per avvicinarsi ai risultati che questa produce è bene che la rete sia allenata a riprodurre esattamente quei risultati. Le simulazioni della Dynamic Programming sono basate su 5 cicli per veicoli heavy-duty. Questi sono:

- WHVC – 1
- ETC – 2
- HDUDDS – 3
- CSC – 4
- HHDDTS – 5

Il primo è World Harmonized Vehicle Cycle, basato su una prova al banco dinamometrico, della durata di 1800 secondi di cui i primi 900 rappresentano una guida urbana, per la quale la velocità media risulta bassa (21,3 km/h) e la velocità massima è 66,2 km/h; i successivi 481 secondi rappresentano una guida rurale con velocità media e massima più elevate; gli ultimi 419 secondi simulano la guida in autostrada, con velocità elevate: una media di 76,7 km/h e una massima di 87,8 km/h. Fasi di start & stop sono davvero poco frequenti.
Il profilo di velocità del veicolo è il seguente:

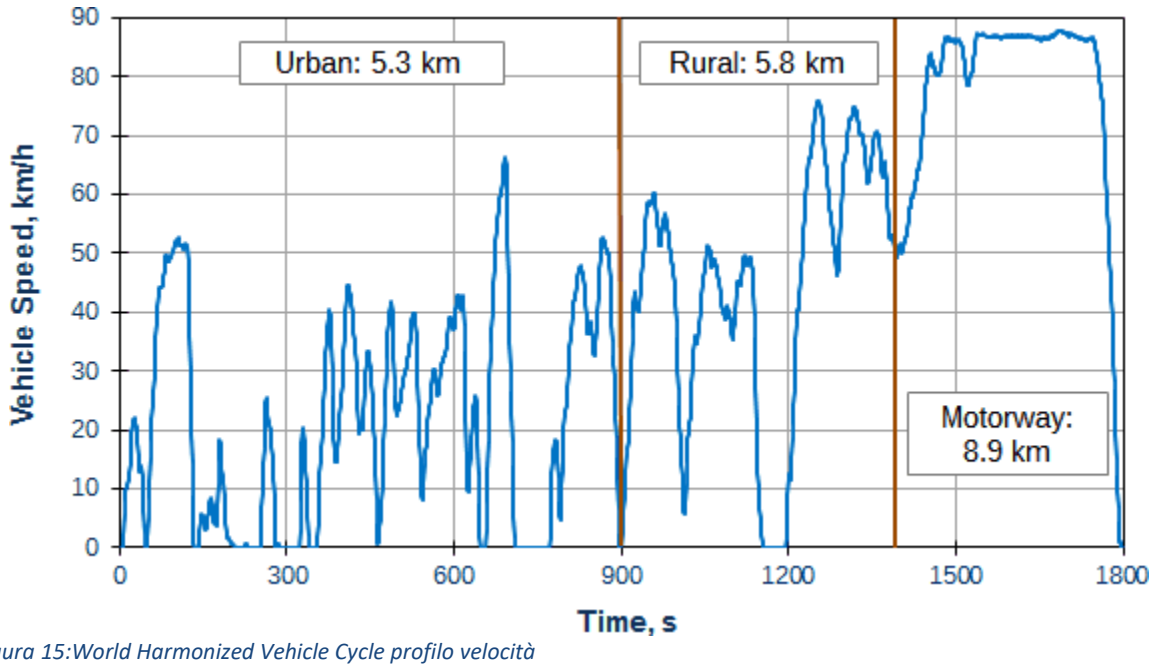


Figura 15:World Harmonized Vehicle Cycle profilo velocità

Il secondo è l'European Transient Cycle, anche questo della durata di 1800 secondi, questa volta divisi equamente rispetto ai tre scenari di guida urbana, rurale e in autostrada, caratterizzate rispettivamente da:

- velocità massima di 50km/h e frequenti start & stop
- Velocità media di 72 km/h e rapida accelerazione
- Velocità media di 88 km/h

Il profilo di velocità è:

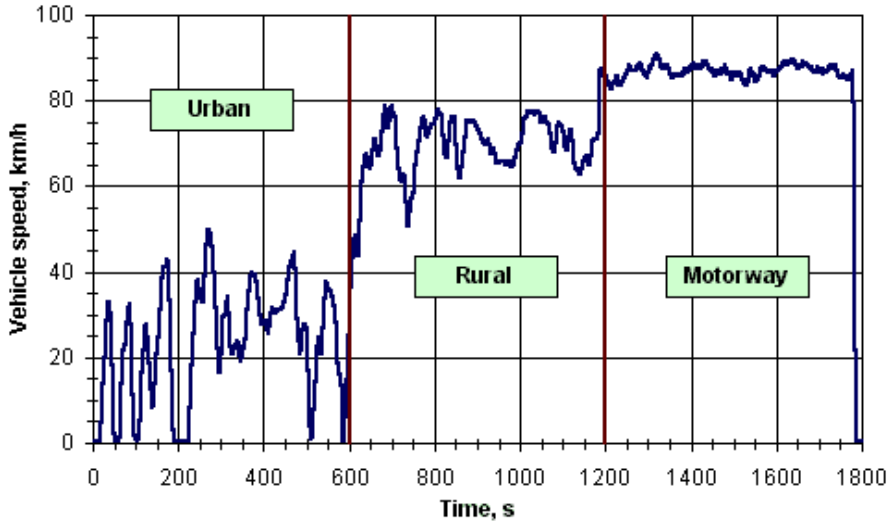


Figura 16: European Transient Cycle profilo velocità

Il terzo è Heavy-Duty Urban Dynamometer Driving Schedule la cui durata è inferiore rispetto ai precedenti. Anche questo è basato su una prova al banco dinamometrico e il profilo di velocità è il seguente:

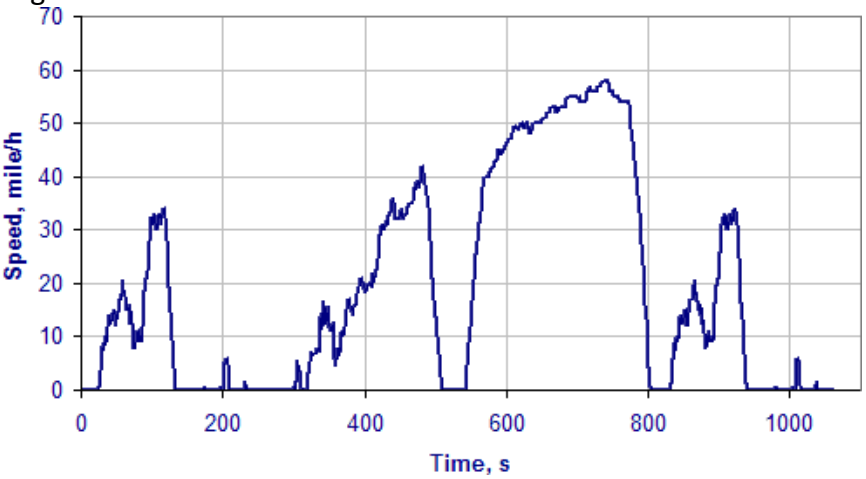


Figura 17:Heavy-Duty Urban Dynamometer Driving Schedule profilo velocità

Anche gli ultimi due cicli, che sono il City Suburban Heavy Vehicle Cycle & Route e l'Heavy Heavy-Duty Diesel Truck Schedule sono basati su prove al banco dinamometrico.

Dei due il primo ha il seguente profilo di velocità:

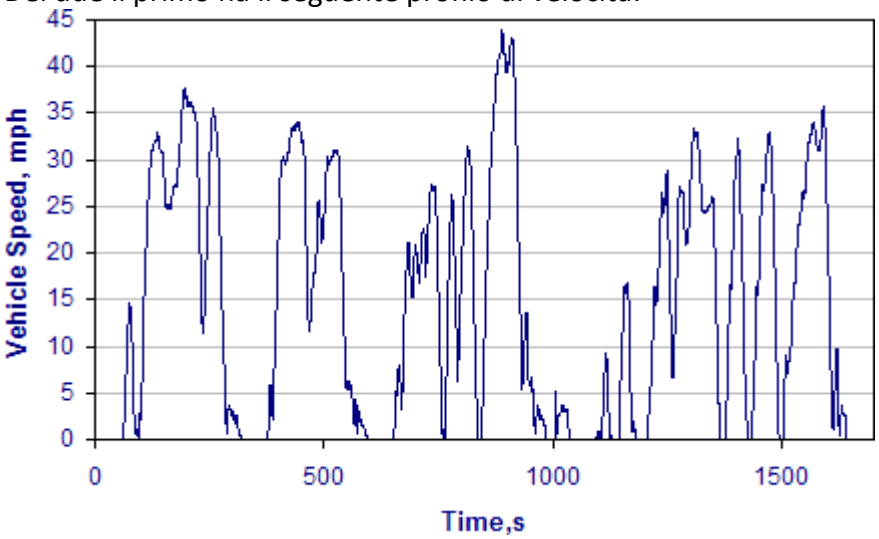


Figura 18:City Suburban Heavy Vehicle Cycle & Route profilo velocità

Infine il profilo di velocità per l'HHDDTS è diviso in quattro modalità: ozioso, strisciante, in transitorio e di crociera. Le ultime tre modalità sono riportate in figura:

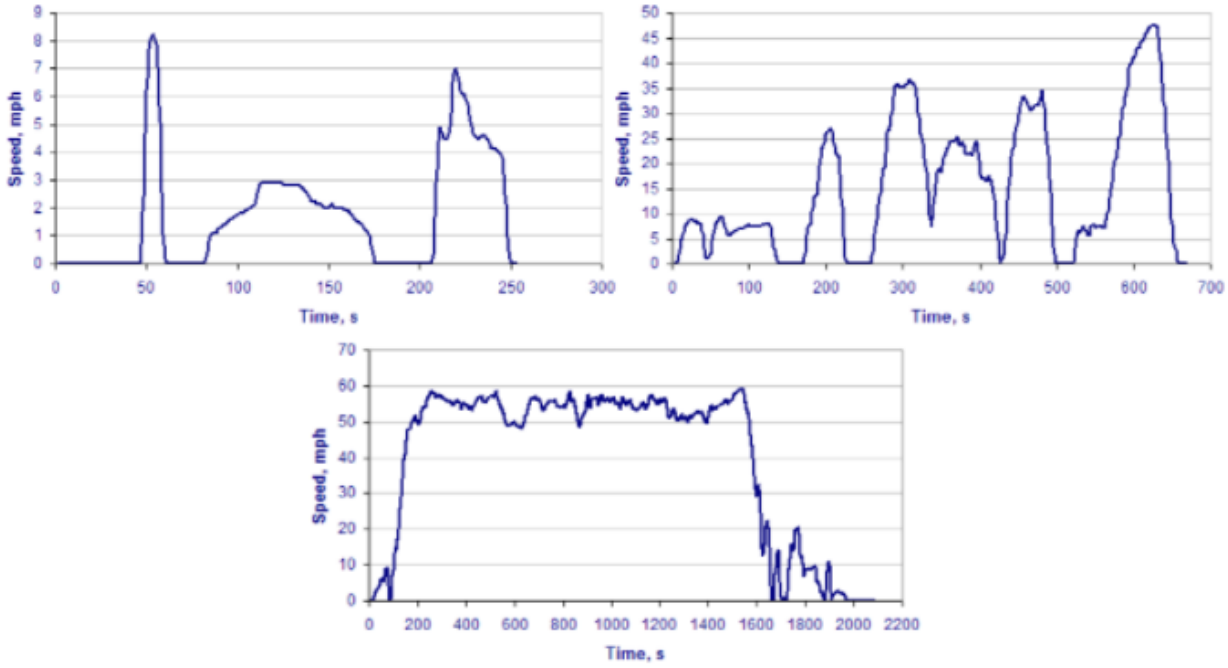


Figura 19:Heavy Heavy-Duty Diesel Truck Schedule

A partire dalle simulazioni effettuate su questi cicli viene composto un unico dataset da 7500 elementi (1500 per ogni ciclo) con l'architettura P2.

Un esempio di architettura P2 è quello riportato nell'immagine seguente:

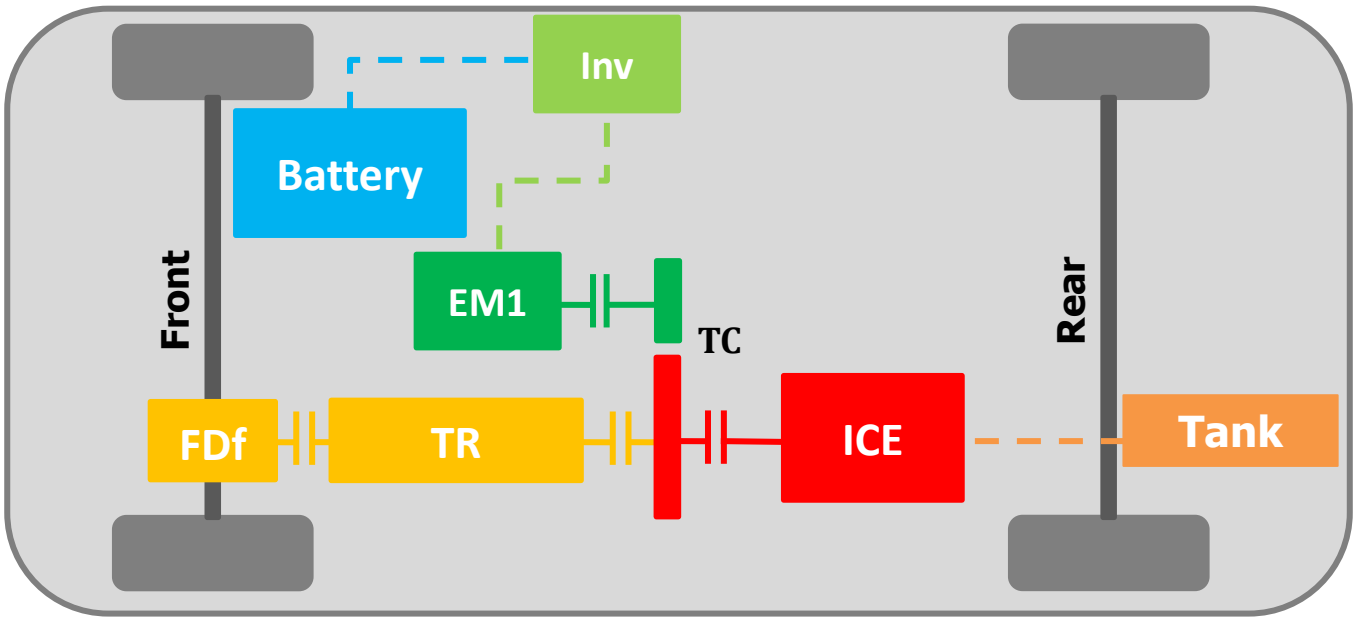


Figura 20:Architettura P2

Come si può vedere la macchina elettrica assiste il motore a combustione interna per fornire la potenza meccanica alla trasmissione e quindi alle ruote motrici, che sono solamente quelle dell'assale anteriore. Questa architettura accoppia la potenza del motore elettrico a quella del motore termico con un dispositivo di accoppiamento di coppia.

L'architettura utilizzata nel progetto è invece caratterizzata da un dispositivo di accoppiamento di velocità tra motore elettrico e termico.

