

ALMA MATER STUDIORUM UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA GESTIONALE

Tesi di laurea in

MANUTENZIONE DEI SISTEMI DI PRODUZIONE

STUDIO E APPLICAZIONE DI MODELLI DI ARTIFICIAL
INTELLIGENCE PER
LA MANUTENZIONE PREDITTIVA DI COMPONENTI
MECCANICI

CANDIDATO

Francesco Porta

RELATORE

Chiar.mo Prof. Alberto Regattieri

CORRELATORI

Ing. Francesca Calabrese

Ing. Raffaele Piscitelli

Sessione III

Anno Accademico 2021/2022

Indice

1	Introduzione	1
2	Introduzione alla manutenzione	4
2.1	L'importanza della manutenzione aziendale.....	5
2.2	Le politiche manutentive	6
2.2.1	La manutenzione correttiva	7
2.2.3	L' industria 4.0 e il suo legame con la manutenzione predittiva	9
2.2.4	Manutenzione predittiva	10
3	Introduzione al machine learning	16
3.1	Tipi di apprendimento e categorizzazione	16
3.2	Il processo di apprendimento di una rete neurale	18
3.3	La scelta del modello migliore	21
4	Il deep learning	30
4.1	Generalità e vantaggi rispetto ai modelli non deep	30
4.2	Modelli convoluzionali.....	32
4.3	Modelli ricorrenti.....	36
5	Il metodo utilizzato e i risultati ottenuti.....	40
5.1	Il caso studio.....	41
5.2	La manipolazione dei dati di ingresso	43
5.3	Costruzione del modello convoluzionale	46
5.4	La programmazione del modello convoluzionale.....	48
5.5	Costruzione del modello convoluzionale con LSTM	50
5.6	Programmazione del modello convoluzionale con LSTM	52
5.7	Risultati ottenuti.....	54
5.8	Previsione in tempo reale del tempo di rottura delle cinghie	61
6	Analisi dei risultati e sviluppi futuri.....	68
6.1	Analisi dei risultati	68
6.2	Sviluppi futuri	71
7	Conclusioni	76

Indice delle figure

2.1 Strategia di tipo I di manutenzione preventiva.....	8
2.2 Strategia di tipo II di manutenzione preventive.....	8
2.3 Le tecnologie abilitanti dell'industria 4.0.....	9
2.4 Processi necessari per applicare manutenzione predittiva nel caso di metodi basati sui dati.....	14
3.1 Tipi di algoritmi presenti in letteratura di apprendimento automatico.....	17
3.2 Esempio di un algoritmo supervisionato: il classificatore di spam.....	17
3.3 Algoritmo supervisionato di regressione.....	18
3.4 Semplice rete neurale a singolo livello.....	19
3.5 Schematizzazione processo di apprendimento erraneo.....	22
3.6 Schematizzazione processo di apprendimento corretto.....	23
3.7 Differenza tra modelli con tanti parametri e con pochi parametri.....	25
3.8 Modalità di ricerca parte 1 algoritmo di tuning di iperparametri.....	27
3.9 Modalità di ricerca parte 2 algoritmo di tuning di iperparametri.....	28
4.1 Rete neurale deep.....	30
4.2 Differenza tra processamenti eseguiti da una rete non deep in alto, e da una rete deep, in basso.....	31
4.3 Schematizzazione rete neurale convoluzionale.....	33
4.4 Convoluzione ottenuta con filtro per l'identificazione di bordi verticali.....	34
4.5 Rappresentazione convoluzioni con stride diversi.....	35
4.6 Rete ricorrente dispiegata, con parametri U,V e W.....	37
4.7 Rappresentazione modello ricorrente con LSTM con struttura many-to-many.....	38
5.1 Schematizzazione metodo utilizzato.....	40
5.2 Segnale di accelerazione, primo asse, primo sensore durante una prova di laboratorio.....	42
5.3 Rappresentazione campionamento eseguito ogni cento secondi.....	43
5.4 Rappresentazione grafica di uno spettrogramma realizzato da un segnale di due secondi.....	44
5.5 Rappresentazione della tecnica di padding speciale.....	45
5.6 Schematizzazione modello convoluzionale usato nella trattazione.....	47

5.7 Scenari testati con il modello convoluzionale.....	48
5.8 Schematizzazione modello convoluzionale con LSTM usato nella trattazione.....	50
5.9 Scenari testati con il modello convoluzionale con LSTM.....	52
5.10 A destra, grafico che mostra le predizioni sui dati test , a sinistra, grafico che valorizza l'errore di previsione.....	56
5.11 Grafici che mostrano le predizioni sui dati di apprendimento.....	57
5.12 A destra, grafico che mostra le predizioni sui dati test , a sinistra, grafico che valorizza l'errore di previsione.....	58
5.13 Grafici che mostrano le predizioni sui dati di apprendimento.....	59
5.14 A destra, grafico che mostra le predizioni sui dati test , a sinistra, grafico che valorizza l'errore di previsione.....	60
5.15 Grafici che mostrano le predizioni sui dati di apprendimento.....	61
5.16 Stampa a video del programma che esegue predizioni.....	63
5.17 A destra, grafico che mostra le predizioni sui dati test , a sinistra, grafico che valorizza l'errore di previsione.....	66
5.18 A destra, grafico che mostra le predizioni sui dati test , a sinistra, grafico che valorizza l'errore di previsione.....	66
5.19 A destra, grafico che mostra le predizioni sui dati test , a sinistra, grafico che valorizza l'errore di previsione.....	67
6.1 Rappresentazione campionamento più frequente.....	72
6.2 Esempio di data augmentation per computer vision.....	73

Indice delle tabelle

5.1 Dati caratteristici per ogni cinghia.....	42
5.2 Descrizione di ogni livello del modello convoluzionale.....	46
5.3 Descrizione di ogni livello del modello convoluzionale con LSTM.....	51
5.4 Risultati modelli.....	54
5.5 Iperparametri scelti dall'algoritmo di tuning.....	55
5.6 Indicatori di performance per scenario 3.....	56
5.7 Iperparametri scelti dall'algoritmo di tuning.....	57
5.8 Indicatori di performance per scenario 5.....	58
5.9 Iperparametri scelti dall'algoritmo di tuning.....	59
5.10 Indicatori di performance per scenario 6.....	60
5.11 Dati caratteristici per ogni nuova cinghia testata.....	64
5.12 Risultati ottenuti sulle nuove cinghie.....	65
5.13 Risultati scenario 3 rispetto alle nuove cinghie.....	65
6.1 Risultati modelli e gap tra le performance.....	69

1 Introduzione

La manutenzione predittiva è una pratica che viene utilizzata per ottimizzare i piani di manutenzione, attraverso la previsione di guasti, con tecniche che sfruttano la predizione di comportamenti sulla base di dati passati. L'applicazione di questa pratica può portare numerosi vantaggi alle aziende manifatturiere, tra cui, la riduzione dei tempi di inattività e l'aumento della qualità del prodotto. Tuttavia, i modelli su cui si affidano affrontano molteplici sfide, che possono limitarne o impedirne l'utilizzo. Dati di buona qualità e numerosi sono al centro della manutenzione predittiva data-driven. Sfortunatamente, però, in ambito aziendale il processo di raccolta dati può essere così lungo e costoso da diventare impraticabile. Il metodo con cui vengono costruiti i modelli "tradizionalmente" rappresenta un ulteriore scoglio: informazioni che rappresentano lo stato di salute del macchinario sono estratte dai dati raccolti e in seguito vengono applicati modelli di intelligenza artificiale. Questo approccio spesso rende il sistema nel suo complesso poco scalabile dato che le informazioni estratte dipendono dal componente e dal macchinario e non sono applicabili in altri contesti industriali.

L'obiettivo di questa trattazione è quello di applicare un modello che risolva in parte le difficoltà sopra descritte e, inoltre, fornire linee guida al fine di poter migliorare i modelli ulteriormente.

Un dataset composto da segnali di accelerazione relativi a cinghie è utilizzato per valutare i modelli proposti. Il dipartimento di ingegneria industriale dell'università di Bologna ha infatti a disposizione una macchina appositamente realizzata per generare grandi quantità di dati di vibrazione. La macchina è costituita principalmente da un motore elettrico il quale moto è scaricato su di una cinghia. Diverse prove di laboratorio sono state condotte per portare allo stato di guasto le cinghie e per poter raccogliere i dati necessari.

I modelli applicati sono moderni algoritmi di *Machine Learning* che sono stati sviluppati in *Python*, uno dei linguaggi di programmazione più usati in *data science*. I modelli di intelligenza artificiale sono stati programmati tramite potenti API (*Application Programming Interface*) che fanno accedere a moduli preimpostati permettendo di velocizzare lo sviluppo del software. *Keras*, una API che consente lo sviluppo di modelli di *deep learning*, viene usata in fase di programmazione per allenare i modelli.

La tesi è articolata in sette capitoli:

- Nel capitolo 1: *Introduzione*

Si delineano gli obiettivi e i risultati attesi. Le tecniche utilizzate sono brevemente menzionate.

- Nel capitolo 2: *Introduzione alla manutenzione*

Una storia della manutenzione viene fornita. Dopodiché, le politiche manutentive più importanti vengono descritte al fine di mettere in evidenza vantaggi e svantaggi di ognuna di esse.

- Il capitolo 3: *Introduzione al machine learning*

Si delineano le differenze tra i vari tipi di apprendimento. L'apprendimento supervisionato, tema centrale di questa trattazione, viene descritto in modo esaustivo e, per finire, viene trattato il tema della scelta del modello migliore.

- Nel capitolo 4: *Il Deep learning*

Vengono introdotti i modelli di *Deep learning*. In seguito, modelli convoluzionali e ricorrenti vengono descritti.

- Nel capitolo 5: *Il metodo utilizzato e i risultati ottenuti*

Il caso studio è introdotto. Dopodiché, è descritto come i modelli di intelligenza artificiale sono applicati al fine di stimare la vita residua dei componenti. Per finire, i risultati ottenuti sono presentati.

- Nel capitolo 6: *Analisi dei risultati e sviluppi futuri*

Viene fornita una breve analisi dei risultati. Al termine del capitolo, alcuni possibili sviluppi futuri sono introdotti.

- Nel capitolo 7: *Conclusioni*

Quello che emerge dal caso studio è riassunto, dopodiché, sono menzionate le competenze che ho acquisito durante questo progetto.

2 Introduzione alla manutenzione

L'enciclopedia Treccani definisce così la manutenzione: “ Il mantenere in buono stato; in partic., insiemi di operazioni che vanno effettuate per tenere sempre nella dovuta efficienza funzionale, in rispondenza agli scopi per cui sono stati costruiti, un edificio, una strada, una nave, una macchina, un impianto” [1].

Le dinamiche manutentive hanno un' origine tanto antica quanto le più antiche pratiche produttive. Rivisitando la storia, infatti, troviamo che i problemi della manutenzione, sia pure in termini impliciti, affioravano anche nella bottega artigiana che, evolvendosi, si trasformò in «manifattura», oggetto dell'analisi di Adam Smith, lo studioso scozzese vissuto fra il 1723 e il 1790, al quale si fa risalire l'inizio delle discipline economiche. In quel contesto, l'artigiano, unico autore del prodotto, si rendeva garante e controllore della sua qualità e della sua conservazione, possedendone per così dire l'integrale *know how* conoscitivo. Quell'artigiano compiva anche implicitamente il controllo di qualità e con esso le azioni manutentive che riteneva necessarie per la sua garanzia. Alla manifattura seguì la rivoluzione industriale, e l'enfasi maggiore nella produzione si spostò sulla dimensione quantitativa. Si introdussero metodi di meccanizzazione del lavoro, che venne organizzato «scientificamente» secondo i criteri teorici esposti da Frederick Winslow Taylor (1856–1915) e applicati in maniera rigida ed esaustiva nelle fabbriche di Henry Ford (1863–1947). Il conseguente «taylorismo–fordismo» ruppe l'identità artigianale prodotto e con essa la presenza implicita degli opportuni interventi manutentivi.

Fu alla fine degli Anni Cinquanta che incominciò a emergere una nuova concezione dell'organizzazione produttiva, tesa a recuperare l'individualità dei collaboratori e con essa la loro motivazione, la loro attenzione, la loro professionalità. Nascono allora nuove tecniche, come quella del *Just in time* e dell'informatizzazione e robotizzazione dei cicli di produzione attraverso i quali si sviluppò la *lean production* e una più ampia flessibilità produttiva. Vi fu, anche, un sempre maggiore coinvolgimento delle persone, il che fece emergere nuovi metodi per il controllo della qualità, rivolti sempre più a seguire il corretto svolgersi delle operazioni in atto e non, quindi, soltanto alla conclusione finale

degli eventi. Si svilupparono così i Circoli di Qualità, le tecniche Zero Difetti, la *Quality Assurance* e ancora altre iniziative che si indirizzarono tutte verso il concetto di *Total quality control*. Oggi il controllo di qualità si trasferisce dal prodotto alle singole fasi che caratterizzano la produzione, con una responsabilizzazione piena degli operatori addetti e quindi dell'individuo che diviene il protagonista assoluto del processo. Contemporaneamente si amplia anche il ciclo produttivo, che tende a estendersi ai rapporti con i clienti utilizzatori dei prodotti per consentirne la migliore utilizzazione.

2.1 L'importanza della manutenzione aziendale

Nel mondo industriale, le pratiche manutentive possono assumere valenza strategica per due ragioni.

In primis, la qualità del prodotto finale, quando il sistema azienda che l'ha realizzato ha adottato delle pratiche manutentive corrette, aumenta notevolmente. L'acquisto di un prodotto più sicuro, o di più alta qualità è un grande vantaggio per il consumatore anche se paga di più. La qualità diventa ancora più importante se pensiamo che la società sta diventando sempre più complessa, la competizione globale aumenta e la ricerca tecnologica trova sempre nuove soluzioni applicative.

In secondo luogo, i costi di mantenimento e di riparazione possono rappresentare una grossa fetta del valore del macchinario preso in considerazione. Si stima che il costo di manutenzione rappresenta una percentuale che va dal 15% al 40% del costo di produzione annuale.

Queste gravi perdite influenzano ogni tipo di settore. Per esempio, nell'industria delle turbine eoliche le spese sopra menzionate possono ammontare al 10-15 % dell'utile generato annualmente[2] e, nel caso di turbine *offshore*, questa percentuale raggiunge il 25% circa.

Un report statunitense ha rivelato che una compagnia IT può perdere dagli 84.000\$ ai 108.000\$ per ora durante un fermo macchina. Stesso discorso per le aziende manifatturiere. Per le piccole e medie imprese i costi sono certamente più bassi, ma inficiano comunque in maniera notevole sul bilancio.

Ancora, il costo di riparazione e di mantenimento sono di circa 0.03 \$ per metro quadro per le opere esterne di edifici.

Per comprendere in modo più profondo il concetto di costo di manutenzione è necessario scomporre il costo totale nelle componenti di costo che lo costituiscono.

I fermi macchina avvengono quando un macchinario necessario allo svolgimento delle attività caratteristiche aziendali si trova inattivo a causa di un guasto. Questo evento si traduce nella necessità di avere a disposizione un'unità manifatturiera di supporto che possa compensare la perdita di produzione. Questa replicazione dei macchinari comporta inevitabilmente un'aumento dei costi aziendali.

I fermi operatori sono analoghi ad i fermi macchina. L'inattività della macchina comporta direttamente l'inattività dell'operatore associato risultando dunque in un'inefficienza.

I costi di manodopera diretta sono gli oneri che l'impresa deve pagare a fronte di un intervento manutentivo. Di solito sono tanto alti quanto il sistema da mantenere è complesso e la causa del danno non è facilmente identificabile.

I costi di scorta sono registrati quando lo spazio a magazzino è impiegato per la conservazione di ricambi. In generale quando la politica manutentiva non è ben sviluppata anche la quantità di ricambi a magazzino è elevata e il costo totale aumenta.

Tutte queste voci di costo influenzano anche un altro tipo di onere che spesso non viene considerato quando si parla di manutenzione: il costo ambientale. In particolare, scorte elevate, magazzini dalle grandi dimensioni e la replicazione di risorse impattano non solo sul sistema azienda ma anche sull'ambiente.

2.2 Le politiche manutentive

L'atto di mantenere può avvenire in diversi modi in funzione del sistema preso in esame, il *budget* disponibile, e la cultura aziendale che lo influenza. In seguito vengono descritte tre metodologie utilizzate in azienda per fare manutenzione.

2.2.1 La manutenzione correttiva

Questo è un approccio tradizionale nel quale i macchinari vengono fatti funzionare fino a quando non si rompono. Questa strategia garantisce in principio il più lungo tempo operativo prima della rottura. Però, dato che intrinsecamente questa tecnica comporta un' azione correttiva che avviene solamente quando il guasto è già avvenuto, i danni a catena non sono in nessun modo evitati e i sistemi si possono danneggiare in modo importante. Oltretutto, con questa strategia manutentiva, i costi di fermo operatore, i costi di fermo macchina e i costi di scorta sono alti, e, spesso, il costo di manodopera diretto sono elevati perché di frequente risulta necessario trovare il componente danneggiato all'interno di un sistema integrato e complesso. Di solito questa politica rimane valida solamente se applicata su macchinari non critici e/o a basso costo per la quale la manutenzione è facile e veloce.

2.2.2 Manutenzione preventiva

Questo approccio alla manutenzione viene anche chiamato *time-based* perché i macchinari vengono fatti funzionare per un certo lasso di tempo e poi vengono controllati: se qualche guasto nascente è facilmente riconoscibile i componenti di interesse vengono sostituiti. La programmabilità degli interventi garantisce, in generale, un' efficientamento rispetto alla manutenzione correttiva. Il tempo tra un controllo e l'altro è molto importante in queste strategie. Se l'intervallo temporale è troppo lungo, qualche guasto imprevisto si può verificare. Al contrario, se l'intervallo è troppo corto, troppa manutenzione sarà svolta e componenti ancora in buono stato saranno sostituiti. In generale, questo approccio alla manutenzione è appropriato quando il tempo di rottura dei componenti è predicibile: in questo caso le RUL (*residual usefull life*) dei sistemi presi in considerazione non devono essere caratterizzate da grande varianza e analisi statistiche svolte su campioni molto numerosi devono essere disponibili.

In letteratura sono molto importanti gli approcci di manutenzione preventiva che prendono il nome di tipo I e di tipo II. Per queste politiche sono disponibili anche dei semplici modelli che cercano di minimizzare il costo totale di mantenimento in funzione

del tempo di ispezione. La strategia di tipo I consiste nel rimpiazzo periodico di componenti deteriorati in un sistema di produzione.

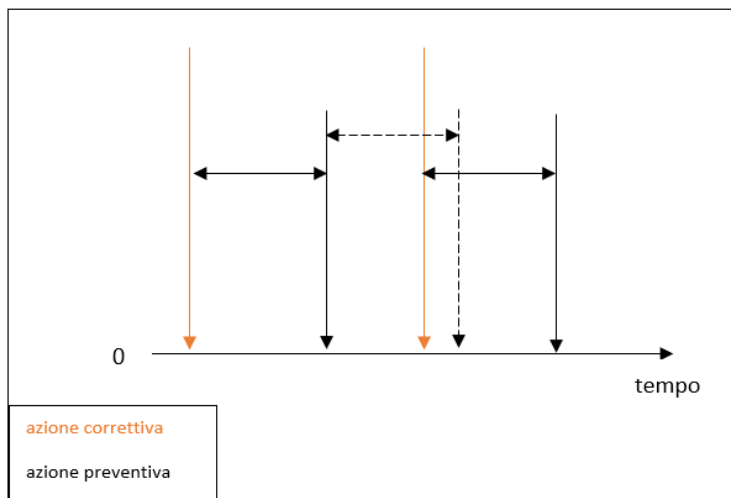


Fig. 2.1 Strategia di tipo I di manutenzione preventiva

Considerando figura 2.1, quando la rottura di un componente avviene e manutenzione correttiva viene eseguita, l'orologio è resettato a 0 e le operazioni di rimpiazzo caratteristiche della politica preventiva avvengono dopo che è trascorso un certo lasso di tempo.

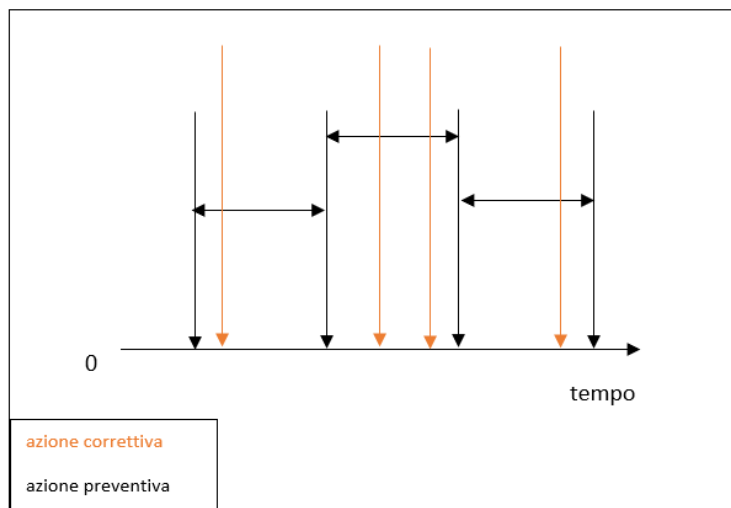


Fig. 2.2 Strategia di tipo II di manutenzione preventiva

Come si può notare da figura 2.2, invece, la politica di tipo II implica che le operazioni preventive vengano svolte ad ogni istante di ispezione prefissato indipendentemente dall'esecuzione di manutenzione correttiva o meno.

2.2.3 L'industria 4.0 e il suo legame con la manutenzione predittiva

L'industria 4.0 si riferisce all'intelligente interconnessione di macchine e processi per l'industria con l'aiuto di vari strumenti che prendono il nome di tecnologie abilitanti. Il ministero dello sviluppo economico ne elenca nove. Queste ultime spaziano dalla robotica alla cybersecurity, dalla nanotecnologia al cloud e al software. Il loro ruolo è decisivo poiché sono considerate *driver* di innovazione. Seppure le aziende possono ottenere il più grande vantaggio competitivo solamente quando sono tutte impiegate in sinergia, anche l'utilizzo singolo di uno di questi strumenti garantisce un grande beneficio.

