UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Gestionale

Stima della vita utile residua per la manutenzione predittiva mediante reti neurali

Remaining useful life estimation for predictive maintenance using neural networks

Relatore: Chiar.mo

Prof. Andrea Monteriù

Tesi di Laurea di:

Lorenzo Paoletti

A.A. 2020/2021

INDICE

1. Introduzione

Uno dei principali costi degli impianti di produzione è senz'altro la manutenzione, argomento principale di questa tesi in cui verrà analizzato e studiato uno dei principali metodi per eseguirla nel modo più efficiente possibile. Si vedrà, in particolare, l'utilizzo della manutenzione predittiva (Predictive Maintenance, PDM) per la stima della vita utile residua (Remaining Useful Life, RUL) attraverso l'ausilio delle reti neurali artificiali (Artificial Neural Network, ANN). La stima sarà effettuata con l'utilizzo di tre reti neurali artificiali diverse. Lo studio inizia con l'introduzione dei vari tipi di manutenzione, e la descrizione del funzionamento del metodo predittivo, in particolare come si stima la vita utile residua di un macchinario e in che modo si può raggiungere la maggiore precisione. Successivamente si vedrà cosa sono e come funzionano le reti neurali artificiali che saranno utilizzate per stimare la vita utile residua; nel caso studio considerato si adopereranno la rete neurale di base (Multilayer Perceptron, MLP), la rete neurale ricorsiva (Recurrent Neural Network, RNN) e la rete neurale convoluzionale (Convolutional Neural Network, CNN). Dopodiché si applicheranno le tre reti neurali artificiali per stimare la vita utile residua ad un dataset della National Aeronautics and Space Administration (NASA) riguardante cento motori a turboventola; si analizzeranno i risultati per confronto, al fine di poter valutare quale rete neurale artificiale risulta più precisa nella stima della vita utile residua. In particolare, saranno confrontati i risultati delle tre reti neurali artificiali riguardo due motori specifici: il numero 33 e il 49. Sicuramente un'applicazione di questo tipo, può rendere la manutenzione predittiva, nonostante sia piuttosto costosa, talmente efficiente da essere

indispensabile; ed incrementarne quindi lo sviluppo in campo industriale. L'ambiente che verrà utilizzato per implementare gli algoritmi della manutenzione predittiva e le reti neurali artificiali, è quello di Matrix Laboratory (MATLAB) con l'utilizzo del toolbox per la manutenzione predittiva; che nello specifico consente di gestire i dati dei sensori, progettare indicatori di condizione e stimare la vita utile residua di una macchina. Si vedrà quindi, di seguito e nel dettaglio, la scrittura dell'algoritmo su MATLAB per le tre diverse reti neurali artificiali.

2. Stato dell'arte

Una delle principali voci di spesa in un sistema di produzione è senz'altro quella della manutenzione. Si possono effettuare diversi tipi di manutenzione; le principali sono tre: la reattiva (Reactive Maintenance, RM), la preventiva (Preventive Maintenance, PVM) e la predittiva. La prima si basa semplicemente sulla risoluzione di un malfunzionamento, ovvero non appena, per esempio, un pezzo si rompe o un macchinario si guasta, sostituisco il pezzo e risolvo il problema. La manutenzione preventiva si basa invece su un controllo periodico dei pezzi e/o dei macchinari dell'impianto al fine di riconoscere se l'oggetto che viene ispezionato è a fine vita o meno; la periodicità è in genere stimata in base alle indicazioni del produttore e si cerca quindi di evitare malfunzionamenti anticipando possibili guasti con questa tipologia di controlli; dunque, si tratta di una manutenzione programmata e a scadenze prefissate. In ogni caso può accadere che la manutenzione preventiva non sia in grado di anticipare un malfunzionamento, infatti si riscontra che generalmente la probabilità di un guasto è più alta nello stato iniziale per la possibilità che un oggetto sia difettoso, successivamente decresce, fino a tornare a risalire quando questo inizia a degradarsi; inoltre l'usura dei materiali non è costante nel tempo, infatti dopo un certo periodo potremmo avere materiali del tutto compromessi e altri per niente degradati; è quindi difficile conoscere il periodo stabile di un'attrezzattura. In una situazione di pezzo difettoso la manutenzione preventiva generalmente non è in grado di intervenire perchè il primo intervento è generalmente programmato in un

momento successivo a guasto già avvenuto. Inoltre, la manutenzione preventiva porta con sé un ulteriore problema molto importante; spesso, per essere eseguita, infatti, richiede che il processo da esaminare venga interrotto con conseguente ritardo di produzione o comunque un costo da sostenere nonostante in quel momento non si crei di certo valore. Quindi è importante usare un monitoraggio continuo e connesso sempre. Per ovviare ai problemi sopra esposti, dove soprattutto è fondamentale che non ci siano guasti, ha preso sempre più piede un ulteriore tipo di manutenzione, ovvero quella predittiva. La manutenzione predittiva cerca di anticipare la probabilità di un guasto, cercando di stimare quanto tempo manca prima che questo avvenga, utilizza modelli matematici che si basano sui dati che vengono raccolti dalle tecnologie 4.0 inserite nell'impianto. Queste tecnologie comprendono generalmente l'internet delle cose (Internet of Things, IoT), sensori e l'intelligenza artificiale (IA); si basano su tecnologie senza fili (wireless), la connessione fra tutto ciò che ci fornisce dati, utilizzo di tecniche di mega dati (Big Data Techniques) e apprendimento profondo (Deep Learning, DL). Per applicarla è fondamentale un monitoraggio continuo e in tempo reale dell'impianto attraverso dei sensori che rilevino le condizioni di funzionamento significative (temperatura, tensione, analisi chimica, ecc.) di ciò che viene controllato, uno spazio per archiviare i dati e soprattutto un approccio, che da questi, riesca a riconoscere quando ci si avvicina ad un guasto e, con l'esperienza raccolta, stimare indicativamente la vita utile residua. Generalmente gli approcci sono basati sulla conoscenza, su delle regole o su dei modelli. Tra gli approcci basati sui modelli troviamo le reti neurali che verranno analizzate nei capitoli successivi. Tendenzialmente si opta per gli approcci basati sui modelli perché molto efficaci e accurati, oltretutto sono facilmente riutilizzabili; al contrario dell'approccio su conoscenza (poco efficiente) o su regole (poco riutilizzabile e molto costoso). In generale quindi la manutenzione predittiva è costosa da implementare a causa del costo delle tecnologie, dei

sistemi di connessione e dei software necessari per salvare i dati e la stima della vita utile (e per questo ancora non molto sviluppata), ma è in grado di essere nettamente più efficiente della preventiva in termini di precisione e costi; infatti potendo stimare la vita utile residua si è in grado di anticipare i guasti molto più spesso di quanto si possa fare con un controllo periodico; inoltre in questo modo si interviene una volta sola (poco prima del termine della vita utile) non dovendo quindi interrompere periodicamente un processo, evitando quindi costi superflui (soprattutto manodopora) e perdite di tempo che non creano valore. Inoltre, con la manutenzione predittiva si è in grado di sfruttare maggiormente un oggetto, evitando di sostituirlo quando ancora ha una vita utile (fatto che invece accade con la manutenzione preventiva dopo un certo periodo), oppure se possibile aggiustarlo invece di sostituirlo e così risparmiare. Chiaramente la manutenzione predittiva è tanto più efficace quanto maggiore è il numero di dati rilevanti che vengono considerati, salvati e modellati; ma ovviamente al crescere della mole di dati presi in considerazione, cresce anche la difficoltà nell'effettuarla e di conseguenza aumenta il costo. Il vantaggio sicuramente più importante è però l'aumento della sicurezza, infatti dove non ci si può permettere che accada un guasto, poiché potrebbe causare problemi per la salute di lavoratori o di chi usufruisce di un servizio; la manutenzione predittiva è l'unico modo per scongiurarlo. Infine, un ulteriore vantaggio è la possibilità di controllare i dati di funzionamento, in tempo reale, a distanza, non dovendo per forza essere nel sito di funzionamento; questo può essere un aspetto importante per quanto riguarda la sicurezza (se il sito in funzionamento è pericoloso) e permette anche di evitare costi, che potrebbero essere elevati, per recarsi nel sito per effettuare la manutenzione.

