UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Utilizzo di tecniche di Machine Learning per l'analisi dello stato di deformazione di microstutture

Use of Machine Learning algorithms to analyse the deformation state of microstructures

Candidato:
Vasco Pesce

Relatore:
Prof. Marco Rossi

Correlatore:
Prof. Emanuele Principi
Ing. Giulia Tanoni

Anno Accademico 2022-2023

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Utilizzo di tecniche di Machine Learning per l'analisi dello stato di deformazione di microstutture

Use of Machine Learning algorithms to analyse the deformation state of microstructures

Candidato:
Vasco Pesce

Relatore:
Prof. Marco Rossi

Correlatore:
Prof. Emanuele Principi
Ing. Giulia Tanoni

Anno Accademico 2022-2023

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica
Via Brecce Bianche – 60131 Ancona (AN), Italy

Ai nonni

Ringraziamenti

I miei ringraziamenti vanno ai compagni di corso che ho incontrato durante questi anni, ai professori che mi hanno aiutato e guidato in questo percorso, agli amici con cui ho condiviso tanto e che sono la soluzione a qualunque brutta giornata. Agli zii e ai cugini che sono stati parte integrante della mia vita sin da piccolo e che sono ancora oggi una presenza importante, ai nonni a cui dedico il mio lavoro: uno che avrebbe voluto farmi tagliare i capelli a cui dico che non deve temere perchè questà facoltà presto o tardi me li farà cadere tutti; l'altro che sarebbe stato fiero di sapere che ho superato l'esame di "ingegneria meccanica" a cui dico a malincuore che non esiste un esame di ingegneria meccanica. Spero che questo traguardo conti lo stesso per lui e sono sicuro che sarà così. Una nonna a cui di cose che mi sono successe in questi anni vorrei racconterne tante ma per ora mi limito a dirle che non sono più "il più bello del mondo"...ora sono anche il più intelligente! E alla nonna qui presente a cui confesso che questi anni di duro lavoro e studio li ho intrapesi solo ed esclusivamente per poterla guardare negli occhi e chiederle:"Adesso che mi sono laureato mi faresti una vaschetta di mascarpone come regalo?". Ma alla fine non c'è bisogno che le dica niente perchè la sua presenza è più che sufficente per rendermi felice. In ultimo ho messo i miei genitori non perchè non siano importanti al contrario sono fondamentali. Sono stati il fulcro della vita, mi hanno insegnato tanto lungo la strada e sostenuto davvero moltissime volte. Chiariamoci, non siamo sempre andati d'accordo ma questo fa parte del gioco, lo scontro con i propri genitori, per quanto doloroso o difficile, serve a farci crescere ed è inevitabile tanto più se si hanno genitori come i miei che provano un amore incondizionato nei confronti del figlio. In definitiva sono il mio pilastro, vi voglio bene.

P.S. Ringrazio Nina, la più importante in assoluto, infatti è l'unico nome che compare nei ringraziamenti, per tutti coloro che hanno commenti su questa parte: teneteveli oramai sono ingegnere e gli ingegneri non perdono tempo con forme di vita inferiori.

Ancona, Luglio 2023

Vasco Pesce

Abstract

The following work consists of three part: the first one is a description about the field and about the tools used during the experimentation, so we talk about machine learning and neural network focusing on the convolutional ones. In the second one we describe the work process used in this experimentation, network model, elements in it and the working steps. In the last one we put the results of our research with the effectiveness of the system used for the analysis of deformation state of microstructure, if the system is able to reach a good value which describe the deformation starting from the provided image and the accuracy of the results even changing the conditions.

Sommario

Il seguente elaborato si divide in tre capitoli: il primo è una descrizione del campo in cui ci muoviamo con tanto di spiegazione degli strumenti utilizzati ai fini di questa ricerca, quindi spiegazioni su machine learning e reti neurali con particolare attenzione a quelle convoluzionali, nel secondo descriviamo il processo lavorativo di cui ci avvaliamo, modello di rete, elementi costituenti e step lavorativi, l'ultimo capitolo riporta l'efficacia dei sistemi usati nell' analisi dello stato di deformazione di microstrutture, se riescono dunque a ricavare un valore approssimato dello stato di deformazione a partire dall'immagine fornita e con quanta accuratezza svolgono questa funzione anche ponendoci in condizioni diverse.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Machine learning e reti neurali

1.1 Introduzione

"Le cose cominciarono a migliorare di nuovo per l' IA nel 1997, quando un sistema chiamato Deep Blue, di proprietà di IBM, sconfisse Garry Kasparov, il campione del mondo di scacchi dell'epoca. Si trattava di una conquista straordinaria, ma ancora più straordinario fu il modo in cui il sistema riuscì a batterlo. Deep Blue non cercava di emulare la creatività di Garry Kasparov, il suo intuito o il suo genio. Non replicava il processo del suo pensiero né imitava il suo ragionamento. Usava piuttosto un' enorme potenza di calcolo e un'amplissima capacità di archiviazione di dati per elaborare fino a 330 milioni di mosse in un secondo. Kasparov, uno dei migliori giocatori di scacchi di tutti i tempi, poteva forse tenere a mente un centinaio di possibili mosse in un momento." [1] ... Il successo ottenuto nel gioco degli scacchi dal programma noto come Deep Blue lo possiamo in pratica definire come il battesimo di uno degli strumenti tecnologici che rivoluzionerà e sta già rivoluzionando il nostro mondo: stiamo parlando del Machine Learning. Il Machine Learning sta trovando sempre più campi d'applicazione come operazioni bancarie, acquisti online, social media, applicazioni ingegneristiche.