4.2 Features

Il dataset è costituito da 10 colonne di cui 2 sono relative alle labels riguardanti la feasibility e l'admissibility, e le altre 8 sono le features. Nello specifico queste sono:

- EngDispl: engine displacement, misurata in litri ed indica la cilindrata del motore termico (ICE)
- PE ratio: è il rapporto tra la potenza della macchina elettrica e l'energia massima che può essere immagazzinata nella batteria, $PE = \frac{P_{em}}{E_{batt}}$;
- EM1Power: potenza del motore elettrico, misurato in kW;
- EM1SpRatio: rapporto di trasmissione al livello del dispositivo di accoppiamento di velocità;
- FDpSpRatio: rapporto di trasmissione dell'assale anteriore;
- CrateDis: Massimo rapporto di scarica della batteria. Definito come: $C_{rateDis} = \frac{1h}{t_{I=max}}$ dove $t_{I=max}$ è il tempo di scarica alla massima intensità di corrente;
- CrateCh: Massimo rapporto di scarica della batteria. Definito come: $C_{rateChar} = \frac{1h}{t_{I=max}}$ dove $t_{I=max}$ è il tempo di carica alla massima intensità di corrente;
- Cycles: è l'indicazione relativa a quale tra i 5 cicli si fa riferimento.

4.3 Normalizzazione dei dati

La normalizzazione dei dati presenti nel dataset è ricorrente nelle applicazioni del Deep Leaning, infatti è dimostrato che la rete può trarre grandi benefici da questa pratica, che risulta particolarmente necessaria quando gli ordini di grandezza delle varie feature sono diversi o non sono noti.
Si noti che la normalizzazione viene applicata solo alle feature e non alle labels. Inoltre, poiché la rete viene allenata su un dataset normalizzato, è opportuno che anche i set di validazione e test siano normalizzati, altrimenti i risultati non sarebbero coerenti.
Come per la maggior parte delle cose, anche per la normalizzazione dei dati non esiste una sola tecnica, e nell'ambito di questo progetto la funzione implementata è la seguente:

$$valore_{norm} = \frac{valore - media}{std}$$

Dove la media e la deviazione standard (std) sono quelle della feature considerata.
Con questa normalizzazione non si possono fissare un limite inferiore e superiore uguali per tutte le features, ma ciò che si ottiene sono delle features dello stesso ordine di grandezza. Questo è utilissimo per accelerare la convergenza al minimo della cost function rispetto ad un dataset non equamente distribuito:

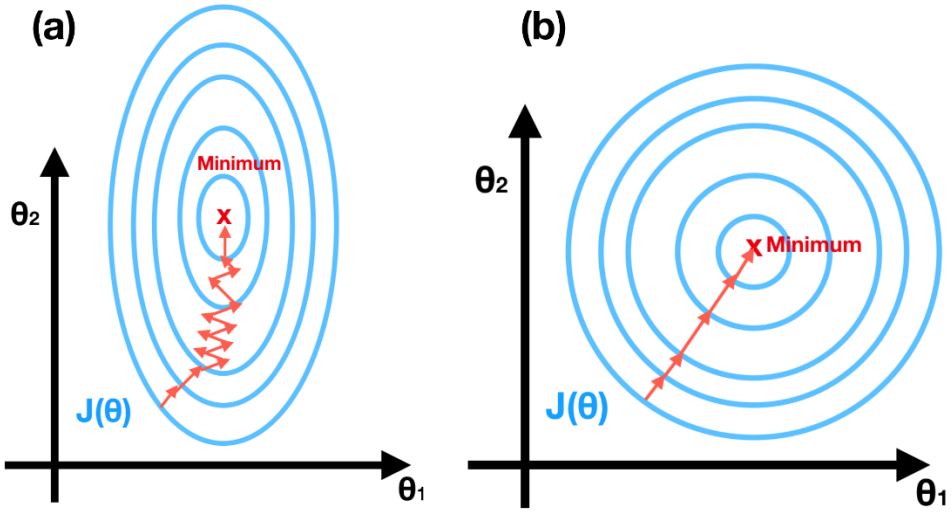


Figura 21:Normalizzazione delle features

Come si può vedere nel dataset non normalizzato i cambi di direzione sono molti e non orientati secondo una direzione di diminuzione globale, con un conseguente numero di passi necessari maggiore. Quindi la normalizzazione velocizza e stabilizza la convergenza.

5 Modello

5.1 Ambiente di lavoro e librerie

Il linguaggio di programmazione usato per il progetto è Python 3 sia per la semplicità della sintassi che per la grande disponibilità di strumenti e librerie disponibili. Alcune delle principali librerie usate sono:

- Tensorflow: è una libreria di Python che opera a basso livello per calcoli numerici e machine learning su larga scala, offrendo una grande varietà di algoritmi e modelli per Machine Learning e Deep Learning.
- Keras: è una libreria in grado di lavorare su librerie a basso livello come Tensorflow ed è stata progettata per permettere una rapida sperimentazione con le reti neurali.
- Scikit-learn: è una libreria che oltre ad algoritmi di classificazione e regressione ne contiene altri di clustering, di processamento dei dati, di previsioni ed è in grado di interagire con NumPy e SciPy.
- SciPy: è usata per risolvere problemi comuni riguardanti calcoli scientifici e per il calcolo delle distanze tra i punti in uno spazio multidimensionale.
- NumPy: usata per le operazioni con vettori multidimensionali e matrici. Inoltre mette a disposizione funzioni matematiche di alto livello da usare con questi vettori.
- Matplotlib: è una libreria per il plotting, può essere considerata come un'estensione grafica di NumPy.
- Pandas: libreria fondamentale per la manipolazione e analisi dei dati, lavora con oggetti di tipo "Dataframe" fornendo le funzioni necessarie per svolgere operazioni su e fra essi.

5.2 Metriche per la valutazione delle performance

Per la fase di classificazione, partendo dalla *confusion matrix*, che riporta i risultati del confronto tra le labels reali e previste dalla rete (veri positivi e veri negativi sono le previsioni corrette, falsi positivi e falsi negativi quelle sbagliate), si possono definire le metriche utilizzate:

- Accuracy: calcolata come le previsioni corrette divise per il numero totale di previsioni (sia corrette che sbagliate), e rappresenta quindi il modo più intuitivo per descrivere le prestazioni in classificazione;

$$Acc = \frac{TP + TN}{(TP + FN) + (TN + FP)} = \frac{TP + TN}{P + N}$$

- F1 score: è una metrica utilizzata quando la distribuzione dei dati tra le due classi non è omogenea (ad esempio tanti esempi negativi e pochi positivi) così da ottenere una valutazione corretta delle performance della rete. Tiene conto sia della precisione (veri positivi/previsti positivi, sia veri che falsi), che della sensibilità (veri positivi/positivi reali, cioè veri positivi+falsi negativi).

$$F1 = 2 * \frac{precisione * sensibilità}{precisione + sensibilità}$$

- Matthews Correlation Coefficient: considera tutte le classi della classificazione binaria ed è ritenuto il miglior indicatore di performance che si affida ad un solo numero[20]. A differenza dell'F1 score l'MCC non è influenzato da quale classe viene scelta come positiva (infatti se ci fossero tanti positivi e pochi negativi l'F1 score potrebbe non dare indicazioni notevoli, a differenza dell'MCC.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Per quanto riguarda il regressore la metrica utilizzata è quella dell'R quadro (R^2) che può assumere valori tra $-\infty$ e 1 e permette di confrontare l'approssimazione ottenibile con una semplice media delle label del dataset con le previsioni del modello. Quanto maggiore è la differenza, tanto maggiore è accurato il modello. La formulazione matematica è:

$$R^2 = 1 - \frac{SSER}{SSEM}$$

Dove SSER sta per Sum of Squared Error by Regression e SSEM Sum of Squared Error by Mean line. Se $R^2 = 1$ le previsioni del modello fittano perfettamente le labels reali, se $R^2 = 0$ il modello si comporta come si comporterebbe un'approssimazione con la media, $R^2 = -\infty$ significa che le previsioni del modello sono in contraddizione con quelle reali.

5.3 Modello di base

Di seguito è presentato uno schema della pipeline di Reti Neurali (DNN) ereditata ed utilizzata in questo progetto. Tutte le reti neurali sono composte da un layer di input, degli hidden layers e un layer di output. Per evitare problemi di gradient vanishing e per migliorare l'efficienza di convergenza viene utilizzata la funzione di attivazione ReLU negli hidden layers. Diversi studi hanno anche dimostrato che un layer di *batch normalization* può accelerare la convergenza [21], motivo per il quale viene applicato in questo progetto. Infine nel regressore (rDNN) viene utilizzato un dropout-layer per ridurre l'overfitting. L'algoritmo di ottimizzazione Adam (stima del momento adattativo), che garantisce una convergenza veloce, è stato scelto come ottimizzatore delle Reti Neurali. Questo consente anche un aggiornamento stabile dei pesi e dei bias, oltre che un efficace controllo automatico del rapporto di apprendimento.
I layer relativi alla batch-normalization ed al dropout sono inclusi in sequenza dopo l'effettivo layer nascosto della rete su cui agiscono. Questa è l'effettiva implementazione in ambiente Keras. Keras, infatti, prevede una struttura sequenziale che consente all'operatore di impilare strati uno sopra l'altro utilizzando semplici righe di codice sequenziali. Questo approccio consente ai meno esperti di costruire architetture anche molto complesse con poche righe di codice, ed è sicuramente uno dei motivi che ha portato a scegliere di utilizzare Keras.

6 Studio del dataset

A questo punto definite anche le metriche per la valutazione delle performance si può passare agli studi fatti sul dataset per cercare di aumentare le performance della rete.
Partendo dai risultati della Permutation Feature Importance, che permette di individuare quali tra le features influenzano maggiormente i risultati della rete, si è scelto un piano rappresentativo per la visualizzazione dei dati:

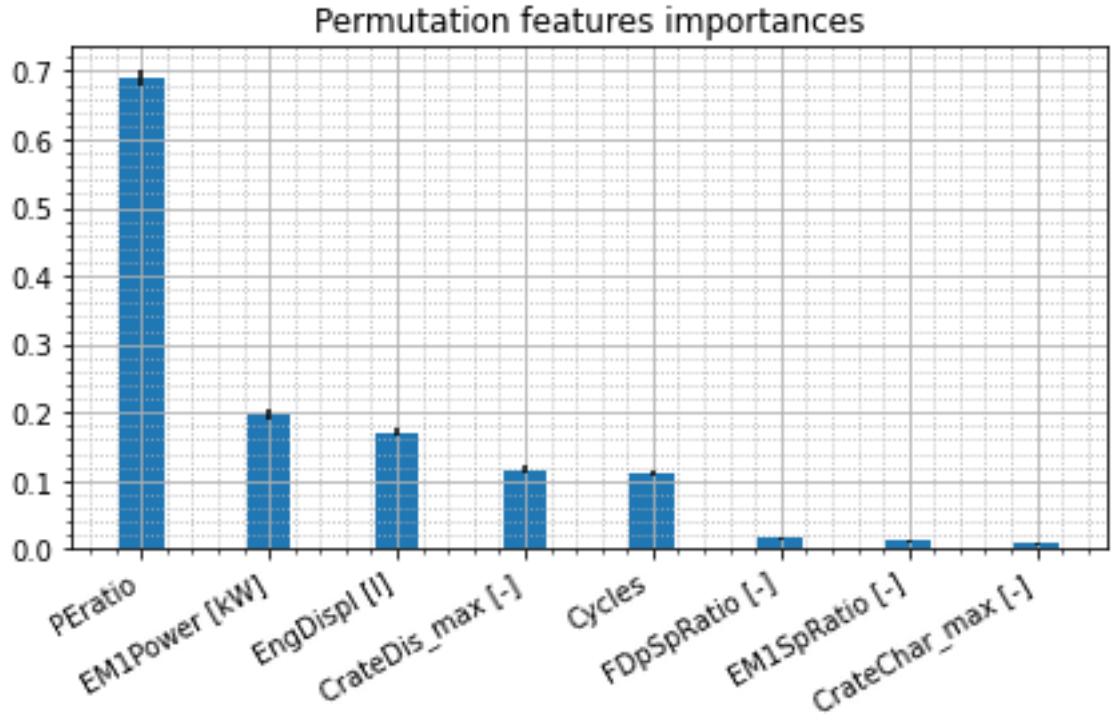


Figura 22:Permutation feature importance

Le due features più importanti sono “PE ratio” ed “EM1Power” e la rappresentazione della distribuzione dei layout nel piano individuato da queste due grandezze è quella riportata in figura 22.
Per evidenziare in un piano la fattibilità o meno del layout sono stati utilizzati due colori diversi per i feasible e gli unfeasible layouts, così da evidenziarne la distribuzione rispetto alle due features più importanti.

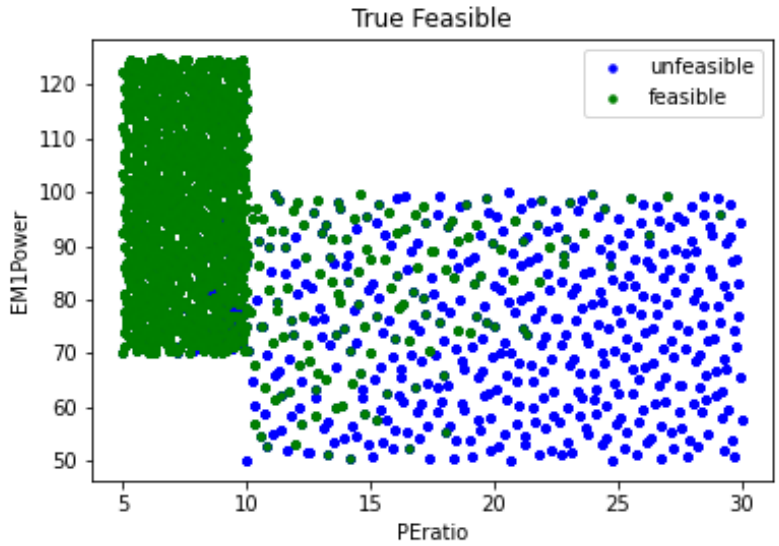


Figura 23:Distribuzione feasible-unfeasible layouts

Come ben visibile risulta presente una zona dove vi è una notevole difficoltà nella discriminazione tra i layout feasible e quelli unfeasible che potrebbe essere la zona di maggiore criticità per le prestazioni della rete.
Lo scopo è quindi quello di riuscire ad estrapolare dal dataset completo questa zona di sovrapposizione per valutare il comportamento della pipeline su di essa. A questo punto le possibilità sono o di delimitare questa zona in base a quanto visibile e definendo quindi un limite superiore ed inferiore o di implementare un algoritmo che sia in grado di individuare in maniera automatica questa zona di sovrapposizione.
Chiaramente un confronto tra le due possibilità è necessario per fare maggiore chiarezza.