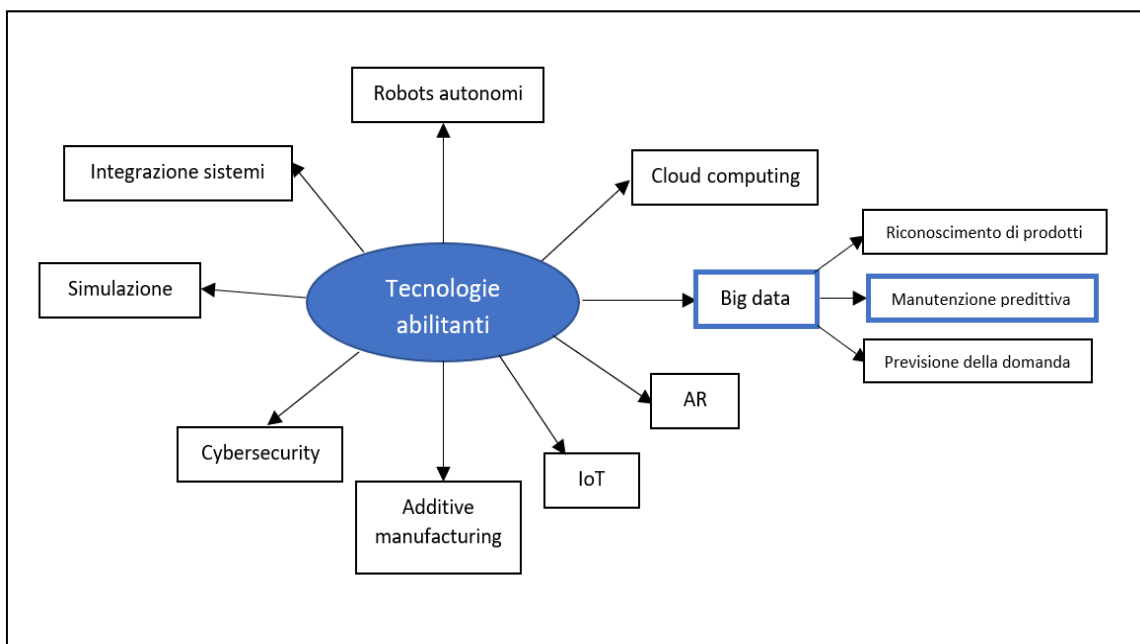


Fig. 2.3 Le tecnologie abilitanti dell'industria 4.0

In particolare, il processamento di una grande quantità di dati (*big data*) allo scopo di imparare relazioni e strutture ha negli ultimi anni attirato un grande interesse. L'utilizzo di questa tecnologia abilitante, il cosiddetto *big data analytics*, ha come fine ultimo quello di sviluppare predizioni accurate riguardo alla realtà circostante che permettano al management di prendere decisioni sempre meno influenzate dall'incertezza. Le tecniche di *machine learning*, impiegate per l'elaborazione dei dati, hanno un campo di applicazione molto ampio: riconoscimento di prodotti, manutenzione predittiva, previsione della domanda, ottimizzazione logistica dei piani di trasporto. La manutenzione predittiva si inserisce dunque in questa intelaiatura come presentato da figura 2.3.

2.2.4 Manutenzione predittiva

Questo approccio alla manutenzione viene chiamato così perché parametri di interesse vengono raccolti tramite sensori, spesso posizionati vicino o sul macchinario, e la RUL viene predetta tramite l'analisi di queste condizioni operative. L'idea principale su cui tale strategia manutentiva si fonda è quella secondo la quale un guasto tipicamente non avviene in maniera istantanea ma è frutto del deterioramento nel tempo delle condizioni di un determinato componente. Il monitoraggio di alcuni parametri può denunciare condizioni di funzionamento anomale e dunque segnalare la necessità di un intervento. Chiari sintomi di un funzionamento fuori standard sono variazioni di temperatura, presenza di particelle di usura degli oli o un anomalo spettro di vibrazioni. Tutte queste grandezze possono essere facilmente misurate e portare alla corretta identificazione del problema. Queste analisi sono spesso svolte utilizzando modelli o tecniche di varia natura. Una volta che la predizione è disponibile, una strategia ottima di manutenzione può essere programmata prima che la rottura vera e propria avvenga. Questa tecnica ha ovviamente diversi vantaggi comparata con le altre due descritte in questo capitolo. E' in grado di predire la rottura di componenti, e, in questo modo, le rotture a catena sono evitate. In più i fermi macchina e i fermi operatore sono limitati quando la predizione è accurata. L'identificazione del componente che si danneggerà, anche questo obiettivo di questa politica, permette una rapida manutenzione e dunque una riduzione dei costi di manodopera diretta associati. Ancora, i costi di scorta, dovuti

al mantenimento di ricambi a magazzino, viene ridotto, perché guasti improvvisi sono meno frequenti consentendo di poter mantenere in giacenza solo i componenti strettamente necessari. Per finire, l'efficientamento generale del sistema azienda e della catena del valore più in generale limita le conseguenze ambientali della manutenzione.

Grazie alla superiorità della manutenzione predittiva, numerose ricerche sono state svolte in questa direzione.

In generale, i metodi utilizzati in letteratura possono essere suddivisi in tre grandi famiglie:

- basati sul modello
- basati sui dati
- basati sul modello e dati

I metodi basati sul modello cercano di realizzare modelli fisici o matematici al fine di descrivere la degradazione dei macchinari, e aggiornare i parametri del modello usando i dati raccolti. Questi metodi, di solito, includono sia l'esperienza dell'uomo, sia le informazioni lette da sensori. Appoggiandosi ad una solida base matematica o fisica, non è necessaria, al fine della previsione accurata della RUL, una grande quantità di dati. Sfortunatamente, nella stragrande maggioranza dei macchinari industriali, i *pattern* di degradazione possono cambiare, e anche di molto, a seconda del tipo di rottura. Per questa ragione diversi modelli andrebbero sviluppati per ogni componente di ogni macchina risultando in analisi poco scalabili e non economicamente fattibili. Per alcuni sistemi complessi, oltretutto, la fisica del guasto diventa di difficile comprensione, impossibilitando la creazione di un modello e restringendo dunque ulteriormente il campo di applicazione. I modelli più utilizzati includono quelli basati su Paris-Erdogan [3], quelli basati sulla legge di Norton [4] e quelli basati sullo spazio di stato.

I metodi basati sui dati derivano il modello di degradazione utilizzando i soli dati forniti dal sistema di monitoraggio, formato principalmente da sensori, senza preoccuparsi del modello analitico del sistema né dei suoi parametri fisici. Generano predizioni di RUL tramite l'uso di dati storici di segnali.

Perciò, l'accuratezza dei metodi basati sui dati dipende direttamente dalla quantità e dalla qualità dei dati presi in considerazione. A livello pratico, è di particolare difficoltà raccogliere una quantità di dati sufficiente ai fini di generare predizioni precise.

I metodi basati sul modello e sui dati cercano di combinare i punti di forza delle tecniche sopra descritte in modo tale da ottenere predizioni ancora più accurate.

Negli ultimi anni, i metodi basati sui dati hanno acquisito una certa importanza. Infatti, tecniche di *machine learning* e in particolare *deep learning* si sono sviluppate con grande rapidità nell'ultimo decennio facilitando il processamento di una grande quantità di dati in tempi ragionevoli. Per questa ragione questa trattazione si basa su questi ultimi strumenti e nelle prossime righe ne sarà presentato un approfondimento generale.

In figura 2.4 sono rappresentati i principali passi necessari che permettono l'applicazione della manutenzione predittiva basata sui dati:

- Per iniziare, segnali devono essere raccolti utilizzando diversi tipi di sensori installati sul macchinario o in sua prossimità. I segnali misurati dovrebbero essere collegati alla vita della macchina monitorata. In altre parole, questi ultimi devono incorporare qualche informazione utile, che possa essere correlata allo stato di salute del macchinario. Ci sono diversi tipi di segnali che possono essere raccolti, come segnali di vibrazione, acustici, temperatura e correnti elettriche. Questi dati vengono poi salvati all'interno di un PC e immagazzinati in memoria. E' importante sottolineare che, in anni recenti, il prezzo della sensoristica necessaria è calato drasticamente rendendo ancora più attraente la raccolta e il seguente processamento dei segnali.

- Il secondo *step* considera i segnali raccolti dallo *step* precedente, li processa e realizza un *health indicator* (HI). L'obiettivo è quello di estrarre informazioni utili dai segnali che siano in grado di rivelare lo stato di salute del macchinario monitorato. In letteratura il processamento dei segnali al fine di realizzare un HI è stato affrontato molte volte e quindi molti algoritmi e tecniche sono state sviluppate. Huang et al. [5], per esempio, ha usato il *Root Mean Square* (RMS) per predire la RUL di cuscinetti. Malhi et al. [6] ha estratto l'RMS e i valori picco dalla trasformata wavelet. Anche il valore di kurtosis proveniente da un segnale processato con un filtro passa-banda è stato utilizzato ai fini della predizione della vita utile di cuscinetti [7]. Gasperin et al. [8] ha estratto la densità

di potenza derivata dall'involuppo del segnale nel dominio della frequenza per predire la RUL di ingranaggi. Le analisi necessarie ai fini di realizzare qualsiasi *health indicator* possono essere classificate in tre categorie: analisi nel dominio del tempo, analisi nel dominio della frequenza e analisi tempo-frequenza. Anche se ogni categoria ha i suoi vantaggi e svantaggi si può affermare che gli HI dipendenti da analisi tempo-frequenza sono i più performanti.

-La fase di divisione in *health stages* (HS) consta nel dividere il processo di degradazione dei macchinari in due o più stadi. Nella prima parte della vita del macchinario, infatti, i segnali, e quindi anche gli HI di conseguenza, non racchiudono informazioni legate all'aspettativa della durata di vita del sistema. Perciò i valori dell'HI non sarebbero significativi quando utilizzati per predire la RUL e il valore di queste ultime sarebbe inevitabilmente incorretto. Di solito, quindi, si evita di predire la vita residua dei macchinari quando ci si trova nel primo stadio e, solo in seguito, viene azionato l'algoritmo predittivo. Anche lo *step* di divisione in HS può essere svolto in innumerevoli modi e la letteratura è ricca di algoritmi e tecniche. Wang et al. [9] ha trovato il punto in cui far partire le predizioni tramite il valore del RMS. Jin et al. [10] ha trasformato l'HI in dati distribuiti in modo gaussiano tramite il metodo Box-Cox e poi ha definito con l'intervallo 3σ la soglia di allarme. Kimotho et al. [11] ha suddiviso il processo di degradazione di cuscinetti in cinque stadi analizzando i cambiamenti nelle ampiezze di frequenza nello spettro di potenza.

- Nella fase finale, l'HI viene dato in pasto ad un algoritmo di intelligenza artificiale che ne fa la regressione, spesso non lineare, in modo tale da ottenere la predizione della vita residua del macchinario monitorato. L'algoritmo, in estrema semplificazione, impara da un certo numero di HI e si occupa di mappare la funzione che va da questi ultimi alla RUL del sistema. Anche in questo caso, i modelli di regressione utilizzati in letteratura sono di svariata natura e numerosi. Molte pubblicazioni [12,13,14,15] usano una rete neurale *feed-forward*. Wang et al. [16] ha usato una rete neurale *feed-forward* a tre livelli per predire il futuro HI, poi ha usato questo dato come *input* di un modello di *proportional hazard* per predire il livello di danno e la probabilità di

sopravvivenza. Benkedjouh et al.[17] ha usato un modello di SVR (*Support Vector Regression*) per mappare l' HI poi ha dato in pasto il risultato ad un modello di potenza.

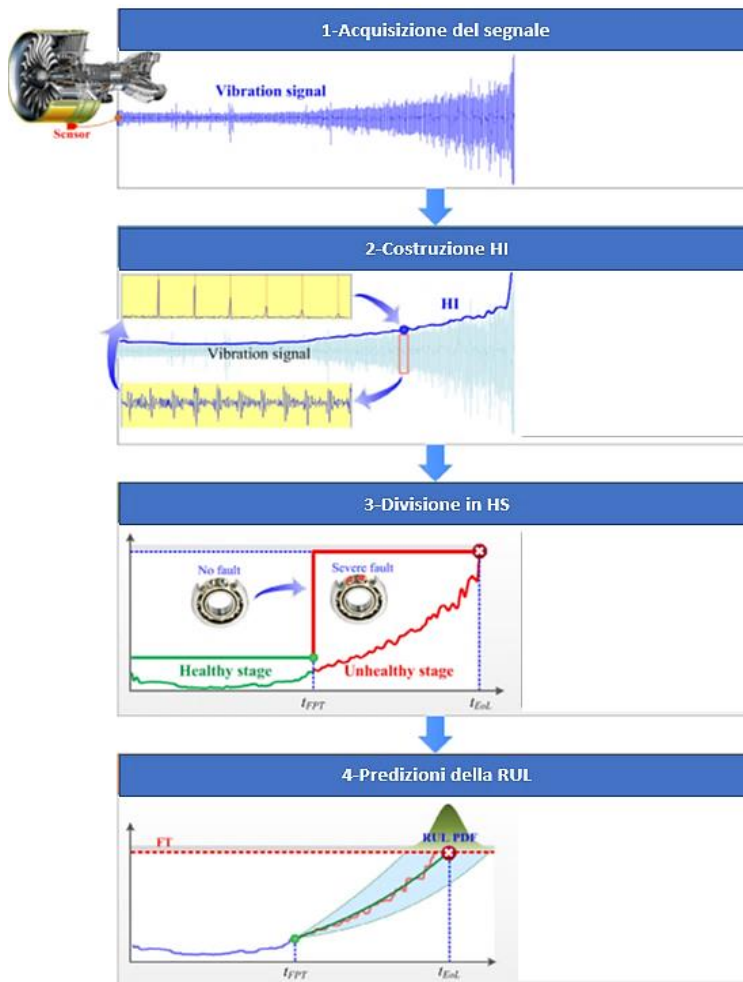


Fig. 2.4 Processi necessari per applicare manutenzione predittiva nel caso di metodi basati sui dati

E' importante sottolineare che queste quattro fasi descrivono il processo predittivo quando si utilizza l' approccio classico. Negli ultimi anni però, come già anticipato, tecniche di *machine learning* via e via più avanzate sono diventate disponibili. La possibilità di processare una grande quantità di dati in tempo limitato ha permesso lo sviluppo di un nuovo approccio che spesso nella *AI community* viene chiamato *end to end learning*. Invece di processare il segnale per ricavare un HI, in questa trattazione lo spettrogramma viene direttamente dato in pasto all' algoritmo predittivo. Sarà dunque il modello responsabile nel trovare le caratteristiche del segnale che possano essere utili

ai fini della previsione della vita utile. In altre parole, è l'algoritmo che impara da solo gli *health indicator* più adatti e che meglio si prestano nell'indicare lo stato di salute della macchina monitorata. Nella letteratura legata alla prognostica lavori di questo tipo sono già presenti. Heimes et al. [18] utilizza una rete neurale ricorrente che prende come *input* un insieme di segnali non elaborati. Meng Ma et al. [19] utilizza una rete neurale convoluzionale ricorrente che prende come *input* spettrogrammi di segnali di vibrazione.

Questo approccio *end to end* ha tre vantaggi rispetto alla metodologia classica:

- 1) La scelta del HI non è necessaria e quindi il tempo di sviluppo di questi algoritmi si riduce notevolmente.
- 2) La scelta del HI dipendeva dall'esperienza del programmatore e la bontà della metrica era legata al sistema preso in considerazione. L' *health indicator* non dava quindi nessuna garanzia di generalizzazione quando impiegato in altre applicazioni.
- 3) Le fasi necessarie alla costruzione degli HI implicano il processamento dei segnali. Non è possibile escludere che informazioni importanti legate allo stato di salute dei macchinari vengano perse proprio durante questo processamento.

3 Introduzione al machine learning

L'intelligenza artificiale e il *machine learning* nell'ultimo decennio hanno suscitato grande interesse nella comunità scientifica. Infatti, la disponibilità di una grande quantità di dati da processare, nonché la possibilità di elaborarli efficacemente ha permesso lo sviluppo delle tecniche descritte in questo capitolo. L'intelligenza artificiale come idea di base è presente da molto tempo, infatti i primi computer europei erano stati concepiti come "macchine logiche" che riproducessero le basi aritmetiche e mnemoniche presenti nel cervello umano. Col progredire della tecnologia e della conoscenza della nostra mente è cambiato il concetto di fondo dell'AI: piuttosto che calcoli sempre più complessi, il lavoro nel campo dell'AI si è concentrato sull'imitazione dei processi decisionali umani e sullo svolgimento di compiti in modi sempre più simili all'uomo. I primi studi riguardanti l'apprendimento automatico vero e proprio iniziano a svilupparsi intorno agli anni '50. Arthur Samuel può essere considerato il fondatore del *machine learning* in quanto nel 1959 ne coniò il termine e ne diede una prima definizione: "Il campo di studi che dà ai computer la capacità di apprendere senza che vengano esplicitamente programmati". A questa prima definizione ora se ne affianca un'altra più operativa e moderna di Tom Mitchell: "si dice che un programma per computer apprenda dall'esperienza (E) rispetto ad alcune classi di attività (T) e alla misura delle prestazioni (P) se la sua prestazione in attività in T, come misurata da P, migliora con l'esperienza E".

3.1 Tipi di apprendimento e categorizzazione

Gli algoritmi di *machine learning* possono essere categorizzati a seconda della quantità di supervisione che ricevono. Ci sono due principali categorie: algoritmi supervisionati e algoritmi non supervisionati.

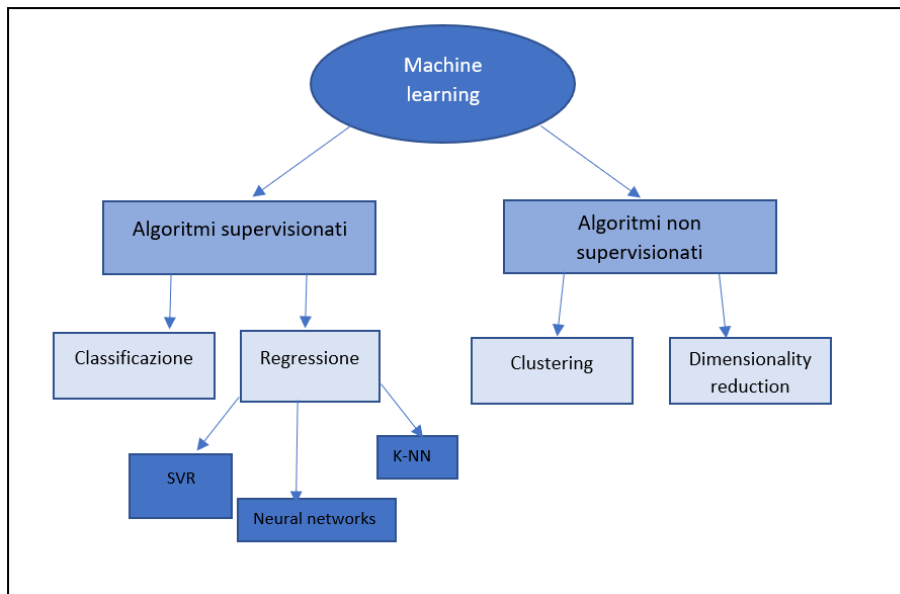


Fig. 3.1 Tipi di algoritmi presenti in letteratura di apprendimento automatico

Nell'apprendimento non supervisionato, i dati che vengono forniti all'algoritmo non includono le soluzioni desiderate. L'obiettivo è ricercare i *pattern* nascosti tra i dati. Di solito viene utilizzato per clustering o *task* associativi, come raggruppare clienti per comportamento.

Nell'apprendimento supervisionato, invece, l'esito desiderato è fornito insieme ai dati. Queste soluzioni sono chiamate etichette.

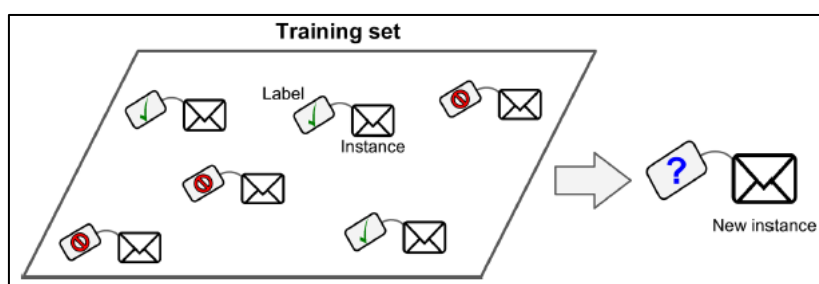


Fig. 3.2 Esempio di un algoritmo supervisionato: il classificatore di spam

Un tipico *task* dell'apprendimento supervisionato è la classificazione. Il filtro di *spam* è un buon esempio: in *input* sono fornite diverse email insieme alle loro classi. L'algoritmo deve imparare come classificare nuove mail.

Un altro tipico *task* prende il nome di regressione. Si tratta di predire un valore numerico e non una classe, come per esempio il prezzo di una macchina, data una serie di caratteristiche (marca, anno, chilometraggio). Per allenare il sistema, è necessario fornire molti dati di macchine insieme alle loro etichette.

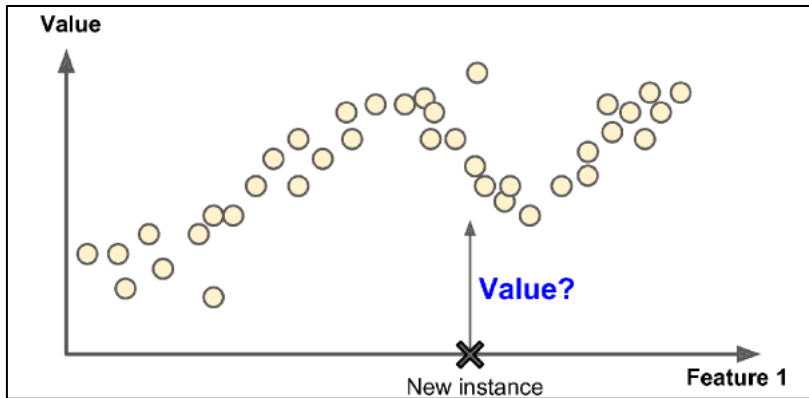


Fig. 3.3 Algoritmo supervisionato di regressione

Dato che la vita utile residua dei componenti viene espressa da un valore numerico, il *task* di interesse in questa trattazione è quello di regressione.

Alcuni tra i più importanti algoritmi di regressione sono:

- K Nearest Neighbors
- Support Vector Regression
- Rete neurali

3.2 Il processo di apprendimento di una rete neurale

Una rete neurale è un modello informatico ispirato alle reti neurali biologiche. Tale modello è costituito da un gruppo di interconnessioni di informazioni costituite da neuroni artificiali e processi che utilizzano un approccio di connessionismo di calcolo. Nella maggior parte dei casi una rete neurale artificiale è un sistema adattivo che cambia la propria struttura in base a informazioni esterne o interne che scorrono attraverso la rete stessa durante la fase di apprendimento.

Questi modelli possono essere rappresentati tramite un insieme di nodi ed archi come mostrato da figura 3.4.