1.2 Reti neurali

Spesso la definizione Machine Learning viene confusa con l'intelligenza artificiale pur non essendo propriamente la stessa cosa. Il Machine Learning è infatti un sottoinsieme dell' IA (sistemi o macchine che imitano l'intelligenza umana) e si occupa di creare sistemi che apprendono e migliorano le capacità e le performance in base ai dati che utilizzano.

Il termine machine learning viene usato per la prima volta nel 1959 dallo scienziato di nazionalità americana Arthur Lee Samuel il quale riuscì a creare un programma per la prima volta basato sulla capacità di apprendimento automatica, una svolta importante nel mondo dell'IA, il programma riguardava nello specifico il gioco della dama per computer. Ulteriori passi avanti in questo

ambito sono stati fatti nel 1969 con il libro Perceprtons scritto dai ricercatori informatici Minsky e Papert. Gli autori dimostravano i limiti di questi sistemi, in particolar moldo i limiti delle rete neurali. Le rete neurali sono la componente fondamentale dei sistemi di apprendimento automatici. Esse hanno il compito di imitare il funzionamento del cervello umano servendosi dei cosidetti neuroni artificiali che si collegano e scambiano informazioni tra loro: così come nel cervello umano avvengono le sinapsi nelle reti neurali i collegamenti tra neuroni permettono il processamento e lo scambio di dati. La prima bozza di rete neurale risale al 1493 quando un neurofisiologo Warren Sturgis e il matematico Warren Pitts modellarono il primo neurone artificiale, il suo scopo era quello di convertire i dati binari multipli nello strato di input in un singolo dato nello strato di output.

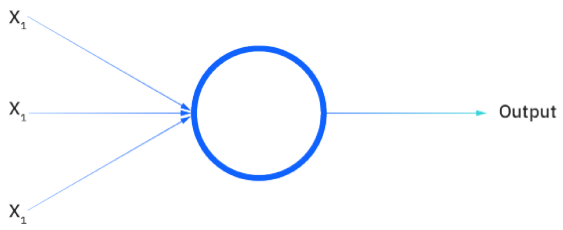

Figura 1.1: neurone artifiale con 3 input e 1 output

La combinaziopne di un numero sufficente di questi neuroni artificiali portò alla prima rete neurale, una forma, se vogliamo, primitiva in grado di risolvere semplici funzioni a due variabili. Nel 1958 Frank Rosenblatt presentò lo schema di rete neurale chiamato Perceptron, che costituisce lo scheletro delle rete neurali ancora oggi utilizzate. Tale schema prevedeva oltre l'utilizzo di uno strato di input e uno di output anche l'utilizzo di un algoritmo (sequenza di calcoli matematici) per la minimizzazione degli errori. Come fecero però notare Minsky e Papert il percettrone costituiva una rete neurale poco potente, non adatta a calcoli complessi in quanto l'algoritmo per la minimizzazione dell' errore alterava i pesi delle connessioni tra i neuroni artificiali. Ciò portava a ottenere un' uscita diversa da quella desiderata. Fu solo nel 1986 che un ricercatore di nome David Rumelhart introdusse un nuovo strato di rete neurale, lo strato detto "Hidden" (nascosto) dando così vita alle MLP (multi-layer-perceptors,ovvero percettrone multi strato). Rumelhart dimostrò che, con l'utilizzo di un algoritmo detto di retropropagazione dell'errore, i pesi delle connessioni venivano modificati in funzione dell'obiettivo: ottenere un uscita quanto più prossima a quella desiderata. Così giungiamo alle rete neurali

odierne, divise quindi in uno strato di input, uno o più strati nascosti, e uno strato di output. Ogni strato(layer) è composto da nodi(che sarebbero i neuroni artificiali); ciascun nodo si connette a quelli del layer successivo e ha un peso e una soglia associati. Il peso serve a determinare l'importanza di qualsiasi variabile specifica, pesi più grandi contribuiscono maggiormente all'output del nodo stesso, questo viene poi passato attraverso una funzione d'attivazione, se il valore dell'output supera la soglia del nodo questo viene attivato permettendo il passaggio dell' informazione al layer successivo e l'output di un nodo diventa l'input del successivo. Questo ci permette di filtrare l'informazione attraverso una serie di calcoli specifici che permettono di giungere al risultato desiderato.