6.1 Zona di sovrapposizione manuale

La zona individuabile manualmente è quella delimitata dalle linee verticali in figura:

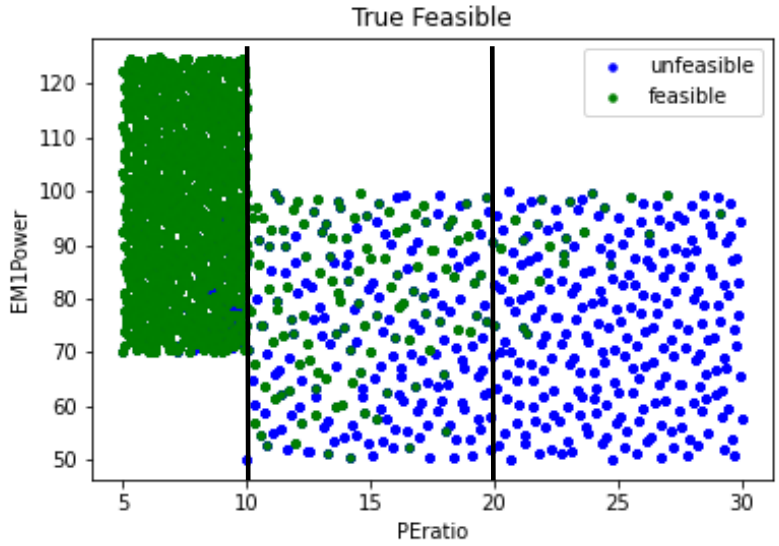


Figura 24:Zona di sovrapposizione manuale

Trattandosi di una scelta manuale risulta influenzata da numerosi fattori, il più importante è sicuramente la chiarezza della rappresentazione, infatti trattandosi di una rappresentazione bidimensionale di uno spazio di 7 dimensioni, ci saranno numerosi punti sovrapposti, quindi potrebbero esserci layout unfeasible anche là dove i layout feasible predominano e viceversa. Inoltre la definizione della zona non può essere tanto accurata in quanto la definizione di una zona i cui limiti non siano delle rette verticali si complica notevolmente dal punto di vista del codice e si perderebbe la praticità di questo tipo di scelta. Infine non trascurabile è la soggettività della scelta: proprio per il fatto che i limiti della zona non saranno molto precisi, la scelta viene lasciata al buon senso di chi la fa, perdendo quindi di ripetibilità e oggettività.
Ciò che si nota dalla figura precedente è che il piano scelto permetta grosso modo di individuare una zona di sovrapposizione, accurata o meno che sia. Però la nota fondamentale è la forte dipendenza dal piano scelto, infatti plottando i punti su un piano diverso la zona di sovrapposizione potrebbe essere non visibile e non portare all'individuazione dei punti corretti. Scegliendo ad esempio un altro piano per la rappresentazione degli stessi punti ciò che si otterrebbe è:

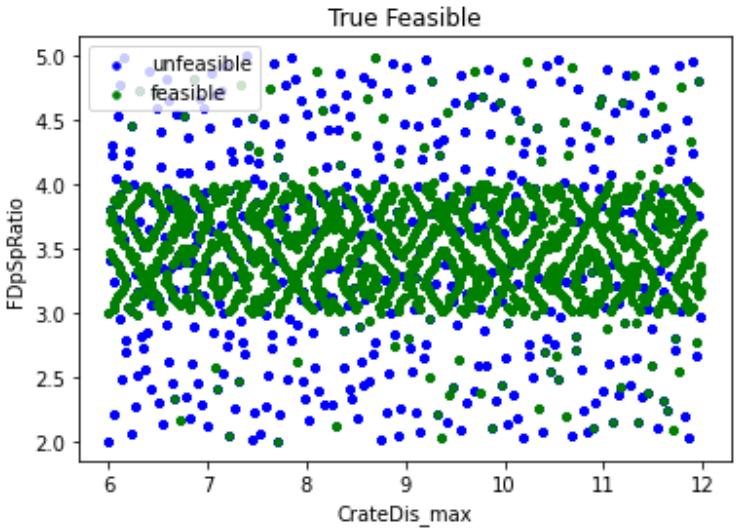


Figura 25:Distribuzione feasible-unfeasible in un altro piano

Come si può ben vedere la scelta di una zona di sovrapposizione risulta impossibile in quanto su tutto il piano non c'è una zona dove la quasi totalità dei layout siano feasible, una dove siano quasi totalmente unfeasible ed una dove i layout feasible e unfeasible coesistono in egual misura. La distribuzione di layout feasible ed unfeasible è abbastanza omogenea e non è quindi discriminabile manualmente la zona di sovrapposizione, che però esiste, in quanto i dati sono gli stessi di prima.

6.2 Zona di sovrapposizione automatica

Questa soluzione ha sicuramente il vantaggio di non dover agire manualmente sul dataset, ma non solo: permette infatti di individuare con maggiore accuratezza la zona di sovrapposizione definendo quindi una zona più appropriata.
Per raggiungere l'obiettivo è stato sviluppato un algoritmo basato sulle distanze euclidee tra i punti del dataset nello spazio di 7 dimensioni individuato dalle features. Per l'implementazione si utilizza la funzione cdist che permette di specificare le matrici tra le quali calcolare la distanza e anche il tipo di distanza, che in questo caso è appunto quella euclidea.
Fin da subito quest'analisi fornisce un risultato interessante, infatti la zona estrapolata è molto accurata, come visibile nella figura seguente:

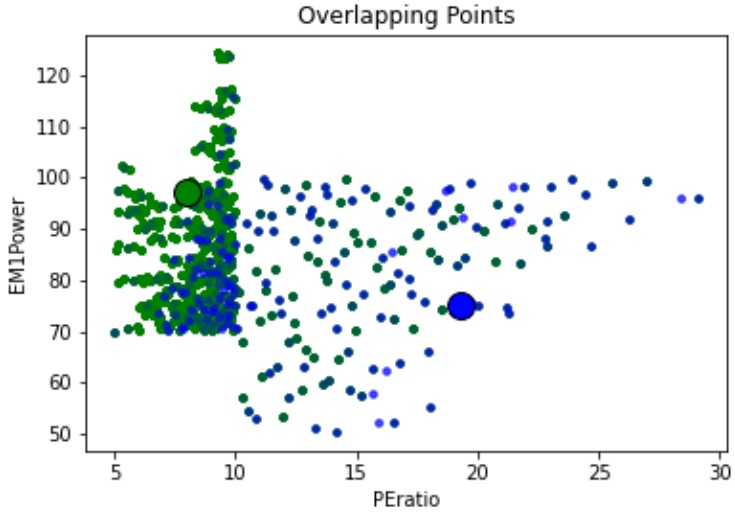


Figura 26:Punti di sovrapposizione

Con questo algoritmo vengono presi singolarmente i punti che vengono identificati come di sovrapposizione e non globalmente una zona di sovrapposizione. Questa caratteristica è contemporaneamente un vantaggio in quanto non presenta il problema di dover definire complessi limiti, ma anche uno svantaggio perché ci sono dei punti che l'algoritmo non etichetta come punti di sovrapposizione, anche se questi appartengono alla stessa zona.
Per risolvere questo problema si sfrutta la clusterizzazione, infatti rimuovendo i punti individuati per isolarli, il dataset costituito dai punti rimanenti è un dataset con due zone a densità diverse ed una zona intermedia tra le due costituita dai pochi punti appartenenti alla zona di sovrapposizione ma che non sono stati rimossi.
In base alle peculiarità degli algoritmi di clustering analizzati in precedenza, la scelta è ricaduta sul DBSCAN:

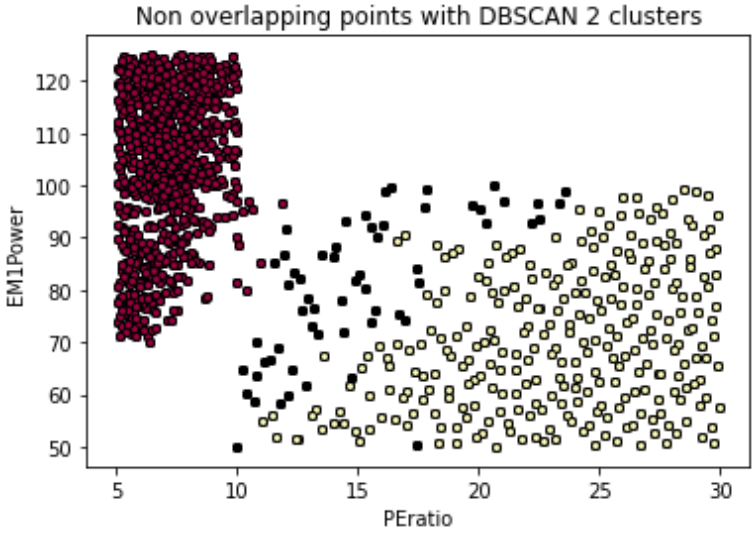


Figura 27: Clusterizzazione punti di non sovrapposizione con DBSCAN

Il risultato è proprio quello atteso. L'algoritmo individua 2 clusters: i layout in rosso appartengono ad un cluster, quelli in giallo sono quelli appartenenti al secondo cluster, e quelli in nero sono gli outlier.
Come ultimo step basta quindi rimuovere gli outlier da questo dataset e inserirli in quello della zona di sovrapposizione, ottenendo quindi una separazione più completa:

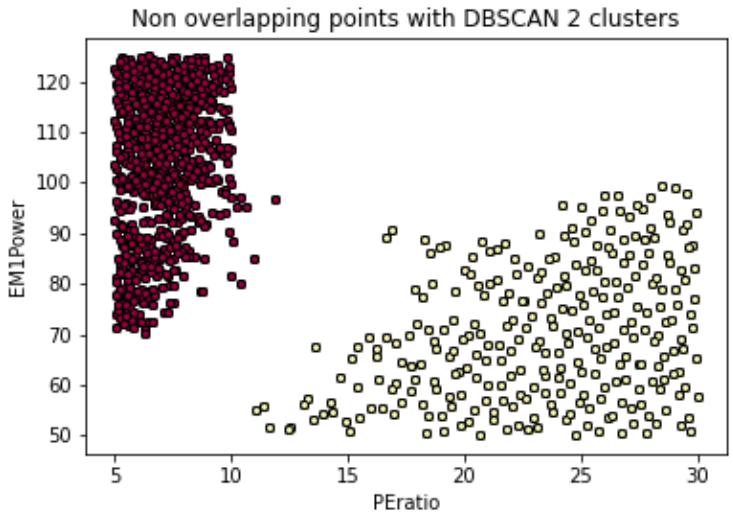


Figura 28: Zona di non sovrapposizione

In figura 27 sono rappresentati gli stessi due cluster di prima senza gli outlier. Questi dati vanno a formare il dataset della zona di non sovrapposizione. In figura 28 invece sono rappresentati i punti che vanno a costituire il dataset della zona di sovrapposizione.

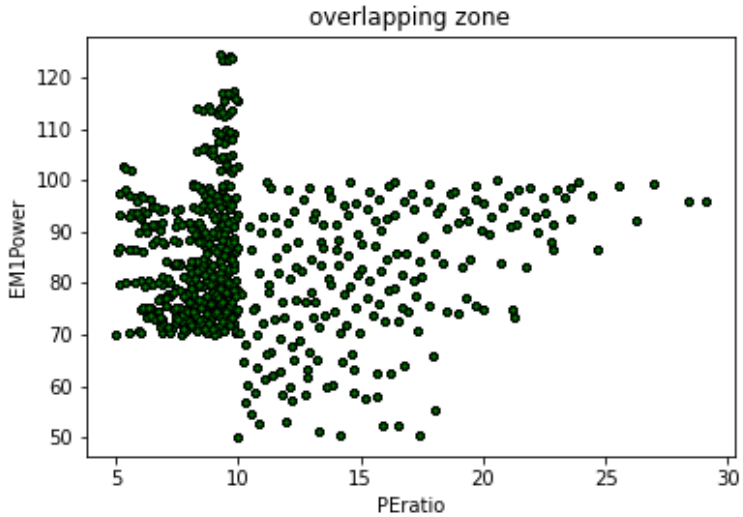


Figura 29: Zona di sovrapposizione automatica

Come risulta già evidente questo algoritmo permette non solo di individuare in maniera molto più accurata la zona di sovrapposizione, ma anche e soprattutto di non dipendere in alcun modo dal piano di rappresentazione. Questo, infatti, è solo una scelta sulla quale rappresentare i risultati

ottenuti dall'algoritmo, a differenza del caso di individuazione manuale dove il piano di rappresentazione è anche il mezzo dal quale i risultati vengono estrapolati.

6.3 Studio con algoritmi di clustering

Viste le potenzialità degli algoritmi di clustering sono stati fatti anche alcuni tentativi di preprocessing del dataset con il k-means e il DBSCAN, sempre nell'ottica di capire se ci fossero possibili soluzioni per migliorare le performance della rete. In particolare si è portato avanti uno studio separando i feasible e gli unfeasible layout e applicando le clusterizzazioni a ciascuna delle due classi, alla ricerca di possibili relazioni interne alle classi stesse che potessero risultare utili in fase di preprocessing.
In quest'ottica sono stati portati avanti più tentativi, a partire da un confronto tra i 3 algoritmi (k-means, meanshift e DBSCAN) in un'applicazione bidimensionale, per passare ad una tridimensionale, che ha permesso di escludere l'eventuale impiego del meanshift e concentrarsi su un confronto tra il k-means e il DBSCAN in uno spazio multidimensionale dove ci si aspettava che il DBSCAN performasse meglio.
In realtà i risultati ottenuti mostrano una migliore capacità da parte del k-means di individuare delle classi ben definite nel dataset, a differenza del DBSCAN che è risultato molto meno performante da quel punto di vista, individuando spesso classi troppo poco numerose da essere significanti. Allo stesso tempo cambiando i parametri con cui lavora il DBSCAN si riescono ad ottenere performance interessanti per quanto riguarda l'individuazione di outlier.
I risultati della clusterizzazione nello spazio individuato da tutte le variabili di design non sono sempre ben rappresentabili in uno spazio 3D (essendo la dimensionalità dello spazio maggiore di 3, chiaramente la rappresentabilità viene meno e bisogna ricorrere ad una rappresentazione in uno spazio che abbia solo 3 delle features totali come assi) costituito da 3 delle 7 feature prese in considerazione. Per questo motivo si utilizza l'analisi delle componenti principali, per visualizzare i risultati in uno spazio 3-dimensionale il più rappresentativo possibile della varianza dei dati:

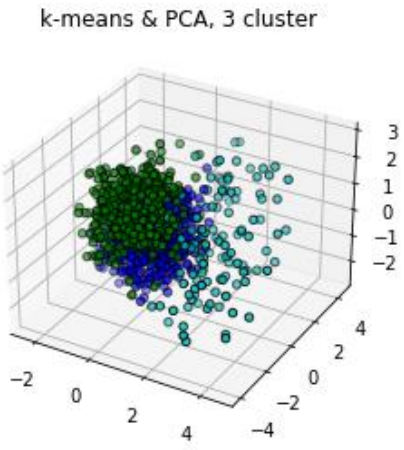


Figura 30:Cluster Feasible layouts e PCA con k-means

In figura 29 vengono riportati i risultati ottenuti con il k-means, invece in figura 30 quelli ottenuti con il DBSCAN.

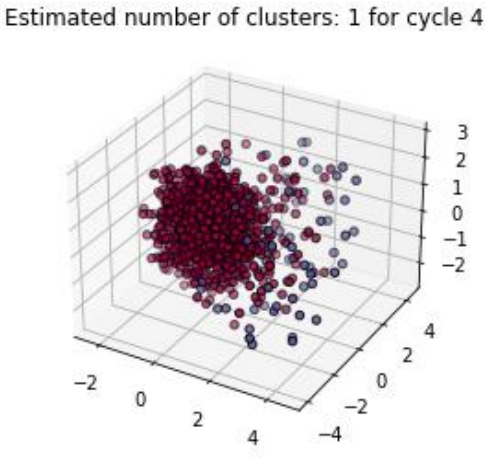


Figura 31: Cluster Feasible layouts e PCA con DBSCAN

Da queste analisi non è emersa qualche caratteristica in particolare che determinasse le classi che si vanno a formare, e potrebbe quindi essere oggetto di ulteriori analisi. Allo stesso tempo però, date le capacità di individuazione degli outlier dimostrate da parte del DBSCAN si è provato ad usare questo algoritmo per verificare se, grazie alla rimozione degli outlier che individua, si riuscissero a migliorare le performance della pipeline.
Le analisi riguardano il dataset composto dai soli feasible layout.