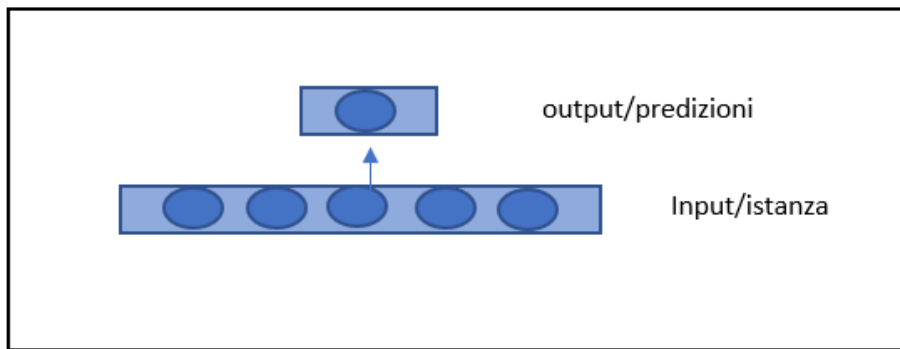


Fig. 3.4 Semplice rete neurale a singolo livello

I parametri di allenamento collegano l'*input* all'*output* della rete. In pratica l'*input* viene elaborato tramite l'utilizzo di questi parametri in modo che una predizione venga resa disponibile. In altre parole, l'*output/predizione* è funzione sia dell'*input* sia dei parametri. Essendo questi parametri i diretti responsabili della bontà delle predizioni, è importante che vengano calcolati in modo accurato e preciso. Il processo di calcolo di questi ultimi viene chiamato apprendimento o allenamento.

L'idea di fondo è ricavarli in modo tale che tutte le predizioni e le soluzioni, alla fine dell'apprendimento, abbiano dei valori simili tra loro. E' allora necessaria la definizione di una funzione costo che misura, ad ogni *step* dell'apprendimento, quanto i pesi del modello sono validi ai fini del predire accuratamente.

Data l'importanza di questi parametri è necessario spiegare in modo più approfondito il modo in cui vengono ricavati. L'algoritmo classico che viene usato per allenare questi modelli e ottenere i pesi θ si chiama *gradient-descent*. I passi fondamentali sono descrivibili come in seguito:

- Solo per la prima iterazione imposto i parametri del modello casualmente
- Eseguo la propagazione di tutti i dati di *input* con i parametri calcolati in precedenza. Significa eseguire le operazioni necessarie per ottenere le predizioni.
- Calcolo l'errore rispetto alle soluzioni. In questo momento entra in gioco la funzione costo che determina, come già anticipato, la bontà delle predizioni quando paragonate alle etichette.

-In seguito vengono ricavate tutte le derivate θ' . Intuitivamente queste indicano di quanto devono essere cambiati i pesi in modo tale che le nuove predizioni, della prossima iterazione, siano più accurate.

-Eseguo un passo elementare di *gradient-descent* :

$$\theta = \theta - \alpha \theta' \quad (3.1)$$

-Ripeto fino a convergenza

Il parametro α è di grande importanza e si chiama tasso di apprendimento: il suo valore detta la velocità del processo e quindi è necessario sceglierlo con attenzione.

Partendo da questo algoritmo di ottimizzazione che ricava i parametri ne sono stati sviluppati altri che ne mantengono la struttura principale ma che lo migliorano. In particolare:

- *Mini-batch gradient-descent* prevede la propagazione di una piccola parte dei dati di *input* ad ogni iterazione. In generale, questo aumenta la velocità di allenamento a patto di avere un leggero disturbo nell'errore. In altre parole, la funzione costo, in questo caso, non cala in modo monotono ad ogni iterazione ma potrebbe presentare dei picchi e delle valli. La quantità di dati di *input* da propagare per ogni iterazione è un parametro importante e si chiama *mini-batch size*.
- *Gradient descent with momentum* [20] prevede l'aggiunta di un termine dipendente dalle iterazioni precedenti, che viene sommato al gradiente nel tentativo di regolarizzare il movimento nello spazio dei parametri. Viene spesso utilizzato in modalità *mini-batch*
- RMSprop [21] prevede l'aggiunta di un termine dipendente dalle iterazioni precedenti. Il gradiente viene diviso per questo fattore nel tentativo di regolarizzare il movimento nello spazio dei parametri. Viene spesso utilizzato in modalità *mini-batch*.
- Adam [22] tenta di combinare le logiche alla base di RMSprop e *gradient descent with momentum*. Viene spesso utilizzato in modalità *mini-batch*.

3.3 La scelta del modello migliore

La funzione costo, come già anticipato, ha il compito di misurare quanto le predizioni siano accurate rispetto alle soluzioni. Permettono all'algoritmo di imparare e indicano anche al programmatore quando è opportuno fermare l'apprendimento. In genere, per il *task* di regressione, le due più importanti sono:

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \bar{y}_i}{y_i} \right| \quad (3.2)$$

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2 \quad (3.3)$$

Sia MAPE il *Mean Absolute Percentage Error*

Sia MSE il *Mean Squared Error*

Sia n il numero di predizioni che voglio valutare

Sia y_i la soluzione o etichetta per l'istanza i

Sia \bar{y}_i la predizione relativa all'istanza i

Entrambe le metriche sono largamente usate in letteratura e quindi empiricamente la loro efficacia è dimostrata. In questa trattazione ne vengono introdotte altre due particolarmente adatte al *task* di previsione della vita utile residua di componenti meccanici:

$$normalized_precision = \frac{\sqrt{\frac{\sum_{i=1}^n (err_i - \overline{err})^2}{n}}}{\bar{y}} \quad (3.4)$$

$$relative_score = \frac{1}{n} \sum_{i=1}^n a_i \quad se \ \bar{y}_i < y_i \quad (3.5)$$

Sia n il numero di predizioni che voglio valutare

Sia err_i l'errore assoluto che si commette per l'istanza i

Sia \overline{err} la media dell'errore rispetto a tutte le predizioni da valutare

Sia \bar{y} la media del valore di tutte le soluzioni o etichette da valutare

Sia a_i un valore che viene sommato alla metrica

Sia y_i la soluzione o etichetta per l'istanza i

Sia \bar{y}_i la predizione relativa all'istanza i

La funzione costo che prende il nome di *normalized_precision* misura la deviazione standard dell'errore. Quella invece che prende il nome di *relative_score* parte del presupposto che modelli che predicono mediamente una soluzione con valore inferiore rispetto al valore effettivo sono da privilegiare. Questa caratteristica è valida soprattutto in *task* di regressione che si prefiggono di prevedere la vita residua di componenti meccanici. Infatti, pronosticare una rottura posteriormente rispetto alla data effettiva porterebbe a guasti improvvisi e diversi problemi legati al mondo della manutenzione correttiva, come già introdotto nel capitolo 2.

All'interno del caso studio presentato da questa trattazione, la funzione costo MSE è stata utilizzata in fase di allenamento. Detto ciò, validare una rete neurale solamente utilizzando i dati di apprendimento porterebbe ad un grave errore. In figura 3.5 è schematizzato il processo di apprendimento come fino ad ora descritto.

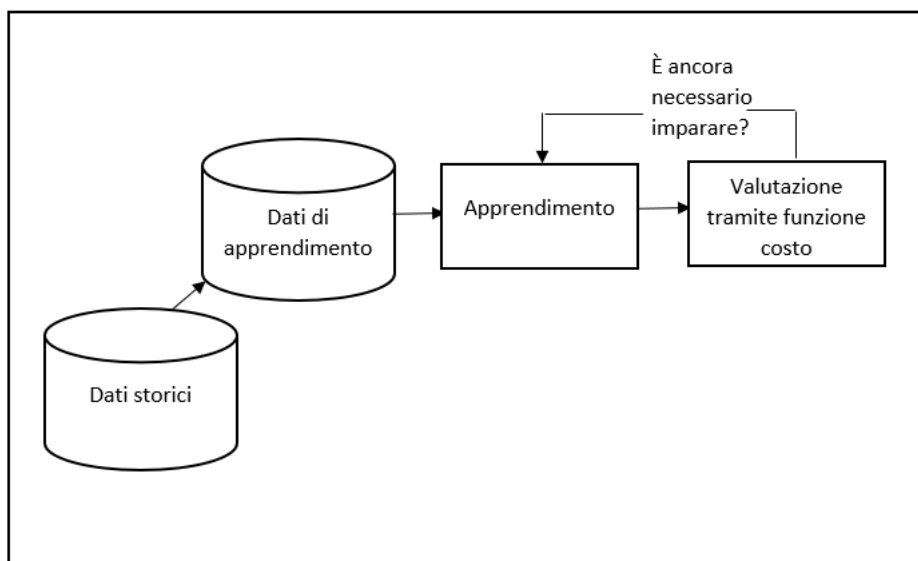


Fig. 3.5 Schematizzazione processo di apprendimento erraneo

Questo processo garantisce in genere dei buoni risultati sui dati di apprendimento ma scarsi risultati quando il modello realizza delle previsioni con dati che non siano quelli su cui è allenato. Questo problema è un importante concetto del *machine learning* e prende il nome di *overfitting*.

Un problema diametralmente opposto è invece quello di *underfitting*. In questo caso il processo di apprendimento è stato arrestato troppo prematuramente e quindi il modello non ha avuto il tempo di imparare il legame tra le istanze e le soluzioni. Questo si traduce in scarse performance anche nei dati di apprendimento. Fortunatamente, gli algoritmi di *machine learning*, in particolar modo le reti neurali, non soffrono quasi mai di *underfitting*.

Il problema di *overfitting*, invece, si risolve, in primo luogo, separando i dati storici in due sottogruppi. Un gruppo è destinato all'apprendimento (dati di apprendimento) e il secondo gruppo è destinato alla valutazione del modello (dati di test). La logica del processo di apprendimento cambia come mostrato in figura 3.6.

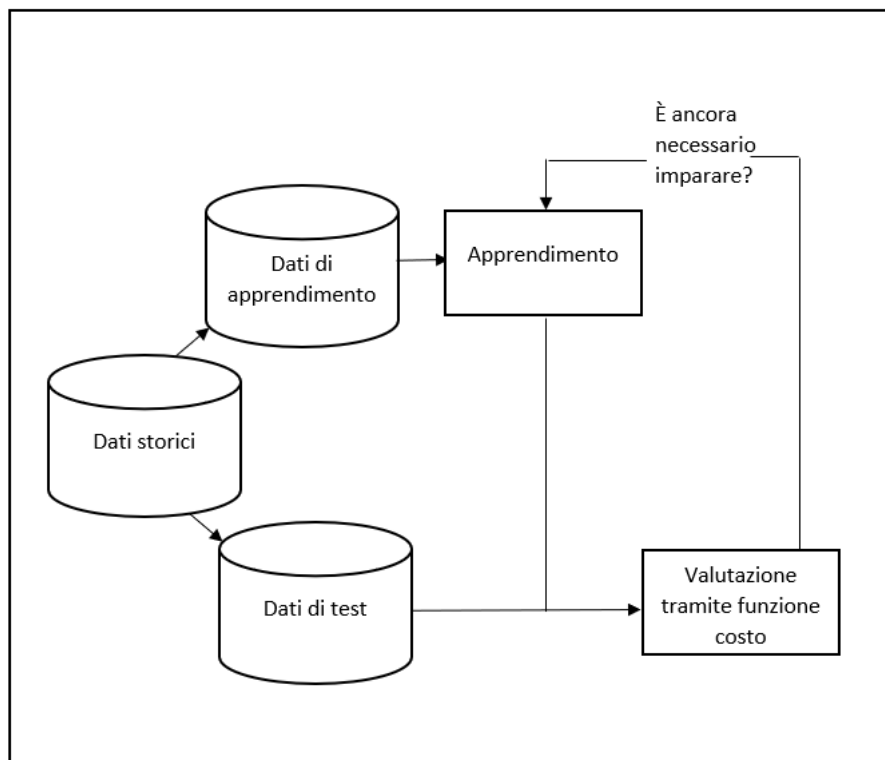


Fig. 3.6 Schematizzazione processo di apprendimento corretto

I risultati sui dati di test fungono da simulazione del comportamento dell'algoritmo quando applicato su nuove istanze. Da notare come l'allenamento viene eseguito solamente sui dati di apprendimento mentre i dati di test sono totalmente esclusi in questa fase.

Un primo e semplice modo per contrastare il problema di *overfitting* consiste, in fase di allenamento, nell'osservare i risultati sui dati di test e sui dati di apprendimento. Fino a quando il costo si riduce su entrambi i *set* di dati l'allenamento può proseguire. Nel momento in cui le performance sui dati di test peggiorano significa che il modello sta andando in *overfitting* e l'allenamento va arrestato. Questa tecnica prende il nome di *early stopping*: l'apprendimento è stato fermato né troppo presto (scongioro problemi di *underfitting*) né troppo tardi (scongioro problemi di *overfitting*).

Altre metodologie per limitare il problema di *overfitting* sono la regolarizzazione L2 oppure la regolarizzazione *dropout*. In breve, la tecnica L2 limita il valore dei parametri del modello assegnando una penalità in fase di allenamento proporzionale al quadrato della somma dei parametri stessi.

$$\text{funzione costo}' = \text{funzione costo} + \text{penalità}(\theta) \quad (3.6)$$

In particolare, più il termine *funzione costo* viene ridotto più il termine *penalità*(θ) cresce. In questo modo, l'algoritmo è disincentivato a limitare eccessivamente il valore della *funzione costo* bensì è chiamato a trovare un giusto bilanciamento tra i due addendi.

La tecnica *dropout*, invece, esclude casualmente alcuni neuroni quando propagato l'*input* risultando in pratica in una rete neurale più piccola. Quindi la numerosità dei parametri è ridotto di conseguenza e il problema di *overfitting* di conseguenza. Un esempio può chiarire il funzionamento intuitivo di questa tecnica.

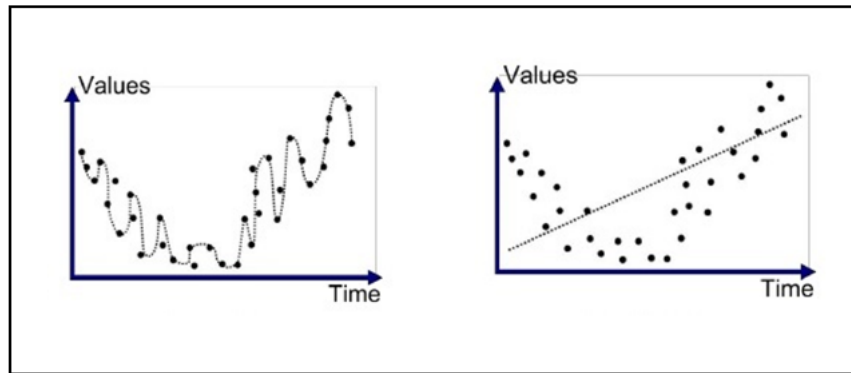


Fig. 3.7 Differenza tra modelli con tanti parametri e con pochi parametri

Nella figura 3.7, il riquadro di sinistra presenta un modello gravemente in *overfitting* nel quale le predizioni sui dati di apprendimento sono state svolte utilizzando un modello polinomiale con grado elevato. Per esempio:

$$y = a3 * time^3 + a2 * time^2 + a1 * time \quad (3.6)$$

Quando la numerosità dei parametri del modello è minore, invece, il modello non riesce a calzare su tutti i dati di apprendimento perfettamente limitando il problema.

Il riquadro di destra della figura 3.7 presenta visibilmente un modello con meno parametri:

$$y = a1 * time \quad (3.7)$$

Si può osservare che nel riquadro di destra, però, il problema opposto è presente. Il numero di parametri è così limitato che la funzione lineare che ne risulta non è in grado di calzare sui dati di apprendimento (*underfitting*). Questo esempio dimostra dunque che la tecnica di *dropout* deve essere calibrata: perdere molti parametri porta a problematiche di *underfitting* mentre mantenere troppi parametri risulta in *overfitting*.

Ai fini di scegliere il modello migliore, resta da introdurre il concetto di *tuning* di iperparametri. Nell'apprendimento automatico, il *tuning* di iperparametri è il problema di scegliere un set di iperparametri ottimali per un algoritmo di apprendimento. Un iperparametro è un parametro il cui valore viene utilizzato per controllare il processo di apprendimento. Alcuni di questi ultimi sono già stati menzionati in questa trattazione. In particolare il tasso di apprendimento, il *mini-batch size*, la quantità di *dropout* da eseguire e la penalità della regolarizzazione L2 sono tutti iperparametri. Ne esistono in realtà molti altri, che, spesso subentrano quando l'architettura della rete neurale si complica. D'ora in poi, se introdotti in questa trattazione, verranno indicati.

Esistono diversi modi per fare *tuning* di iperparametri. In seguito vengono descritti i metodi più popolari usati in letteratura.

-*tuning* manuale: la tecnica consiste nel costruire un modello ed eseguire un allenamento come illustrato da figura 3.6. Dopo aver osservato le performance sui dati di test il programmatore decide di modificare gli iperparametri del modello e lanciare un nuovo allenamento. Per esempio, in caso venga diagnosticato *overfitting* dopo un primo allenamento, la quantità di *dropout* o la penalità in funzione obiettivo può essere modificato e, in seguito, un nuovo allenamento viene svolto.

-*grid search*: l'algoritmo consiste nel valutare tutte le possibili soluzioni di un modello di intelligenza artificiale quando lo spazio di iperparametri è stato limitato ad un sottospazio. Di solito la migliore soluzione corrente è valutata sui dati di test. Quest'ultima è naturalmente memorizzata in modo tale che il possa essere richiamata in futuro.

-*random search*: l'algoritmo consiste nell'estrarre casualmente i valori dal sottospazio di iperparametri di interesse. Ogni soluzione è poi valutata come nell'algoritmo sopra descritto e le soluzioni migliori correnti vengono memorizzate. Si ripete il processo fino a quando non si verifica una certa condizione di arresto preimpostata dal programmatore.

In questa trattazione, un algoritmo un po' più complesso è stato sviluppato ai fini di trovare gli iperparametri ottimali. L'algoritmo può essere suddiviso in due parti che si susseguono una dopo l'altra. In parte 1 vengono selezionati inizialmente alcuni

iperparametri di cui si vuole fare il *tuning*. Vengono poi estratti casualmente i valori e viene fatto l'allenamento del modello. Ogni soluzione è valutata sui dati di test. In particolare, il costo o *Score* di una soluzione dipende da tre componenti:

$$Score = \frac{2}{3} \times MAPE + \frac{1}{6} \times normalized_precision + \frac{1}{6} \times relative_score \quad (3.8)$$

Nuove combinazioni di iperparametri vengono continuamente estratte casualmente e per ognuna di esse un allenamento viene eseguito. Questo processo si ripete per 3 ore. Naturalmente il modello migliore, ossia quello che ha minimizzato lo *Score*, viene salvato. Da notare che la parte 1 di questo algoritmo di *tuning* è una *random search* canonica.

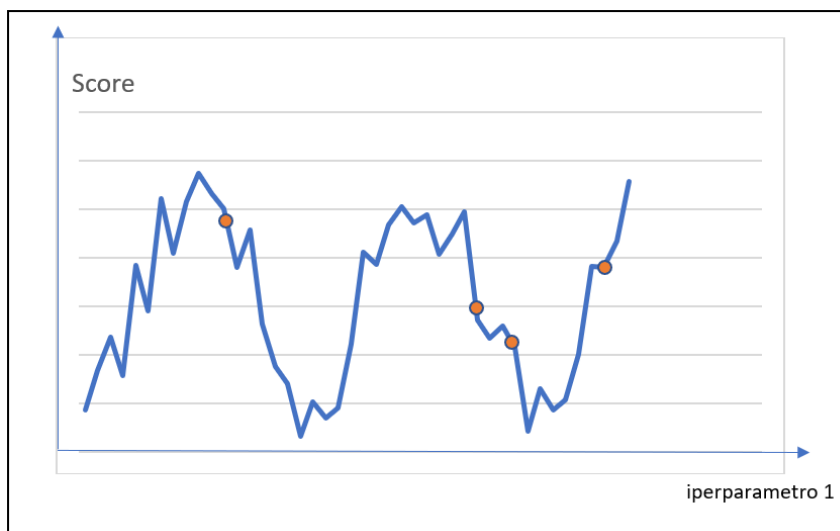


Fig. 3.8 Modalità di ricerca parte 1 algoritmo di tuning di iperparametri

In parte 2 vengono utilizzati gli iperparametri selezionati in parte 1, che hanno generato il modello migliore, come *input*. A questi valori viene applicata una perturbazione randomica e viene valutato il modello così ottenuto ancora una volta utilizzando lo *Score* come sopra definito. Se il modello risultante è migliore, gli iperparametri di riferimento, quelli sui quali si applica una perturbazione casuale, vengono aggiornati e diventano quelli di quest'ultima architettura. Ancora una volta il modello migliore, ossia quello che

ha minimizzato lo *Score*, viene salvato. Le iterazioni di questa parte dell'algoritmo si ripetono per 2 ore

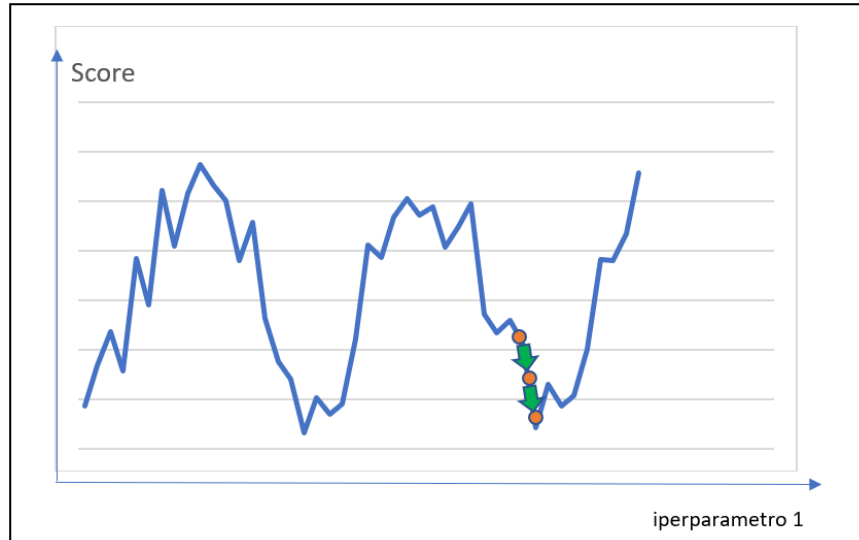


Fig. 3.9 Modalità di ricerca parte 2 algoritmo di tuning di iperparametri

4 Il deep learning

Il deep learning è sicuramente stato uno degli sviluppi più importanti avvenuto nel campo dell'intelligenza artificiale. Si tratta in pratica di reti neurali, introdotte già nel capitolo 3, di carattere multi-livello. Il recente successo del Deep Learning è dovuto a fattori che hanno contribuito a colmare il vuoto in alcune aree che in passato non hanno permesso di raggiungere i risultati attesi, ad esempio:

- L'incremento dei dati a disposizione, che ormai hanno raggiunto quantità molto elevate;
- Lo sviluppo di sistemi di calcolo parallelo altamente performanti basati su GPU (*Graphics Processing Unit*);
- L'ottimizzazione dei metodi di addestramento delle reti neurali, in modo che possano ottenere soluzioni migliori ai problemi che, in passato, hanno impedito ai ricercatori di ottenere risultati significativi.

Grazie alle reti neurali artificiali in grado di analizzare automaticamente dati quali immagini, video, audio o serie temporali, questo ambito sta vivendo anni di rapido progresso, arrivando anche, in molti casi, a superare le prestazioni degli esseri umani.