3. Reti neurali

Al fine di stimare la vita utile residua si utilizzeranno le reti neurali artificiali, ovverosia un modello matematico costituito da gruppi di informazioni interconnesse in grado di risolvere alcuni problemi matematici, tra cui quello della stima della vita utile residua. In particolare, per risolvere il problema in questione, si utilizzeranno tre tipi di reti neurali artificiali: una rete neurale di base o percettrone multistrato; una rete neurale ricorsiva e una rete neurale convoluzionale. Le reti neurali artificiali cercano di imparare dall'esperienza pregressa, facendo generalizzazioni da situazioni simili per cercare di portare ad un risultato affine. Le prestazioni delle reti neurali artificiali dipendono principalmente dal numero di neuroni negli strati nascosti e dalle connessioni di rete, ma questi aspetti sono direttamente dipendenti dal problema in questione. Per la stima della vita utile residua, la rete neurale presenta una struttura con tre strati nascosti di neuroni, uno per l'input e uno per l'output (in questo caso la vita utile residua). In input si utilizzano i dati raccolti dai sensori, ma questi essendo grezzi non sono facili da assimilare; come soluzione si adottano funzionalità aggiuntive con caratteristiche realizzate manualmente, le quali si basano sull'utilizzo del dominio del tempo, della frequenza o di quello tempo-frequenza. La rete neurale convoluzionale è il modello più importante dell'apprendimento profondo, dotato di un'importante precisione grazie al fatto che utilizza pesi condivisi e alla capacità di rappresentare il campo locale; è in grado di estrarre le caratteristiche dei dati di input e di combinarli strato per strato in modo da generare

caratteristiche di alto livello. La struttura tipica della rete neurale convoluzionale consiste essenzialmente dello strato di input, dello strato di convoluzione, dello strato di pooling e dello strato completamente connesso. Lo strato di input può essere presentarsi in maniera bidimensionale come spettro tempo-frequenza o unidimensionale come dati di serie temporali. Nello strato di convoluzione, il nucleo convoluzionale lavora sui dati di input, attraverso un sistema di pesi, per generare un output di dettagli chiamato mappa di caratteristiche. Lo strato di pooling serve per campionare i dati, in modo da ridurre i parametri del modello mantenendolo efficace; ciò serve per migliorare la velocità di allenamento dello stesso. La rete neurale convoluzionale non mostra buone capacità di estrazione di funzionalità utili dai dati di monitoraggio, perciò è stata costruita, nel campo unidimensionale, la 1D-CNN end-to-end che è in grado di riflettere i segnali di vibrazione grezzi ai tipi di guasto; con questo modello è stata dimostrata una precisione vicina al 99% ottimizzandone i parametri. Questa particolare rete comprende due strati convoluzionali, due di pooling e uno strato completamente connesso. La possibilità di operare direttamente sui dati grezzi evita l'uso di qualsiasi algoritmo di estrazione delle caratteristiche. La rete neurale convoluzionale è spesso combinata con un algoritmo di elaborazione del segnale per migliorare le prestazioni di diagnosi dei guasti. La 1D-CNN richiede pochi dati per essere allenata in modo efficace e anche un hardware poco costoso per l'implementazione, ma mantiene il problema della difficoltà ad estrarre delle funzionalità dal sensore nel formato unidimensionale; per risolvere questo problema la rete neurale convoluzionale è combinata con l'algoritmo S-transform, che converte automaticamente i dati dei sensori in una matrice tempo-frequenza (quindi bidimensionale e senza un intervento manuale). In alternativa, però, la rete neurale convoluzionale è in grado di apprendere le specifiche di input in modo bidimensionale, in realtà anche in modo più efficiente rispetto al metodo unidimensionale. Generalmente per indicare il degrado

dei macchinari è necessaria una conoscenza preliminare importante per poter progettare algoritmi di estrazione e fusione dei dati; ma utilizzando una rete neurale convoluzionale che abbia diversi strati di convoluzione e pooling impilati tra loro per imparare le caratteristiche e mapparle, e rimuovendo le regioni anomale, si è visto che si ottiene una precisione, non in grado di essere raggiunta con qualunque altro modello. In particolare, per quanto riguarda la stima della vita utile residua, si è visto che combinando la rete neurale convoluzionale con un metodo di livellamento per la predizione, in cui il segnale delle vibrazioni è convertito in spettro di frequenza discreto tramite trasformata di Fourier e da questo la CNN è in grado di completare il calcolo. In questo modo si è visto che la predizione è particolarmente accurata. Una ulteriore alternativa è l'utilizzo della rete neurale convoluzionale bidimensionale (quella che verrà utilizzata nel caso in esame) basata su serie temporali variabili normalizzate dai segnali del sensore, dove una delle dimensioni è il numero dei sensori. In questo caso, la struttura della rete neurale convoluzionale è impiegata per estrarre le caratteristiche dei dati locali attraverso la rete di apprendimento profondo per una migliore prognosi. Le informazioni si ottengono nel dominio tempo-frequenza applicando la trasformata di Fourier a breve termine ed esplorata per prognostica, e l'estrazione delle specifiche multi-scala avviene attraverso la rete neurale convoluzionale. In precedenza, la diagnosi guasti e la stima della vita utile residua sono state sempre studiate separatamente, nonostante che una condivisione delle informazioni ne avrebbe sicuramente migliorato l'efficienza. A tal fine, sono in fase di sviluppo delle reti neurali convoluzionali congiunte (JL-CNN). Le reti neurali ricorrenti trattano i dati sequenziali. Come modello sequenziale, la rete neurale ricorrente è in grado di costruire collegamenti tra le unità nascoste e mantenere memoria dei vari ingressi precedenti negli stati interni della rete. L'output finale è la rappresentazione appresa di tutti i dati sequenziali di input. La rete neurale ricorrente utilizza la memoria a lungo termine (long short-term memory,

LSTM) che è in grado di memorizzare e modellare a lungo termine i dati e si utilizza principalmente con dati dipendenti dal tempo. La rete neurale ricorrente viene sempre più utilizzata di recente per la diagnosi dei guasti perché è in grado di apprendere molto più facilmente sequenze importanti di dati. Data l'ampia capacità di memorizzare i dati dipendenti dal tempo nella memoria a lungo termine, le reti neurali ricorrenti sono state utilizzate nella stima della vita utile residua. La stima viene fatta utilizzando l'analisi dei componenti principali del kernel per estrarre le funzionalità non lineari. Questa analisi insieme alla enorme mole di dati acquisita (grazie alla LSTM) danno la possibilità a una rete neurale ricorrente di effettuare un'analisi del degrado anche non lineare e di conseguenza una buona predizione della vita utile residua. Fondamentale per una buona stima è comunque la costruzione di un indicatore di degrado del macchinario adeguato, talvolta questi indicatori sono basati proprio sulla rete neurale ricorrente che sembra garantire al momento prestazioni più che accettabili.