7 Outlier con DBSCAN

7.1 Individuazione e rimozione outlier

L'idea di base è quella appena proposta. Si parte plottando gli andamenti delle emissioni di CO2 di tutti i layout relativi a un ciclo guida, ordinati in maniera crescente:

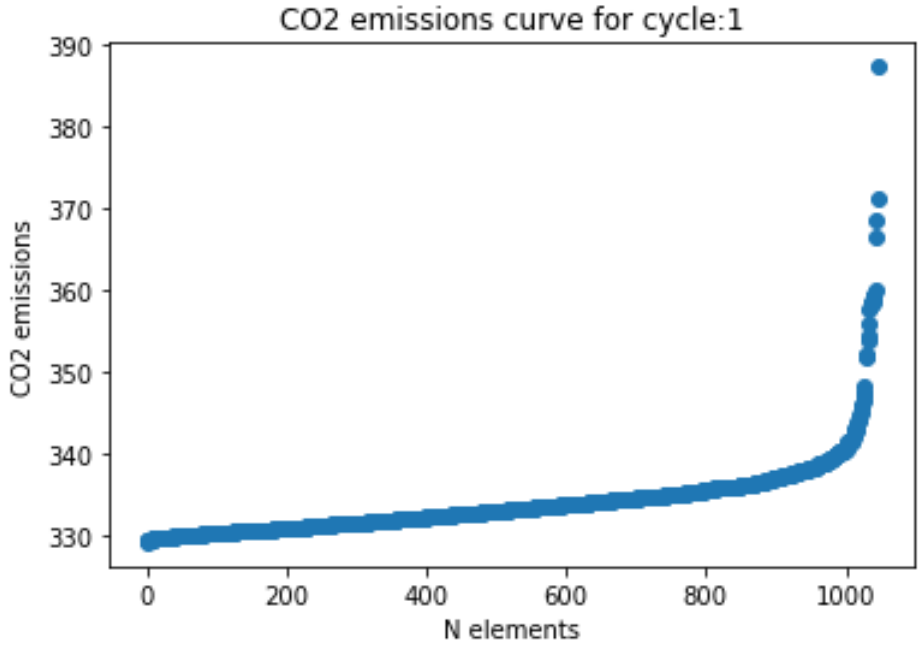


Figura 32: Andamento delle emissioni di CO2

Ciò che salta subito all'occhio è l'impennata finale. La spiegazione matematica è molto semplice, e cioè che essendoci in ascisse la numerosità di elementi, l'impennata sta solamente ad indicare la scarsità di layout caratterizzati da emissioni di CO2 così elevate. Dal punto di vista fisico però la domanda che sorge è se ci sono delle correlazioni tra alcune features e l'innalzarsi delle emissioni inquinanti e se, data la scarsità, questi layout ritenuti feasible non siano invece degli outlier che possono mettere in difficoltà l'apprendimento della rete.
Per prima cosa quindi sono state cercate le eventuali correlazioni con le features. Per fare ciò si è proceduto plottando i valori della feature in questione sempre rispetto al numero di elementi. Questo perché così facendo si riesce a mettere meglio in evidenza la relazione tra l'andamento delle emissioni e quello delle feature stesse.

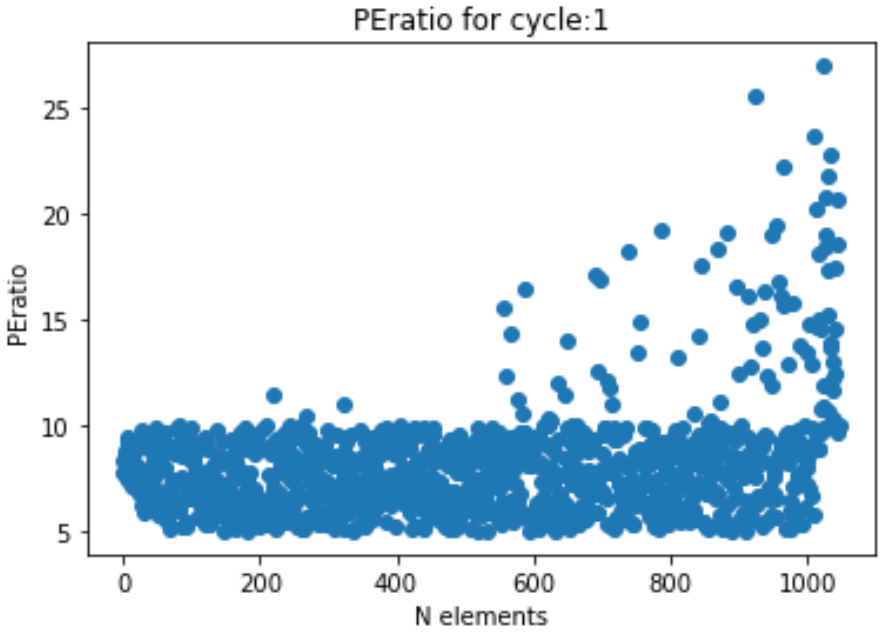


Figura 33: Andamento del PEratio

Per quanto riguarda il PE ratio si nota proprio che, per la maggior parte dei layout, è compreso tra 5 e 10, invece da un numero di layout da circa 600 in poi si nota, in primis con bassa densità, una presenza di layout caratterizzati da PE ratio più elevati, e alla fine della curva si nota come, con maggiore densità, si presenti un andamento simile a quello dell'impennata di CO2.

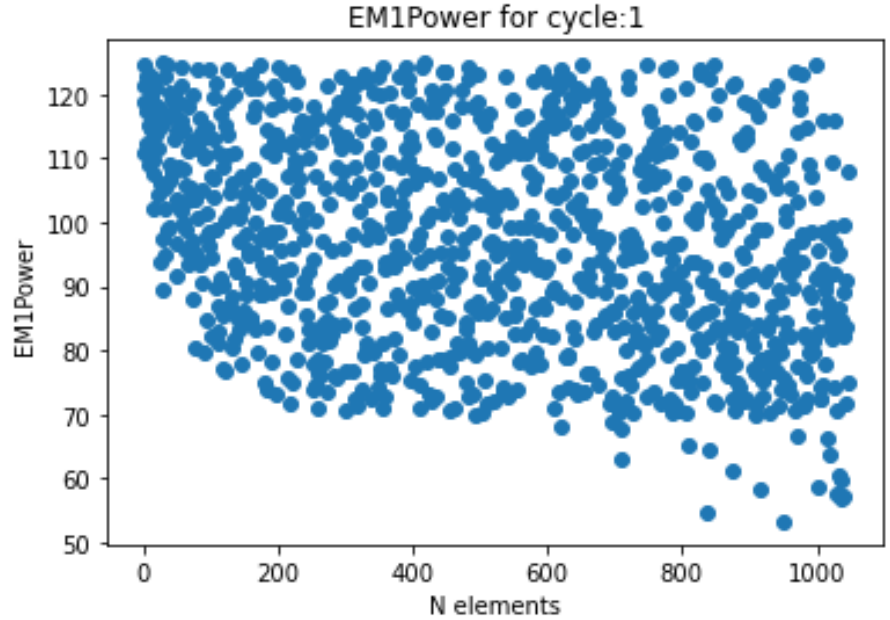


Figura 34:Andamento dell'EM1Power

Per l'EM1 Power la distribuzione di layout è molto uniforme tra 70 e più di 120 per la grande maggioranza di layout, con un basso numero di layout caratterizzati da valori compresi tra 50 e 70, alcuni dei quali appartenenti alla zona dell'impennata di CO2.

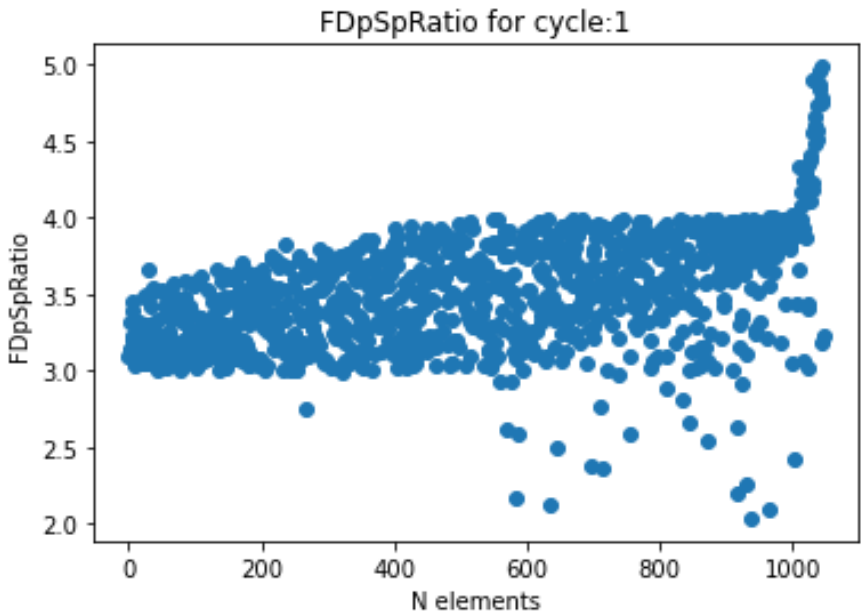


Figura 35:Andamento dell'FDpSpRatio

Per questo ciclo l'FDpSpRatio è la feature che presenta l'andamento più netto e simile a quello della CO2. La maggior parte dei layout hanno un valore compreso tra 3 e 4, pochi layout sparsi con valore compreso tra 2 e 3, e poi la zona dell'impennata di CO2 corrisponde alla zona di impennata del FDpSpRatio tra 4 e 5.

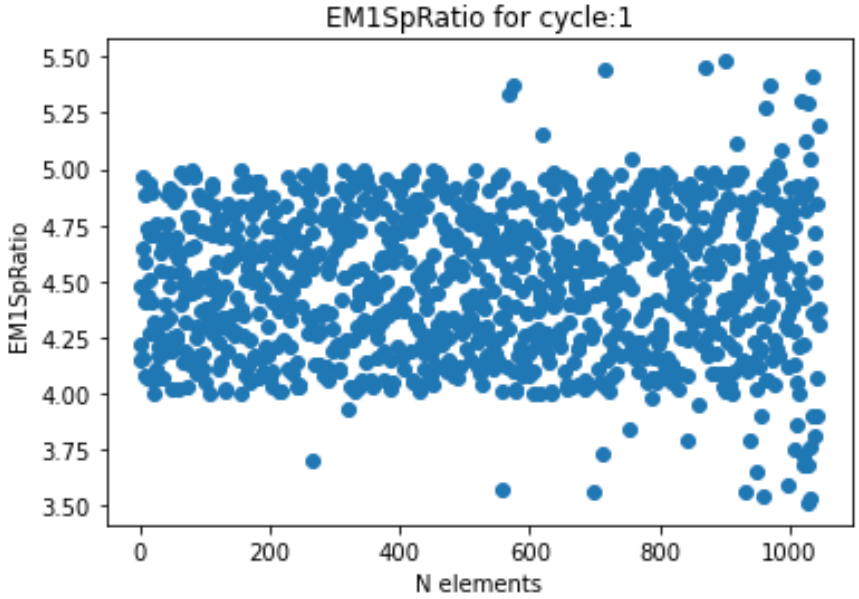


Figura 36:Andamento dell'EM1SpRatio

Infine per l'EM1SpRatio non si nota una impennata netta, ma anche questa feature, nella zona corrispondente all'impennata della CO2 presenta una distribuzione diversa dal resto dei layout.

Si nota come effettivamente per le features riportate vi siano delle distribuzioni diverse dei punti nella stessa zona in cui si notano delle emissioni di CO2 più elevate.
Le features non riportate invece risultano uniformemente distribuite per tutto lo spettro dei valori di CO2.
Ottenuti questi risultati ci sia aspetta quindi che il DBSCAN possa essere in grado di individuare delle differenze con i restanti punti e identificare correttamente i presunti outlier.
Dopo aver eliminato gli outlier individuati da DBSCAN ciò che si ottiene è:

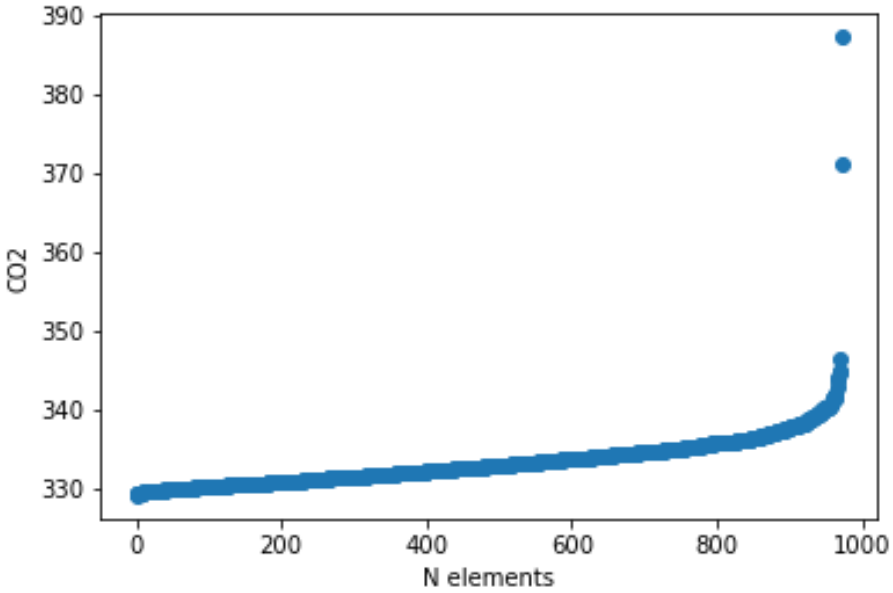


Figura 37:Rimozione outliers

La maggior parte dei layout ad alte emissioni di CO2 e separati dalla distribuzione continua sono stati eliminati, invece i punti rimanenti sono caratterizzati da valori delle feature nel range dei valori medi e non eliminati in quanto non ritenuti outlier.
A questo punto si ripete questa procedura anche per gli altri cicli dopo di che si ricompone il dataset complessivo depurato dagli outlier e si valutano le performance della rete.

7.2 Risultati della pipeline

Nella seguente tabella vengono riportati i risultati relativi alla fase di testing sia per il classificatore che per il regressore sia per il dataset depurato dagli outlier che quello non depurato.

original dataset	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	98,00%	95,48%	0,1	4,01	98,47%	3,59	12,01	0	1,03	3,3	0
Simulazione 2	97,33%	93,96%	0,2	5,26	97,30%	4,76	20,46	0,02	1,42	5,5	0,01
Simulazione 3	98,00%	95,46%	0,1	3,08	99,12%	2,69	11,86	0,01	0,76	3,01	0
Simulazione 4	97,33%	93,90%	0	3,29	99,17%	2,65	11,48	0,03	0,79	3,73	0,01
MEDIA	97,67%	94,70%	0,10	3,91	98,52%	3,4225	13,9525	0,015	1	3,885	0,005
STD	0,33%	0,77%	0,070711	0,8523204	0,007539	0,858818	3,76207	0,01118	0,264102	0,966967	0,005

CO2 outlier dataset	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	96,76%	92,83%	0	4,68	97,89%	4,09	17,43	0,03	1,2	4,41	0,01
Simulazione 2	97,54%	94,26%	0,2	2,47	99,47%	1,93	12,95	0,01	0,57	3,61	0
Simulazione 3	96,67%	92,21%	0,2	3,21	99,40%	2,5	11,74	0,03	0,77	3,09	0,01
Simulazione 4	97,40%	93,96%	0,1	2,62	99,41%	2,38	11,83	0,01	0,69	3,85	0
MEDIA	97,09%	93,32%	0,13	3,245	99,04%	2,725	13,4875	0,02	0,8075	3,74	0,005
STD	0,38%	0,83%	0,082916	0,8734558	0,006659	0,816226	2,325579	0,01	0,237526	0,474447	0,005

Tabella 1 Confronto performance Dataset completo-Dataset depurato

Al contrario di quanto atteso, i risultati in fase di classificazione risultano leggermente peggiori rispetto al dataset non modificato, il che sta ad indicare che i punti rimossi non sono outlier che influenzano negativamente l'apprendimento della rete, ma al contrario sono punti rappresentativi dai quali la rete apprende.
Invece in termini di regressione le prestazioni risultano effettivamente leggermente migliorate, ma nel complesso si ritiene che reputare quei punti come outlier sarebbe stato un errore che è importante aver evitato per non rimuovere dei punti dai quali il classificatore è in grado di imparare.

8 Analisi e risultati sulla zona di sovrapposizione

Una volta trovato il modo migliore per individuare la zona di sovrapposizione, e cioè quella automatica, la prima prova portata avanti è stata la valutazione delle performance proprio sulla zona di sovrapposizione.