4.1 Generalità e vantaggi rispetto ai modelli non deep

La logica multi-livello di una rete neurale *deep* può essere descritta tramite una rappresentazione ad archi e nodi.

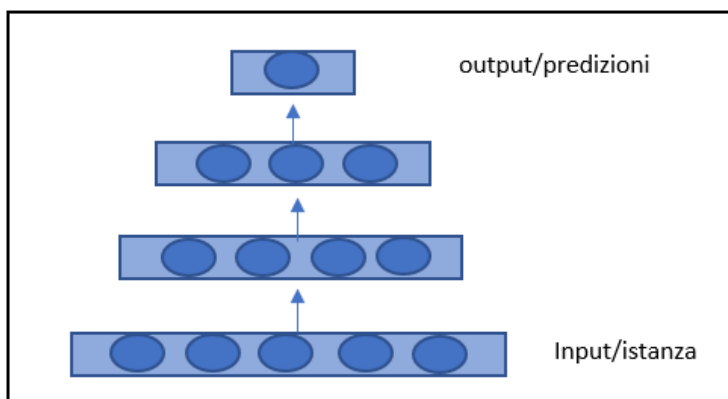


Fig. 4.1 Rete neurale deep

Ogni nodo di figura 4.1, prende il nome di neurone. Quest'ultimo può essere inteso come l'unità elementare di elaborazione della rete. Elaborano le informazioni presenti nei neuroni dei livelli precedenti, tramite l'utilizzo di parametri, e restituiscono un risultato. I parametri, dunque, collegano l'*input* della rete all'*output* in modo tale che le predizioni sono funzione dell'*input* e dei parametri stessi. L'unica vera grande differenza tra modelli *deep* e modelli non *deep* sta nel numero di processamenti che la rete compie prima di ottenere l'*output*.

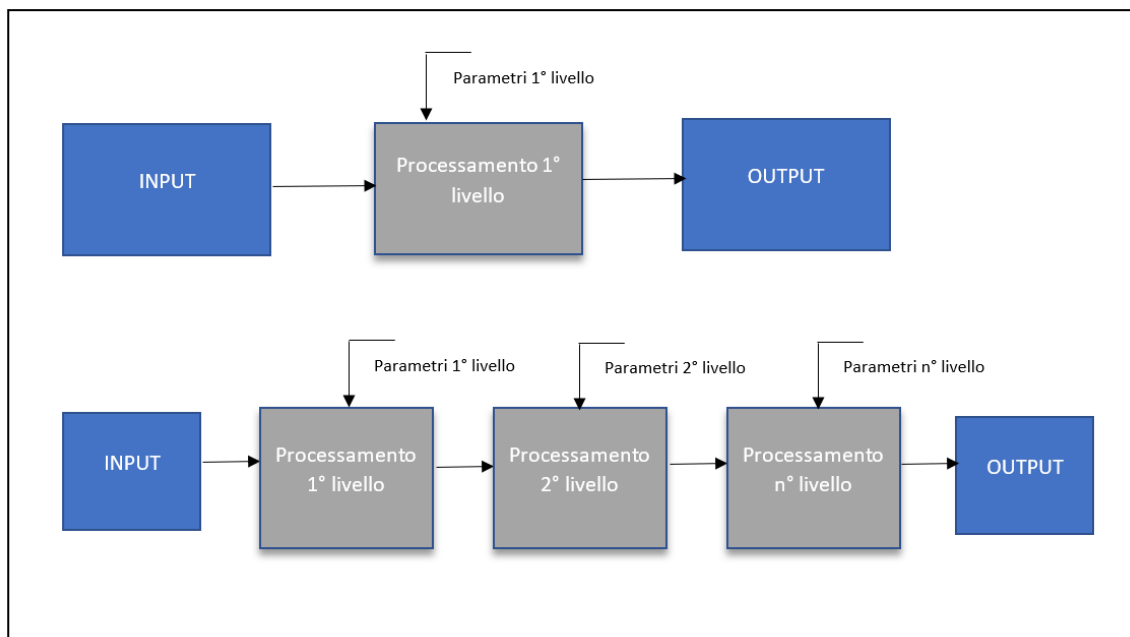


Fig. 4.2 Differenza tra processamenti eseguiti da una rete non deep, in alto, e da una rete deep, in basso

E' importante sottolineare che il processo di apprendimento di un modello *deep* rimane pressoché invariato rispetto a quello descritto in sezione 3.3 precedentemente.

Un ulteriore aspetto che deve essere chiarito è la natura dei dati di *input*: una rete si definisce *deep* per il numero di processamenti che vengono eseguiti in successione, come esemplificato in figura 4.2, e non per la natura dei dati di *input*. Per esempio, il formato caratteristico dei dati può essere un vettore, come in figura 4.1, nel caso in cui si abbiano a disposizione una serie di predittori in sequenza, oppure, può essere una matrice, nel caso in cui sia richiesto dall'applicazione dare in pasto all'algoritmo una struttura 2D.

La figura 4.2 racchiude anche la chiave del successo di questi modelli. All'aumentare del numero di processamenti e del numero di livelli il numero di parametri dell'algoritmo aumenta di conseguenza e, come già discusso in sezione 3.4, il problema di *underfitting* viene contrastato. La legge che detta la numerosità dei parametri può essere scritta in questo modo:

$$\text{numero parametri} = \sum_{i=0}^l (n_i * n_{i+1} + n_{i+1}) \quad (4.1)$$

Sia l il numero di livelli della rete

Sia n_i il numero di neuroni del livello i

Sia n_{i+1} il numero di neuroni del livello $i+1$

Naturalmente, a causa di tutti questi parametri, il problema opposto potrebbe nascere. Sappiamo però che il problema di *overfitting* può essere limitato tramite la regolarizzazione L2 oppure la regolarizzazione *dropout*.

La numerosità dei livelli e la quantità dei neuroni sono personalizzabili e dettano i risultati del processo di allenamento: sono dunque iperparametri.

4.2 Modelli convoluzionali

I modelli convoluzionali hanno rivoluzionato il campo dell'intelligenza artificiale che si occupa del riconoscimento di oggetti, classificazione, e riconoscimento facciale. Questo settore di studio viene spesso chiamato dalla comunità scientifica *computer vision*. Sono stati introdotti da Yann LeCun negli anni 80 ma hanno trovato una propria applicabilità solamente nell'ultimo decennio a causa del loro importante costo computazionale.

In pratica, le reti convoluzionali possono essere intese come una complicazione delle reti neurali che prendono come dati di *input* delle matrici e non dei vettori. Le matrici vengono, nei primi livelli, processate generando altre matrici e in seguito vengono appiattite originando dei vettori che possono essere dati in pasto ad una struttura uguale a quella già descritta nella scorsa sezione. In figura 4.3 le strutture 2D vengono schematizzate con un rettangolo blu. E' importante sottolineare che nella figura 4.3 la semplice schematizzazione non comprende quei livelli, normalmente frapposti tra quelli

convoluzionali, che prendono il nome di *max pooling*. Questi livelli si occupano di propagare solo le caratteristiche più importanti della rete, tralasciando quelle non rilevanti e riducendo inoltre il costo computazionale di tutto il modello.

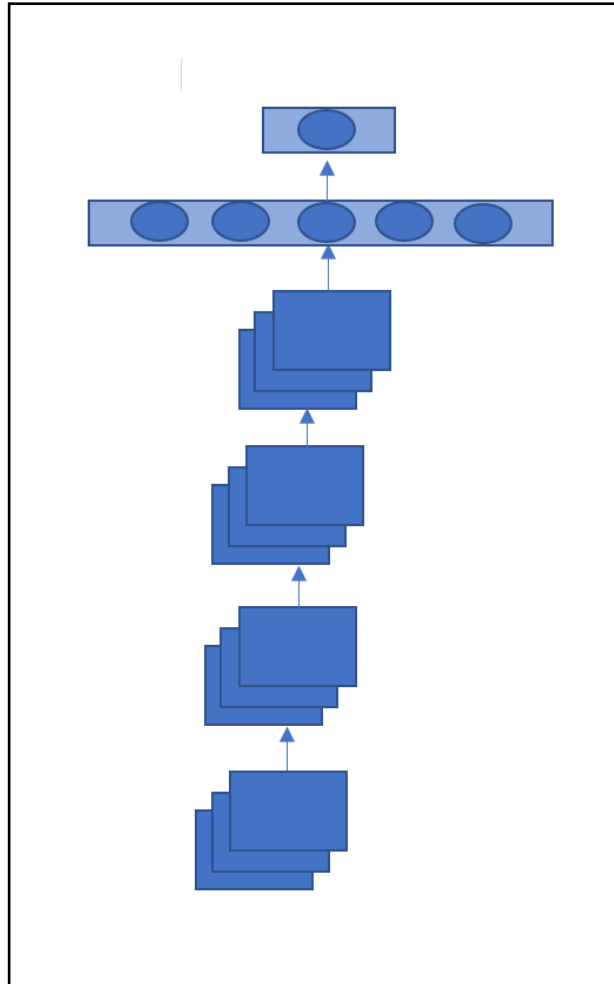


Fig. 4.3 Schematizzazione rete neurale convoluzionale

Le matrici vengono processate, nei primi livelli, con l'operazione di convoluzione. In quest'ultima si può ritrovare la chiave del successo di queste reti. In figura 4.4 sono visualizzabili gli *input* e gli *output* di una operazione di convoluzione.

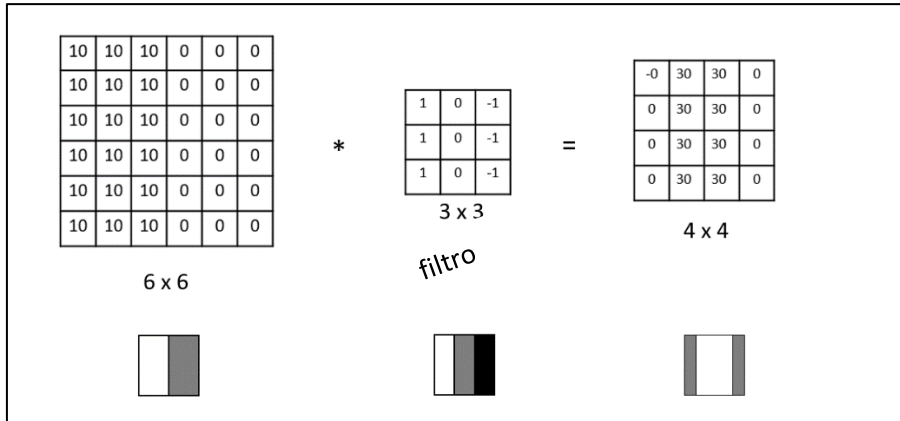


Fig. 4.4 Convoluzione ottenuta con filtro per l'identificazione di bordi verticali

L'operazione si esegue facendo scorrere una matrice più piccola, chiamata filtro o *kernel* sulla matrice di *input*. Si effettua dunque una moltiplicazione cella a cella per ottenere il valore da inserire nella matrice di *output*. In una rete convoluzionale, però, i valori assegnati al filtro non sono predeterminati, come in figura 4.4, bensì sono i parametri che la rete può imparare tramite il già descritto processo di apprendimento. In genere, si applicano più operazioni di convoluzione sullo stesso *input*. Ciò significa che più filtri sono applicati alla stessa struttura iniziale generando tante strutture 2D di *output* quanti sono i *kernel* utilizzati. Per questa ragione, nella schematizzazione di figura 4.3, più matrici sono impilate.

Un'altra importante caratteristica dei modelli convoluzionali che li differenzia da quelli descritti nella sezione precedente è relativa alla gestione delle matrici di *input* e delle matrici intermedie. Infatti, ogni matrice, è sottoponibile potenzialmente ad un processo chiamato *padding* che ne estende la dimensione. Questa operazione prevede che la struttura 2D venga incorniciata con degli zeri lungo tutti i bordi. In questo modo, nel momento in cui la convoluzione viene eseguita, ad ogni cella viene data pari importanza e si evita una veloce riduzione delle dimensioni delle matrici. La legge che determina la riduzione della dimensione da un livello all'altro, quando applicata la convoluzione, è:

$$n' = n - f + 1 + 2p \quad (4.2)$$

Sia n la dimensione (altezza o larghezza) della matrice di *input*

Sia n' la dimensione della matrice di *output*

Sia f la dimensione del filtro (altezza o larghezza)

Sia p la dimensione della cornice di *padding*

Dalla formula è facile evincere che, senza il termine *padding*, la matrice di *output* diventa più piccola. Questo problema diventa molto grave quando più livelli convoluzione sono applicati in sequenza. Il *padding* offre una rapida ed efficace soluzione.

Esiste un altro parametro, chiamato *stride*, che detta la riduzione della dimensione delle matrici. Se il filtro scorre lungo la matrice di *input* spostandosi di volta in volta di una sola cella si dice che l'operazione di convoluzione ha *stride* di uno. Allo stesso modo, se il filtro scorre lungo la matrice di *input* spostandosi di volta in volta di n celle si dice che l'operazione di convoluzione ha *stride* di n . In figura 4.5 è rappresentato graficamente l'utilizzo di *stride* diversi in una convoluzione.

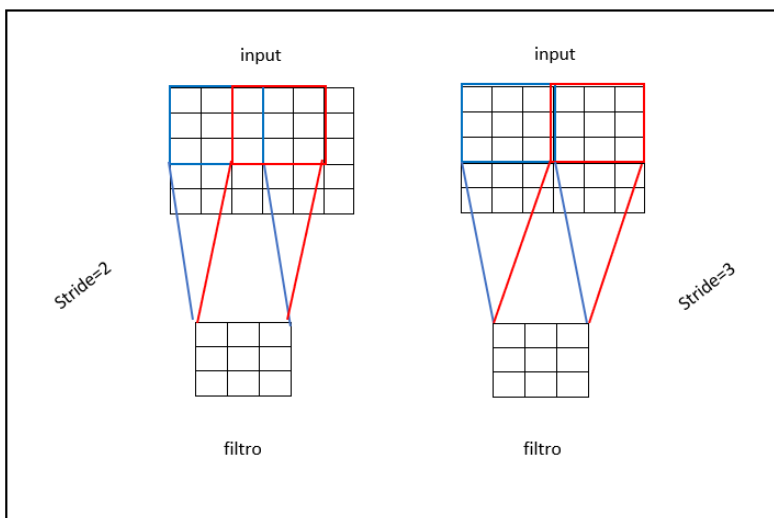


Fig. 4.5 Rappresentazione convoluzioni con stride diversi

Dopo questa breve introduzione alle reti convoluzionali è forse più facile comprendere le ragioni per le quali sono così performanti. A parità di numero di livelli e dimensione dell'*input* quelle convoluzionali hanno in genere una numerosità di parametri ridotta rispetto alle reti neurali classiche. La quantità di questi ultimi dipende solamente dalla dimensione dei filtri che vengono utilizzati e non sono funzione della dimensione dell'*input* e dei livelli della rete, come avveniva per i modelli precedentemente descritti.

Di conseguenza, garantiscono un'attenuazione naturale del problema di *overfitting*, e, aspetto non meno importante, richiedono un minore spazio di memoria designato al salvataggio dei parametri.

Anche in questa sezione sono stati incontrati diversi iperparametri: la dimensione della cornice di *padding*, la quantità di filtri applicati, la dimensione dei filtri e il parametro *stride*.

4.3 Modelli ricorrenti

I modelli ricorrenti si usano soprattutto nel campo dell'intelligenza artificiale che si occupa della traduzione automatica, riconoscimento vocale e di modelli linguistici. Questo settore di studio viene spesso chiamato dalla comunità scientifica *natural language processing* (NLP). Sono stati introdotti da John Hopfield negli anni 80 ma hanno trovato una propria applicabilità solamente a partire dal 2007 quando la rete ricorrente chiamata LSTM (*Long Short-Term Memory*) ha iniziato a rivoluzionare il mondo del riconoscimento vocale.

Seppur esistano innumerevoli varianti di questa rete, sia a livello strutturale, sia a livello di equazioni che ne determinano le logiche e gli *output*, hanno tutte delle caratteristiche comuni. L'idea di fondo di tutte le reti ricorrenti è di utilizzare le informazioni sequenzialmente: in una rete tradizionale si assume che tutti gli *input* e *output* siano indipendenti gli uni dagli altri, ma per alcuni *tasks* non è un approccio corretto. Infatti se ad esempio si desidera predire la parola successiva all'interno di una frase è importante sapere quale parola sia stata elaborata in precedenza. Queste reti sono chiamate "ricorrenti" perché eseguono lo stesso *task* per ogni elemento della sequenza in esame, con l'*output* che dipende dalla computazione precedente. Si può pensare alle reti ricorrenti come se avessero una "memoria" che catturi l'informazione su quello che è stato già calcolato. Teoricamente, le ricorrenti possono utilizzare informazioni in sequenze arbitrariamente lunghe, ma in pratica sono limitate a guardare indietro solo di qualche step. Il grafo in figura 4.6 mostra una rete ricorrente che viene dispiegata in una rete completa. Dispiegare significa semplicemente di riscrivere la rete nella sua sequenza completa.

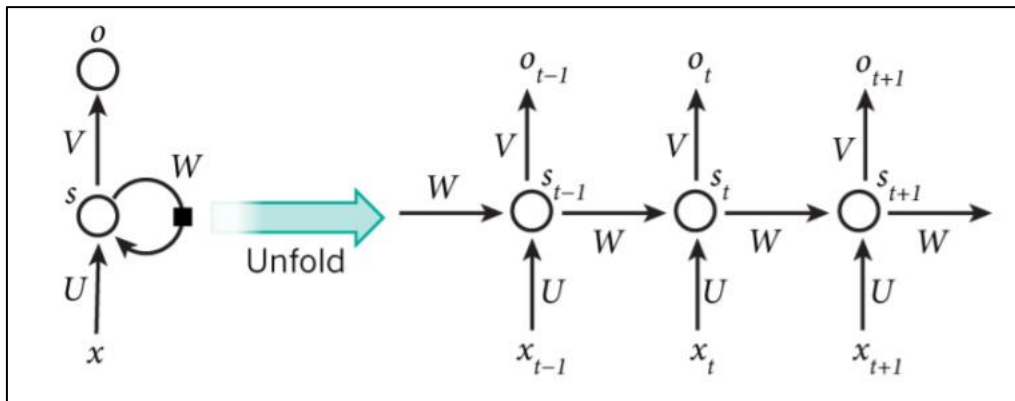


Fig. 4.6 Rete ricorrente dispiegata, con parametri U, V e W

Le formule che regolano il calcolo in una rete ricorrente sono le seguenti:

- x_t è l'*input* allo *step* t .
- s_t è lo stato nascosto allo *step* t . E' la "memoria" della rete; s_t è calcolato in base allo stato nascosto precedente e l'*input* allo *step* corrente corrisponde: $s_t = f(U * x_t + W * s_{t-1})$.

Si può pensare allo stato nascosto s_t come la memoria della rete, il quale cattura l'informazione su cosa sia successo negli *step* precedenti. A differenza di una Rete Neurale tradizionale, che utilizza diversi parametri ad ogni livello, una rete ricorrente condivide gli stessi parametri (U, V, W sopra) in tutti gli *step*. Questo riflette il fatto che si stia eseguendo lo stesso *task* ad ogni *step*, solo con diversi *input*. Ciò riduce notevolmente il numero totale di parametri che la rete deve imparare.

Detto ciò, la figura 4.6 rappresenta una delle più semplici reti ricorrenti che si possono trovare in letteratura. Nel corso degli anni sono state sviluppate reti molto sofisticate come le già citate LSTM (*Long Short Term Memory*).

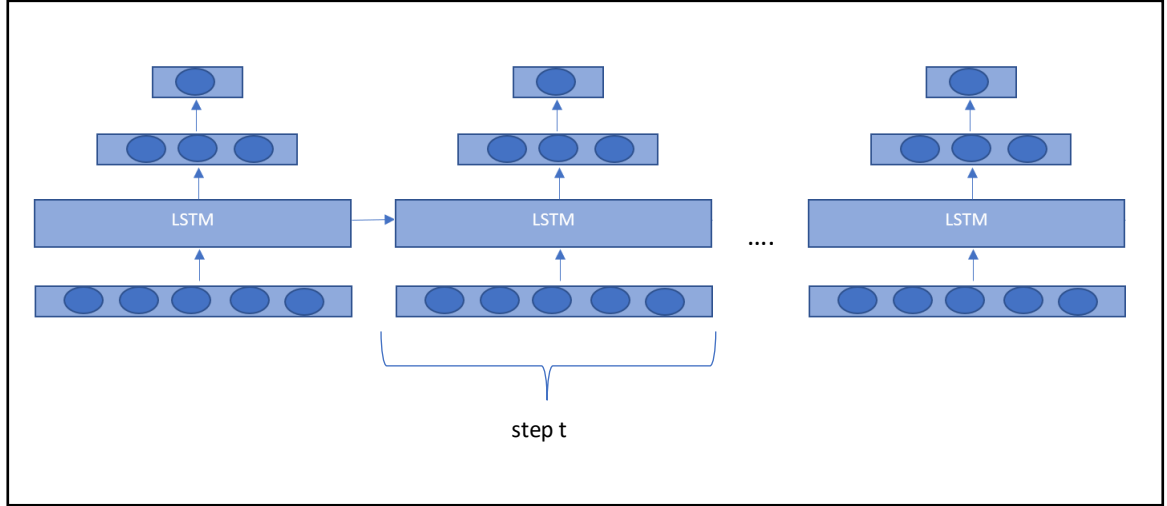


Fig. 4.7 Rappresentazione modello ricorrente con LSTM con struttura *many-to-many*

Data la notazione seguente, nelle successive righe sono riportati i calcoli che ne dettano il funzionamento.

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \quad (4.3)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \quad (4.4)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \quad (4.5)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \quad (4.6)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \quad (4.7)$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>}) \quad (4.8)$$

Sia $\tilde{c}^{<t>}$ il valore candidato per l'aggiornamento di $c^{<t-1>}$

Sia $x^{<t>}$ l'input allo step temporale t

Sia Γ_u il risultato dell'equazione che detta l'aggiornamento dello stato di memoria $c^{<t>}$

Sia Γ_f il risultato dell'equazione che detta la conservazione dello stato di memoria $c^{<t>}$

Sia Γ_o un valore che detta in quale proporzione $c^{<t>}$ influenzerà $a^{<t>}$

Siano i valori in W e in b parametri che il modello impara

$c^{<t>}$ è da intendere come la memoria della rete. Come si può notare, dipende infatti da $c^{<t-1>}$ ossia la memoria della rete dello *step* precedente. Esiste anche un valore candidato a rimpiazzare $c^{<t-1>}$ che si chiama $\tilde{c}^{<t>}$. Il valore aggiornato $c^{<t>}$ è controllato anche da Γ_u e Γ_f che vengono anche chiamati *update gate* e *forget gate*. Γ_u è un numero tra 0 e 1 che detta in quale proporzione il valore candidato è da considerare. Γ_f è anch'esso un numero tra 0 e 1 che decide in quale proporzione la memoria della rete debba essere preservata.