4. Stima della RUL mediante reti neurali

In questo capitolo, si vedrà come sia possibile efffettuare la stima della vita utile residua di motori a turboventola mediante l'utilizzo di MATLAB e, in particolare, si descriverà il codice da utilizzare indicandone la funzione che svolge. Il primo passo è quello di scaricare il dataset e vedere sinteticamente da cosa è composto. Per scaricarlo e decomprimerlo in una cartella, denominata"data", è sufficiente scrivere:

```
filename = "CMAPSSData.zip";
if ~exist(filename,'file')
    url = "https://ti.arc.nasa.gov/c/6/";
    websave(filename,url);
end

dataFolder = "data";
if ~exist(dataFolder,'dir')
    mkdir(dataFolder);
end
unzip(filename,dataFolder)
```

Si può ora vedere che questa cartella è composta da 4 diversi set (FD001, FD002, FD003, FD004) di dati di serie temporali, che sono stati simulati in diverse condizioni operative e modalità di guasto. In questo studio si utilizzerà il set FD001 che è ulteriormente suddiviso in un insieme di addestramento e uno di test. L'insieme di addestramento contiene dati di

serie temporali simulati per cento motori. Ogni motore ha ventuno sensori i cui valori vengono registrati in una data istanza in un processo continuo; quindi, la sequenza dei dati registrati varia in lunghezza e corrisponde a un'istanza RTF (full run-to-failure). L'insieme di test contiene cento sequenze parziali che corrispondono ai valori della vita utile alla fine di ogni sequenza. La cartella "data" ora conterrà file di testo con 26 colonne di numeri, separati da spazi. Ogni riga contiene i dati acquisiti durante un singolo ciclo operativo e ogni colonna rappresenta una variabile diversa. In particolare, la prima colonna contiene il numero dell'unità; la seconda contiene data e ora; terza, quarta e quinta contengono le impostazioni operative; infine, le restanti contengono le misurazioni dei ventuno sensori. Il passo successivo consiste nella preelaborazione dei dati di addestramento, in cui si utilizzerà la funzione "localLoadData" per caricare i dati, in particolare la funzione estrae i dati da "filenamePredictors" e li riporta in una tabella che contiene le predizioni di addestramento e la sequenza di risposta corrispondente (RUL); ogni riga rappresenta un motore differente. Quindi si scriverà:

```
filenameTrainPredictors = fullfile(dataFolder,"train_FD001.txt");
rawTrain = localLoadData(filenameTrainPredictors);
```

Il passo successivo sarà quello di rimuovere i sensori che non variano, in modo da mantenere solo quelli significati per la stima della vita utile residua. Per farlo si utilizzerà la funzione "prognosability" per misurare la variabilità dei sensori, e se il valore sarà pari a zero quel sensore verrà scartato. Si scriverà dunque:

```
prog = prognosability(rawTrain.X,"timeStamp");
idxToRemove = prog.Variables==0 | isnan(prog.Variables);
featToRetain = prog.Properties.VariableNames(~idxToRemove);
```

```
for i = 1:height(rawTrain)
    rawTrain.X{i} = rawTrain.X{i}{:,featToRetain};
end
```

Successivamente si normalizzeranno i predittori di addestramento per avere media zero e varianza unitaria:

```
[~,Xmu,Xsigma] = zscore(vertcat(rawTrain.X{:}));
preTrain = table();
for i = 1:numel(rawTrain.X)
    preTrain.X{i} = (rawTrain.X{i} - Xmu) ./ Xsigma;
end
```

Affinché la rete si concentri sulla parte dei dati in cui i motori hanno maggiori probabilità di guastarsi, si può decidere di ritagliare le risposte a una determinata soglia (nel caso studio pari a 150). Questo significa che se i valori della RUL sono maggiori, li considera tutti uguali; a tal fine è sufficiente scrivere:

```
clipResponses = true;

if clipResponses
    rulThreshold = 150;
    for i = 1:numel(rawTrain.Y)
        preTrain.Y{i} = min(rawTrain.Y{i},rulThreshold);
    end
end
```

Il passo successivo è quello di preparare i dati per il riempimento. Per farlo bisogna ordinare i dati di addestramento in base alla lunghezza della

sequenza, successivamente bisogna scegliere una dimensione “mini-batch” che divida i dati di allenamento in modo uniforme e riduca la quantità di riempimento dei “mini-batch” stessi. Infine, ordiniamo i dati di allenamento in base alla lunghezza della sequenza. Per farlo si scrive:

```
for i = 1:size(preTrain,1)
    preTrain.X{i} = preTrain.X{i}';
    preTrain.Y{i} = preTrain.Y{i}';
    sequence = preTrain.X{i};
    sequenceLengths(i) = size(sequence,2);
end

[sequenceLengths,idx] = sort(sequenceLengths,'descend');
XTrain = preTrain.X(idx);
YTrain = preTrain.Y(idx);
```

Il passo successivo è quello di scrivere l’architettura della rete da utilizzare e, a tal fine, si vedranno tre diverse tipologie di reti. Successivamente è necessario addestrare la rete; per farlo bisogna impostare quante epochs addestrare, la dimensione dei “mini-batch”, il tasso di apprendimento; infine, per l’addestramento si utilizza il comando “trainNetwork”. Si scrive dunque:

```
maxEpochs = 20;
miniBatchSize = 512;

options = trainingOptions('adam', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.3,...
    'MaxEpochs',maxEpochs, ...
```

```
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',0.003, ...
    'Shuffle','every-epoch',...
    'Plots','training-progress',...\
    'Verbose',0);

net = trainNetwork(dataTrain,layers,options);
```

Infine, non rimane che testare la rete ed eventualmente è possibile evidenziare le performance della stessa. Per farlo è necessario ricordare che anche la cartella test conteneva centro brevi sequenze che stavano ad indicare la RUL test. Il set di dati test va preparato allo stesso modo del set di dati di addestramento. Si creano, inoltre, una tabella con la previsione calcolata e quella reale per verificarne la differenza. Inoltre, si possono introdurre anche dei grafici che le confrontino e magari si può effettuare anche il calcolo dell'errore quadratico medio (RMSE). Si scrive quindi:

```
filenameTestPredictors = fullfile(dataFolder,'test_FD001.txt');
filenameTestResponses = fullfile(dataFolder,'RUL_FD001.txt');
dataTest = localLoadData(filenameTestPredictors,filenameTestResponses);

for i = 1:numel(dataTest.X)
    dataTest.X{i} = dataTest.X{i}{:,featToRetain};
    dataTest.X{i} = (dataTest.X{i} - Xmu) ./ Xsigma;
    if clipResponses
        dataTest.Y{i} = min(dataTest.Y{i},rulThreshold);
    end
end

unitLengths = zeros(numel(dataTest.Y),1);
```

```
for i = 1:numel(dataTest.Y)
    unitLengths(i) = numel(dataTest.Y{i,:});
end
dataTest(unitLengths<WindowLength+1,:) = [];

predictions = table('Size',[height(dataTest) 2],'VariableTypes',["cell","cell"],'VariableNames',["Y","YPred"]);

for i=1:height(dataTest)
    unit = localGenerateSequences(dataTest(i,:),WindowLength,Stride);
    predictions.Y{i} = unit.Y;
    predictions.YPred{i} = predict(net,unit,'MiniBatchSize',miniBatchSize);
end

for i = 1:size(predictions,1)
    predictions.RMSE(i) = sqrt(mean((predictions.Y{i} - predictions.YPred{i}).^2));
end

figure;
histogram(predictions.RMSE,'NumBins',10);
title("RMSE ( Mean: " + round(mean(predictions.RMSE),2) + " , StDev: " + round(std(predictions.RMSE),2) + " )");
ylabel('Frequency');
xlabel('RMSE');

figure;
localLambdaPlot(predictions,"random");
```

Infine, per vedere la stima della vita utile residua di un motore specifico (cosa che verrà effettuata per vedere il numero 33 e il 49) è sufficiente scrivere nella linea di comando “figure” seguito da:

```
y = predictions.Y{idx};
yPred = predictions.YPred{idx};
x = 0:numel(y)-1;
plot(x,y,x,yPred)
legend("True RUL","Predicted RUL")
xlabel("Time stamp (Test data sequence)")
ylabel("RUL (Cycles)")

title("RUL for Test engine #"+idx+ " ("+lambdaCase+" case)")
```

in cui al posto di “idx” e di “lambdaCase” si deve inserire il numero del motore di interesse. Ora si analizzeranno in dettaglio le architetture delle tre reti che sono state testate, iniziando dalla rete neurale artificiale di base, successivamente la rete neurale ricorrente e infine la rete neurale convoluzionale.