8.1 Confronto tra zona di sovrapposizione automatica e manuale

Per completezza è stato comunque portato avanti un confronto tra i risultati ottenuti sulla zona di sovrapposizione individuata automaticamente dall'algoritmo sviluppato e quelli della zona individuata manualmente.
Quest'analisi consiste nell'allenare, validare e testare la rete sulla sola zona di sovrapposizione.
I risultati che seguono sono quelli ottenuti in training e validation, che riportano l'andamento di Accuracy ed MCC (metriche per la valutazione delle perfomance) oltre che l'andamento della loss function in funzione del numero di epoche:

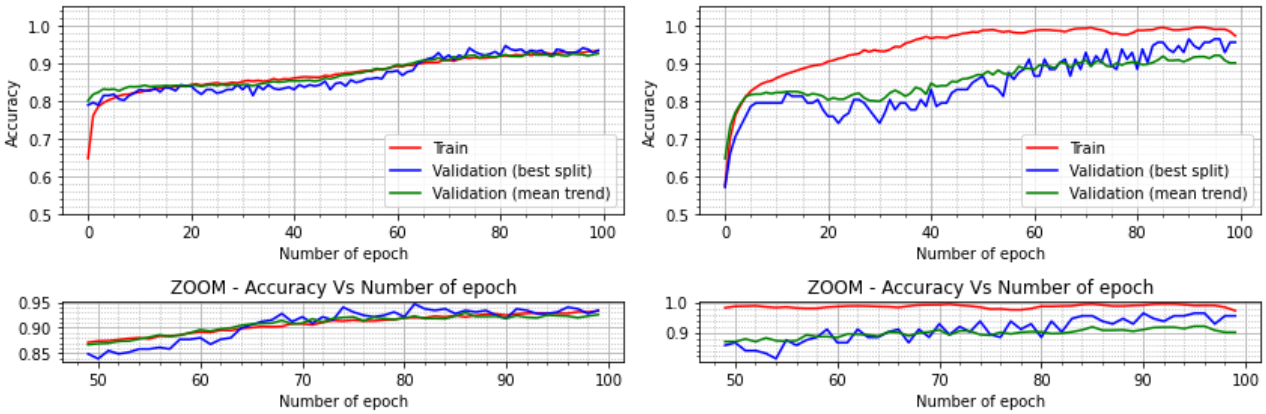


Figura 38:Accuracy zona di sovrapposizione automatica(sinistra) e manuale(destra)

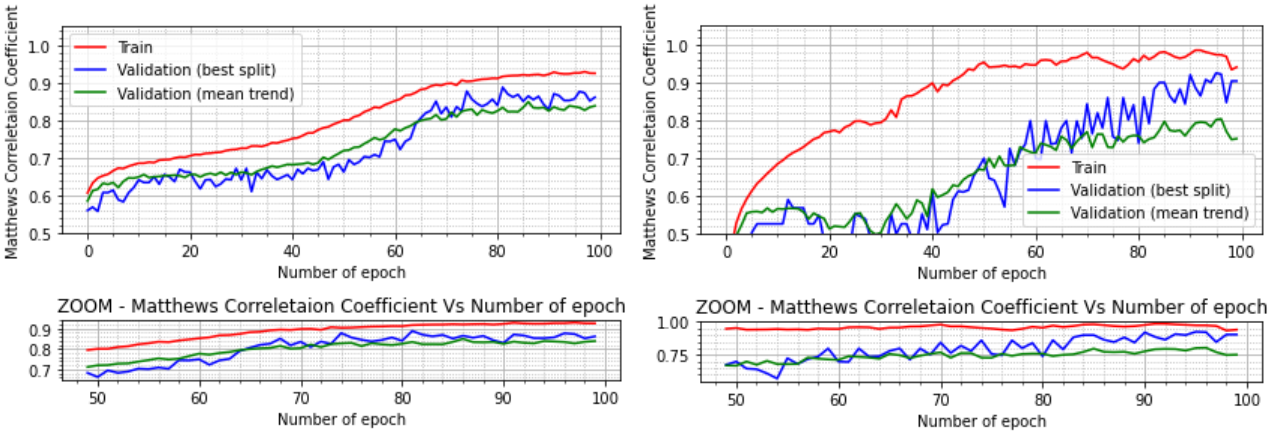


Figura 39:MCC zona di sovrapposizione automatica(sinistra) e manuale(destra)

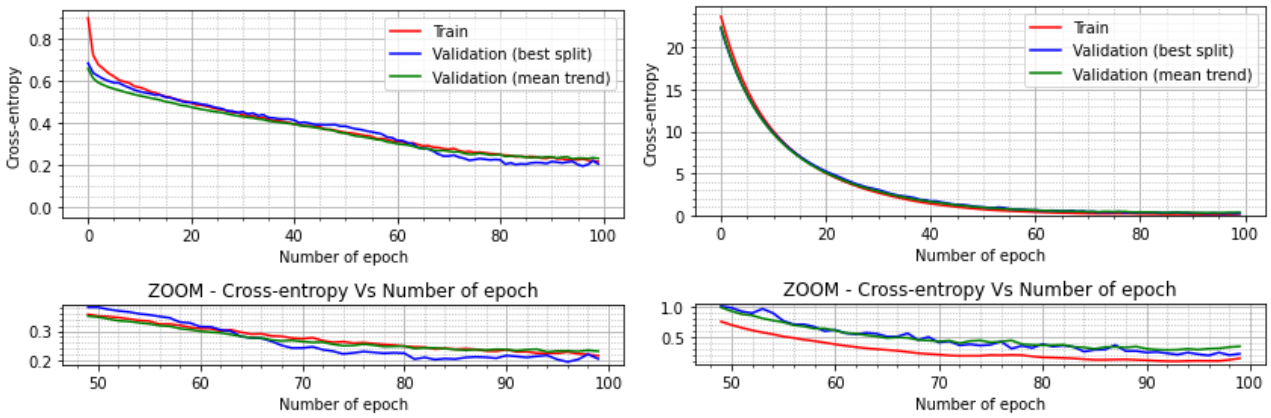


Figura 40: Cross-entropy zona sovrapp. automatica(sinistra) e manuale(destra)

Come si vede dai grafici riportati nella zona di sovrapposizione individuata in maniera automatica (che prende in considerazione una zona più specifica e più numerosa) le performances in fase di training sono peggiori rispetto a quelle della zona individuata manualmente. Questo conferma come la zona individuata in questo modo sia effettivamente più rappresentativa della sovrapposizione tra feasible ed unfeasible layouts. In questa zona i risultati in validation sono più simili al training rispetto alla zona di sovrapposizione manuale, facendo auspicare che se si riuscissero a migliorare le prestazioni in training migliorerebbero anche quelle in validation (e test).
Queste sono inoltre già migliori di quelle ottenute con il dataset selezionato manualmente.
Andando a vedere i risultati per il set di test si ottiene:
Test score (Zona di sovrapposizione automatica)

- Accuracy on test: 93.29%
- Area Under the Curve on test: 99.01%
- Precision on test: 98.22%
- Recall on test: 91.21%
- F1_score on test: 94.59%
- Matthews correlation coefficient on test: 86.19%

Test score (Zona di sovrapposizione manuale)

- Accuracy on test: 89.90%
- Area Under the Curve on test: 96.73%
- Precision on test: 80.00%
- Recall on test: 85.71%
- F1_score on test: 82.76%
- Matthews correlation coefficient on test: 75.72%

Ripetendo più volte le simulazioni si ottengono i seguenti risultati:

Zona sovrapp. Manuale	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	89,90%	75,72%	0	5,81	94,26%	3,55	15,17	0,12	1,07	4,21	0,04
Simulazione 2	95,96%	90,04%	0,2	5,75	95,90%	4,29	11,28	0,03	1,28	3,82	0,01
Simulazione 3	89,90%	75,10%	0	9,68	87,33%	7,3	21,64	1,26	2,25	6,02	0,41
MEDIA	91,92%	80,29%	0,07	7,08	92,50%	5,04667	16,03	0,47	1,53333	4,68333	0,15333
STD	2,86%	6,90%	0,09428	1,8386408	0,03714	1,62173	4,27295	0,55982	0,51396	0,95848	0,1819

Zona sovrapp. Automatica	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	93,29%	86,19%	0,3	6,38	95,26%	5,39	21,56	0,27	1,55	6,4	0,08
Simulazione 2	90,81%	79,78%	0,3	2,79	99,39%	2,12	8,95	0,03	0,66	2,53	0,01
Simulazione 3	92,23%	82,94%	0,2	7,65	94,44%	6,1	56,49	0,12	1,85	15,3	0,03
MEDIA	92,11%	82,97%	0,27	5,6066667	96,36%	4,53667	29	0,14	1,35333	8,07667	0,04
STD	1,02%	2,62%	0,04714	2,0580627	0,02166	1,73325	20,1085	0,09899	0,50533	5,34644	0,02944

Tabella 2:Confronto performance zona sovrapposizione manuale-automatica

Come si può notare i risultati relativi alla fase di testing sono confrontabili con quelli di validation in entrambi i casi. I valori medi sono abbastanza simili a causa della seconda simulazione relativa alla zona manuale, che presenta valori molto più alti e che vanno ad aumentare sia la media che la deviazione standard e facendo apparire i risultati molto simili, anche se comunque leggermente migliori per la zona di sovrapposizione automatica.

8.2 Incremento numero di epoche

Per entrambi i casi gli andamenti ottenuti lasciano immaginare la possibilità che le performance possano migliorare ulteriormente aumentando il tempo di allenamento. Si è quindi andati ad aumentare il numero di epoche di training per verificare l'eventuale presenza di underfitting e, nel caso, risolverlo.

- Zona di sovrapposizione manuale:

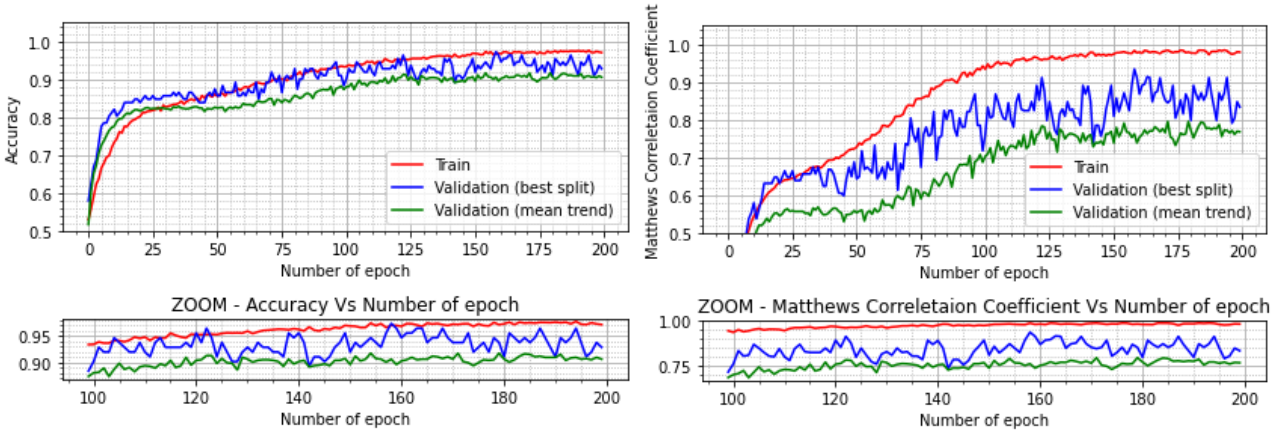


Figura 41:Accuracy (sinistra) MCC (destra) zona di sovrapposizione manuale 200Epochs

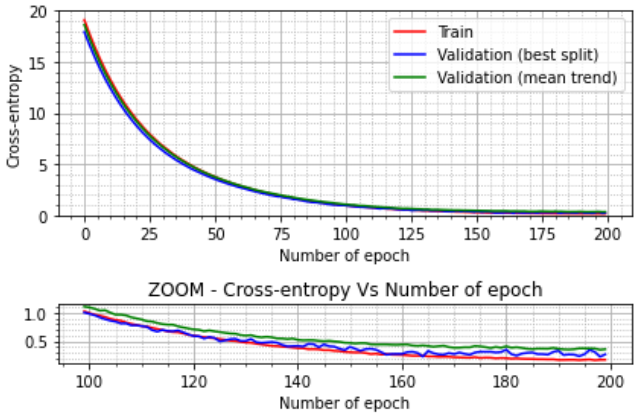


Figura 42:cross-entropy zona di sovrapposizione manuale 200Epochs

- Zona di sovrapposizione automatica:

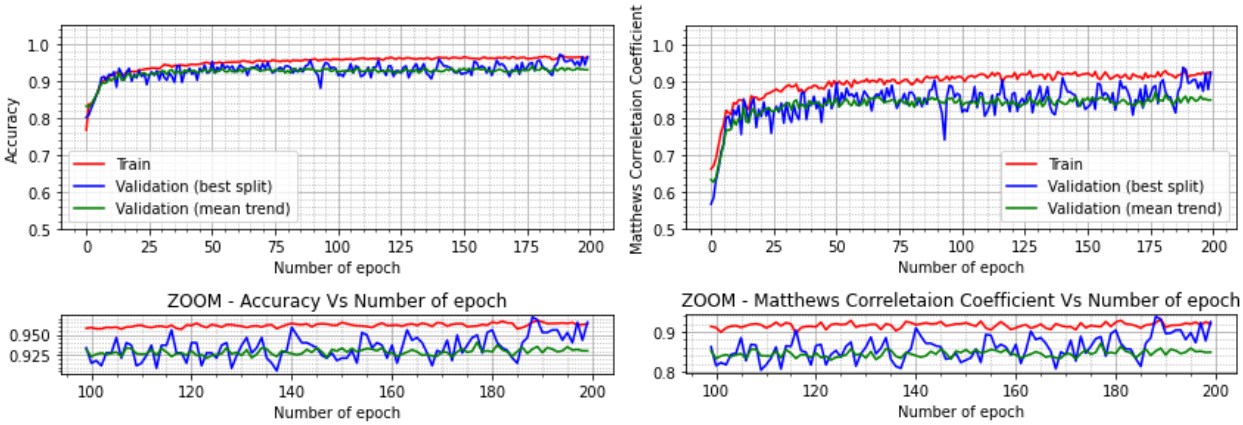


Figura 43:Accuracy (sinistra) MCC (destra) zona di sovrapposizione automatica 200Epochs

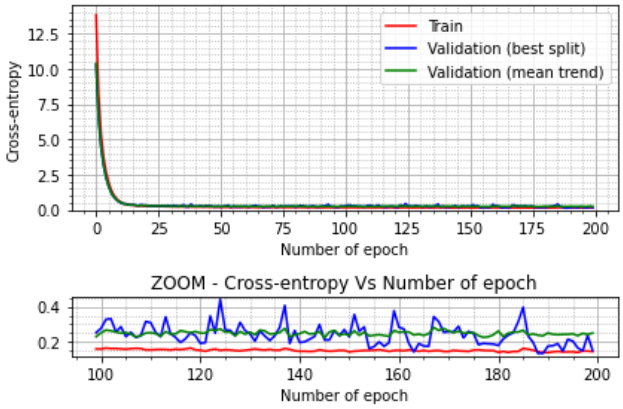


Figura 44: cross-entropy zona di sovrapposizione automatica 200Epochs

Già da questi grafici si può osservare che i risultati raggiunti in training e validation sono molto simili a quelli ottenuti con 100 epoche. Per completezza si riportano anche i valori ottenuti in fase di test:

Zona sovrapp. Manuale	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	89,90%	75,72%	0	5,81	94,26%	3,55	15,17	0,12	1,07	4,21	0,04
Simulazione 2	95,96%	90,04%	0,2	5,75	95,90%	4,29	11,28	0,03	1,28	3,82	0,01
Simulazione 3	89,90%	75,10%	0	9,68	87,33%	7,3	21,64	1,26	2,25	6,02	0,41
MEDIA	91,92%	80,29%	0,07	7,08	92,50%	5,04667	16,03	0,47	1,53333	4,68333	0,15333
STD	2,86%	6,90%	0,09428	1,8386408	0,03714	1,62173	4,27295	0,55982	0,51396	0,95848	0,1819

Zona sovrapp. Manuale 200Epoche	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	93,94%	85,06%	0,1	11,9	79,92%	6,15	48,36	0,06	1,8	13,1	0,02
Simulazione 2	89,90%	74,65%	0,2	16,75	71,85%	6,4	73,71	0,35	1,8	18,98	0,11
Simulazione 3	91,92%	80,08%	0	23,27	14,66%	22,34	31,3	7	7,06	9,47	1,92
MEDIA	91,92%	79,93%	0,10	17,306667	55,48%	11,63	51,1233	2,47	3,55333	13,85	0,68333
STD	1,65%	4,25%	0,08165	4,6584428	0,29049	7,5738	17,4237	3,20538	2,47959	3,91849	0,87523

Tabella 3: Confronto Performance Zona di sovrapposizione manuale 100 vs 200 Epoche

Da questo confronto si può notare come in media le prestazioni ottenibili raddoppiando le epoche di training non siano migliori di quelle con 100 epoche, e anche considerando la prova 2 a 100 epoche come un outlier, l'incremento di performance apportato dall'incremento di epoche non sarebbe tale da giustificare la durata della fase di allenamento e la scelta di questa zona di sovrapposizione.