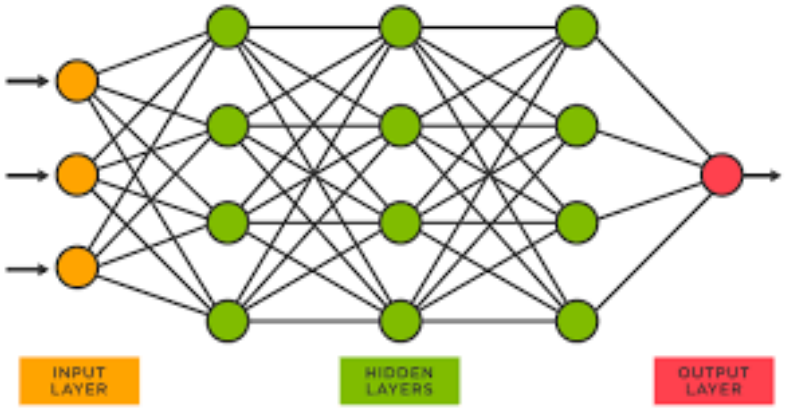

Figura 1.2: rete neurale stratificata

Facciamo un piccolo esempio: immaginiamo di dover decidere se andare a fare surf e l'output che stiamo cercando è quindi una risposta precisa e cioé "sì" che corrisponde nel linguaggio binario a 1 oppure "no" che corrisponde a 0. Le tre variabili (fattori) che influenzano la decisione verrano definite con la X: $X1$ = condizioni del mare,$X2$ = condizioni fisiche,$X3$ = condizioni climatiche.

Ora presupponiamo che le condizioni climatiche siano buone $X3 = 1$,le condizioni del mare siano favorevoli X1=1, ma accusassimo dolori alla schiena $X2 = 0$.A questo punto assegnamo ad ogni input un peso a seconda dell'importanza: $W1 = 5$ le condizioni del mare hanno quindi un notevole peso, $W2 = 2$ visto che il problema alla schiena non è invalidante, $W3 = 4$ il clima è un fattore importante. Infine prendiamo come valore di soglia 3.

Tabella 1.1: Esempio.

fattori	pesi
X_1	W_1
X_2	W_2
X_3	W_3

Utiliziamo ora la formula:

$$\text{output} = f(x) = 1 \text{ if } \sum (W_i X_i) + b \geq 0 \tag{1.1}$$

da cui otteniamo

$$\text{output} = (W_1 X_1) + (W_3 X_3) - 3 \tag{1.2}$$

il meno davanti al tre viene messo perchè il valore dell'output deve superare la soglia, quindi la loro differenza deve essere maggiore di zero, in questo caso:

$$(5\text{x}1) + (4\text{x}1) - 3 = 6 > 0 \tag{1.3}$$

quindi l'uotput, essendo maggiore di 0, è 1 ovvero la risposta alla domanda se fare o meno surf è sì.

1.3 Reti neurali convoluzionali

Ci sono vari tipi di reti neurali con campi d'applicazione differenti, basti pensare alle reti neurali riccorenti che hanno vasto utilizzo nel riconoscimento vocale. Oppure le reti convoluzionali più utilizzate in ambito di riconoscimento visivo e classificazione per immagini. E' proprio su quest'ultime che concentreremo la nostra attenzione.

Le Convolutional neural network (CNN) hanno rappresentato una svolta importante nelle attività di identificazione di persone e oggetti a partire da un immagine, infatti i metodi utilizzati in precedenza per il riconoscimento e la classificazione visiva richidevano lavori meticolosi e dispensivi, sopratutto in termini di tempo.

Le CNN sfruttano principi dell'algebra lineare per l'identificazione all'interno di immagini, si avvalgono però di calcoli complessi e necessitano di una qualità grafica non da poco quindi di schede GPU adatte. Le CNN hanno poi tre tipi di livello:

- livello convoluzionale
- livello di pooling
- livello completamente connesso (Fully-connected)

1.3.1 Livello convoluzionale

Il livello convoluzionale è il punto dove si verificano la maggior parte dei calcoli e quindi il principale elemento costituente le CNN. E' dotato di un filtro e una mappa delle funzioni che servono per il processo detto appunto

di convoluzione. Il termine convoluzione, da un punto di vista matematico, significa proprio "far scorrere" una funzione sopra un' altra al fine di mescolarle, il risultato sarà una terza funzione, prodotto delle altre due.

Prendiamo ad esempio come input un'immagine, questa sarà costituita da pixels, in particolare una matrice di questi; essendo in 3D l'input sarà dato da tre dimensioni:larghezza, lunghezza, profondità. Il filtro (rilevatore di funzioni) dovrà spostarsi attraverso i campi dell'immagine e verificare la presenza della funzione. Tale filtro è una raccolta di elementi, che sarebbero i pesi, specificamente ordinati detta array, ispirati ai vettori matematici nel caso in cui siano monodimensionali o ispirati alle matrici quando sono multidimensionali. Nel caso dei filtri costituenti le reti convoluzionali sono bi-dimensionali (2D).

L'array bi-dimensionale in questione rappresenta semplicemente parti dell'immagine.A questo punto il filtro viene applicato ad un'area dell'immagine dove viene effettuato un prodotto di punti tra input e filtro cioé tra le dimensioni e i pesi, il prodotto andrà a costituire poi un array di output.