4.1 Stima della RUL mediante reti neurali di base (MLP)

La struttura della MLP è sicuramente la più semplice come accennato nel capitolo precedente, ed è infatti composta da due livelli “fullyConnectedLayer” (che restituiscono l’output) e uno “reluLayer” in mezzo per l’attivazione. Si scrive dunque:

```
numHiddenUnits = 100;
numResponses = 1;

layers = [
   imageInputLayer(InputSize)
   fullyConnectedLayer(numHiddenUnits)
   reluLayer()
   fullyConnectedLayer(numResponses)
   regressionLayer()];
```

4.2 Stima della RUL mediante reti neurali ricorrenti (RNN)

L'architettura della rete neurale artificiale ricorrente è simile alla MLP, ma come già anticipato nel capitolo precedente vede l'introduzione della Long Short Time Memory (LSTM). Perciò il codice sarà:

```
numHiddenUnits = 100;
numResponses=1;

layers = [
   sequenceInputLayer(40)
   lstmLayer(numHiddenUnits, "OutputMode","last")
   fullyConnectedLayer(numResponses)
   regressionLayer];
```

4.3 Stima della RUL mediante reti neurali convoluzionali (CNN)

La stima della vita utile residua, utilizzando una rete neurale artificiale convoluzionale è stata ottenuta utilizzando la convolution2dlayer. In questo caso, i dati di input vengono elaborati e ordinati in un formato di sequenza, con la prima dimensione che rappresenta il numero di caratteristiche selezionate e la seconda dimensione che rappresenta la lunghezza della sequenza temporale. I livelli convoluzionali sono raggruppati attraverso il livello fullyConnected seguito da quello di attivazione (relu) e quindi impilati assieme per l'estrazione delle caratteristiche. I livelli completamente connessi sono utilizzati per fornire l'output finale ovvero la vita utile residua. Dunque, il codice specifico risulta:

```
filterSize = [5, 1];
numHiddenUnits = 100;
numResponses = 1;

layers = [
    imageInputLayer(InputSize)
    convolution2dLayer(filterSize,10)
    reluLayer()
    convolution2dLayer(filterSize,20)
    reluLayer()
    convolution2dLayer(filterSize,10)
    reluLayer()
    convolution2dLayer([3 1],5)
```

```
reluLayer()
fullyConnectedLayer(numHiddenUnits)
reluLayer()
dropoutLayer(0.5)
fullyConnectedLayer(numResponses)
regressionLayer()];
```

5. Risultati

Si procederà ora all'analisi dei risultati ottenuti utilizzando le tre reti. Per confrontarle, si possono scegliere due motori di riferimento, in questo caso i numeri 33 e 49, e verificare quanto la vita utile residua predetta è diversa da quella realmente testata. Nel caso della rete neurale di base, per il motore 33 dopo 158 iterazioni si ottiene una vita utile predetta di 15.4618 cicli, contro quella testata che è pari a 11 (fig.1); mentre per il numero 49 abbiamo una RUL stimata di 26.3797 cicli contro i 29 di quella del test (fig.2); si ottiene quindi, una buona stima della vita utile residua dei due motori in questione. Per la rete neurale artificiale ricorsiva, dopo 158 iterazioni, nel motore 33 si ottiene una vita utile residua predetta di 13.5593 cicli rispetto agli 11 di test (fig.33), nel motore 49 abbiamo una RUL predetta di 41.512 cicli (dopo 149 iterazioni) contro quella di test pari a 29 (fig.4); anche in questo caso si ottiene una buona stima della vita residua dei motori. Nel caso della rete neurale convoluzionale, per il motore 33 ad esempio, si ottiene una vita utile residua predetta di 10.228 cicli contro gli 11 di quella "reale" dopo 158 iterazioni (fig.5); per il numero 49 si ottiene, invece, una RUL stimata in 23.5148 contro quella di test che è pari a 29 dopo 149 iterazioni (fig.6); in definitiva, si riesce ad ottenere una previsione molto vicina a quella realmente testata.

Fig.1

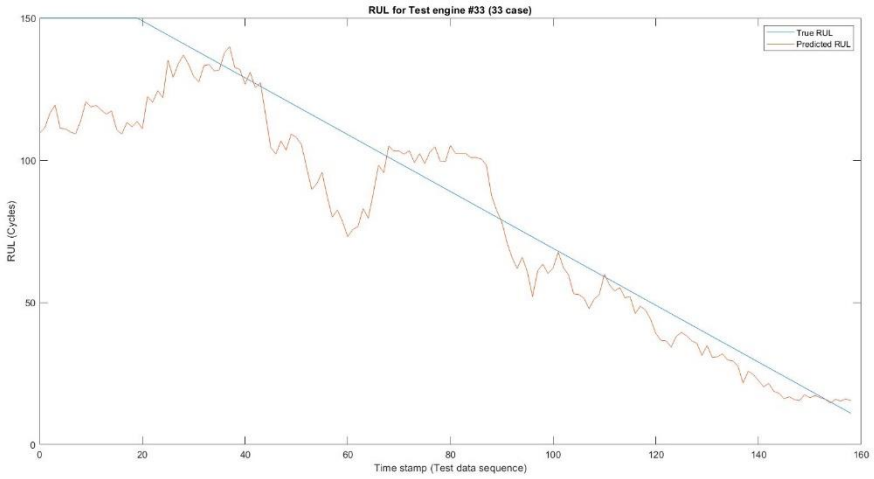

RUL for Test engine #33 (33 case)
True RUL
Predicted RUL
RUL (Cycles)
Time stamp (Test data sequence)
0
50
100
150
0
20
40
60
80
100
120
140
160

Fig.2

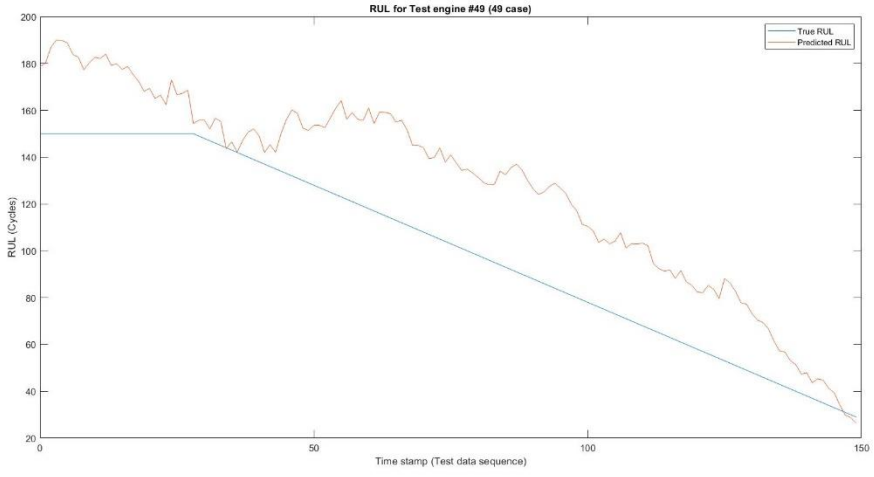

RUL for Test engine #49 (49 case)
True RUL
Predicted RUL
RUL (Cycles)
Time stamp (Test data sequence)
20
40
60
80
100
120
140
160
180
200
0
50
100
150