Zona sovrapp. Automatica	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	93,29%	86,19%	0,3	6,38	95,26%	5,39	21,56	0,27	1,55	6,4	0,08
Simulazione 2	90,81%	79,78%	0,3	2,79	99,39%	2,12	8,95	0,03	0,66	2,53	0,01
Simulazione 3	92,23%	82,94%	0,2	7,65	94,44%	6,1	56,49	0,12	1,85	15,3	0,03
MEDIA	92,11%	82,97%	0,27	5,6066667	96,36%	4,53667	29	0,14	1,35333	8,07667	0,04
STD	1,02%	2,62%	0,04714	2,0580627	0,02166	1,73325	20,1085	0,09899	0,50533	5,34644	0,02944

Zona sovrapp. Automatica (200Epoche)	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	91,52%	81,39%	0	4,19	98,40%	3,27	23,76	0,02	1,04	7,72	0,01
Simulazione 2	90,81%	79,78%	0,1	5,87	97,31%	4,82	18,25	0,05	1,45	5,74	0,01
Simulazione 3	93,29%	85,41%	0,3	8,05	94,06%	6,34	24,27	0,03	1,92	7,25	0,01
MEDIA	91,87%	82,19%	0,13	6,0366667	96,59%	4,81	22,0933	0,03333	1,47	6,90333	0,01
STD	1,04%	2,37%	0,12472	1,5802391	0,01843	1,25334	2,72561	0,01247	0,35954	0,84468	0

Tabella 4: Confronto Performance Zona di sovrapposizione automatica 100 vs 200 Epoche

In entrambi i casi i risultati sono molto simili per 100 e per 200 epoche, quindi non vale la pena allenare la rete per il doppio delle epoche senza migliorarne le prestazioni.
Inoltre per le analisi successive non si prenderà più in considerazione la zona di sovrapposizione manuale in quanto: meno accurata, presenta performance comunque peggiori, presenta una minore numerosità di dati.

8.3 Confronto tra zona di sovrapposizione e dataset completo

A questo punto si opera un secondo confronto. Questa volta tra i risultati ottenuti sulla zona di sovrapposizione e quelli ottenuti sul dataset completo.
I risultati confermano le attese, e cioè che le prestazioni sono migliori sull'intero dataset in quanto questo contiene la zona di sovrapposizione, ma anche tutti i dati restanti che alzano i valori delle metriche utilizzate per la valutazione delle performance:

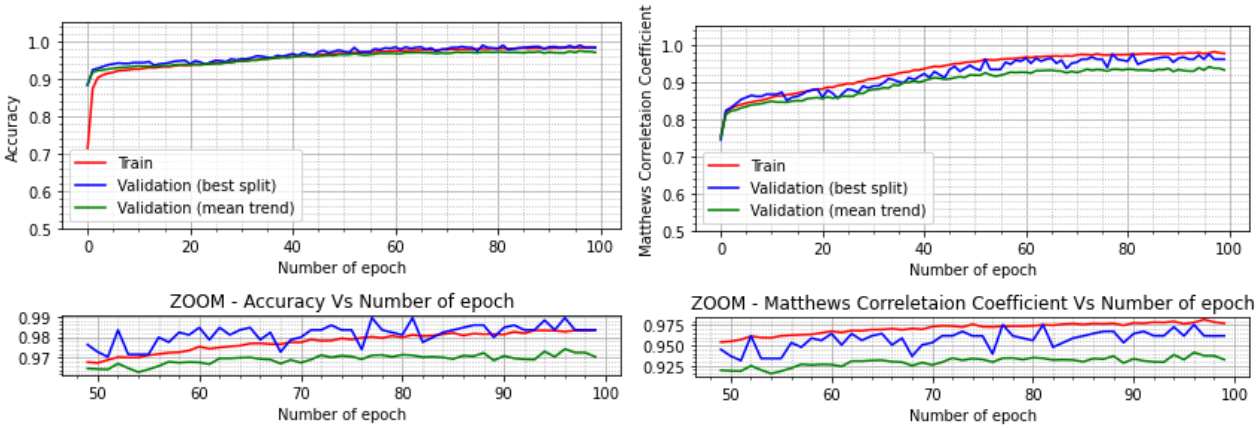


Figura 45:Accuracy (sinistra), MCC (destra) per Dataset completo

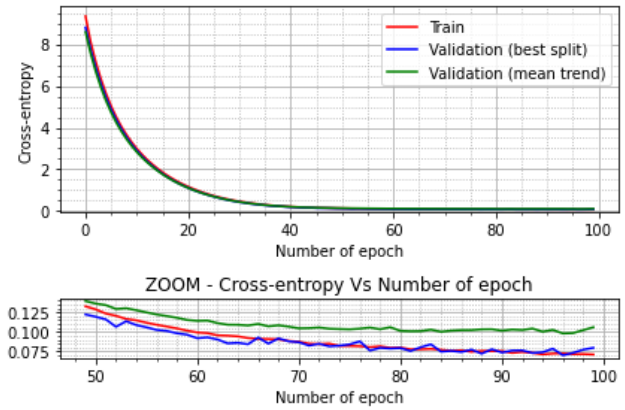


Figura 46: cross-entropy per Dataset completo

Si può notare immediatamente come non solo vi sia una maggiore vicinanza tra i risultati in allenamento ed in validazione, ma anche come le prestazioni siano migliori in senso assoluto.

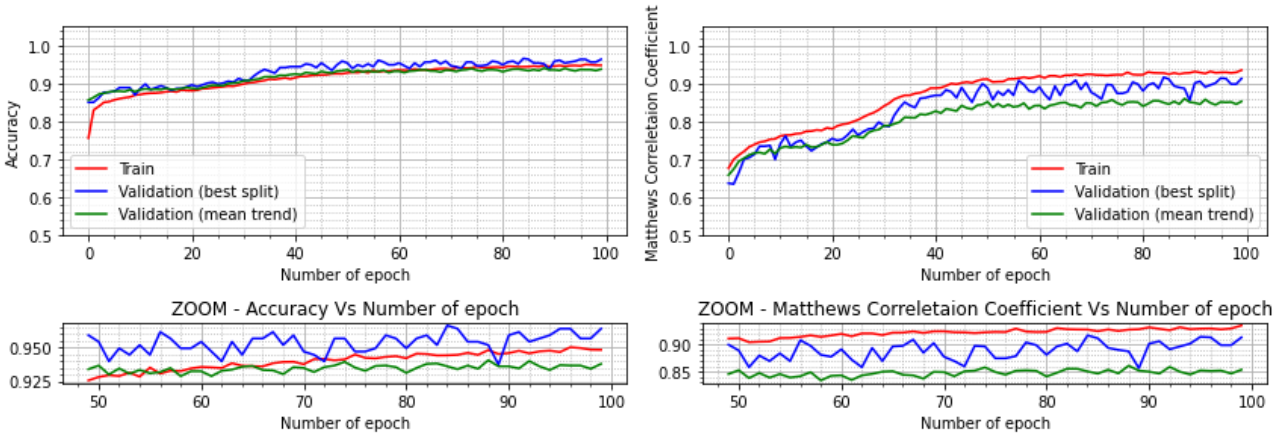


Figura 47:Accuracy (sinistra), MCC (destra) per Zona di sovrapposizione

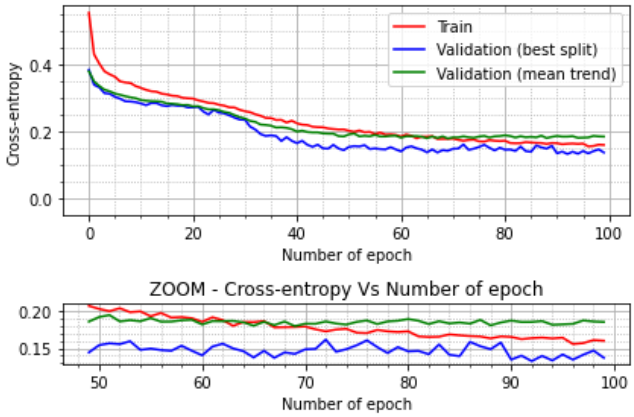


Figura 48: cross-entropy per Zona di sovrapposizione

Nel complesso i risultati di classificatore e regressore sono riportati nel confronto di 4 simulazioni per ciascuno dei due dataset. Anche in fase di test i risultati sono coerenti con quanto appena visto per allenamento e validazione, risultando migliori per il dataset completo:

original dataset	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	98,00%	95,48%	0,1	4,01	98,47%	3,59	12,01	0	1,03	3,3	0
Simulazione 2	97,33%	93,96%	0,2	5,26	97,30%	4,76	20,46	0,02	1,42	5,5	0,01
Simulazione 3	98,00%	95,46%	0,1	3,08	99,12%	2,69	11,86	0,01	0,76	3,01	0
Simulazione 4	97,33%	93,90%	0	3,29	99,17%	2,65	11,48	0,03	0,79	3,73	0,01
MEDIA	97,67%	94,70%	0,10	3,91	98,52%	3,4225	13,9525	0,015	1	3,885	0,005
STD	0,33%	0,77%	0,07071	0,8523204	0,00754	0,85882	3,76207	0,01118	0,2641	0,96697	0,005

Zona sovrapp. 7D	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	94,81%	88,81%	0,2	6,79	91,25%	5,14	19,81	0,21	1,63	6,37	0,06
Simulazione 2	91,55%	79,92%	0,2	2,9	98,58%	2,05	6,98	0,2	0,63	1,85	0,07
Simulazione 3	85,71%	69,39%	0,1	3,74	97,55%	2,97	8,35	0,1	0,91	2,31	0,03
Simulazione 4	88,37%	73,60%	0	3,29	98,81%	2,77	5,9	0,63	0,85	1,82	0,17
MEDIA	90,11%	77,93%	0,13	4,18	96,55%	3,2325	10,26	0,285	1,005	3,0875	0,0825
STD	3,41%	7,31%	0,08292	1,5359199	0,03095	1,15322	5,58164	0,20378	0,3756	1,90508	0,05262

Tabella 5:Confronto dataset completo e zona di sovrapposizione

8.4 Train/val su zona di sovrapposizione o dataset completo

Un ulteriore confronto ha permesso di capire che le prestazioni sulla zona di sovrapposizione risultano migliori allenando (e quindi validando) la rete sul dataset completo, per poi svolgere la sola fase di testing sulla zona di sovrapposizione, infatti proprio dal confronto tra i risultati di test si ottiene quanto riportato:

Zona sovrapp. 7D	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	94,81%	88,81%	0,2	6,79	91,25%	5,14	19,81	0,21	1,63	6,37	0,06
Simulazione 2	91,55%	79,92%	0,2	2,9	98,58%	2,05	6,98	0,2	0,63	1,85	0,07
Simulazione 3	85,71%	69,39%	0,1	3,74	97,55%	2,97	8,35	0,1	0,91	2,31	0,03
Simulazione 4	88,37%	73,60%	0	3,29	98,81%	2,77	5,9	0,63	0,85	1,82	0,17
MEDIA	90,11%	77,93%	0,13	4,18	96,55%	3,2325	10,26	0,285	1,005	3,0875	0,0825
STD	3,41%	7,31%	0,08292	1,5359199	0,03095	1,15322	5,58164	0,20378	0,3756	1,90508	0,05262

Zona sovrapp. 7D solo test	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	93,59%	83,88%	0	5,06	97,95%	3,77	20,98	0	1,17	6,3	0
Simulazione 2	92,88%	83,92%	0,1	2,97	99,32%	2,13	18,36	0	0,63	5,78	0
Simulazione 3	93,00%	83,36%	0,1	2,4	99,49%	2,03	8,11	0,01	0,6	2,34	0
Simulazione 4	93,51%	84,42%	0	1,43	99,82%	0,95	8,26	0	0,28	2,72	0
MEDIA	93,25%	83,90%	0,05	2,965	99,15%	2,22	13,9275	0,0025	0,67	4,285	0
STD	0,31%	0,38%	0,05	1,3289564	0,00713	1,00742	5,81697	0,00433	0,31961	1,76971	0

Tabella 6: Confronto risultati zona di sovrapposizione con allenamento su zona di sovrapposizione vs dataset completo

Non si riporta il confronto sui risultati relativi alla fase di allenamento e validazione in quanto sono gli stessi del confronto precedente, dal quale in sostanza differisce solo la zona di test e quindi i risultati riportati.
Questi risultati forniscono il punto di partenza per le analisi successive.

8.5 Nuova architettura del modello

Ottenute sia la zona di sovrapposizione che le migliori prestazioni su essa ottenibili utilizzando la stessa e sola pipeline di reti neurali usata in precedenza per il dataset completo, si passa ad una fase di ottimizzazione che possa, creando una pipeline che vada a lavorare in maniera indipendente sulla zona di sovrapposizione, aumentare le prestazioni dello strumento.
L'idea che sta alla base delle analisi successive è che le prestazioni della rete sulla zona di sovrapposizione possano migliorare cambiando gli iper-parametri che vanno a definire la rete stessa.
Per cambiare gli iper-parametri bisogna cambiare l'iperspazio all'interno del quale vengono cercati. La direzione in cui cambiarli è sconosciuta quindi sono necessari diversi tentativi e simulazioni.
Per questo motivo, data la lunghezza della fase di training necessaria con un dataset composto da dati appartenenti a 5 cicli diversi, si è fatta la scelta di utilizzare un dataset composto dai dati relativi ad un solo ciclo guida, il World Harmonized Vehicle Cycle (whvc), così da ridurre i tempi di simulazione. Si passa quindi da un'analisi cycle-independent (indipendente dal ciclo guida da portare a termine) ad una cycle-dependent, ossia, con una feature esplicativa della missione di guida.
L'architettura che si intende quindi realizzare è la seguente:

- Prendere il dataset grezzo (cycle-dependent) e applicare l'algoritmo per definire la zona di sovrapposizione ed eliminare eventuali outliers. Algoritmo che può essere chiamato "multi-dimensional Euclidean distances - with DBSCAN".
- Separare la zona di sovrapposizione dal resto dei punti.
- Creare due pipeline che lavorano in parallelo, entrambe allenate sull'intero dataset ma con una calibrazione degli iper-parametri diversa.
- Salvare gli output delle reti (classificazione feasibility e regressione CO2).
- Riunire gli output delle reti in un unico database che faccia riferimento al dataset originale.