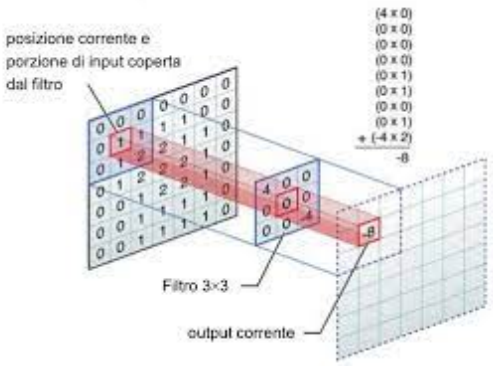

Figura 1.3: filtro di convoluzione

Il processo vieni ripetuto spostando ogni volta il filtro di un passo fino a quando non avrà ricoperto l'intera immagine, l'output finale è la mappa delle funzioni del livello convoluzionale stesso anche detta funzione convoluta.

C'è da dire che dopo ogni convoluzione si aggiunge una trasforamzione che introduce la non linearità del modello chiamata ReLu (Rectified Linear Unit). Questa serve ad annulare quei valori ottenuti in precedenza che non sono utili.Una convoluzione; può inoltre essere seguita da un'altra convoluzione creando così una gerarchia di convoluzioni. Passiamo a fare un esempio, Ipotizziamo di voler trovare un' auto in un immagine, l'auto la possiamo considerare scomponibile in varie parti (ruote, telaio, vetri) così ogni componente costituisce un modello di basso livello della rete. L'immagine del auto nel suo insieme ne costituisce uno di alto livello e così via a creare una gerarchia di funzioni.

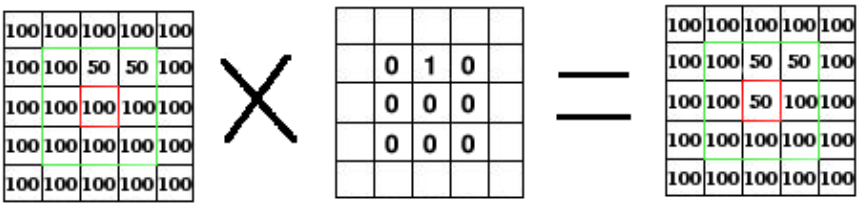

Figura 1.4: matrice di convoluzione

1.3.2 Livelli di pooling

Passiamo ora ai livelli di pooling, conosciuti anche come livelli di sottocampionamento. Il loro scopo è quello di ridurre il numero dei parametri di input, e quindi ridurre la dimensionalità.

Come nei livelli di convoluzione anche in quelli di pooling abbiamo la presenza di un filtro; la differenza sostanziale e che in questo non abbiamo pesi associati, è dunque privo di array bi-dimensionale. Il filtro in sostanza si limita ad applicare una funzione d'aggregazione ai valori del campo recettivo e fornisce un'array di output.

Il procedimento può essere definito di massimo pooling o di medio pooling. In quello di massimo pooling, maggiormente diffuso, il filtro prende il pixel con valore più alto e lo invia all'array di output. Nel medio pooling invece il filtro calcola il valore medio nel campo recettivo e lo invia all'array di output. In sostanza il livello di pooling è responsabile di una scrematura di informazioni, che può essere visto come un aspetto negativo se si pensa alla quantità di informazioni perse. Ha però sicuramente dei vantaggi come, ad esempio, consente di ottenere una minore complessità del sistema e inoltre riduce il rischio di overfitting o sovradattamento.

Il termine overfitting sta ad indicare un fenomeno che può verificarsi durante la fase d'addestramento delle reti neurali, dove la rete non apprende correttamente i calcoli che deve eseguire e il processo con cui deve valutare le informazioni ma semplicemente impara a memoria tali dati. Quando poi si vogliono fare valutazioni su un set di dati ex-novo la rete non riesce a fornire risultati accettabili, il fenomeno dell'overfitting e l'addestramento della rete neurale sono temi che affronteremo nel capitolo successivo.

1.3.3 Livello completamente connesso

In quest'ultimo livello si esegue l'attività di classificazione che si basa sulle funzioni estratte nei livelli precedenti e, a differenza di questi livelli, c'è una connessione diretta per ogni nodo del livello di output con un nodo del livello

precedente. Non troviamo inoltre la funzione ReLu ma bensì una funzione d'attivazione che classifica gli input in modo appropriato.