Fig.3

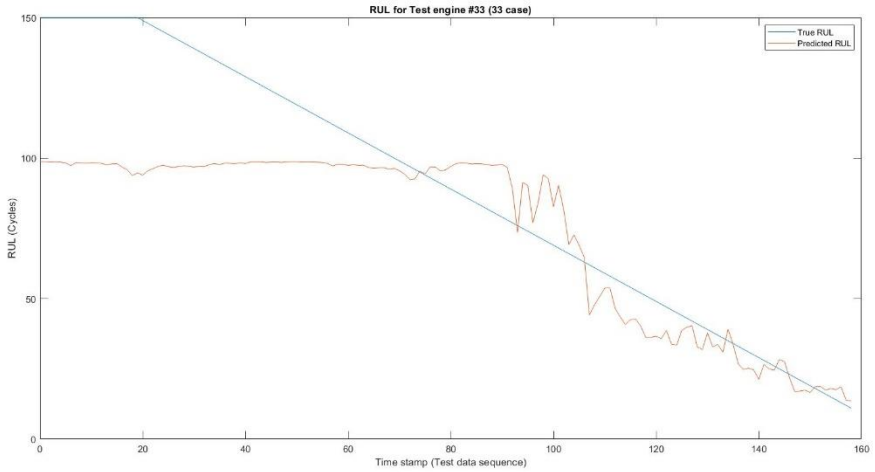


Fig.4

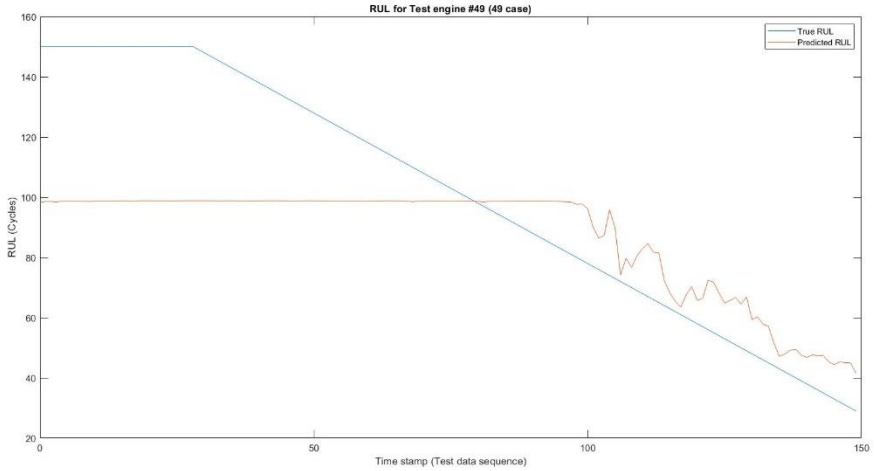


Fig.5

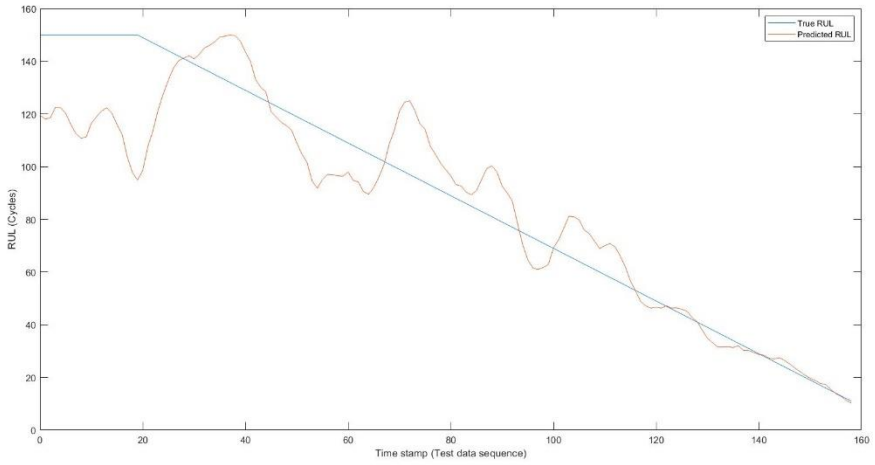
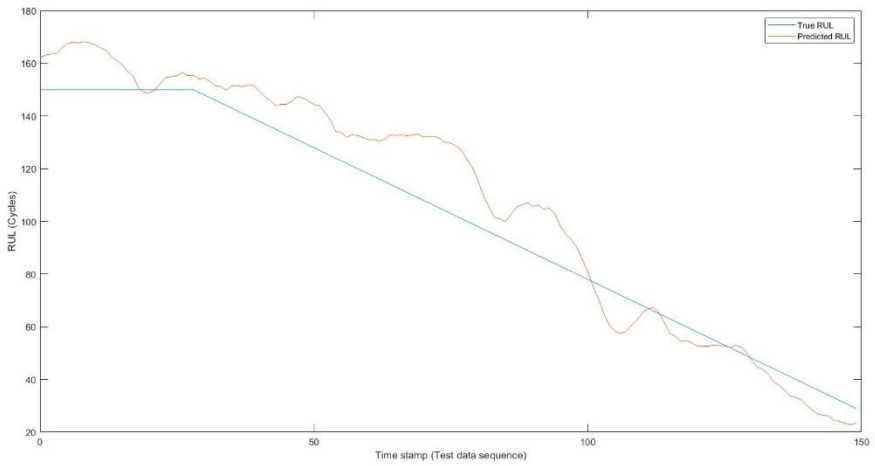

True RUL
Predicted RUL
RUL (Cycles)
Time stamp (Test data sequence)
0
20
40
60
80
100
120
140
160

Fig.6


True RUL
Predicted RUL
RUL (Cycles)
Time stamp (Test data sequence)
20
40
60
80
100
120
140
160
180
0
50
100
150

6. Conclusioni

In un mondo sempre più tecnologico e interconnesso è sempre più necessario ed efficiente l'utilizzo della manutenzione predittiva a discapito della preventiva, e non solo, è inoltre sempre più semplice da effettuare grazie ai nuovi sviluppi nel campo. Le reti neurali artificiali sono sicuramente fondamentali per l'introduzione della manutenzione predittiva per la stima dei guasti e della vita utile residua. Ne sono state riportate e testate tre con differenti risultati che di seguito sono analizzati. La rete neurale artificiale di base (MLP) ha riportato un errore del 40,56 % per il motore 33 e del 9,04 % per il motore 49, abbiamo quindi ottenuto una buona stima attraverso questa soluzione. La rete neurale artificiale ricorrente (RNN) ha riportato un errore del 23,27% per il motore 33 e del 43,14% del motore 49, abbiamo quindi ottenuto una buona stima, ma peggiore di quella ottenuta con la multilayer perceptron. La rete neurale artificiale convoluzionale ha riportato un errore del 7,02% per il motore 33 e del 18,91% per il motore 49; ovvero una stima piuttosto precisa e in media anche migliore della rete neurale di base. Perciò si può affermare che in questo caso la scelta migliore è la rete neurale artificiale convoluzionale (che ricordiamo nel caso in specie è stata utilizzata nella versione 2D) con cui si riesce a ottenere i risultati migliori in percentuale con l'utilizzo di questo algoritmo di predizione.

Sicuramente per il futuro tutto ciò potrebbe essere migliorato per trovare una soluzione ancora più precisa, ma nel frattempo è possibile che questo sia d'impulso per lo sviluppo di ulteriori studi e per un'introduzione maggiore nel campo industriale di questo tipo di manutenzione.