5 Il metodo utilizzato e i risultati ottenuti

In questo capitolo viene descritto il *dataset* su cui sono stati costruiti i modelli sopra menzionati. Viene poi presentato il metodo e i risultati ottenuti. Infine vengono descritte le prove eseguite in laboratorio in cui i modelli più performanti della fasi precedenti sono utilizzati per stimare la vita residua di componenti meccanici. In figura 5.1 sono schematizzate le attività svolte che verranno descritte in modo più approfondito nelle prossime sezioni.

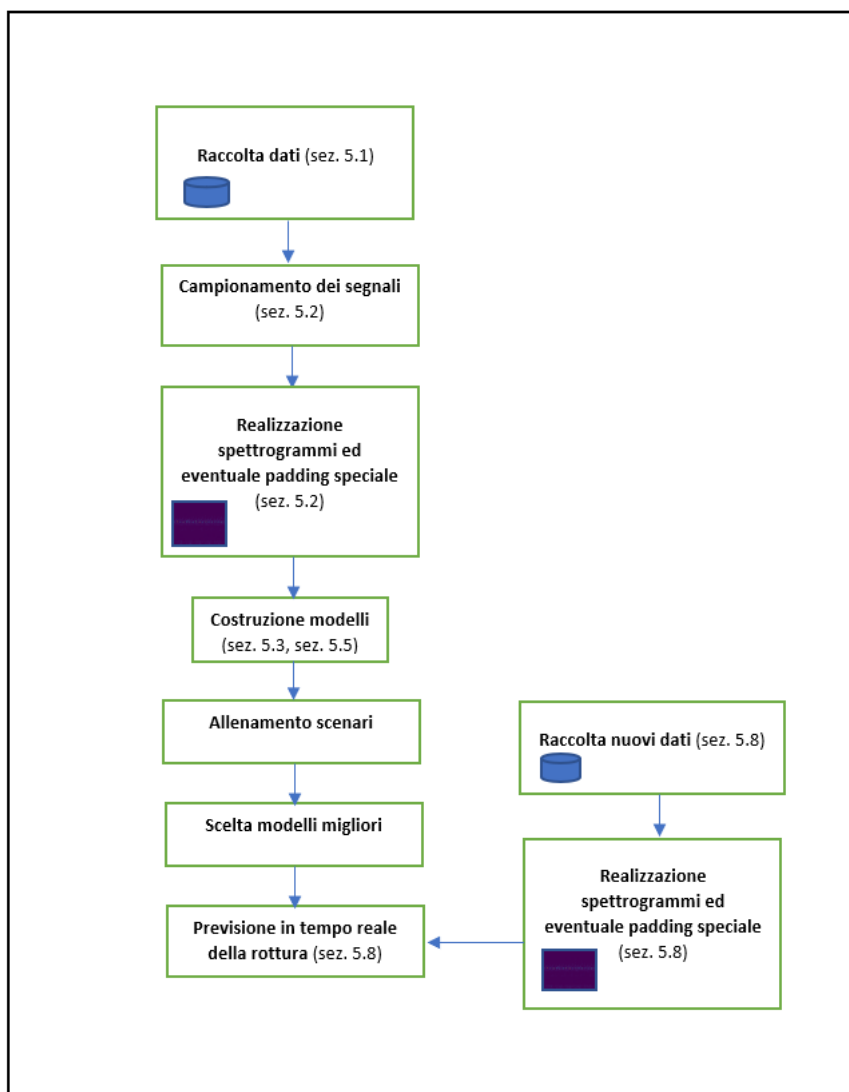


Fig. 5.1 Schematizzazione metodo utilizzato

5.1 Il caso studio

Il dipartimento industriale dell'università di Bologna ha a disposizione una macchina appositamente realizzata per generare grandi quantità di dati di vibrazione. La macchina è costituita principalmente da un motore elettrico il quale moto è scaricato su di una cinghia. Quest'ultima è sottoposta ad uno stress molto elevato perché la velocità del motore è di 700 rpm e le coppie frenanti sono molto elevate.

La cinghia è un organo di trasmissione meccanica utilizzato per collegare in modo leggermente elastico, ma comunque solidale, due alberi, mediante l'uso di pulegge, il cui interesse è piuttosto elevato. Grazie alla loro flessibilità possono essere utilizzate in vari ambiti, dai motori come cinghia dei servizi, cinghia di distribuzione o cinghia di trazione, come essere utilizzate in varie macchine operatrici o utensili, così come nell'elettronica, come nei lettori ottici. Il campo di applicabilità così grande di questi elementi rende attraente lo sviluppo di algoritmi di manutenzione predittiva. Il caso studio, inoltre, assume ancora più rilevanza se si prende in considerazione che in letteratura il problema dello stimare la vita residua di cinghie è raramente affrontato. Per finire, le cinghie, se si rompono, sono spesso responsabili di danni che si possono propagare anche verso altri componenti (danni a catena). Perciò la prevenzione del guasto del componente cinghia ha una notevole importanza.

La macchina a disposizione dell'università di Bologna ha installati tre sensori di accelerazione (accelerometri) che rilevano i dati di vibrazione. In particolare, ogni sensore è triassiale: effettuano misurazioni simultanee in tre direzioni ortogonali, per l'analisi di tutte le vibrazioni a cui viene sottoposta la struttura. Ciò implica che in totale i segnali monitorati sono nove.

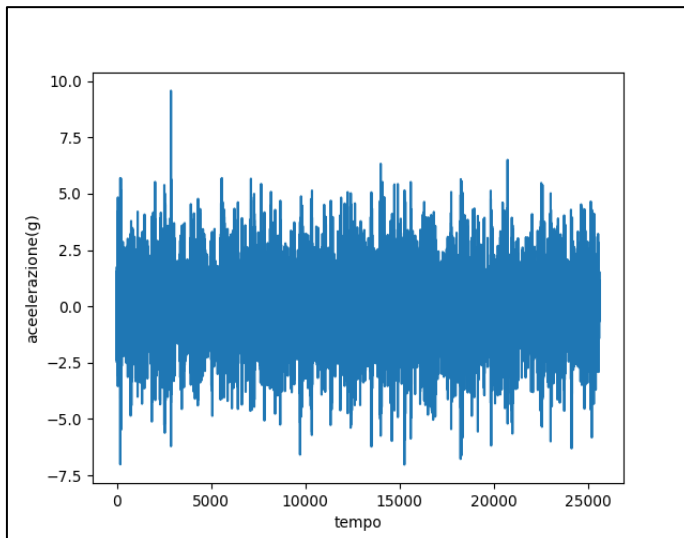


Fig. 5.2 Segnale di accelerazione, primo asse, primo sensore durante una prova di laboratorio

In figura 5.2 sono graficati due secondi di segnale di vibrazione relativi a quando è stato svolta la prova su cinghia 14. La frequenza di campionamento è di 12800 Hz dunque 12800 punti al secondo sono generati per ogni segnale. Diverse prove sono state eseguite utilizzando cinghie diverse. I segnali sono stati misurati e immagazzinati per tutta la durata della vita delle cinghie. Le cinghie, sottoposte a tensione maggiore, si rompono velocemente rendendo il processo di raccolta dati agevole. In particolare, la vita totale di un componente non supera mai le cinque ore.

Nome cinghia	Coppia frenante	Modalità operativa	Tempo di rottura (min)	Data esecuzione prova
cinghia 12	10%	1	269,3	15/04/2021
cinghia 14	10%	1	238,9	15/04/2021
cinghia 15	20%	2	125,7	17/05/2021
cinghia 16	20%	2	195,9	20/05/2021
cinghia 17	20%	2	215,86	24/05/2021

Tabella 5.1 Dati caratteristici per ogni cinghia

In tabella 5.1 sono mostrati alcuni dati caratteristici per ogni cinghia. Si può notare che le prove sono state eseguite utilizzando due modalità operative. La modalità operativa 1 indica che è stata utilizzata, per quella prova, una coppia frenante nel sistema del 10%. La modalità operativa 2 denota invece che è stata utilizzata invece una coppia frenate del 20 %.

5.2 La manipolazione dei dati di ingresso

I nove segnali rilevati dai tre accelerometri sono stati processati prima di essere inseriti all'interno dei modelli di apprendimento. In primo luogo, sono stati considerati solamente due secondi di segnale ogni cento secondi di vita del componente. Dunque, una parte dei dati è stata ignorata appositamente per evitare problemi legati al costo computazionale che gli algoritmi di apprendimento avrebbero dovuto sostenere. In particolare, la quantità di dati prima dell'inizio di questo *step* era di circa 160 gigabytes mentre, terminato il processamento, la totalità dei dati si è ridotta a circa 1,8 gigabytes.

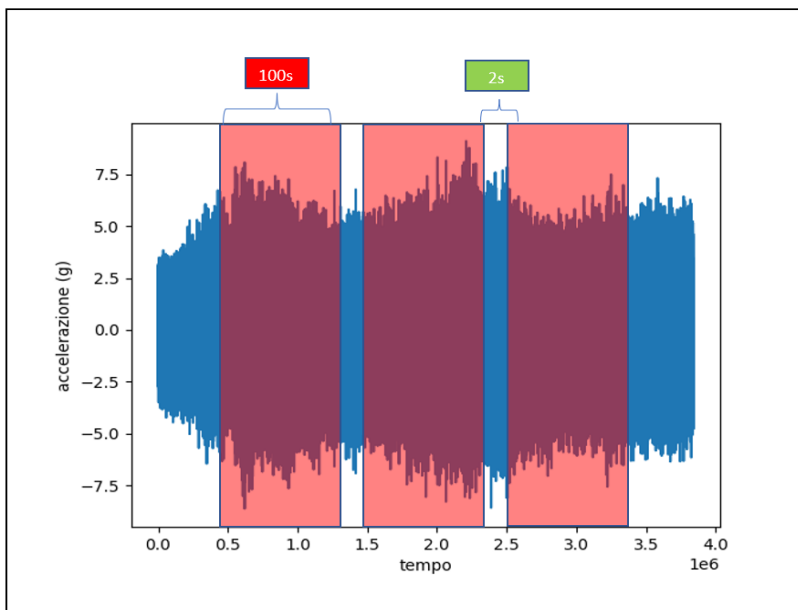


Fig. 5.3 Rappresentazione campionamento eseguito ogni cento secondi

I segnali così elaborati subiscono un ulteriore processo. Di ognuno di essi viene calcolato lo spettrogramma. Quest'ultimo è una struttura 2D che raccoglie informazioni di tempo e frequenza riguardo al segnale iniziale. Uno spettrogramma è la rappresentazione

grafica dell'intensità di accelerazione in funzione del tempo e della frequenza. In altre parole, indica le frequenze che si attivano maggiormente in un determinato istante temporale.

Questo strumento è molto usato in analisi vibrazionale perché rivela caratteristiche importanti dei segnali che altrimenti, nel dominio del tempo, non sarebbero riconoscibili. Il ciò rende lo spettrogramma una tecnica robusta che permette di snellire il processo di apprendimento successivo degli algoritmi di intelligenza artificiale.

Uno spettrogramma si ottiene suddividendo l'intervallo della forma d'onda da analizzare in sotto-intervalli uguali, dette finestre temporali, di durata da cinque a dieci ms e calcolando la trasformata di Fourier della parte di forma d'onda contenuta in ciascuna finestra. Quest'ultima fornisce l'intensità di vibrazione in funzione della frequenza. Le Trasformate di Fourier, relative alle diverse finestre temporali, vengono poi assemblate a formare lo spettrogramma.

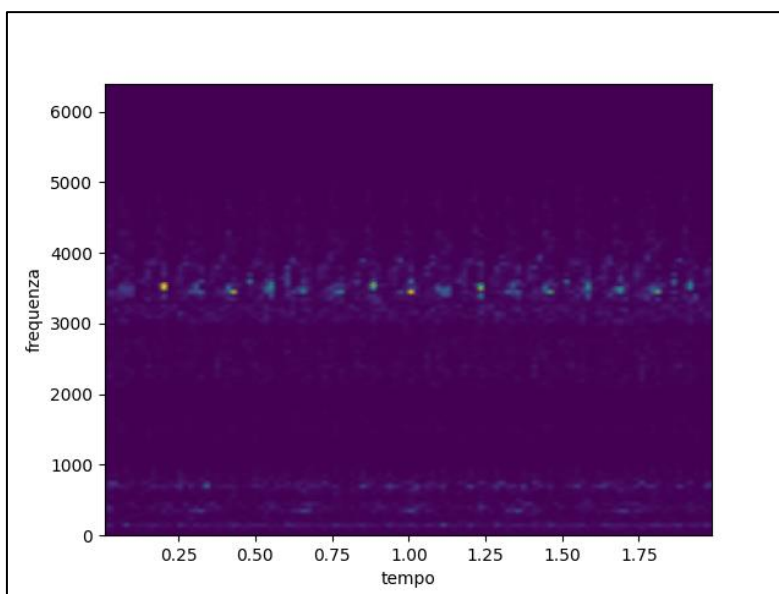


Fig. 5.4 Rappresentazione grafica di uno spettrogramma realizzato da un segnale di due secondi

Gli spettrogrammi così realizzati sono i dati di *input* (insieme alle etichette/soluzioni che valorizzano la vita residua del macchinario) dei modelli *deep* che vengono utilizzati in questo caso studio. Questi algoritmi cercano di apprendere il legame presente tra gli

spettrogrammi dati in *input* e la vita residua della cinghia. Si parte dal presupposto che i segnali cambino in funzione dello stato di salute della cinghia. Per esempio, si può supporre che determinate frequenze si attivino solo in caso in cui la cinghia si trovi in stadio avanzato della propria vita. Questo permette all'algoritmo di riconoscere il nuovo stato e modificare le previsioni di conseguenza.

In alcuni allenamenti, poi, sono stati modificati leggermente gli spettrogrammi di ingresso per fornire all'algoritmo informazioni riguardo alla cinghia presa in considerazione. In particolare, è stata applicata una cornice di *padding* differente in funzione della modalità operativa alla quale era sottoposta la cinghia. Il che dovrebbe permettere all'algoritmo di rivelare immediatamente le diverse modalità operative e regolare le previsioni di conseguenza. In figura 5.5 è rappresentata la tecnica appena descritta che nelle prossime pagine di questa trattazione assumerà il nome di *padding* speciale.

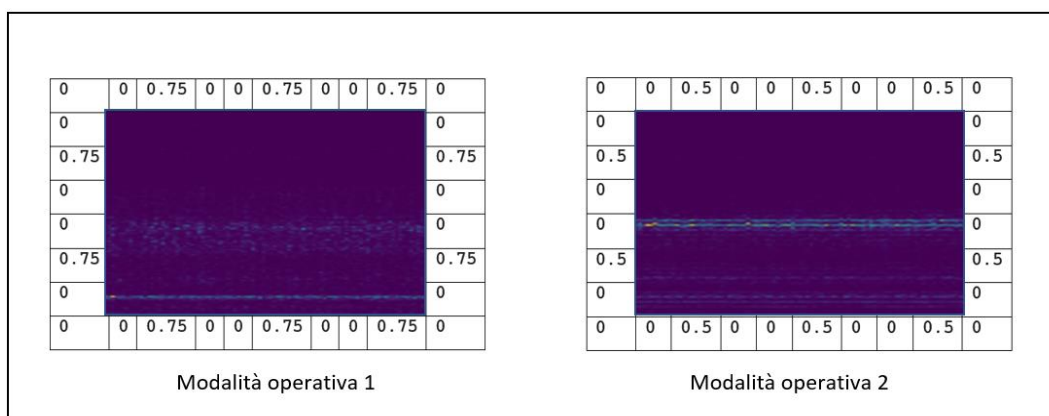


Fig. 5.5 Rappresentazione della tecnica di padding speciale

In seguito verrà approfondito il modello convoluzionale mentre in sezione 5.5 sarà descritto il modello convoluzionale con LSTM.

5.3 Costruzione del modello convoluzionale

Supponiamo di misurare segnali all'istante i e che la rottura della cinghia avvenga all'istante $i + t$. I dati che vanno forniti a questo modello sono gli spettrogrammi relativi ai segnali dell'istante i e il tempo t (etichetta/soluzione). Questo processo viene ripetuto ogni qualvolta i segnali sono stati considerati, quindi ogni cento secondi. La totalità degli spettrogrammi e delle etichette così associate rappresentano i dati di apprendimento per questo modello. Nella tabella 5.2 sono descritti i vari livelli di questo modello. In particolar modo, è specificato per ogni livello quali iperparametri sono soggetti all'algoritmo di *tuning*, come descritto in sezione 3.4, e quali sono fissati.

Numero livello	tipo	iperparametri fissi	iperparametri soggetti a tuning
1	convoluzionale	dimensione filtro=5x5, padding=1, max pooling è presente	numero filtri, stride, percentuale dropout
2	convoluzionale	dimensione filtro=3x3, padding=0, max pooling è presente	numero filtri, stride, percentuale dropout
3	convoluzionale	dimensione filtro=3x3, padding=0, max pooling è presente	numero filtri, stride, percentuale dropout
4	feed-forward		numero neuroni, percentuale dropout

Tabella 5.2 Descrizione di ogni livello del modello convoluzionale

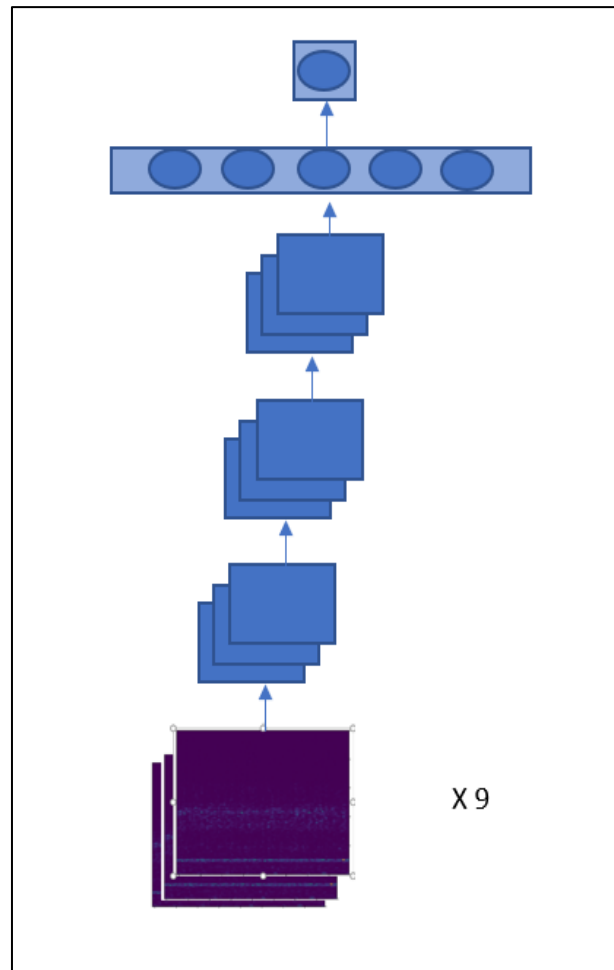


Fig. 5.6 Schematizzazione modello convoluzionale usato nella trattazione

Il modello così sviluppato viene allenato diverse volte, come descritto precedentemente, per valutare diverse configurazioni degli iperparametri soggetti a *tuning*. I dati di una cinghia sono scelti come dati di test per simulare il comportamento dell'algoritmo quando nuove istanze sono date in pasto al modello. Diverse combinazioni di cinghie sono state inserite all'interno dei dati di apprendimento e di test per scoprire come l'aumento del numero di componenti potesse influenzare la bontà delle predizioni. In figura 5.7 sono presentati i vari scenari testati e le cinghie che entrano in gioco.

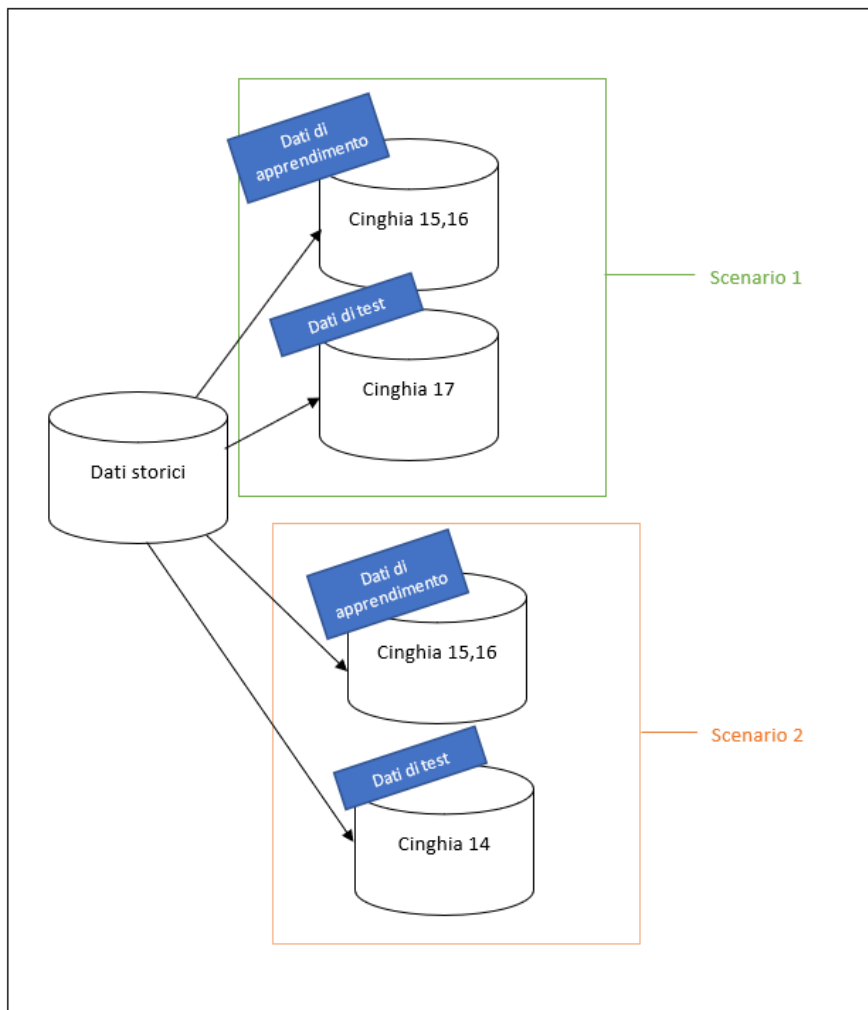


Fig. 5.7 Scenari testati con il modello convoluzionale

A causa dei risultati poco soddisfacenti rispetto alla metrica chiamata Score (3.8), solo pochi scenari sono stati considerati. In sezione 5.5 , verrà descritto un modello più robusto.