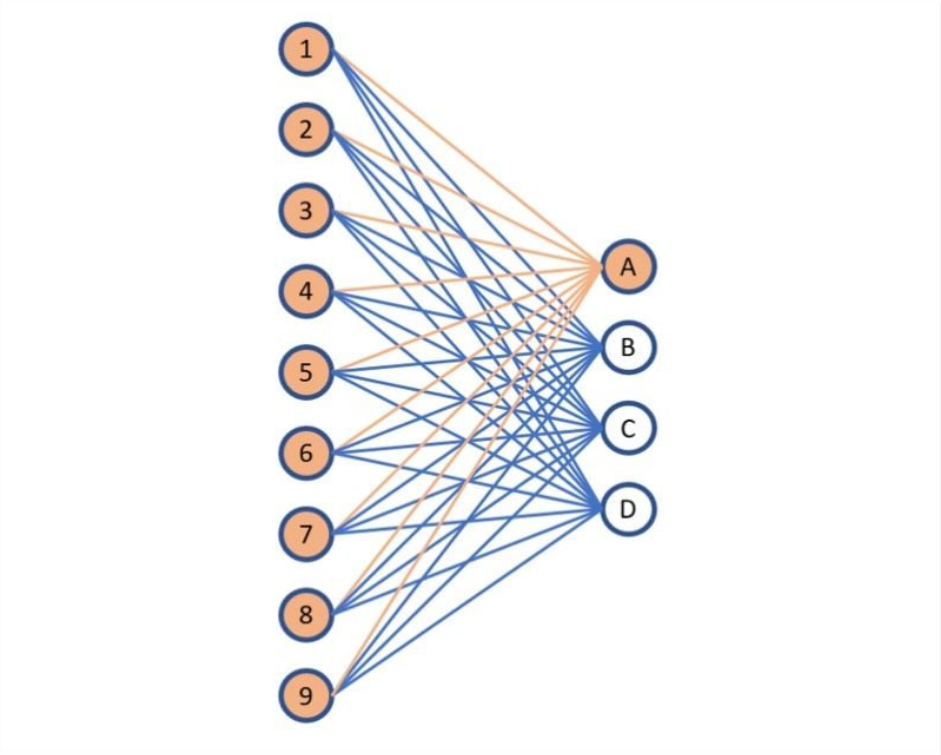

Figura 1.5: layer fully-connected

Otteniamo quindi alla fine un rete neurale convoluzionale.

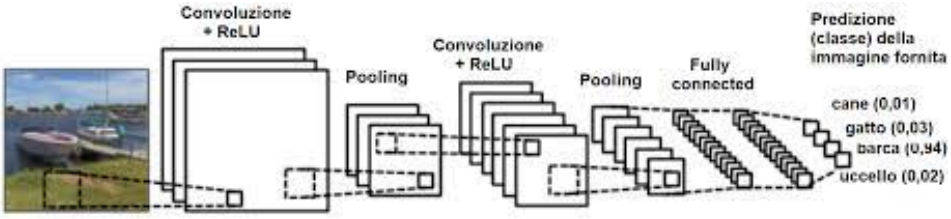

Figura 1.6: rete neurale convoluzionale

1.4 Machine learning

Finora abbiamo parlato di reti neurali, la base dei sistemi di appredimento automatici. Ora spendiamo qualche parola per descrivere meglio il Machine learning.

Come abbiamo già detto, per machine learning intendiamo un sottoinsieme dell'intelligenza artificiale basato sullo sviluppo e l'implementazione di algoritmi

che devono fornire ai sistemi di calcolo la capacità di imparare e migliorare automaticamente dopo ogni utilizzo, a livello pratico.

Partendo da un set di dati noto che costituisce l'input bisogna risalire ad un risultato preciso:l'output, Tanto sarà in grado di automigliorarsi il sistema tanto sarà minimizzato l'errore in uscita e sarà quindi fornito un output quanto più prossimo al valore desiderato.

Esistono due tipi di machine learning e si basano sui diversi approcci all'apprendimento che vengono usati, il primo viene definito machine learning supervisionato, l'altro machine learning non supervisionato.

In realtà bisognerebbe aggiungere in questa differenziazione anche il machine learning semi-supervisionato e quello per rinforzo.

1.4.1 Machine learning supervisionato

In questa tipologia, che risulta essere la più utilizzata, è fondamentale la presenza di un operatore il cui scopo è quello di guidare l'allenamento della rete spingendola a generare determinati risultati seguendo precisi calcoli. In che modo?

Nella fase di addestramento l'operatore fornisce alla rete un set di dati in input e un set di dati di output predefinito. La rete dunque si allena ad ottenere l'output dato a partire dal input fornito, impara cioé i calcoli e le operazioni da dover svolgere. Quando poi il sistema è sufficentemente pronto, viene testato ed eventualmente utilizzato, migliorando sempre più le sue capacità dopo ogni processo di apprendimento.

Campi di utilizzo che possiamo trovare nella nostra quotidianeità sono i motori di suggerimento nei siti di acquisti o nei navigatori per le automobili [2].

1.4.2 Machine learning non supervisionato

Con questo tipo di approccio, fortemente indipendente, l'operatore, che prima era fondamentale, diventa quasi superfluo . Il sistema impara a riconoscere e gestire schemi complessi e qui i dati in input non sono più etichettati e l'output non è predefinito ma costituisce un' incognita.

Campi di utilizzo possono essere il riconoscimento facciale e analisi di sequenze genetiche.

1.4.3 Machine learning semi-supervisionato

Generalmente tali modelli si utilizzano quando i dati di input da analizzare sono in numero così elevato che diventa impossibile etichettarli tutti, così se ne etichetta solo una porzione. Lo scopo è quello di fornire al sistema una base da cui partire che possa accelerare il processo d'apprendimento o quanto meno

semplificarlo almeno nella fase iniziale. Inoltre l' algoritmo dei modelli semi-supervisionati analizza i set di dati etichettati cercando pattern o correlazioni da trovare ove possibile nei set non etichettati.