7. Bibliografia

- Ran Y., Zhou X., Lin P., Wen Y., Deng R., A Survey of Predictive Maintenance: Systems, Purposes and Approaches, IEEE Comunications Surveys & Tutorials vol. XX, 2019
- Krupitzer C., Wagenhals T., Zufle M., Lesch V., Schafer D., Mozaffarin A., Edinger J., Becker C., Kounev S., A Survey on Predictive Maintenance for Industry 4.0, 2020
- Lu B., Durocher D., Stemper P., Predictive Maintenance Techniques, IEEE Industry Applications Magazine, 2009
- Hashemian H. M., State-of-the-Art Predictive Maintenance Techniques, IEEE Transactions on instrumentation and measurement vol. 60, 2011
- Zhang W., Yang D., Wang H., Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey, IEEE Systems Journal vol. 13, 2019
- Patwardhan A., Verma A. K., Kumar U., A Survey on Predictive Maintenance Through Big Data, Springer International Publishing, 2016
- Matzka S., Explainable Artificial Intelligence for Predictive Maintenance Applications, Third International Conference on Artificial Intelligence for Industries, 2020
- Mathworks inc., Predictive Maintenance Toolbox User's Guide, 2020

- Manutenzione Predittiva con MATLAB: un esempio di prognostica, https://www.mathworks.com/videos/predictive-maintenance-with-matlab-a-prognostic-case-study-119606.html
- Model-Based Design per la Manutenzione Predittiva, https://www.mathworks.com/videos/model-based-design-for-predictive-maintenance-1582821305754.html
- Examples of Data Analytics for Predictive Maintenance, https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/63012/versions/4/previews/PredictiveMaintenanceDemo_r1/html/Demo0_PreProcessing.html
- Remaining Useful Life Estimation using Convolutional Neural Network, https://it.mathworks.com/help/predmaint/ug/remaining-useful-life-estimation-using-convolutional-neural-network.html

8. Appendice (codice MATLAB)

Codice MATLAB dell'algoritmo con rete neurale artificiale di base (MLP):

```
filename = "CMAPSSData.zip";
if ~exist(filename,'file')
    url = "https://ti.arc.nasa.gov/c/6/";
    websave(filename,url);
end

dataFolder = "data";
if ~exist(dataFolder,'dir')
    mkdir(dataFolder);
end
unzip(filename,dataFolder)
filenameTrainPredictors = fullfile(dataFolder,"train_FD001.txt");
rawTrain = localLoadData(filenameTrainPredictors);
head(rawTrain.X{1},8)
rawTrain.Y{1}(1:8)
stackedplot(rawTrain.X{1},[3,5,6,7,8,15,16,24],'XVariable','timeStamp')
prog = prognosability(rawTrain.X,"timeStamp");
idxToRemove = prog.Variables==0 | isnan(prog.Variables);
featToRetain = prog.Properties.VariableNames(~idxToRemove);
for i = 1:height(rawTrain)
    rawTrain.X{i} = rawTrain.X{i}{:,featToRetain};
```

```
end
[~,Xmu,Xsigma] = zscore(vertcat(rawTrain.X{:}));
preTrain = table();
for i = 1:numel(rawTrain.X)
    preTrain.X{i} = (rawTrain.X{i} - Xmu) ./ Xsigma;
end
clipResponses = true;

if clipResponses
    rulThreshold = 150;
    for i = 1:numel(rawTrain.Y)
        preTrain.Y{i} = min(rawTrain.Y{i},rulThreshold);
    end
end

WindowLength = 40;
Stride = 1;
dataTrain = localGenerateSequences(preTrain,WindowLength,Stride);

numFeatures = size(dataTrain.X{1},2);
InputSize = [WindowLength numFeatures 1];
for i = 1:size(dataTrain,1)
    dataTrain.X{i} = reshape(dataTrain.X{i},InputSize);
end

numHiddenUnits = 100;
numResponses = 1;

layers = [
    imageInputLayer(InputSize)
```

```
    fullyConnectedLayer(numHiddenUnits)
    reluLayer()
    fullyConnectedLayer(numResponses)
    regressionLayer()];

maxEpochs = 20;
miniBatchSize = 512;

options = trainingOptions('adam', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.3,...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',0.003, ...
    'Shuffle','every-epoch',...
    'Plots','training-progress',...
    'Verbose',0);

net = trainNetwork(dataTrain,layers,options);

figure;
lgraph = layerGraph(net.Layers);
plot(lgraph)

filenameTestPredictors = fullfile(dataFolder,'test_FD001.txt');
filenameTestResponses = fullfile(dataFolder,'RUL_FD001.txt');
dataTest = localLoadData(filenameTestPredictors,filenameTestResponses);

for i = 1:numel(dataTest.X)
    dataTest.X{i} = dataTest.X{i}{:,featToRetain};
```

```
    dataTest.X{i} = (dataTest.X{i} - Xmu) ./ Xsigma;
    if clipResponses
        dataTest.Y{i} = min(dataTest.Y{i},rulThreshold);
    end
end

unitLengths = zeros(numel(dataTest.Y),1);
for i = 1:numel(dataTest.Y)
    unitLengths(i) = numel(dataTest.Y{i,:});
end
dataTest(unitLengths<WindowLength+1,:) = [];

predictions = table('Size',[height(dataTest)
2],'VariableTypes',["cell","cell"],'VariableNames',["Y","YPred"]);

for i=1:height(dataTest)
    unit = localGenerateSequences(dataTest(i,:),WindowLength,Stride);
    predictions.Y{i} = unit.Y;
    predictions.YPred{i} = predict(net,unit,'MiniBatchSize',miniBatchSize);
end

for i = 1:size(predictions,1)
    predictions.RMSE(i) = sqrt(mean((predictions.Y{i} -
predictions.YPred{i}).^2));
end

figure;
histogram(predictions.RMSE,'NumBins',10);
title("RMSE ( Mean: " + round(mean(predictions.RMSE),2) + " , StDev: "
+ round(std(predictions.RMSE),2) + " )");
```

```
ylabel('Frequency');
xlabel('RMSE');

figure;
localLambdaPlot(predictions,"random");

function data = localLoadData(filenamePredictors,varargin)

if isempty(varargin)
    filenameResponses = [];
else
    filenameResponses = varargin{:};
end


rawData = readtable(filenamePredictors);


VarNames = {...
    'id', 'timeStamp', 'op_setting_1', 'op_setting_2', 'op_setting_3', ...
    'sensor_1', 'sensor_2', 'sensor_3', 'sensor_4', 'sensor_5', ...
    'sensor_6', 'sensor_7', 'sensor_8', 'sensor_9', 'sensor_10', ...
    'sensor_11', 'sensor_12', 'sensor_13', 'sensor_14', 'sensor_15', ...
    'sensor_16', 'sensor_17', 'sensor_18', 'sensor_19', 'sensor_20', ...
    'sensor_21'};
rawData.Properties.VariableNames = VarNames;

if ~isempty(filenameResponses)
    RULTest = dlmread(filenameResponses);
end
```

```
IDs = rawData{:,1};
nID = unique(IDs);
numObservations = numel(nID);


data = table('Size',[numObservations 2],...
   'VariableTypes',{'cell','cell'},...
   'VariableNames',{'X','Y'});

for i=1:numObservations
   idx = IDs == nID(i);
   data.X{i} = rawData(idx,:);
   if isempty(filenameResponses)

      data.Y{i} = flipud(rawData.timeStamp(idx))-1;
   else

      sequenceLength = sum(idx);
      endRUL = RULTest(i);
      data.Y{i} = [endRUL+sequenceLength-1:-1:endRUL]';
   end
end
end

function seqTable =
localGenerateSequences(dataTable,WindowLength,Stride)

getX = @(X) generateSequences(X,WindowLength,Stride);
```

```
getY = @(Y) Y(WindowLength:Stride:numel(Y));

seqTable = table;
temp = cellfun(getX,dataTable.X,'UniformOutput',false);
seqTable.X = vertcat(temp{:});
temp = cellfun(getY,dataTable.Y,'UniformOutput',false);
seqTable.Y = vertcat(temp{:});
end


function seqCell = generateSequences(tsData,WindowLength,Stride)



getSeq = @(idx) tsData(1+idx:WindowLength+idx,:);

idxShift = num2cell(0:Stride:size(tsData,1)-WindowLength)';

seqCell = cellfun(getSeq,idxShift,'UniformOutput',false);
end

function localLambdaPlot(predictions,lambdaCase)

if isnumeric(lambdaCase)
   idx = lambdaCase;
else
   switch lambdaCase
      case {"Random","random","r"}
         idx = randperm(height(predictions),1);
      case {"Best","best","b"}
```

```
            idx = find(predictions.RMSE == min(predictions.RMSE));
        case {"Worst","worst","w"}
            idx = find(predictions.RMSE == max(predictions.RMSE));
        case {"Average","average","a"}
            err = abs(predictions.RMSE-mean(predictions.RMSE));
            idx = find(err==min(err),1);
    end
end
y = predictions.Y{idx};
yPred = predictions.YPred{idx};
x = 0:numel(y)-1;
plot(x,y,x,yPred)
legend("True RUL","Predicted RUL")
xlabel("Time stamp (Test data sequence)")
ylabel("RUL (Cycles)")

title("RUL for Test engine #"+idx+ " ("+lambdaCase+" case)")
end
```