5.4 La programmazione del modello convoluzionale

Utilizzando la potente API chiamata *Keras* disponibile sul linguaggio di programmazione *Python* possono essere realizzati modelli di *deep learning* elencando i livelli che lo costituiscono. Per esempio, nel codice 5.1 riportato di seguito, sono presentati livelli del modello convoluzionale usato in questa trattazione. I dati di *inputs* di riga 2 vengono

trasformati tramite l'applicazione di livelli convoluzionali (riga 3, 7 e 11) e in seguito viene applicato un livello lineare per fornire l' *output* (riga 19).

```

1  def model_conv2D(nh,nw,nchan,size_conv,size_dense,drop,drop_1,ac
    tivations,size_stride):
2      inputs = Input(shape=(nh,nw,nchan))
3      X = Conv2D(int(nchan*size_conv)+1, 5, int(size_stride[0]),padd
        ing="same", activation=activations[0])(inputs)
4      X=Dropout(drop)(X)
5      X=BatchNormalization()(X)
6      X=MaxPool2D(2)(X)
7      X = Conv2D(int(nchan*size_conv*1.5), 2, int(size_stride[1]), a
        ctivation=activations[1])(X)
8      X=Dropout(drop)(X)
9      X = BatchNormalization()(X)
10     X=MaxPool2D(3)(X)
11     X = Conv2D(int(nchan*size_conv*2.25), 3, int(size_stride[2]),a
        ctivation=activations[2])(X)
12     X=Dropout(drop)(X)
13     X = BatchNormalization()(X)
14     X=MaxPool2D(2)(X)
15     X = Flatten()(X)
16     X = Dense(size_dense+1, activations[3])(X)
17     X=Dropout(drop_1)(X)

18     X = BatchNormalization()(X)
19     outputs=Dense(1, activation="linear")(X)
20
21     model = keras.Model(inputs=inputs, outputs=outputs)
22
23     return model

```

Listing 5.1 Definizione modello convoluzionale

Il codice 5.2 mostra invece il modo in cui il modello convoluzionale è allenato. Prima di tutto è stato specificato l'algoritmo di ottimizzazione, ossia Adam, dopodiché è indicata la funzione costo (riga 2) che, come già anticipato, è il MSE. L'allenamento inizia dopo il comando di riga 3 che specifica i dati di apprendimento. Questi ultimi sono incamerati nelle variabili `X_train` e `Y_train`.

```

1 opt = keras.optimizers.Adam(learning_rate=alpha)
2 model.compile(optimizer=opt, loss='mean_squared_error')
3 history=model.fit(X_train,Y_train,batch_size=5,epochs=epo,verbo
  e=0)

```

Listing 5.2 Allenamento modello convoluzionale

5.5 Costruzione del modello convoluzionale con LSTM

Il modello convoluzionale con LSTM, appartenente alla classe dei ricorrenti, non considera solamente lo spettrogramma del momento corrente per eseguire una previsione bensì si avvale di tutto lo storico degli spettrogrammi raccolti. In figura 5.8 si può osservare in forma schematica questa caratteristica.

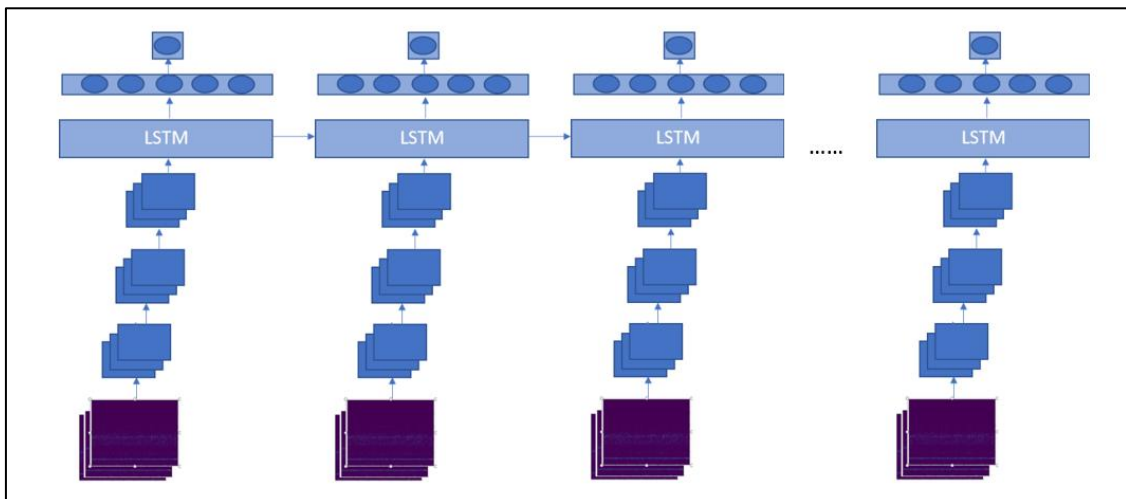


Fig. 5.8 Schematizzazione modello convoluzionale con LSTM usato nella trattazione

Nella tabella 5.3 sono descritti i vari livelli di questo modello.

Numero livello	tipo	iperparametri fissi	iperparametri soggetti a tuning
1	convoluzionale	dimensione filtro=5x5, padding=1, max pooling è presente	numero filtri, stride, percentuale dropout
2	convoluzionale	dimensione filtro=3x3, padding=0, max pooling è presente	numero filtri, stride, percentuale dropout
3	convoluzionale	dimensione filtro=3x3, padding=0, max pooling è presente	numero filtri, stride, percentuale dropout
4	ricorrente LSTM		numero neuroni, percentuale dropout
5	feed-forward		numero neuroni, percentuale dropout

Tabella 5.3 Descrizione di ogni livello del modello convoluzionale con LSTM

Il modello così sviluppato viene allenato diverse volte, come descritto precedentemente, per valutare diverse configurazioni degli iperparametri soggetti a *tuning*. I dati di una cinghia sono scelti come dati di test per simulare il comportamento dell'algoritmo quando nuove istanze sono date in pasto al modello. Anche applicando questo modello, diverse combinazioni di cinghie sono state inserite all'interno dei dati di apprendimento e di test per scoprire come l'aumento del numero di componenti potesse influenzare la bontà delle predizioni. In figura 5.9 sono presentati i vari scenari testati.

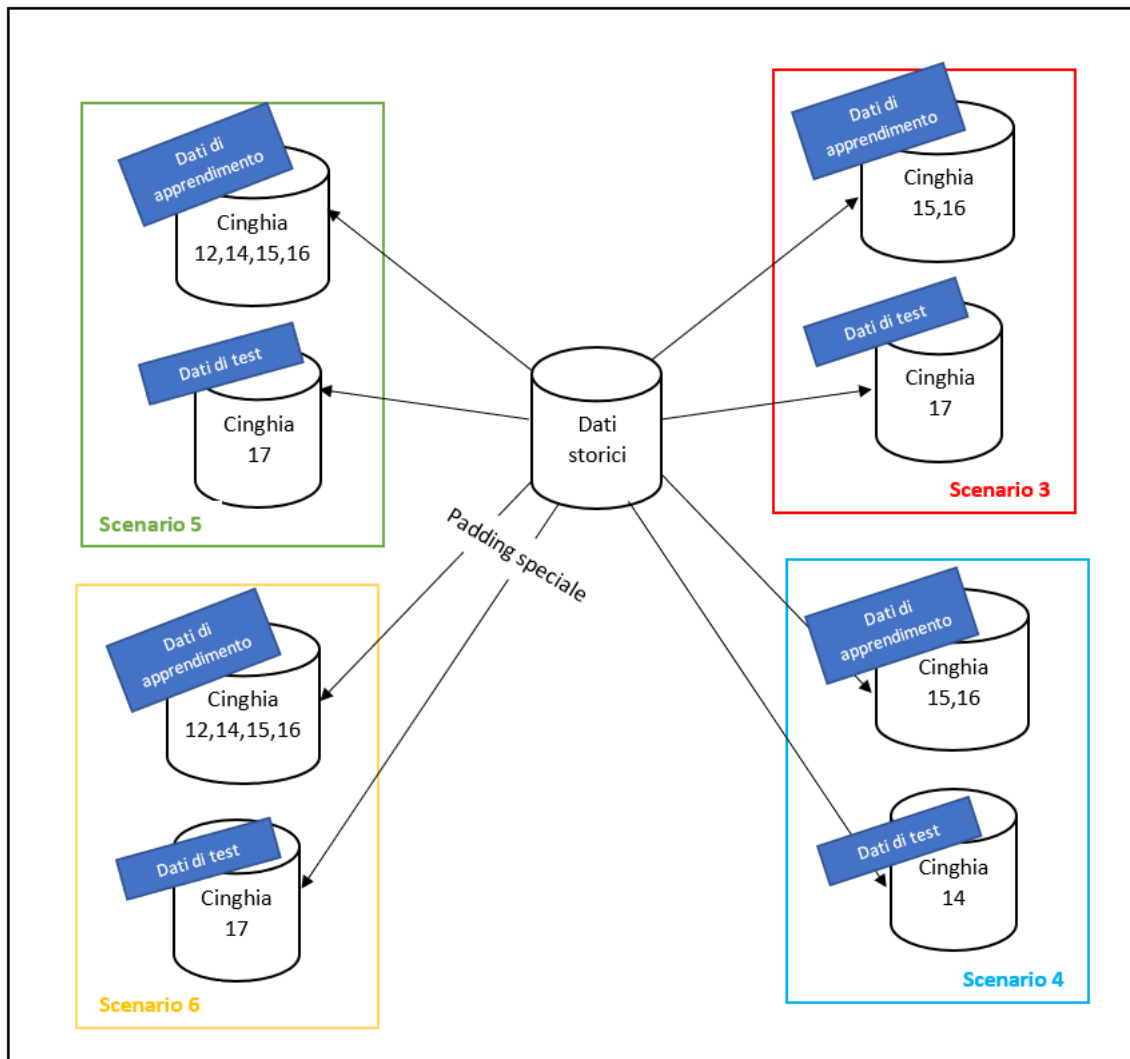


Fig. 5.9 Scenari testati con il modello convoluzionale con LSTM

5.6 Programmazione del modello convoluzionale con LSTM

Anche i modelli ricorrenti sono programmabili utilizzando *Keras*. Per esempio, nel codice 5.3, sono presentati livelli del modello convoluzionale con LSTM usato in questa trattazione. I dati di *inputs* di riga 2 vengano trasformati tramite l'applicazione di livelli convoluzionali (riga 3, 7 e 11). La grande differenza tra i due modelli è data dalla presenza del livello ricorrente LSTM di riga 16.

```

1  def model_conv_lstm(Tx,nh,nw,nchan,size_conv,size_lstm,size_dense,drop,drop_1,activations,size_stride):
2      inputs =Input(shape=(Tx,nh,nw,nchan))

3      X = TimeDistributed(Conv2D(int(nchan*size_conv)+1, 5, int(size_stride[0]),padding="same", activation=activations[0]))(inputs)
4      X=Dropout(drop)(X)
5      X=BatchNormalization()(X)
6      X=TimeDistributed(MaxPool2D(2))(X)
7      X=TimeDistributed(Conv2D(int(nchan*size_conv*1.5),3,int(size_stride[1]), activation=activations[1]))(X)
8      X=Dropout(drop)(X)
9      X=BatchNormalization()(X)
10     X=TimeDistributed(MaxPool2D(3))(X)
11     X=TimeDistributed(Conv2D(int(nchan*size_conv*2.25),3,int(size_stride[2]), activation=activations[2]))(X)
12     X=Dropout(drop)(X)
13     X=BatchNormalization()(X)
14     X =TimeDistributed(MaxPool2D(2))(X)
15     X=TimeDistributed(Flatten()(X)

16     X=LSTM(size_lstm,return_sequences=True)(X)

17     X=BatchNormalization()(X)
18     X=Dropout(drop)(X)
19     X = TimeDistributed(Dense(size_dense+1, activation = activations[3]))(X)
20     X=BatchNormalization()(X)
21     X=Dropout(drop_1)(X)
22     outputs = TimeDistributed(Dense(1, activation = "linear"))(X)

23
24     model = keras.Model(inputs=inputs, outputs=outputs)
25     return model

```

Listing 5.3 Definizione modello convoluzionale con LSTM

Le righe di codice che permettono l'avvio del processo di apprendimento sono identiche a quelle del modello convoluzionale, dunque, per evitare inutili ripetizioni, in questa sezione non sono presentate.

5.7 Risultati ottenuti

I risultati degli scenari descritti nelle sezioni precedenti sono presentati in tabella 5.4 . In particolare, ogni casella rappresenta lo Score (3.8) sui dati di test ottenuto dopo che è stato applicato l'algoritmo di *tuning* per cinque ore (tre ore per parte 1 e due ore per parte 2). Lo Score, come si può notare in sezione 3.4, racchiude informazione riguardo al MAPE, la *normalized_precision* e il *relative_score*. Si tratta quindi di una metrica estremamente compatta e informativa rispetto alla bontà delle previsioni. Dunque , per questa ragione, è stato deciso di inserire nella tabella proprio questa misura.

Nome scenario	Score	Tipo
Scenario 1	0,57	Convoluzionale
Scenario 2	0,38	Convoluzionale
Scenario 3	0,095	Convoluzionale con LSTM
Scenario 4	0,26	Convoluzionale con LSTM
Scenario 5	0,19	Convoluzionale con LSTM
Scenario 6	0,15	Convoluzionale con LSTM

Tabella 5.4 Risultati modelli

Come si può notare dai risultati in tabella, i modelli convoluzionali con LSTM sono molto più robusti rispetto a quelli che non considerano le informazioni nel tempo . In particolar modo, gli scenari evidenziati in verde sono stati identificate come quelli più promettenti. Questi tre modelli, per l'esattezza, sono anche quelli utilizzati per predire la rottura di ulteriori cinghie in tempo reale. La prossima sezione si occuperà di descrivere con più precisione lo svolgimento di queste prove mentre nelle righe che verranno saranno riportati i risultati relativi agli scenari evidenziati in verde in modo più dettagliato.

Per ogni scenario viene riportato:

- 1) Una tabella riassuntiva degli iperparametri scelti dall'algoritmo di *tuning*
- 2) Il MAPE, la *normalized_precision*, il *relative_score* e lo Score sui dati di apprendimento
- 3) Un grafico che rappresenta insieme il *trend* delle previsioni (in blu) e il *trend* della vita residua (in arancione) per i dati di test. Questo strumento è molto utile perché permette di osservare tutte le predizioni che i modelli effettuano e paragonarle con la vita utile residua reale immediatamente.
- 4) Un grafico che rappresenta insieme il *trend* delle previsioni (in blu) e il *trend* della vita residua (in arancione) per i dati di apprendimento
- 5) Un grafico che rappresenta l'errore percentuale per ogni predizione effettuata. Questo grafico serve ad evidenziare in quale punto della vita utile della cinghia il modello effettua maggiori errori e in quali, invece, le previsioni sono affidabili.

Scenario 3

Numero livello	tipo	Iperparametri soggetti a tuning
1	convoluzionale	numero filtri=11, stride=2, percentuale dropout=4%
2	convoluzionale	numero filtri=15, stride=1, percentuale dropout=4%
3	convoluzionale	numero filtri=22, stride=2, percentuale dropout=4%
4	Ricorrente LSTM	numero neuroni=128, percentuale dropout=4%
5	feed-forward	numero neuroni=6, percentuale dropout=6%

Tabella 5.5 Iperparametri scelti dall'algoritmo di tuning

Indicatore di performance	Valore
MAPE	0,122
normalized_precision	0,063
relative_score	0,023
Score	0,095
MAPE rispetto ai dati di apprendimento	0,088

Tabella 5.6 Indicatori di performance per scenario 3

Come si può notare da tabella 5.6, lo scenario 3 presenta risultati ottimi. Lo Score è il più basso in assoluto, l'errore di predizione non possiede una deviazione standard elevata e, anche rispetto ai dati di apprendimento, il MAPE è minimo. Sintomo del fatto che l'algoritmo ha imparato dei *trend* dalle cinghie 15 e 16 e li ha riproposti per cinghia 17.

In seguito sono invece riportati i grafici prima descritti:

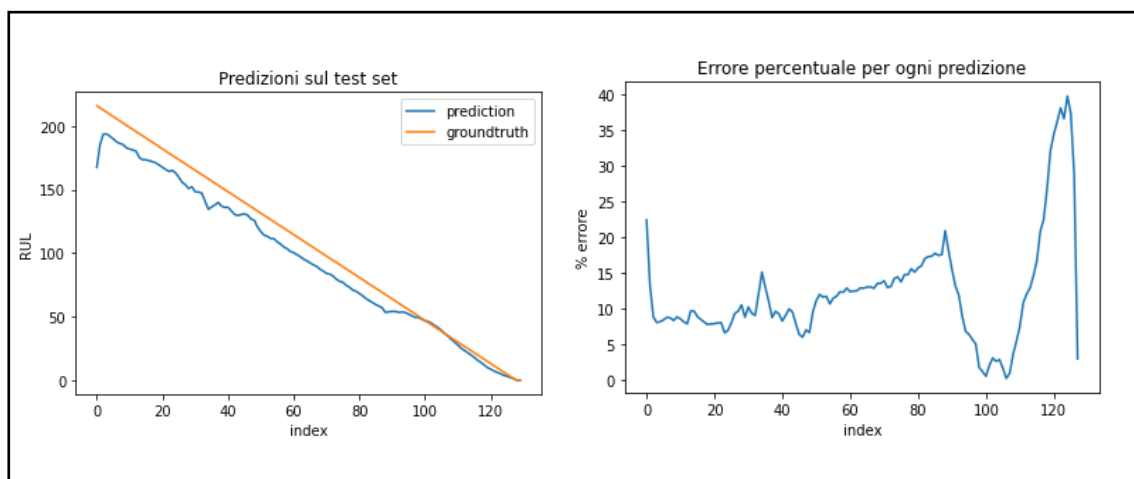


Fig. 5.10 A destra, grafico che mostra le predizioni sui dati test, a sinistra, grafico che valorizza l'errore di previsione

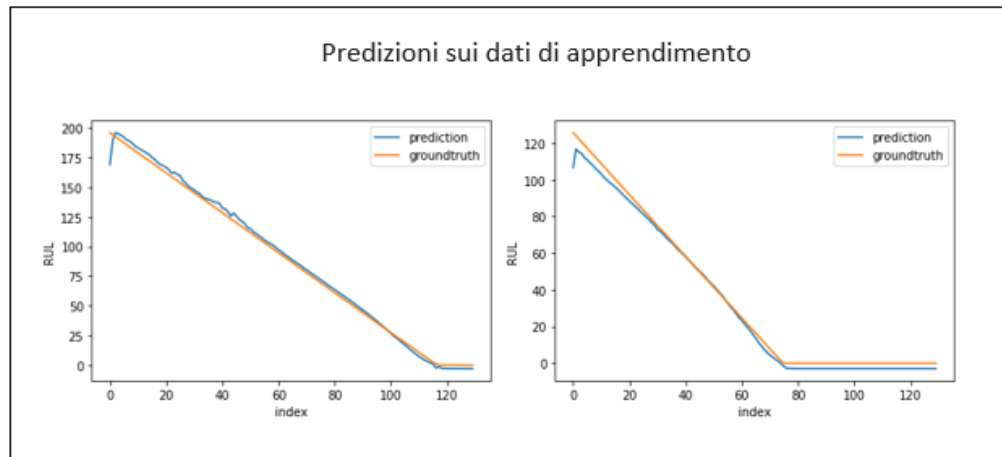


Fig. 5.11 Grafici che mostrano le predizioni sui dati di apprendimento

Scenario 5

Numero livello	tipo	Iperparametri soggetti a tuning
1	convoluzionale	numero filtri=16, stride=2, percentuale dropout=2,7%
2	convoluzionale	numero filtri=23, stride=1, percentuale dropout=2,7%
3	convoluzionale	numero filtri=35, stride=2, percentuale dropout=2,7%
4	Ricorrente LSTM	numero neuroni=128, percentuale dropout=2,7%
5	feed-forward	numero neuroni=11, percentuale dropout =7,2%

Tabella 5.7 Iperparametri scelti dall'algoritmo di tuning

Indicatore di performance	Valore
MAPE	0,161
normalized_precision	0,084
relative_score	0,419
Score	0.191
MAPE rispetto ai dati di apprendimento	0,138

Tabella 5.8 Indicatori di performance per scenario 5

Nello scenario 5 il MAPE, come si può notare da tabella 5.8, è soddisfacente. Fatta 100 la vita utile residua del componente l'errore è di solo 16 unità di tempo. Il relative score, però, è elevato. Ciò significa che mediamente il modello prevede una vita residua maggiore rispetto a quella effettiva.

In seguito sono invece riportati i grafici prima descritti:

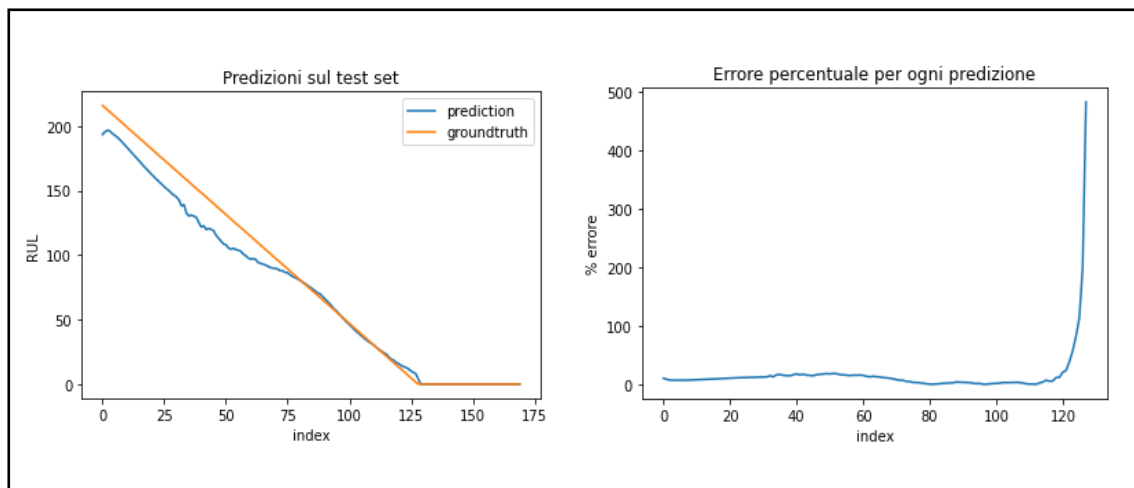


Fig. 5.12 A destra, grafico che mostra le predizioni sui dati test, a sinistra, grafico che valorizza l'errore di previsione

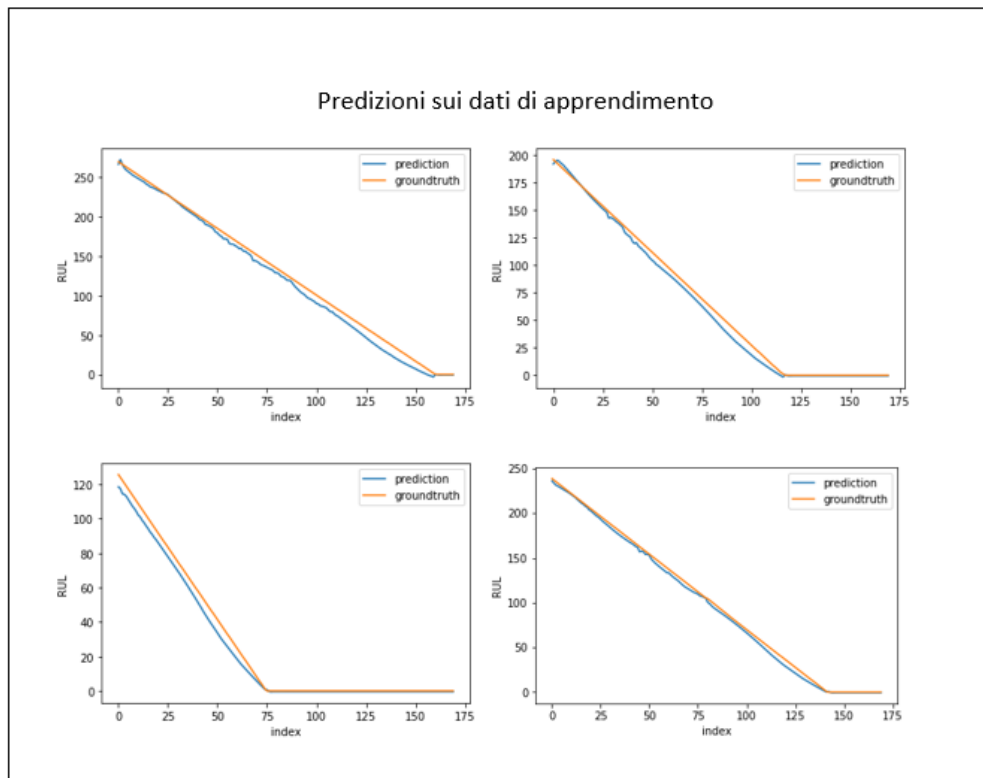


Fig. 5.13 Grafici che mostrano le predizioni sui dati di apprendimento

Scenario 6

Si tratta della stessa combinazione presentata nelle due pagine precedenti. Deve essere sottolineato, però, che in questo caso il *padding* speciale è stato eseguito.