Campi di utilizzo sono il riconoscimento vocale, analisi linguistica e sta trovando spazio anche in campo medico, dove viene usato per categorizzare le proteine.

1.4.4 Apprendimento per rinforzo

Nell' apprendimento per rinforzo, il sistema viene spinto dalla ricerca di un obiettivo che può essere un insieme di azioni o uno stato finale ammissibile. L'obiettivo può essere fisso o mutevole. Nel tal caso l' obiettivo è definito premio numerico già programmato nel algoritmo come meta da raggiungere del sistema. In un certo senso anche qui come nel modello supervisionato il sistema impara per esempio mentre la ricompensa serve da rinforzo.

Un esempio efficace sarebbe il gioco degli scacchi, dove si insegnano all'allievo le regole piuttosto che spiegargli le infinite mosse o combinazioni; sta poi a lui sviluppare una strategia di gioco. In questo paragone il premio sarebbe la conquista dei pezzi avvresari.

Campi di utilizzo sono le contrattazioni sui mercati azionari o lo sviluppo di modalità di intrattenimento nel settore videoludico.

1.4.5 Algoritmi

Detto questo è doveroso fare una precisazione. Quando si parla della differenziazione tra le varie tipologie di apprendimento non è sbagliato usare il termine Machine Learning ma sarebbe più corretto fare tale distinzione riferendosi all' algoritmo: è infatti il diverso algoritmo usato a determinare i diversi metodi di apprendimento del sistema.

Per dare un' idea di cosa sia un algoritmo potremmo definirlo come un insieme di regole e operazioni da fare per arrivare a un detreminato risulato. Per quanto riguarda gli algoritmi per l'apprendimento supervisionato, su cui ci concentriamo, possiamo distinguerli in classificatori e quelli di regressione.

Nei classificatori l'output è detto etichetta o label, ed è una variabile qualitativa binaria: vero o falso, sì o no. Oppure può essere una varibile nominale: tipo di macchina, colore dei capelli... Infine può essere una variabile ordinale: junior, senior. Nei regressori l'output è un valore numerico continuo, i valori numerici compresi in un intervallo.

Oltre all'output la differenza tra i due algoritmi sta nella modalità di valutazione della qualità del modello: per i classificatori avviene un conteggio delle etichette predette correttamente e quelle non. Poi si stimano dei coefficenti

di accuratezza, precisione, sensibilità. Nei regressori invece tali coefficenti si basano sulla valutazione di una funzione di errore.

1.4.6 Funzionamento del modello supervisionato

Come già detto il modello supervisionato è il più diffuso. Oltre però ad essere il più diffuso è anche quello maggiormente elaborato, si costituisce infatti di diverse fasi:

- preparazione dei dati di addestramento; in questa fase ad ogni set di dati di input viene associato un set di dati di output, la presenza di un operatore è richiesta per svolgere tale associazione in alternativa ci si può affidare a dei sistemi di raccolta dati che svolgano la stessa funzione
- divisione del dataset in training e test
- fase di training dove si utilizzano i dataset omonimi e si addestra la rete facendole apprendere le correlazioni tra input e output prescelti. Tale fase è ripetuta più volte così da affinare le capacità della rete fino a quando non raggiungono stardard accettabili; In più oltre ai cicli di addestramento sono presenti cicli di validazione: il dataset viene scomposto in più parti e rotazione e una di queste viene utilizzata per la validazione invece che per l'addestramento
- fase di test in cui si sperimenta il modello sul dataset di test appunto, dati su cui il modello non si è addestrato, una volta generati gli output si confrontano con quelli reali e si valuta la qualità del modello sulla base di diversi parametri quali l' accuratezza, la precisione etc

Per capire meglio questo procedimento facciamo un esempio. Immaginiamo di dover classificare 100 e-mail a seconda delle tipologie e per rendere le cose ancora più semplici ipotiziamo di dover fare una classificazione binaria: spam/non-spam. Conosciamo quindi i dati input e sarà facile associarli ad un preciso output, la prima fase è dunque conclusa passiamo alla seconda.

Nella seconda fase dovremmo dividere il dataset in 3 parti: training, validation, test. Generalmente la parte più corposa di questi dati viene utilizzata per addestrare la rete quindi nella fase di training. Una parte più contenuta viene destinata alla validation purché si faccia attenzione alla quantità di dati usati in questa operazione perché una quantità esigua rischierebbe di invalidare il processo in quanto gli output sarebbero varati in un campo troppo stretto. Al contrario, l'uso di troppi dati per la validazione potrebbe sottrarre gli stessi al delicato processo di apprendimento. L'ultima parte infine viene dedicata al test. Per rendere un idea più precisa potremmo dire che una suddivione appropriata

di 100 elementi sarebbe 80-10-10: 80 per il training, 10 per la validation, 10 per il test.