8.6 Dataset ridotto

Per prima cosa si riportano le differenze tra i risultati ottenuti sulla zona di sovrapposizione per il dataset completo e quelli relativi al dataset composto dal solo ciclo whvc.
La zona di sovrapposizione relativa alla condizione cycle-dependent, riportata nel piano PEratio – EM1Power è molto simile a quella del dataset complessivo:

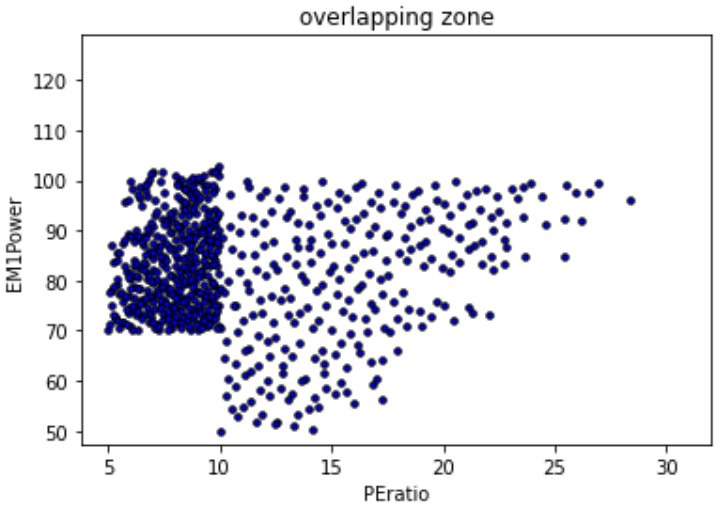


Figura 49:Zona di sovrapposizione whvc

Per quanto riguarda il confronto a parità di iperspazio i risultati sono i seguenti:
I risultati della rete (training e validation) sul dataset ridotto sono quelli nelle figure a sinistra tra le seguenti, confrontati con i risultati ottenuti in condizioni cycle-indipendent. Da quello che si può vedere i risultati sono abbastanza simili:

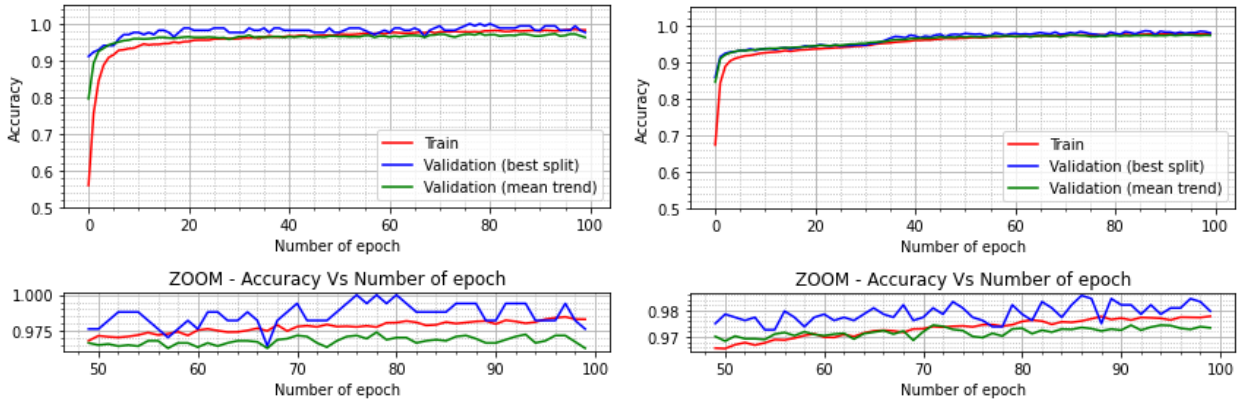


Figura 50 Accuracy cycle-dependent (sinistra), cycle-independent (destra)

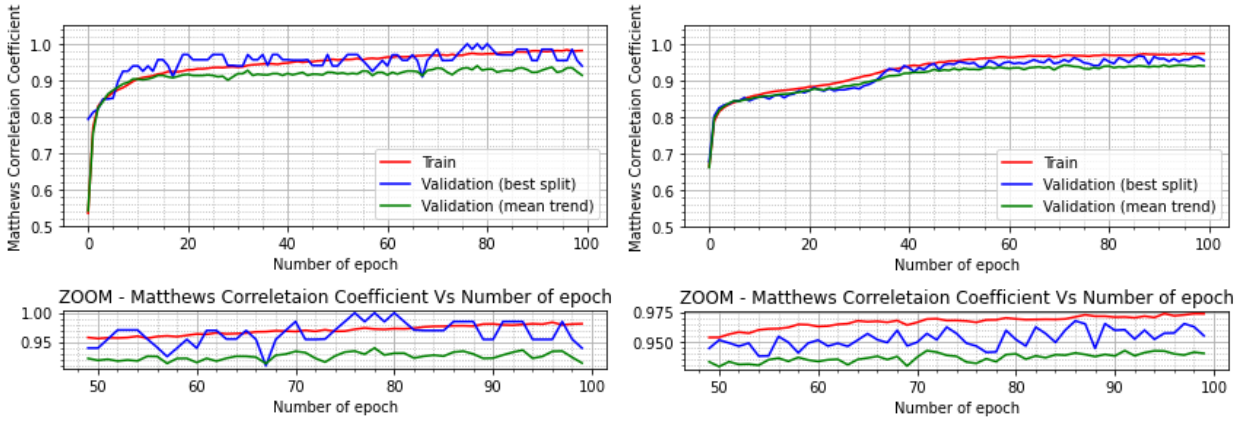


Figura 51 MCC cycle-dependent (sinistra), cycle-independent (destra)

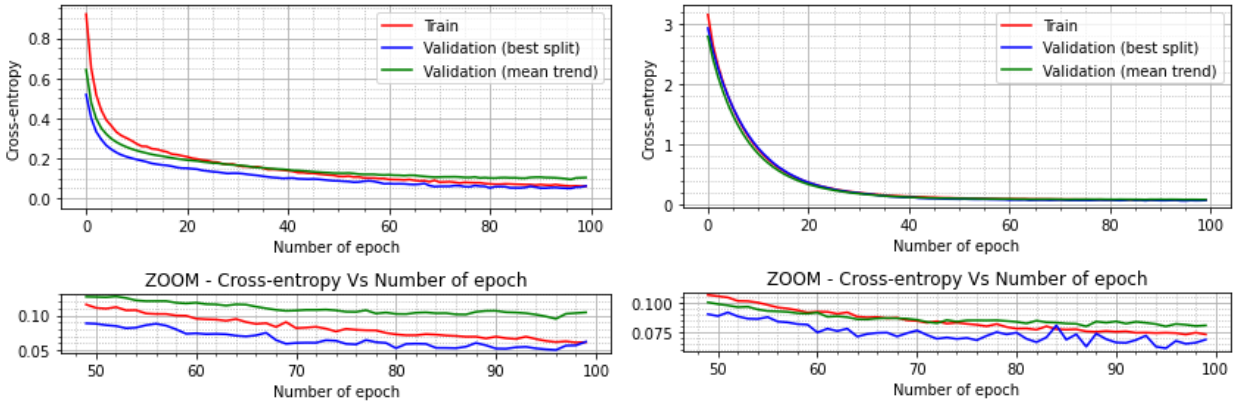


Figura 52 Cross-entropy cycle-dependent (sinistra), cycle-independent (destra)

L'MCC è la metrica che presenta maggiore differenza tra il caso cycle-independent e quello cycle-dependent.
Inoltre i risultati sul dataset di test per la condizione cycle-dependent sono più bassi:

original hyperspace	CLASSIFICATORE FEASIBILITY		REGRESSORE CO2							
					Absolute Error			Relative Error		
	Accuracy	MCC	Loss Testir	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	86,36%	67,63%	2,57	80,56%	2,03	13,38	0,01	0,59	3,65	0
Simulazione 2	85,71%	68,40%	0,97	88,31%	0,66	2,78	0	0,2	0,83	0
Simulazione 3	92,42%	81,79%	3,33	6,77%	2,26	19,15	0,05	0,66	5,53	0,01
Simulazione 4	91,80%	80,67%	1,09	95,48%	0,87	4,21	0,01	0,26	1,24	0
Simulazione 5	93,55%	82,41%	1,33	94,74%	1,06	4,59	0,09	0,32	1,28	0,03
Simulazione 6	91,55%	79,71%	1,9	56,34%	1,45	10,1	0,04	0,43	2,94	0,01
Simulazione 7	91,07%	79,89%	1,29	77,51%	0,98	3,56	0,01	0,29	1,08	0
Simulazione 8	89,23%	76,44%	1,61	57,80%	1,31	7,65	0	0,39	2,25	0
MEDIA	90,21%	77,12%	1,76	0,70	1,33	8,18	0,03	0,39	2,35	0,01
STD	0,026724494	0,055151762	0,761732	0,2760698	0,527441	5,367746	0,029554	0,150976	1,519079	0,009922

Tabella 7 Risultati con iperspazio originale ciclo whvc

Rispetto a quelli ottenuti sempre testando sulla sola zona di sovrapposizione ma cycle-independent:

Zona sovrapp. 7D solo test	CLASSIFICATORE FEASIBILITY			REGRESSORE CO2							
						Absolute Error			Relative Error		
	Accuracy	MCC	Dropout	Loss Testing	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	93,59%	83,88%	0	5,06	97,95%	3,77	20,98	0	1,17	6,3	0
Simulazione 2	92,88%	83,92%	0,1	2,97	99,32%	2,13	18,36	0	0,63	5,78	0
Simulazione 3	93,00%	83,36%	0,1	2,4	99,49%	2,03	8,11	0,01	0,6	2,34	0
Simulazione 4	93,51%	84,42%	0	1,43	99,82%	0,95	8,26	0	0,28	2,72	0
MEDIA	93,25%	83,90%	0,05	2,965	99,15%	2,22	13,9275	0,0025	0,67	4,285	0
STD	0,31%	0,38%	0,05	1,3289564	0,00713	1,00742	5,81697	0,00433	0,31961	1,76971	0

Tabella 8Results original hyperspace cycle-independent

Ancora una volta la differenza maggiore è quella sul Mattews Correlation Coefficient.

8.7 Ottimizzazione iperspazio

8.7.1 Pipeline per la zona di sovrapposizione

A questo punto, considerando la nuova architettura che si desidera implementare, ciò che manca è il tuning degli iper-parametri delle due nuove pipeline che lavoreranno in parallelo.
Si presenta di seguito l'iperspazio di partenza, ereditato dai progetti precedenti, e quello che porta i risultati migliori a seguito di vari tentativi:
Iperspazio iniziale:

- Learning rate: 0.0001 – 0.005
- Hidden Layers: 2 – 4
- Neurons first hidden layer: 30 – 140
- L2 regularizer: 0.0001 – 0.1
- Batch size: 32 – 256

Nuovo iperspazio:

- Learning rate: 0.01 – 1
- Hidden Layers: 2 – 6
- Neurons first hidden layer: 10 – 130
- L2 regularizer: 0.001 – 0.1
- Batch size: 32 – 256

I risultati in fase di training e validation sono abbastanza costanti per le diverse simulazioni effettuate, inoltre per non appesantire eccessivamente la trattazione non si riportano tutti i grafici ottenuti:

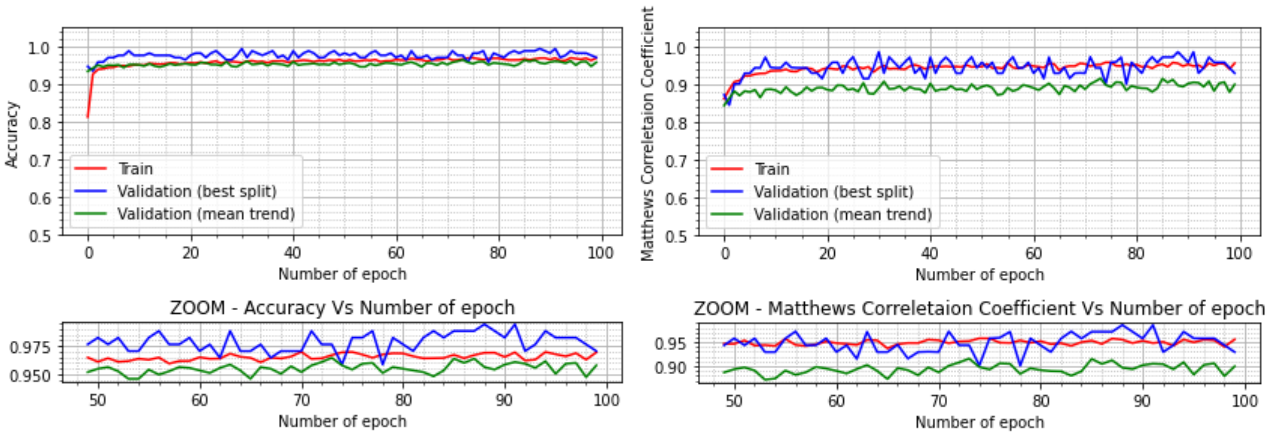


Figura 53: Accuracy (sinistra) MCC (destra) whvc nuovo iperspazio

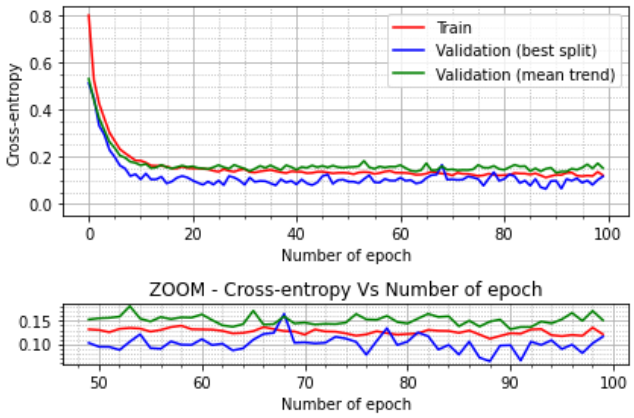


Figura 54: cross-entropy whvc nuovo iperspazio

Risulta ben evidente come i grafici siano in linea con quelli ottenuti con il precedente iperspazio, con delle performances in validation, soprattutto per l'MCC, peggiori rispetto al training e comunque sempre intorno al 90%.
Quello che si ottiene però in fase di testing è che le performance della rete nella zona di sovrapposizione risultano migliorate dal punto di vista del classificatore di feasibility, con un aumento sia dell'accuracy che dell' MCC.
Come conseguenza anche le performance in regressione risultano migliori, con una loss in testing minore, errori sia assoluti che relativi minori ed anche l'R on test è maggiore:

new hyperspace	CLASSIFICATORE FEASIBILITY		REGRESSORE CO2							
					Absolute Error			Relative Error		
	Accuracy	MCC	Loss Testir	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	94,20%	86,31%	1,42	73,72%	1,11	4,31	0,06	0,33	1,28	0,02
Simulazione 2	96,77%	92,17%	1,51	89,43%	1,09	3,41	0,01	0,33	1,01	0
Simulazione 3	97,18%	94,11%	1,29	93,03%	0,93	6,51	0,01	0,27	1,78	0
Simulazione 4	98,44%	96,38%	1,45	92,32%	0,94	6,23	0,01	0,28	1,87	0
Simulazione 5	89,23%	76,26%	1,31	92,15%	0,98	3,57	0,04	0,29	1,05	0,01
Simulazione 6	93,22%	83,18%	1,34	95,30%	0,87	6,94	0	0,26	1,93	0
Simulazione 7	95,45%	90,59%	1,94	86,40%	1,36	9,98	0,02	0,4	2,9	0,01
Simulazione 8	93,75%	86,06%	1,49	89,17%	1,22	4,71	0,04	0,36	1,31	0,01
MEDIA	94,78%	88,13%	1,47	88,94%	1,06	5,71	0,02	0,32	1,64	0,01
STD	0,026971744	0,061154798	0,193871	0,0629781	0,155222	2,045903	0,019325	0,045552	0,583919	0,00696