Numero livello	tipo	Iperparametri soggetti a tuning
1	convoluzionale	numero filtri=19, stride=1, percentuale dropout=4%
2	convoluzionale	numero filtri=27, stride=2, percentuale dropout=4%
3	convoluzionale	numero filtri=41, stride=2, percentuale dropout=4%
4	Ricorrente LSTM	numero neuroni=64, percentuale dropout=4%
5	feed-forward	numero neuroni=5, percentuale dropout =5,95%

Tabella 5.9 Iperparametri scelti dall'algoritmo di tuning

Indicatore di performance	Valore
MAPE	0,159
normalized_precision	0,117
relative_score	0,174
Score	0,155
MAPE rispetto ai dati di apprendimento	0,188

Tabella 5.10 Indicatori di performance per scenario 6

Come si può notare da tabella 5.10, la `normalized_precision` in questo caso è leggermente più alta se paragonata a quella degli altri scenari esaminati. Osservando però qualitativamente figura 5.14, la deviazione standard dell'errore non sembra inficiare la validità del modello. In particolar modo, verso fine vita, l'errore percentuale di predizione cala e si mantiene entro limiti ragionevoli.

In seguito sono invece riportati i grafici prima descritti:

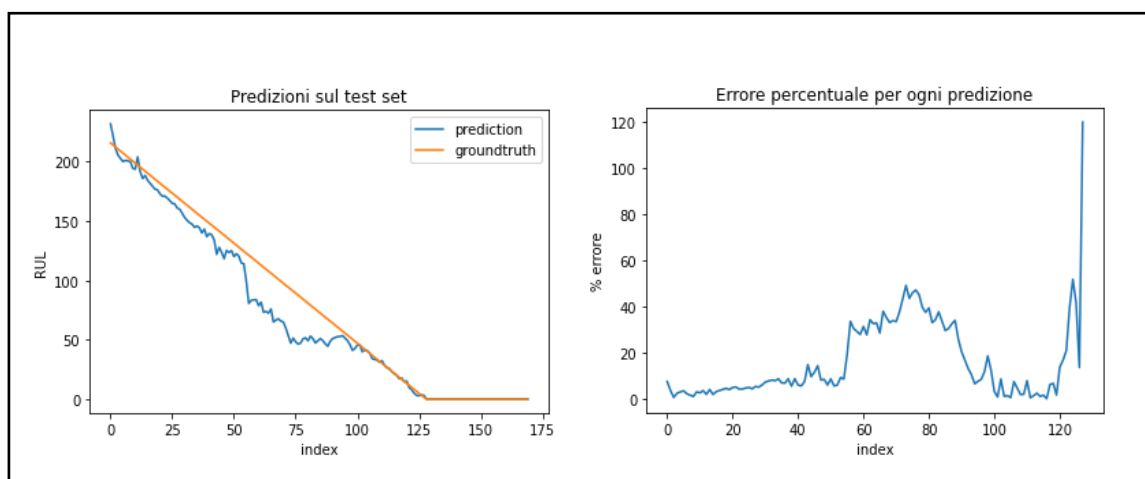


Fig. 5.14 A destra, grafico che mostra le predizioni sui dati test, a sinistra, grafico che valorizza l'errore di previsione

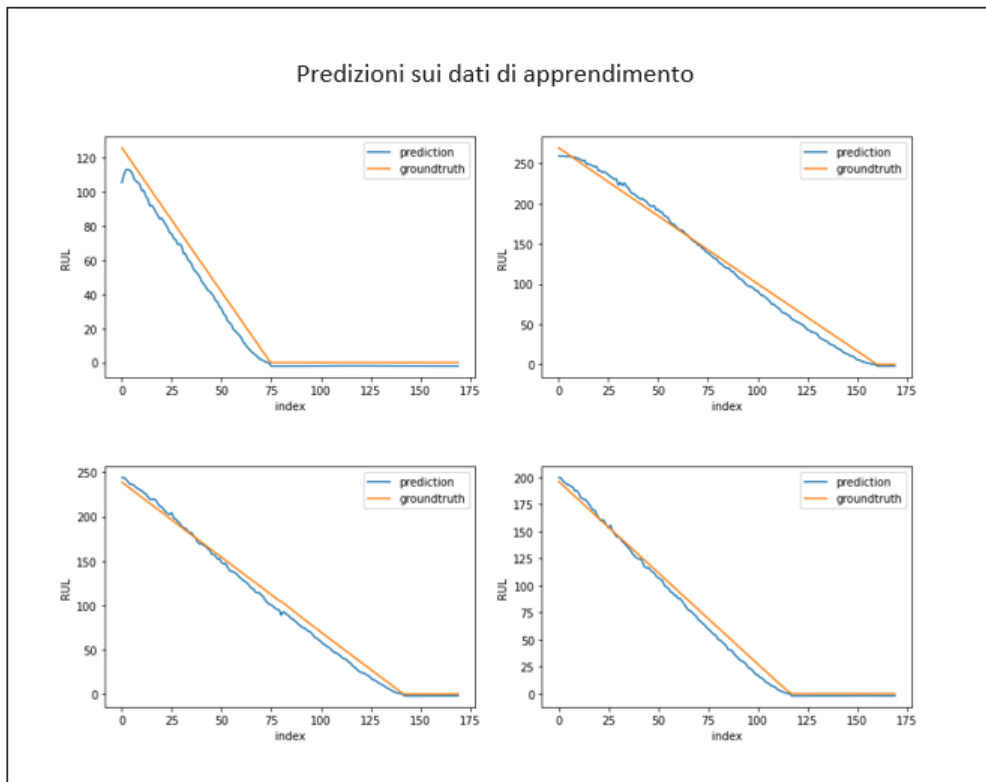


Fig. 5.15 Grafici che mostrano le predizioni sui dati di apprendimento

5.8 Previsione in tempo reale del tempo di rottura delle cinghie

I tre scenari precedentemente descritti sono stati utilizzati per predire la vita residua di alcune altre cinghie in tempo reale. Sono stati eseguiti dunque altre prove in cui cinghie sono state portate alla rottura.

In questa fase doveva essere verificato che i modelli potessero prevedere effettivamente in tempo reale la vita utile residua della cinghia. Due complicazioni che potevano verificarsi a questo punto erano vincoli temporali e limiti di memoria. Per quanto riguarda il vincolo di tempo, il programma che applica i modelli deve fornire una predizione prima che inizino i calcoli necessari per fornire l' *output* successivo. Per quanto riguarda la memoria, invece, era da verificare se le strutture dati, salvate dal programma, nel corso della rottura, fossero sufficientemente "leggere" da non mandare in collasso la RAM del calcolatore impiegato.

Un altro obiettivo che era stato fissato era quello di valutare ulteriormente i modelli. Confrontare, cioè, le previsioni dei modelli con la vita residua effettiva dei componenti.

Il programma di predizione ha incorporate diverse funzionalità. Si interfaccia con i sensori di accelerazione tramite la libreria di Python chiamata nidaqmx, considera nove segnali di due secondi ogni cento secondi, ne computa lo spettrogramma per ognuno e poi fornisce una predizione tramite l'applicazione dei modelli già menzionati. Nel codice 5.4 sono mostrate le librerie che sono state importate.

```
1.  import time
2.  import numpy as np
3.  import tensorflow as tf
4.  from tensorflow import keras
5.  import pickle
6.  import nidaqmx
7.  from nidaqmx import constants
8.  from nidaqmx import stream_readers
9.  import os
10. import Paddings
```

Listing 5.4 Importazione delle librerie necessarie

Il codice 5.5 mostra invece il collegamento tra i sensori e Python. In riga 1 è stato definito un task, che prende il significato di misurazione. Per fare in modo di misurare nove segnali contemporaneamente sono stati aggiunti uno per uno i nove assi relativi ai tre accelerometri (righe 2-10). Nella riga 11 avviene la misurazione vera e propria.

```
1.  with nidaqmx.Task() as task:
2.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod1/ai0")
3.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod1/ai1")
4.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod1/ai2")
5.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod2/ai0")
6.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod2/ai1")
7.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod2/ai2")
8.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod3/ai0")
9.      task.ai_channels.add_ai_accel_chan("cDAQ3Mod3/ai1")
10.     task.ai_channels.add_ai_accel_chan("cDAQ3Mod3/ai2")

11.     task.timing.cfg_samp_clk_timing(rate=12800, sample_mode=constants.AcquisitionType.FINITE, samps_per_chan=25600)
```

```
12. nidaqmx.stream_readers.AnalogMultiChannelReader(task.in_stream).read_many_sample(data, 25600)
```

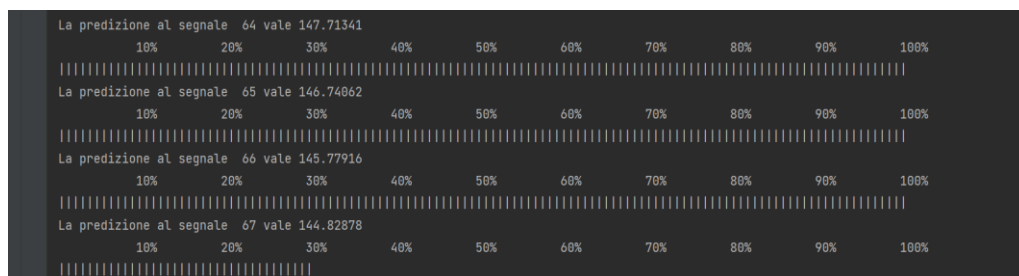
Listing 5.5 Collegamento tra sensori e Python

Nello script 5.6 sono presentati i comandi necessari per eseguire una predizione. Il modello di riferimento è caricato in riga 2 dopodiché la predizione avviene in riga 3.

```
1.
os.chdir("C:\\Users\\utente\\Desktop\\PORTA\\modelli\\new_models\\12-14-15-16_17")
2. model=tf.keras.models.load_model('LSTM_12-14-15-16_17.h5')
3. predictions = np.ravel(model.predict(X))
```

Listing 5.6 Predizione fatta a partire dagli spettrogrammi che si trovano in X

Il programma consente di valutare scenari con e senza *padding* speciale e , quando la cinghia si rompe, è in grado di impostare quell'istante come quello di rottura determinando le soluzioni/etichette corrette.



```
La predizione al segnale 64 vale 147.71341
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
La predizione al segnale 65 vale 146.74062
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
La predizione al segnale 66 vale 145.77916
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
La predizione al segnale 67 vale 144.82878
10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
```

Fig. 5.16 Stampa a video del programma che esegue predizioni

Oltre ad effettuare predizioni, lo *script* memorizza i segnali di *input* (ancora due secondi) ogni cinquanta secondi e li salva in *file* di testo. Quest'ultima caratteristica permette un

futuro riutilizzo dei dati nel caso in cui si volessero applicare o allenare nuovi modelli con diverse caratteristiche.

In figura 5.16 è riportata la stampa a video del programma che applica i modelli di intelligenza artificiale e fornisce predizioni. Nell'immagine si può notare una barra di caricamento che mostra all'utente quando una nuova predizione sarà resa disponibile. I due limiti, presentati ad inizio sezione, non sono risultati vincolanti in nessun modo. Il limite temporale è sempre stato rispettato e lo stato della memoria RAM non ha mai influenzato l'esecuzione del programma.

Per valutare nuovamente i modelli ed eseguire predizioni in tempo reale altre cinghie sono state portate alla rottura. In particolare, tre cinghie sono state portate allo stato di guasto. In tabella 5.11 sono mostrati i dati caratteristici per ogni cinghia.

Nome cinghia	Coppia frenante	Modalità operativa	Tempo di rottura (min)	Data esecuzione prova
cinghia 18	20%	2	134,5	10/12/2021
cinghia 19	20%	2	196,2	10/12/2021
cinghia 20	10%	1	350,4	17/12/2021

Tabella 5.11 Dati caratteristici per ogni nuova cinghia testata

Di seguito sono riportati gli score dei tre scenari, quando applicati alle nuove cinghie.

Score				
	cinghia 18	cinghia 19	cinghia 20	Media per cinghia
Scenario 3	1,51	0,41	0,49	0,80
Scenario 5	1,13	0,25	0,51	0,63
Scenario 6	0,43	0,38	0,48	0,43

Tabella 5.12 Risultati ottenuti sulle nuove cinghie

Per lo scenario che presenta gli Score migliori agglomerati (scenario 6) sono riportati i risultati in modo più dettagliato. In particolare, in tabella 5.13 è mostrato come questo modello performa rispetto a tutte le componenti dello Score.

	cinghia 18	cinghia 19	cinghia 20	Media per cinghia
MAPE	0,45	0,51	0,69	0,55
normalized_precision	0,34	0,19	0,11	0,21
relative_score	0,42	0,08	0	0,16

Tabella 5.13 Risultati scenario 3 rispetto alle nuove cinghie

E' interessante anche visualizzare il *trend* delle predizioni rispetto al *trend* della vita residua dei componenti. In seguito sono riportati, per ogni cinghia, il grafico che mostra questi *trend* e il grafico che mostra l'errore percentuale per ogni predizione.

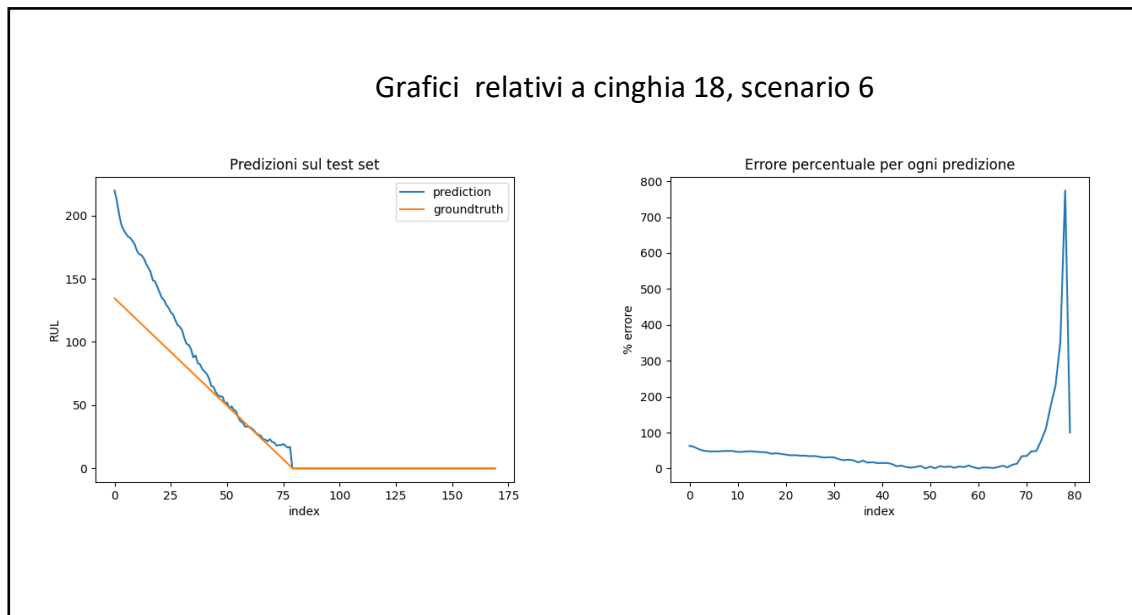


Fig. 5.17 A destra, grafico che mostra le predizioni sui dati test, a sinistra, grafico che valorizza l'errore di previsione

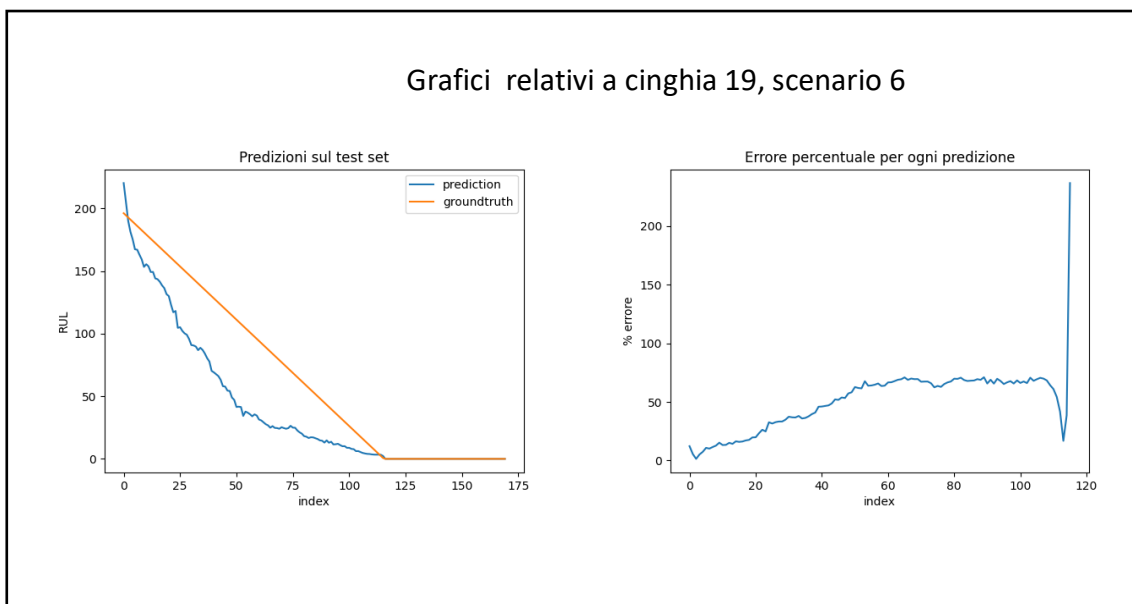


Fig. 5.18 A destra, grafico che mostra le predizioni sui dati test, a sinistra, grafico che valorizza l'errore di previsione

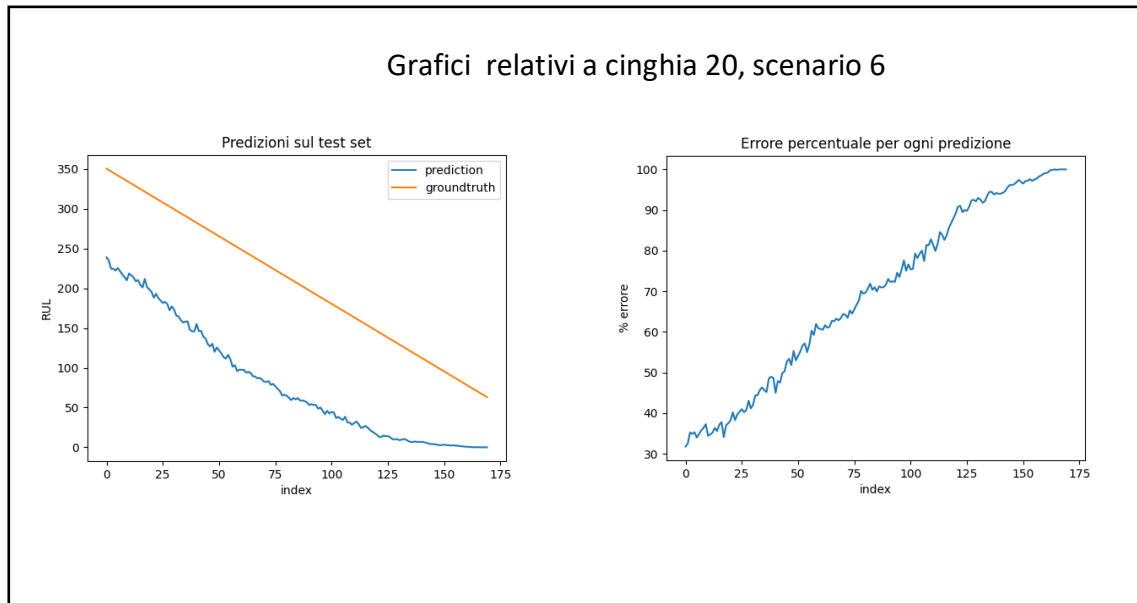


Fig. 5.19 A destra, grafico che mostra le predizioni sui dati test, a sinistra, grafico che valorizza l'errore di previsione

6 Analisi dei risultati e sviluppi futuri

In questo capitolo, una breve analisi dei risultati è svolta al fine di fare chiarezza sui modelli fino ad ora descritti. Cosa accomuna i modelli migliori? I risultati finali sono soddisfacenti? Rispondere a queste domande è importante perché indica quali potrebbero essere sviluppi per nuove ricerche accademiche svolte su questo *dataset*. Al termine di questo capitolo, viene sviluppata qualche idea per continuare lo studio e migliorare così ulteriormente i risultati.

6.1 Analisi dei risultati

In primo luogo, è interessante paragonare i migliori modelli ottenuti dopo il processo di *tuning* di iperparametri. L'algoritmo di ricerca converge verso strutture di reti neurali e iperparametri simili se i modelli sono prestanti?

Si può notare, esaminando sezione 5.5, che l'iperparametro chiamato percentuale di *dropout* rimane pressoché invariato in scenario 3, scenario 5 e scenario 6. Tra tutta la forbice di valori possibili (0-100%) il dropout non supera mai il 7%. Sintomo del fatto che la presenza di una regolarizzazione non troppo accentuata ma esistente è una caratteristica importante di un modello prestante. Questi modelli sono accumulati anche da quell'iperparametro chiamato *stride*. In tutti i casi, infatti, lo *stride* è fissato a 2 per due livelli convoluzionali e fissato a 1 per il restante.