Si passa poi alla fase di training dove il modello imparerà le correlazioni tra input e output. Nell' esempio che stiamo affrontando, in particolare, il modello imparerà a riconoscere, sulla base di alcuni fattori, quali e-mail sono da catalogare come spam e quali no. Inutile dire che la parte più delicata e importante di tutto il processo risiede proprio qui. I problemi che si possono riscontrare in questa fase, citando i più noti, sono l'overfitting e l'underfitting cioé l'incapacità del modello di apprendere e generalizzare su dati ex-novo, ecco perchè esiste la sottofase di validazione.

Nella validazione uitilizzeremo dei dati etichettati che però la rete non ha mai visto. Ovviamente per ognuno dei dati inseriti conosciamo già l'output. Per riprendre l'esempio, possiamo utilizzare delle e-mail su cui il modello non si è addestrato e di cui già sappiamo quali sono spam e quali no. Tale informazione tuttavia non viene inserita nel sistema, in quanto dobbiamo verificare la sua capacità predittiva. A questo punto il modello ci darà una predizione in base a ciò che ha appreso durante l'addestramento e catalogherà, in maniera predittiva, le e-mail spam e le e-mail non-spam. Confronteremo dunque le predizioni del modello con le informazioni reali di cui siamo in possesso, prendendo così consapevolezza delle capacità del nostro modello predittivo Se queste sono scarse dovremo ripetere la fase di training, magari cambiando i parametri. Nel caso in cui siano buone possiamo passare alla fase successiva.

La fase di test è la conclusiva. In poche parole si testa il modello su altri dati e si osservano le sue prestazioni.

1.4.7 Problema del overfitting e del underfitting

Il fenomeno dell'overfitting noto anche come sovradattamento si verifica quando il sistema appunto si adatta troppo ai dati che gli sono stati forniti e non riesce quindi a generalizzare su nuovi dati. Generalmente tale fenomeno si verifica quando si addestra la rete troppo a lungo su un set di dati specifico oppure troppo complesso. La rete dunque assorbe anche informazioni irrilevanti adattantosi troppo a quello specifico set di dati usato per addestrarla non riuscendo così a svolgere la sua funzione su dataset nuovi. Per riuscire ad individuare tale fenomeno si fanno dei test con un set di dati diverso da quello di addestramento. Stiamo parlando della fase di test. Se il sistema registra pochi errori durante il training e molti durante il test o la validation, allora siamo in presenza di overfitting.

Non serve dire quindi che evitare il problema dell'overfitting è cruciale, come evitarlo però è la vera domanda, infatti strumenti o passaggi come la validazione ci aiutano ad individuare questo problema, non a risolverlo. Per risolverlo

bisogna intervenire già fin dalle prime fasi del processo di allenamento: occorre infatti ridurre la complessità dei dati da elaborare, magari arrestando il processo precocemente o semplicemente andando ad escludere un certo quantitativo di dati, partendo ovviamente da quelli più ininfluenti.

Anche in questa operazione bisogna però mostrare particolare attenzione perchè la rimozione di troppi dati o l'arresto eccessivamente precoce dell' addestramento possono condurre al fenomeno contrario, ugualmente dannoso, del underfitting. In questo caso il sistema non si è addestrato a sufficenza per riuscire a riconoscere una correlazione input-output. Quindi in poche parole la rete non si è addestrata a dovere e non ce ne possiamo servire.

Fenomeni di overfitting o underfitting portano allo stesso risultato, cioè invalidano il sistema. ma Le caratteristiche che riscontriamo nei due casi tuttavia sono diverse: nell' underfitting si registrano distorsioni (bias) elevate e varianze ridotte nelle previsioni, accade ilk contrario nell' overfitting.

Diamo ora una rapida spiegazione di cosa siano varianza e distorsione. Quando si parla di previsione dobbiamo sempre associare la presenza di un errore che può essere scomposto in tre parti:

- errore di bias
- errore di varianza
- errore irriducibile

L'errore di bias si verifica nell' apprendimento ed è dovuto alla presenza di alcune ipotesi errate prese nella fase di apprendimento che produco risultati distorti. Ssi distinguono bias di selezione, di misurazione, di richiamo.

Per varianza si intende invece la differenza che c'è nell' accuratezza del sistema nel momento in cui si passa dai dataset usati nel training a quelli usati nel test. L'obiettivo sarebbe quello di avere un bias e una varianaza bassi cosa però impossibile in quanto sono inversamente proporzionali tra loro, quindi il massimo che si può fare è raggiungere un buon compromesso.

In definitiva la soluzione ideale si trova come molte volte nel mezzo, il punto d'equilibrio e di corretto funzionamento del sistema si trova a metà tra le condizioni di underfitting e overfitting.