Tabella 9: Risultati con nuovo iperspazio ciclo whvc

original hyperspace	CLASSIFICATORE FEASIBILITY		REGRESSORE CO2							
					Absolute Error			Relative Error		
	Accuracy	MCC	Loss Testir	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	86,36%	67,63%	2,57	80,56%	2,03	13,38	0,01	0,59	3,65	0
Simulazione 2	85,71%	68,40%	0,97	88,31%	0,66	2,78	0	0,2	0,83	0
Simulazione 3	92,42%	81,79%	3,33	6,77%	2,26	19,15	0,05	0,66	5,53	0,01
Simulazione 4	91,80%	80,67%	1,09	95,48%	0,87	4,21	0,01	0,26	1,24	0
Simulazione 5	93,55%	82,41%	1,33	94,74%	1,06	4,59	0,09	0,32	1,28	0,03
Simulazione 6	91,55%	79,71%	1,9	56,34%	1,45	10,1	0,04	0,43	2,94	0,01
Simulazione 7	91,07%	79,89%	1,29	77,51%	0,98	3,56	0,01	0,29	1,08	0
Simulazione 8	89,23%	76,44%	1,61	57,80%	1,31	7,65	0	0,39	2,25	0
MEDIA	90,21%	77,12%	1,76	0,70	1,33	8,18	0,03	0,39	2,35	0,01
STD	0,026724494	0,055151762	0,761732	0,2760698	0,527441	5,367746	0,029554	0,150976	1,519079	0,009922

Tabella 10: Risultati con vecchio iperspazio ciclo whvc

Come si può vedere nella simulazione 5 ci sono dei casi in cui le performance risultano decisamente basse rispetto alle altre, ma il miglioramento nelle altre simulazioni è comunque notevole. Da questo si deduce che l'iperspazio selezionato potrebbe ancora non essere quello ottimale per la stabilità dei risultati, ma si potrebbe in parte attribuire la causa di questo al ridotto numero di dati.
Un'ulteriore osservazione importante è che in alcune simulazioni si ottengono risultati, ad esempio guardando l'MCC, migliori in test che in validation.
Stando a questi risultati la considerazione che si può fare è che l'iperspazio individuato forza la scelta di iper-parametri tali da far performare bene la rete in un dataset che è un sottogruppo di quello complessivo, ma è costituito allo stesso tempo da una distribuzione più complicata.
Per avere un idea di cosa abbia influenzato maggiormente i risultati relativi al nuovo iperspazio vengono riportati gli iper-parametri della nuova rete di classificazione a confronto con quelli della precedente:

original hyperspace	CLASSIFICATORE FEASIBILITY		IPERPARAMETRI					
	Accuracy	MCC	HIDDEN LA	Neuron first	Learning ra	L2 Regular	Batch size	Dropout
Simulazione 1	86,36%	67,63%	3	110	0,00345	0,00083	66	0,1
Simulazione 2	85,71%	68,40%	3	126	0,00441	0,00428	141	0
Simulazione 3	92,42%	81,79%	2	76	0,00334	0,00798	63	0,2
Simulazione 4	91,80%	80,67%	2	69	0,00116	0,0451	85	0,1
Simulazione 5	93,55%	82,41%	2	40	0,00372	0,0012	182	0,2
Simulazione 6	91,55%	79,71%	2	99	0,00238	0,02365	35	0
Simulazione 7	91,07%	79,89%	3	121	0,00489	0,00561	141	0,2
Simulazione 8	89,23%	76,44%	2	51	0,0009	0,01196	135	0,2
MEDIA	90,21%	77,12%	2,38	86,50	0,00	0,01	106,00	0,13
STD	0,026724494	0,055151762	0,484123	30,203477	0,001349	0,014083	47,43153	0,082916

Tabella 11: iper-parametri per la rete ottimizzata sul dataset completo

new hyperspace	CLASSIFICATORE FEASIBILITY		IPERPARAMETRI					
	Accuracy	MCC	HIDDEN LA	Neuron first	Learning ra	L2 Regular	Batch size	Dropout
Simulazione 1	94,20%	86,31%	2	112	0,01006	0,00881	78	0
Simulazione 2	96,77%	92,17%	2	91	0,02251	0,00858	119	0
Simulazione 3	97,18%	94,11%	3	43	0,01561	0,00157	81	0
Simulazione 4	98,44%	96,38%	3	49	0,01291	0,00231	92	0,1
Simulazione 5	89,23%	76,26%	3	103	0,01729	0,00108	119	0,2
Simulazione 6	93,22%	83,18%	3	110	0,01098	0,06719	198	0
Simulazione 7	95,45%	90,59%	3	45	0,01114	0,00168	45	0
Simulazione 8	93,75%	86,06%	3	11	0,01439	0,00194	101	0
MEDIA	94,78%	88,13%	2,75	70,50	0,01	0,01	104,13	0,04
STD	0,026971744	0,061154798	0,433013	35,651087	0,003856	0,021203	42,01915	0,069597

Tabella 12:iperparametri per la rete ottimizzata sulla zona di sovrapposizione

Come si può notare la differenza principale sta nel parametro relativo al learning rate, che per la rete ottimizzata per la zona di sovrapposizione è dell'ordine di 10^{-2} invece per la rete ottimizzata per il dataset completo è dell'ordine di 10^{-3}. L'altra differenza importante che si può notare è la profondità della rete stessa. La differenza non è elevata ma ciò che si nota è che nella maggior parte delle simulazioni (sei su otto) per la rete ottimizzata sulla zona di sovrapposizione è composta da 3 *hidden layers*, a differenza dei soli 2 che compongono la rete nelle restanti due simulazioni. Per quanto riguarda l'altra rete la situazione che si verifica è quasi opposta: in cinque simulazioni su otto il numero di *hidden layers* è 2, e solo nelle restanti tre questo sale a 3.

8.7.2 Pipeline zona di non sovrapposizione

Per quanto riguarda l'iperspazio relativo alla pipeline che lavora sulla zona di non sovrapposizione si usa lo stesso iperspazio utilizzato in partenza per la pipeline singola.
Infatti stando ai risultati ottenuti dovrebbe essere la sola zona di sovrapposizione a mettere in difficoltà la rete allenata con quella calibrazione degli iper-parametri.
Per questo motivo, allenando e validando la rete sull'intero dataset e cambiando solo la zona di testing ci si aspetta che i risultati in fase di test possano essere migliori rispetto a quelli in fase di allenamento e validazione.
Gli andamenti delle due metriche e della loss function sono più o meno sempre simili e dipendono dalle scelte degli iper-parametri, ciò che cambia sono i risultati in fase di testing (essendo cambiata la zona del dataset testata). In particolare i risultati confermano come sulla zona considerata sia decisamente più facile performare bene. Si ottiene infatti:

Zona di non sovrapposizione	CLASSIFICATORE FEASIBILITY		REGRESSORE CO2							
					Absolute Error			Relative Error		
	Accuracy	MCC	Loss Testir	R Testing	Mean	Max	Min	Mean	Max	Min
Simulazione 1	98,81%	97,09%	1,09	77,00%	0,77	3,72	0,01	0,23	1,11	0
Simulazione 2	98,86%	97,28%	1,31	77,34%	0,94	3,54	0,02	0,28	1,07	0
Simulazione 3	100,00%	100,00%	1,55	54,75%	0,99	6,07	0	0,3	1,8	0
Simulazione 4	98,78%	97,13%	1,97	42,37%	1,09	14,4	0,02	0,32	4,17	0,01
Simulazione 5	100,00%	100,00%	1,13	68,77%	0,96	2,85	0,03	0,29	0,86	0,01
Simulazione 6	100,00%	100,00%	1,47	78,43%	1,2	3,48	0,03	0,36	1,02	0,01
Simulazione 7	100,00%	100,00%	1,42	77,16%	1	3,08	0	0,3	0,91	0
Simulazione 8	100,00%	100,00%	0,98	85,40%	0,75	2,51	0,04	0,23	0,74	0,01
MEDIA	99,56%	98,94%	1,37	70,15%	0,96	4,96	0,02	0,29	1,46	0,01
STD	0,005732351	0,013725956	0,295042	0,1350774	0,140334	3,70943	0,013636	0,040755	1,067052	0,005

Tabella 13:Risultati sulla zona di non sovrapposizione ciclo whvc

Come si può notare per la maggior parte delle prove il classificatore prevede perfettamente l'appartenenza all'una o all'altra classe non sbagliando neanche una previsione:

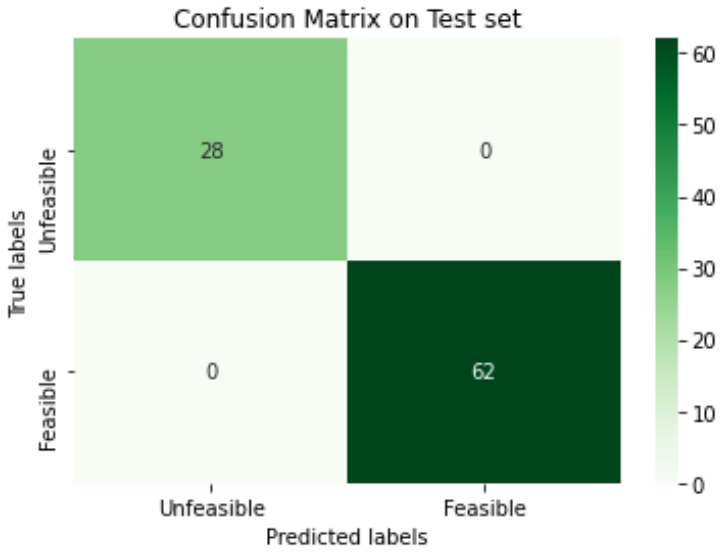


Figura 55: Confusion matrix su test set, zona di non sovrapposizione

Nei 3 casi in cui le metriche non sono pari al 100% la rete classifica in maniera errata un solo layout, le percentuali non sono le stesse solamente per la diversa numerosità del dataset di test. Inoltre questa diversa numerosità è dovuta al fatto che, anche se lo split ratio tra train_val_dataset e test_dataset è sempre 0.9, la composizione degli stessi non è costante, e quindi il numero di layout del dataset di test appartenenti alla zona di sovrapposizione e non può subire delle variazioni.

Si riportano infine le prestazioni medie complessive in classificazione (rete con una diversa calibrazione degli iper-parametri tra le due che compongono ciascuna pipeline), ottenibili dalle due pipeline in parallelo, unendo i risultati di ciascuno dei due rami:

- Accuracy = 98,05%
- Mattews Correlation Coefficient = 95,78%

Soprattutto per l'MCC l'incremento è importante rispetto ai valori medi ottenibili in fase di testing, sempre sull'intero dataset, ma ottenibile con una sola pipeline che lavora su tutto il dominio (Accuracy = 97,00% MCC = 92,74%).

Conclusioni

Per riassumere, il presente progetto ha utilizzato un algoritmo capace di individuare in maniera automatica una zona di dataset su cui è difficile performare bene. Gli obiettivi sono 2: il primo è eseguire la Classificazione ed ha l'obiettivo di produrre Previsioni di Fattibilità; il secondo è eseguire la regressione con l'obiettivo di prevedere le emissioni di CO2 Tank-to-Wheel.
I risultati delle simulazioni sono più che convincenti sia per la previsione di fattibilità che di regressione. L'algoritmo realizzato e la successiva fase di tuning degli iper-parametri permettono di ottenere delle performance nella zona di sovrapposizione decisamente migliori rispetto al caso iniziale.
Inoltre dai risultati ottenuti si può concludere che non solo la zona di sovrapposizione è quella che comporta più difficoltà alla rete di classificazione, ma che addirittura nella maggior parte dei casi sia l'unica zona in cui la rete sbaglia delle previsioni, il che fa capire quanto importante fosse focalizzarsi in modo diretto su questa zona.
Chiaramente sarebbe necessario svolgere altre simulazioni ed effettuare un tuning degli iper-parametri per il dataset complessivo composto dai 5 cicli a disposizione.
In conclusione, l'integrazione dell'algoritmo multi-dimensional Euclidean distances - with DBSCAN, in un'architettura di due pipeline di reti neurali è un ottimo strumento per aumentare ancora di più le performance di un tool che riscontrava delle difficoltà in una zona del dataset.
Eventuali sviluppi futuri potrebbero riguardare un'ulteriore fase di preprocessing, proprio tramite l'utilizzo di algoritmi di clustering. Partendo dai risultati ottenuti infatti si potrebbero applicare gli algoritmi già proposti in questo progetto ad esempio ad un dataset con una distribuzione diversa o casuale, al fine di generalizzare sempre più l'applicabilità del tool.

Bibliografia

[1] J. Miller, L. Du, D. Kodjak, and ICCT, "Impacts of World-Class Vehicle Efficiency and Emissions Regulations in Select G20 Countries," *ICCT - Int. Counc. Clean Transp.*, no. January, p. 17, 2017, [Online]. Available: https://www.theicct.org/sites/default/files/publications/ICCT_G20-briefing-paper_Jan2017_vF.pdf%0Ahttp://www.theicct.org/sites/default/files/publications/ICCT_G20-briefing-paper_Jan2017_vF.pdf.

[2] F. Un-Noor, S. Padmanaban, L. Mihet-Popa, M. N. Mollah, and E. Hossain, "A comprehensive study of key electric vehicle (EV) components, technologies, challenges, impacts, and future direction of development," *Energies*, vol. 10, no. 8, 2017, doi: 10.3390/en10081217.

[3] M. Venditti, "Innovative Models and Algorithms for the Optimization of Layout and Control Strategy of Complex Diesel HEVs," no. May, 2015.

[4] R. Finesso, "HEV layout optimization." Cirrincione, Giansalvo.

[5] A. Di Mauro, "Design optimization of Hybrid Electric Vehicles based on Deep Learning algorithms," p. 87316161, 2020.

[6] S. Quaranta, "Deep learning solution for the performance analysis of hybrid powertrains," no. March, 2021.

[7] G. Cirrincione, "deep learning." .

[8] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, doi: 10.1016/j.neunet.2014.09.003.

[9] M. J. Brown, L. A. Hutchinson, M. J. Rainbow, K. J. Deluzio, and A. R. De Asha, "A comparison of self-selected walking speeds and walking speed variability when data are collected during repeated discrete trials and during continuous walking," *J. Appl. Biomech.*, vol. 33, no. 5, pp. 384–387, 2017, doi: 10.1123/jab.2016-0355.

[10] S. Ruder, "An overview of gradient descent optimization algorithms," pp. 1–14, 2016, [Online]. Available: http://arxiv.org/abs/1609.04747.

[11] B. J. Taylor, M. A. Darrah, and C. D. Moats, "Verification and validation of neural networks: a sampling of research in progress," *Intell. Comput. Theory Appl.*, vol. 5103, p. 8, 2003, doi: 10.1117/12.487527.

[12] F. Nie, Z. Hu, and X. Li, "An investigation for loss functions widely used in machine learning," *Commun. Inf. Syst.*, vol. 18, no. 1, pp. 37–52, 2018, doi: 10.4310/cis.2018.v18.n1.a2.

[13] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[14] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[15] A. Barredo Arrieta *et al.*, "Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Inf. Fusion*, vol. 58, pp. 82–115, 2020, doi: 10.1016/j.inffus.2019.12.012.

[16] É. D. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, "Scikit-learn: Machine Learning in Python." 2011.

[17] "Elbow Method for optimal value of k in KMeans," 2021, [Online]. Available: www.geeksforgeeks.org.

[18] "Analisi delle componenti principali," [Online]. Available: https://it.wikipedia.org/wiki/Analisi_delle_componenti_principali.

[19] "DBSCAN." https://en.wikipedia.org/wiki/DBSCAN.

[20] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, pp. 1–13, 2020, doi: 10.1186/s12864-019-6413-7.

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.