Codice MATLAB dell'algoritmo con rete neurale artificiale ricorrente (RNN):

```
filename = "CMAPSSData.zip";
if ~exist(filename,'file')
    url = "https://ti.arc.nasa.gov/c/6/";
    websave(filename,url);
end
```

```
dataFolder = "data";
if ~exist(dataFolder,'dir')
   mkdir(dataFolder);
end
unzip(filename,dataFolder)
filenameTrainPredictors = fullfile(dataFolder,"train_FD001.txt");
rawTrain = localLoadData(filenameTrainPredictors);
head(rawTrain.X{1},8)
rawTrain.Y{1}(1:8)
stackedplot(rawTrain.X{1},[3,5,6,7,8,15,16,24],'XVariable','timeStamp')
prog = prognosability(rawTrain.X,"timeStamp");
idxToRemove = prog.Variables==0 | isnan(prog.Variables);
featToRetain = prog.Properties.VariableNames(~idxToRemove);
for i = 1:height(rawTrain)
   rawTrain.X{i} = rawTrain.X{i}{:,featToRetain};
end
[~,Xmu,Xsigma] = zscore(vertcat(rawTrain.X{:}));
preTrain = table();
for i = 1:numel(rawTrain.X)
   preTrain.X{i} = (rawTrain.X{i} - Xmu) ./ Xsigma;
end
clipResponses = true;

if clipResponses
   rulThreshold = 150;
   for i = 1:numel(rawTrain.Y)
      preTrain.Y{i} = min(rawTrain.Y{i},rulThreshold);
   end
end
```

```
WindowLength = 40;
Stride = 1;
dataTrain = localGenerateSequences(preTrain,WindowLength,Stride);

numFeatures = size(dataTrain.X{1},2);
InputSize = [WindowLength numFeatures 1];
for i = 1:size(dataTrain,1)
   dataTrain.X{i} = reshape(dataTrain.X{i},InputSize);
end

numHiddenUnits = 100;
numResponses = 1;

layers = [
   sequenceInputLayer(40)
   lstmLayer(numHiddenUnits, 'OutputMode','last')
   fullyConnectedLayer(numResponses)
   regressionLayer];

maxEpochs = 20;
miniBatchSize = 512;

options = trainingOptions('adam', ...
   'LearnRateSchedule','piecewise',...
   'LearnRateDropFactor',0.3,...
   'MaxEpochs',maxEpochs, ...
   'MiniBatchSize',miniBatchSize, ...
   'InitialLearnRate',0.003, ...
   'Shuffle','every-epoch',...
```

```
    'Plots','training-progress',...
    'Verbose',0);

net = trainNetwork(dataTrain.X,dataTrain.Y,layers,options);

figure;
lgraph = layerGraph(net.Layers);
plot(lgraph)

filenameTestPredictors = fullfile(dataFolder,'test_FD001.txt');
filenameTestResponses = fullfile(dataFolder,'RUL_FD001.txt');
dataTest = localLoadData(filenameTestPredictors,filenameTestResponses);

for i = 1:numel(dataTest.X)
    dataTest.X{i} = dataTest.X{i}{:,featToRetain};
    dataTest.X{i} = (dataTest.X{i} - Xmu) ./ Xsigma;
    if clipResponses
        dataTest.Y{i} = min(dataTest.Y{i},rulThreshold);
    end
end

unitLengths = zeros(numel(dataTest.Y),1);
for i = 1:numel(dataTest.Y)
    unitLengths(i) = numel(dataTest.Y{i,:});
end
dataTest(unitLengths<WindowLength+1,:) = [];

predictions = table('Size',[height(dataTest)
2],'VariableTypes',["cell","cell"],'VariableNames',["Y","YPred"]);
```

```
for i=1:height(dataTest)
    unit = localGenerateSequences(dataTest(i,:),WindowLength,Stride);
    predictions.Y{i} = unit.Y;
    predictions.YPred{i} =
predict(net,unit.X,'MiniBatchSize',miniBatchSize);
end

for i = 1:size(predictions,1)
    predictions.RMSE(i) = sqrt(mean((predictions.Y{i} -
predictions.YPred{i}).^2));
end

figure;
histogram(predictions.RMSE,'NumBins',10);
title("RMSE ( Mean: " + round(mean(predictions.RMSE),2) + " , StDev: "
+ round(std(predictions.RMSE),2) + " )");
ylabel('Frequency');
xlabel('RMSE');

figure;
localLambdaPlot(predictions,"random");

function data = localLoadData(filenamePredictors,varargin)

if isempty(varargin)
    filenameResponses = [];
else
    filenameResponses = varargin{:};
end
```

```
rawData = readtable(filenamePredictors);


VarNames = {...
    'id', 'timeStamp', 'op_setting_1', 'op_setting_2', 'op_setting_3', ...
    'sensor_1', 'sensor_2', 'sensor_3', 'sensor_4', 'sensor_5', ...
    'sensor_6', 'sensor_7', 'sensor_8', 'sensor_9', 'sensor_10', ...
    'sensor_11', 'sensor_12', 'sensor_13', 'sensor_14', 'sensor_15', ...
    'sensor_16', 'sensor_17', 'sensor_18', 'sensor_19', 'sensor_20', ...
    'sensor_21'};
rawData.Properties.VariableNames = VarNames;

if ~isempty(filenameResponses)
    RULTest = dlmread(filenameResponses);
end


IDs = rawData{:,1};
nID = unique(IDs);
numObservations = numel(nID);


data = table('Size',[numObservations 2],...
    'VariableTypes',{'cell','cell'},...
    'VariableNames',{'X','Y'});

for i=1:numObservations
    idx = IDs == nID(i);
    data.X{i} = rawData(idx,:);
```

```
    if isempty(filenameResponses)

        data.Y{i} = flipud(rawData.timeStamp(idx))-1;
    else

        sequenceLength = sum(idx);
        endRUL = RULTest(i);
        data.Y{i} = [endRUL+sequenceLength-1:-1:endRUL]';
    end
end
end

function seqTable =
localGenerateSequences(dataTable,WindowLength,Stride)

getX = @(X) generateSequences(X,WindowLength,Stride);
getY = @(Y) Y(WindowLength:Stride:numel(Y));

seqTable = table;
temp = cellfun(getX,dataTable.X,'UniformOutput',false);
seqTable.X = vertcat(temp{:});
temp = cellfun(getY,dataTable.Y,'UniformOutput',false);
seqTable.Y = vertcat(temp{:});
end



function seqCell = generateSequences(tsData,WindowLength,Stride)
```

```
getSeq = @(idx) tsData(1+idx:WindowLength+idx,:);

idxShift = num2cell(0:Stride:size(tsData,1)-WindowLength)';

seqCell = cellfun(getSeq,idxShift,'UniformOutput',false);
end

function localLambdaPlot(predictions,lambdaCase)

if isnumeric(lambdaCase)
   idx = lambdaCase;
else
   switch lambdaCase
      case {"Random","random","r"}
         idx = randperm(height(predictions),1);
      case {"Best","best","b"}
         idx = find(predictions.RMSE == min(predictions.RMSE));
      case {"Worst","worst","w"}
         idx = find(predictions.RMSE == max(predictions.RMSE));
      case {"Average","average","a"}
         err = abs(predictions.RMSE-mean(predictions.RMSE));
         idx = find(err==min(err),1);
   end
end
y = predictions.Y{idx};
yPred = predictions.YPred{idx};
x = 0:numel(y)-1;
plot(x,y,x,yPred)
legend("True RUL","Predicted RUL")
xlabel("Time stamp (Test data sequence)")
```

```
ylabel("RUL (Cycles)")

title("RUL for Test engine #"+idx+ " ("+lambdaCase+" case)")
end
```