Si può notare invece che gli altri iperparametri non sembrano essere relazionati in nessun modo. Per esempio, il parametro che definisce il numero di neuroni nell'unità LSTM è fissato a 128 per scenario 3 e scenario 5 ma poi diventa 64 per scenario 6.

Un'altra importante caratteristica che accomuna i modelli di successo è dettata dalla quantità dei dati di apprendimento. Inizialmente, dopo le fasi di allenamento iniziale, questa caratteristica non era stata rilevata. Infatti, scenario 3 presenta poche cinghie a dati di apprendimento ed era quello sulla carta che avrebbe dovuto performare meglio (score di 0,095 di scenario 3 rispetto a score di 0,15 di scenario 6). In seguito, però, quando i modelli sono stati testati nuovamente su cinghia 18 19 e 20 la validità di scenario 3 è stata riconsiderata dato che gli altri due modelli si sono rilevati di gran lunga

più efficaci. Dunque, si può concludere che reti con più dati di apprendimento, come scenario 5 e 6, hanno una capacità di generalizzazione migliore e dunque performano in modo migliore quanto testati su nuove istanze.

E' importante notare anche che scenario 6, quello che presenta risultati medi migliori in assoluto, si differenzia da scenario 5 solo dalla cornice di *padding speciale* che è stata applicata e descritta in sezione 5.2. Il che suggerisce un' effetto positivo di quest'ultima quando applicata a modelli di questo tipo. Naturalmente, questa affermazione necessita di ulteriori analisi per poter essere effettivamente confermata.

Per concludere, è necessario considerare il *gap* tra le performance dei modelli quando testati su cinghia 17 (quando veniva eseguito l'algoritmo di *tuning*) e quando testati in seguito su cinghia 18, 19 e 20.

Performance: Score	
Performance modelli testati su cinghia 17	Performance modelli testati su cinghia 18,19 e 20
0,14	0,62

Tabella 6.1 Risultati modelli e *gap* tra le performance

Come è evidenziato da tabella 6.1, le performance dei modelli relativi a scenari 3, 5 e 6 cambiano in modo importante quando testati su cinghia 18, 19 e 20. Esistono due ragioni che possono spiegare il peggioramento delle performance.

Una spiegazione è da ricercare nelle dinamiche che governano l'algoritmo di *tuning*. Queste possono infatti aver mascherato le effettive performance. Ciò è possibile perché intrinsecamente l'algoritmo valuta diverse configurazioni di iperparametri ai fini di ottimizzare le performance sui dati di test. Calibra dunque la rete per ottenere buoni risultati su cinghia 17 ma non necessariamente queste performance si trasferiscono direttamente a tutte le cinghie valutate in seguito. Questo problema è comunemente chiamato *overfitting* dei dati di test perché evidentemente è un problema molto simile a quello di *overfitting* dei dati di apprendimento descritto in sezione 3.4. I modelli infatti

apprendono dai dati di apprendimento e da quelli di test ma non sono in grado di generalizzare.

Inoltre, le performance potrebbero essere calate perché la distribuzione dei dati di ingresso ha subito un cambiamento dopo che sono state eseguite le prove di rottura di cinghia 12,14, 15,16 e 17.

Affermare che le distribuzioni di dati abbiano subito un cambiamento significa sostenere che i segnali abbiano subito una variazione importante nelle loro caratteristiche distintive: ampiezza, frequenze presenti, energia etc. Seppure sono necessarie nuove analisi per dimostrare che questo sia avvenuto, alcune circostanze fanno sospettare l'effettivo cambiamento della distribuzione dei dati:

-Dopo che sono state eseguite le prove che hanno portato alla rottura cinghia 12,14,15,16 e 17 sono stati cambiati diversi organi della macchina che raccoglie i dati. Cuscinetti ed ingranaggi sono stati sostituiti con parti nuove e non usurate. Questo cambiamento di componenti può aver modificato la meccanica del sistema e di conseguenza i segnali rilevati dai sensori.

-Tra l'esecuzione delle prove su cui sono stati allenati i modelli e l'esecuzione delle prove su cinghia 18,19 e 20 sono passati diversi mesi. Questo tempo di pausa potrebbe aver influito sulle caratteristiche delle cinghie in poliuretano utilizzate nonché su altri componenti del macchinario. Anche il cambio di temperatura dell'ambiente potrebbe essere ragione di un cambiamento nella distribuzione dei dati.

Se il cambiamento nella distribuzione dei dati fosse verificato, sarebbe in parte spiegato il calo di performance. Questa osservazione rende evidente la complessità nell'implementare un sistema di questo tipo in azienda. Anche in industria, naturalmente, componenti di macchinari complessi vengono di continuo sostituiti. I modelli idealmente dovrebbero essere robusti a questi cambiamenti e ai cambiamenti della distribuzione dei dati di ingresso che ne conseguono. Il caso studio proposto ha cercato di mettere in luce le problematiche di questi algoritmi nel caso in cui venissero impiegati industrialmente in modo tale da essere pragmatico e vicino alle esigenze aziendali.

Un ultimo commento su cinghia 20 è necessario. Come si può notare da sezione 5.6, questa cinghia ha una durata di vita che supera di gran lunga la vita di tutte le altre cinghie. E' ragionevole pensare che i modelli abbiano performato mediamente peggio su questo componente proprio per questa ragione. In altre parole, l'algoritmo ha appreso di prevedere al più 269 minuti quando una cinghia si trova in perfette condizioni (durata di cinghia 12) ma, la nuova cinghia testata ha una vita di circa 350 minuti. Inevitabilmente l'algoritmo si affida allo storico di dati a disposizione e prevede in accordo con la forbice di valori di etichette/soluzioni che gli sono stati forniti (0-269,4) risultando in stime meno affidabili.

6.2 Sviluppi futuri

Una delle barriere che ha da sempre disincentivato l'adozione di sistemi di manutenzione predittiva di questo tipo in azienda è la disponibilità di dati. Da un lato gli algoritmi funzionerebbero bene con una grande quantità di dati di apprendimento. Dall'altro c'è la necessità di implementare algoritmi che utilizzino pochi dati. Le rotture di componenti in industria, infatti, avvengono tipicamente dopo diversi giorni, se non mesi, di funzionamento nominale. Ciò porta il processo di raccolta dati ad essere lungo, costoso e la maggior parte non praticabile.

Nel caso studio di questa trattazione il *trade-off* appena descritto è stato preso in considerazione e dunque il numero di cinghie inserite a dati di apprendimento non è mai stato elevato. Ciononostante, dopo i risultati ottenuti e i commenti di sezione 6.1 è chiaro che l'espansione dei dati di apprendimento permetterebbe una capacità di generalizzazione migliore nonché predizioni più accurate. Anche i problemi già descritti di *overfitting* dei dati di test e di cambiamento nella distribuzione dei dati di ingresso sono risolvibili accumulando più dati.

Eseguire più prove in laboratorio rompendo più cinghie ed inserendole direttamente all'interno dei dati di apprendimento sarebbe sicuramente il metodo più diretto per raggiungere questo fine. Deve essere ricordato però che i dati spesso non sono disponibili nell'industria perciò utilizzare un approccio del genere renderebbe lo studio astratto e meno connesso alle esigenze delle aziende.

Esistono però altri modi per limitare il problema della scarsità di dati senza però eseguire direttamente le prove di rottura in laboratorio:

-Eseguire un campionamento più frequente

In sezione 5.2 è stato introdotto il tipo di campionamento eseguito. I segnali di accelerazione di *input* sono stati considerati solo ogni cento secondi per limitare il costo computazionale dell'algoritmo. I dati grezzi potrebbero essere però considerati più di frequente in modo tale da espandere la quantità di dati di apprendimento. Per esempio, le accelerazioni potrebbero essere memorizzate ed utilizzate ogni cinquanta secondi ai fini di raddoppiare i dati.

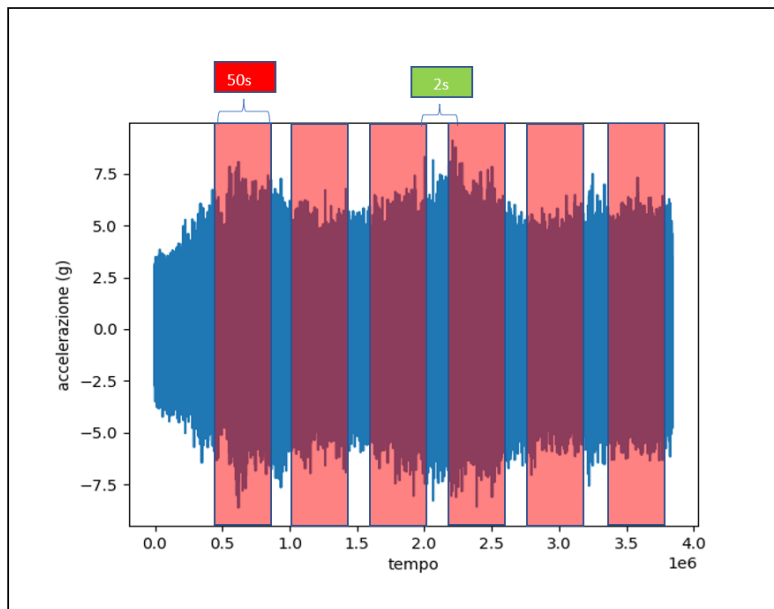


Fig. 6.1 Rappresentazione campionamento più frequente

L'utilizzo di questa tecnica può risultare interessante ma allo stesso tempo è importante sottolineare che, pur applicandola, il numero effettivo di cinghie rimane invariato. La qualità di informazioni aggiuntive che i modelli possono utilizzare, dopo l'applicazione di questa modifica, non è paragonabile a quella che avrebbe a disposizione l'algoritmo nel caso in cui venisse aggiunta una cinghia.

-Utilizzare tecniche di data augmentation

Data augmentation significa, letteralmente, “aumento dei dati”. È un insieme di tecniche che ampliano il dataset a disposizione senza effettivamente raccogliere nuovi elementi: *la data augmentation* applica ai dati già esistenti dei cambiamenti casuali controllati, realizzandone delle copie modificate.

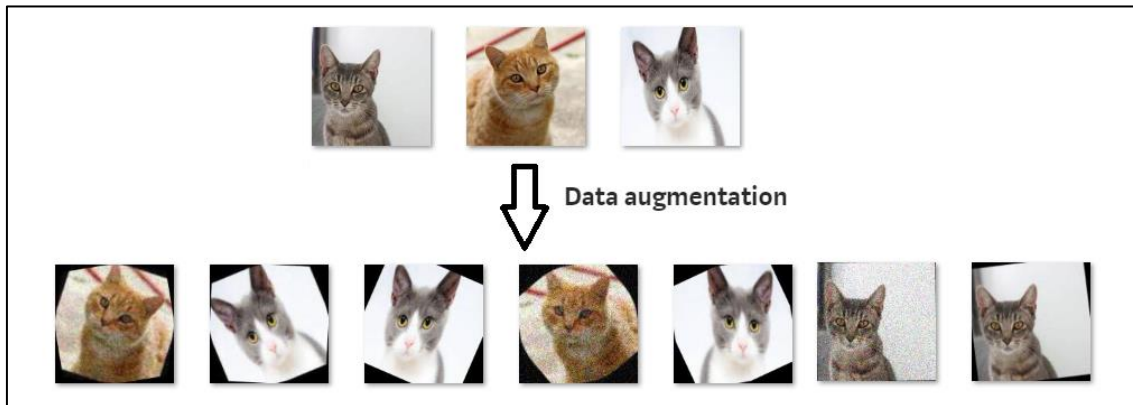


Fig. 6.2 Esempio di data augmentation per computer vision

Questa tecnica è utilizzata in particolar modo in *computer vision* per poter generare più varianti della stessa immagine. La validità di questa tecnica è stata dimostrata da diversi studi. Anche nel campo della manutenzione predittiva *data augmentation* è spesso utilizzata. L’aggiunta di un disturbo randomico ai segnali, l’ utilizzo di metodi di *time wrapping*, lo spostamento delle frequenze e altri metodi sono stati utilizzati al fine di espandere i dati di apprendimento. Hendricks et al. [23] paragona diversi metodi di *data augmentation* per valutare quale possa essere il migliore al fine della predizione della RUL di cuscinetti. Seokgoo Kim et al. [24] utilizza l’approccio di *dynamic time wrapping* per eseguire previsioni ed incrementare la quantità dei dati nella sua applicazione. Sempre più spesso, ultimamente, modelli GANs sono applicati per espandere la quantità di dati a disposizione. Lu et al. [25] predice la vita residua di cuscinetti applicando un modello ricorrente. Si serve di un GAN per espandere la numerosità delle istanze di *input*.

-Utilizzare il transfer learning

Esiste un'altra tecnica che può limitare il problema della scarsa quantità di dati. In questo caso è necessario avere a disposizione però altri *pattern* di rottura non necessariamente

legati al sistema di partenza. Per esempio, se si hanno a disposizione segnali di accelerazione di cuscinetti si può applicare questa tecnica e migliorare le performance rispetto all'obiettivo di stimare la vita residua di cinghie. Questa tecnica avanzata prende il nome di *transfer learning*. Nel campo della manutenzione predittiva questo strumento è stato molte volte utilizzato. Hu et al. [26] utilizza approcci di *transfer learning* e modelli di *deep-learning* per predire la vita residua di cuscinetti. Marei et al. [27] stima la vita residua di taglienti di macchine utensili tramite reti neurali convoluzionali ricorrenti e *transfer learning*.

Un altro possibile sviluppo futuro prevede l'aggiunta della suddivisione in *health stages* (HS) ai modelli già descritti. Come introdotto in sezione 2.2.4, nella prima parte della vita dei componenti i segnali non racchiudono informazioni legate all'aspettativa di tempo di funzionamento del sistema. Solo in un secondo momento, molto spesso prossimo alla rottura vera e propria, l'usura inizia ad influenzare i segnali raccolti. Ciò significa che tendenzialmente previsioni di inizio vita sono spesso poco accurate. La divisione in HS consentirebbe di rilevare quel momento temporale in cui l'usura della cinghia inizia ad essere rilevante e, solo in seguito, far partire l'algoritmo che fornisce le predizioni.

7 Conclusioni

Questo studio è nato con lo scopo di valutare l'efficacia di modelli di manutenzione predittiva basati sui dati. Prima di tutto, sono state confrontate diverse politiche di manutenzione. Sono stati sottolineati le caratteristiche distintive della manutenzione predittiva che la rende più sofisticata e superiore rispetto ad altre. In particolare, il sistema azienda, nel caso in cui fosse adottata questa pratica, registrerebbe dei *savings* importanti grazie al minor tempo di fermo macchina, minor tempo di fermo operatore, costi di scorta ridotti e riduzione di costi di manodopera diretta.

Dopodiché, algoritmi di intelligenza artificiale sono stati applicati per eseguire manutenzione predittiva su cinghie. I tre modelli più performanti sono stati utilizzati per predire la RUL di altre cinghie in diretta.

In questa tesi ho affrontato diversi problemi:

- la grande quantità di dati memorizzati avrebbe reso difficile il processo di apprendimento delle reti neurali. Per questa ragione, è stato proposto un campionamento dei segnali che ha consentito la riduzione della quantità dei dati di *input* fino a 1,8 gigabytes.
- le diverse modalità operative utilizzate durante le prove di laboratorio avrebbero potuto rendere il *task* di stima della RUL molto complesso. Per limitare questa problematica è stata introdotta una cornice di *padding* speciale. In alcuni scenari, i modelli sono stati informati della modalità operativa delle cinghie in modo tale che potessero regolare le predizioni di conseguenza.
- inizialmente l'apprendimento si è dimostrato poco stabile. Per risolverlo, ho utilizzato un algoritmo di *tuning* di iperparametri automatico. Il che ha permesso di valutare diversi scenari senza la necessità di dover eseguire *tuning* manuali imprecisi e snervanti.

Al termine di questa trattazione, un' attenta analisi dei risultati ha permesso di evidenziare i punti chiave su cui fare leva al fine di migliorare ulteriormente le

performance. L'espansione della quantità dei dati di apprendimento è probabilmente la strada da compiere anche se si disincentiva la raccolta diretta di nuovi segnali tramite l'esecuzione di altre prove di rottura. E' infatti preferibile l'utilizzo di altre tecniche per creare segnali sintetici da dare poi in pasto ai modelli, aumentare la frequenza di campionamento oppure adottare strategie di *transfer learning*.

Sviluppare algoritmi che stimassero la RUL di componenti è stato un invito che ho colto con piacere. Perché, innanzitutto, questa tesi mi ha permesso di approfondire le conoscenze che già possedevo rispetto alla manutenzione aziendale.

Inoltre, questo progetto di tesi mi ha permesso di sviluppare competenze di *machine learning* che sono piuttosto distanti rispetto a quelle che ho ottenuto nel corso di ingegneria gestionale. Il che mi ha consentito di espandere il mio sapere, cosa che trovo molto importante.

Sitografia

[1] <https://www.treccani.it/vocabolario/manutenzione/>

[21] https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Bibliografia

[2] Lu, B., Li, Y., Wu, X., Yang, Z.A., 2009. Review of recent advances in wind turbine condition monitoring and fault diagnosis. *Power Electronics and Machines in Wind Applications, PEMWA*, 24–26 June 2009, Lincoln, NE. IEEE, pp. 1–7.

[3] P.C. Paris, F. Erdogan, A critical analysis of crack propagation laws, *J. Fluids Eng.* 85 (1963) 528–533.

[4] P. Baraldi, F. Mangili, E. Zio, A Kalman filter-based ensemble approach with application to turbine creep prognostics, *IEEE Trans. Reliab.* 61 (2012) 966–977.

[5] Z. Huang, Z. Xu, X. Ke, W. Wang, Y. Sun, Remaining useful life prediction for an adaptive skew-Wiener process model, *Mech. Syst. Signal Process.* 87 (2017) 294–306

[6] A. Malhi, R.Q. Yan, R.X. Gao, Prognosis of defect propagation based on recurrent neural networks, *IEEE Trans. Instrum. Meas.* 60 (2011) 703–711.

[7] Z.X. Zhang, X.S. Si, C.H. Hu, An age- and state-dependent nonlinear prognostic model for degrading systems, *IEEE Trans. Reliab.* 64 (2015) 1214–1228.

[8] M. Gašperin, Đ. Juric'ić, P. Boškoski, J. Viz'ntin, Model-based prognostics of gear health using stochastic dynamical models, *Mech. Syst. Signal Process.* 25 (2011) 537–548.

[9] W. Wang, A model to predict the residual life of rolling element bearings given monitored condition information to date, *IMA J. Manage. Math.* 13 (2002) 3–16.

[10] X. Jin, Y. Sun, Z. Que, Y. Wang, T.W.S. Chow, Anomaly detection and fault prognosis for bearings, *IEEE Trans. Instrum. Meas.* 65 (2016) 2046–2054.

[11] J.K. Kimotho, C. Sondermann-Woelke, T. Meyer, W. Sextro, Machinery prognostic method based on multi-class support vector machines and hybrid differential evolution–particle swarm optimization, *Chem. Eng. Trans.* 33 (2013) 619–624.

[12] J. Sun, H. Zuo, H. Yang, P. Michael, Study of ensemble learning-based fusion prognostics, in: *Prognostics and System Health Management Conference*, Macao, 2010, pp. 1–7.

[13] A.K. Mahamad, S. Saon, T. Hiyama, Predicting remaining useful life of rotating machinery based artificial neural network, *Comput. Math. Appl.* 60 (2010) 1078–1087.

[14] R. Huang, L. Xi, X. Li, C. Richard Liu, H. Qiu, J. Lee, Residual life predictions for ball bearings based on self-organizing map and back propagation neural network methods, *Mech. Syst. Signal Process.* 21 (2007) 193–207.

- [15] N. Gebraeel, M. Lawley, R. Liu, V. Parmeshwaran, Residual life predictions from vibration-based degradation signals: a neural network approach, *IEEE Trans. Industr. Electron.* 51 (2004) 694–700.
- [16] L. Wang, L. Zhang, X.-Z. Wang, Reliability estimation and remaining useful lifetime prediction for bearing based on proportional hazard model, *J Cent South Univ* 22 (2015) 4625–4633.
- [17] T. Benkedjouh, K. Medjaher, N. Zerhouni, S. Rechak, Remaining useful life estimation based on nonlinear feature reduction and support vector regression, *Eng. Appl. Artif. Intell.* 26 (2013) 1751–1760
- [18] Heimes, Felix. (2008). Recurrent neural networks for remaining useful life estimation. 1 - 6. 10.1109/PHM.2008.4711422.
- [19] M. Ma and Z. Mao, "Deep-Convolution-Based LSTM Network for Remaining Useful Life Prediction," in *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 1658-1667, March 2021, doi: 10.1109/TII.2020.2991796.
- [20] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533-536. doi:10.1038/323533a0
- [22] D. P. Kingma and J. L. Ba, Adam: A Method for stochastic Optimization. San Diego: The International Conference on Learning Representations (ICLR), 2015.
- [23] Hendriks, J, & Dumond, P. "Data Augmentation for Roller Bearing Health Indicator Estimation Using Multi-Channel Frequency Data Representations." *Proceedings of the ASME 2021 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Volume 10: 33rd Conference on Mechanical Vibration and Sound (VIB)*. Virtual, Online. August 17–19, 2021. V010T10A011.
- [24] Kim, S., Kim, N. H., & Choi, J. -. (2020). Prediction of remaining useful life by data augmentation technique based on dynamic time warping. *Mechanical Systems and Signal Processing*, 136 doi:10.1016/j.ymssp.2019.106486
- [25] Lu, H., Barzegar, V., Nemani, V. P., Hu, C., Laflamme, S., & Zimmerman, A. T. (2021). GAN-LSTM predictor for failure prognostics of rolling element bearings. Paper presented at the *2021 IEEE International Conference on Prognostics and Health Management, ICPHM 2021*, doi:10.1109/ICPHM51084.2021.9486650
- [26] Hu, T., Guo, Y., Gu, L., Zhou, Y., Zhang, Z., & Zhou, Z. (2022). Remaining useful life prediction of bearings under different working conditions using a deep feature disentanglement based transfer learning method. *Reliability Engineering and System Safety*, 219
- [27] Marei, M., & Li, W. (2022). Cutting tool prognostics enabled by hybrid CNN-LSTM with transfer learning. *International Journal of Advanced Manufacturing Technology*, 118(3-4), 817-836.

Uno speciale ringraziamento va al Chiar.mo professor Alberto Regattieri che si è dimostrato sempre disponibile nel chiarirmi dubbi e che mi ha introdotto al problema oggetto di questa trattazione. Ringrazio di cuore l'ingegnere Francesca Calabrese e l'ingegnere Raffaele Piscitelli che mi hanno assistito lungo tutto il percorso di redazione della tesi. Il loro aiuto è stato indispensabile e i loro feedback sempre puntuali e rilevanti.