Codice MATLAB dell'algoritmo con rete neurale artificiale convoluzionale (CNN):

```
filename = "CMAPSSData.zip";
if ~exist(filename,'file')
    url = "https://ti.arc.nasa.gov/c/6/";
    websave(filename,url);
end

dataFolder = "data";
if ~exist(dataFolder,'dir')
    mkdir(dataFolder);
end
unzip(filename,dataFolder)
filenameTrainPredictors = fullfile(dataFolder,"train_FD001.txt");
rawTrain = localLoadData(filenameTrainPredictors);
head(rawTrain.X{1},8)
rawTrain.Y{1}(1:8)
stackedplot(rawTrain.X{1},[3,5,6,7,8,15,16,24],'XVariable','timeStamp')
prog = prognosability(rawTrain.X,"timeStamp");
idxToRemove = prog.Variables==0 | isnan(prog.Variables);
featToRetain = prog.Properties.VariableNames(~idxToRemove);
```

```
for i = 1:height(rawTrain)
    rawTrain.X{i} = rawTrain.X{i}{:,featToRetain};
end
[~,Xmu,Xsigma] = zscore(vertcat(rawTrain.X{:}));
preTrain = table();
for i = 1:numel(rawTrain.X)
    preTrain.X{i} = (rawTrain.X{i} - Xmu) ./ Xsigma;
end
clipResponses = true;

if clipResponses
    rulThreshold = 150;
    for i = 1:numel(rawTrain.Y)
        preTrain.Y{i} = min(rawTrain.Y{i},rulThreshold);
    end
end

WindowLength = 40;
Stride = 1;
dataTrain = localGenerateSequences(preTrain,WindowLength,Stride);

numFeatures = size(dataTrain.X{1},2);
InputSize = [WindowLength numFeatures 1];
for i = 1:size(dataTrain,1)
    dataTrain.X{i} = reshape(dataTrain.X{i},InputSize);
end

filterSize = [5, 1];
numHiddenUnits = 100;
numResponses = 1;
```

```
layers = [
    imageInputLayer(InputSize)
    convolution2dLayer(filterSize,10)
    reluLayer()
    convolution2dLayer(filterSize,20)
    reluLayer()
    convolution2dLayer(filterSize,10)
    reluLayer()
    convolution2dLayer([3 1],5)
    reluLayer()
    fullyConnectedLayer(numHiddenUnits)
    reluLayer()
    dropoutLayer(0.5)
    fullyConnectedLayer(numResponses)
    regressionLayer()];

maxEpochs = 20;
miniBatchSize = 512;

options = trainingOptions('adam', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.3,...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',0.003, ...
    'Shuffle','every-epoch',...
    'Plots','training-progress',...
    'Verbose',0);
```

```
net = trainNetwork(dataTrain,layers,options);

figure;
lgraph = layerGraph(net.Layers);
plot(lgraph)

filenameTestPredictors = fullfile(dataFolder,'test_FD001.txt');
filenameTestResponses = fullfile(dataFolder,'RUL_FD001.txt');
dataTest = localLoadData(filenameTestPredictors,filenameTestResponses);

for i = 1:numel(dataTest.X)
    dataTest.X{i} = dataTest.X{i}{:,featToRetain};
    dataTest.X{i} = (dataTest.X{i} - Xmu) ./ Xsigma;
    if clipResponses
        dataTest.Y{i} = min(dataTest.Y{i},rulThreshold);
    end
end

unitLengths = zeros(numel(dataTest.Y),1);
for i = 1:numel(dataTest.Y)
    unitLengths(i) = numel(dataTest.Y{i,:});
end
dataTest(unitLengths<WindowLength+1,:) = [];

predictions = table('Size',[height(dataTest)
2],'VariableTypes',["cell","cell"],'VariableNames',["Y","YPred"]);

for i=1:height(dataTest)
    unit = localGenerateSequences(dataTest(i,:),WindowLength,Stride);
    predictions.Y{i} = unit.Y;
```

```
    predictions.YPred{i} = predict(net,unit,'MiniBatchSize',miniBatchSize);
end

for i = 1:size(predictions,1)
    predictions.RMSE(i) = sqrt(mean((predictions.Y{i} -
predictions.YPred{i}).^2));
end

figure;
histogram(predictions.RMSE,'NumBins',10);
title("RMSE ( Mean: " + round(mean(predictions.RMSE),2) + " , StDev: "
+ round(std(predictions.RMSE),2) + " )");
ylabel('Frequency');
xlabel('RMSE');

figure;
localLambdaPlot(predictions,"random");

function data = localLoadData(filenamePredictors,varargin)

if isempty(varargin)
    filenameResponses = [];
else
    filenameResponses = varargin{:};
end


rawData = readtable(filenamePredictors);
```

```
VarNames = {...
    'id', 'timeStamp', 'op_setting_1', 'op_setting_2', 'op_setting_3', ...
    'sensor_1', 'sensor_2', 'sensor_3', 'sensor_4', 'sensor_5', ...
    'sensor_6', 'sensor_7', 'sensor_8', 'sensor_9', 'sensor_10', ...
    'sensor_11', 'sensor_12', 'sensor_13', 'sensor_14', 'sensor_15', ...
    'sensor_16', 'sensor_17', 'sensor_18', 'sensor_19', 'sensor_20', ...
    'sensor_21'};
rawData.Properties.VariableNames = VarNames;

if ~isempty(filenameResponses)
    RULTest = dlmread(filenameResponses);
end



IDs = rawData{:,1};
nID = unique(IDs);
numObservations = numel(nID);



data = table('Size',[numObservations 2],...
    'VariableTypes',{'cell','cell'},...
    'VariableNames',{'X','Y'});

for i=1:numObservations
    idx = IDs == nID(i);
    data.X{i} = rawData(idx,:);
    if isempty(filenameResponses)

        data.Y{i} = flipud(rawData.timeStamp(idx))-1;
    else
```

```
        sequenceLength = sum(idx);
        endRUL = RULTest(i);
        data.Y{i} = [endRUL+sequenceLength-1:-1:endRUL]';
    end
end
end

function seqTable =
localGenerateSequences(dataTable,WindowLength,Stride)

getX = @(X) generateSequences(X,WindowLength,Stride);
getY = @(Y) Y(WindowLength:Stride:numel(Y));

seqTable = table;
temp = cellfun(getX,dataTable.X,'UniformOutput',false);
seqTable.X = vertcat(temp{:});
temp = cellfun(getY,dataTable.Y,'UniformOutput',false);
seqTable.Y = vertcat(temp{:});
end



function seqCell = generateSequences(tsData,WindowLength,Stride)




getSeq = @(idx) tsData(1+idx:WindowLength+idx,:);

idxShift = num2cell(0:Stride:size(tsData,1)-WindowLength)';
```

```
seqCell = cellfun(getSeq,idxShift,'UniformOutput',false);
end

function localLambdaPlot(predictions,lambdaCase)

if isnumeric(lambdaCase)
    idx = lambdaCase;
else
    switch lambdaCase
        case {"Random","random","r"}
            idx = randperm(height(predictions),1);
        case {"Best","best","b"}
            idx = find(predictions.RMSE == min(predictions.RMSE));
        case {"Worst","worst","w"}
            idx = find(predictions.RMSE == max(predictions.RMSE));
        case {"Average","average","a"}
            err = abs(predictions.RMSE-mean(predictions.RMSE));
            idx = find(err==min(err),1);
    end
end
y = predictions.Y{idx};
yPred = predictions.YPred{idx};
x = 0:numel(y)-1;
plot(x,y,x,yPred)
legend("True RUL","Predicted RUL")
xlabel("Time stamp (Test data sequence)")
ylabel("RUL (Cycles)")

title("RUL for Test engine #"+idx+ " ("+lambdaCase+" case)")
end
```