Generalmente il fenomeno dell'overfitting si verifica molto più spesso rispetto al suo contrario, ecco perchè di seguito elencheremo alcune tecniche per evitarlo, l'ideale sarebbe utilizzare un modello lineare ma nella realtà ci capita spesso di imbatterci in problemi non lineari, quindi vediamo alcune tecniche di prevenzione o eliminazione di questo problema:

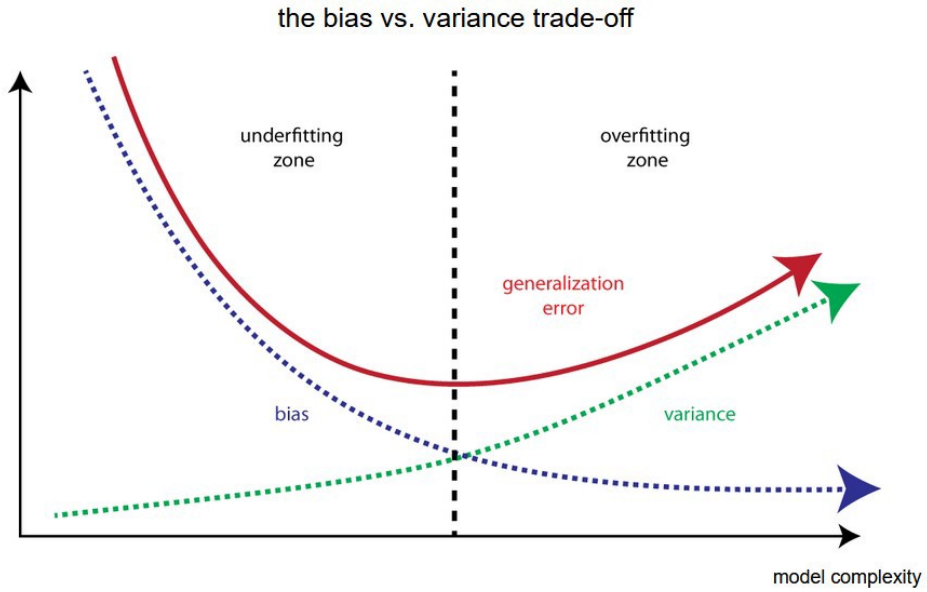

Figura 1.7: andamento delle curve di bias e varianza

- l' arresto anticipato, già accenato, è un sistema di prevenzione che si basa sull' interuzzione prematura del addestramento e che può portare all'effetto indesiderato dell'underfitting.

- l' aggiunta di nuovi dati, nuovi dati possono fornire al sistema più opportunità per capire il rapporto che deve esserci tra input e output, il tutto però regge solo nel ipotesi in cui i dati aggiunti siano "puliti" ovvero pertinentie utili, se però sono dati di poco conto il rischio è quello di aumentare la complessità del modello e di incentivare l'overfitting anziché evitarlo

- la selezione di funzioni dove la costruzione di un modello prevede l'utilizzo di parametri e funzioni usati nella predizione dei risultati, ma molte di queste funzioni sono superflue o ridondanti, un' accurata selezione delle più importanti può aiutare a ridurre le probabilità di overfitting

- i metodi d'insieme, costituiti da più classificatori le cui predizioni vengono mediate tra loro alla ricerca del risultato più diffuso. Questa tecnica è usata per la riduzione della varianza.

Una volta evitati questi due problemi il modello, essendo in grado di generalizzare su nuovi dati, è finalmente pronto per l'utilizzo.

1.5 Breve accenno al Deep learning

Il Deep learning è un sottoinsieme del Machine learning in cui le rete neurali sono allenate a sopportare grandi quantità di dati.

Come discusso in precedenza nelle reti neurali i dati vengono trasmessi da ogni livello a quello sucessivo fino ad arrivare al livello di output, nelle reti neurali di Machine learning possono esserci diversi livelli intermedi definiti nascosti comunque meno di una decina, nel Deep learning ce ne sono centinaia. La enorme quantità di livelli presenti nelle reti di Deep learning permettono di avere di una potenza di calcolo maggiore e quindi di poter gestire dati complessi con maggiore precisione per quanto poi una maggiore complessità della rete comporta anche l'utilizzo di più parametri e lo sfruttamentodi GPU ottimizzate.

Alcuni vantaggi propri di questo sistema sono:

- analisi di dati non strutturati, raccolti da varie fonti
- etichettatura dei dati, il sistema è in grado di etichettare autonomamente i set di data che gli vengono forniti
- velocità e precisione migliore, una volta addestrata adeguatemente la rete di Deep learning è in grado di fornire risultati precisi in tempi brevi rispetto al ML
- capacità di analizzare dati più complessi
- capacità di autogestirsi, molto più che nei sistemi di ML sono infatti in grado senza l'intervento umano di creare funzioni da soli

I campi in cui si possono applicare i sistemi di Deep learning sono numerosi, si va dal utilizzo nei social media (raccolta di informazioni utilizzate per annunci mirati) fino al settore finanziario (previsione trend economici), dal settore sanitario (predizione di malattie e disturbi in base ad abitudini e condizioni familiari) fino alla sicurezza informatica [3] ....

Ci sono comunque dei contro come ad esempio la inflessibilità del sistema cioè l'incapacità si adattarsi al di fuori del ambito molto specifico in cui la rete è stata addestrata oppure l'impossibilità di riuscire a scorgere il modo con cui la rete giunge alla soluzione, rendendo arduo così anche il compito di trovare e eliminare eventuali distorsioni durante i calcoli.

Nonostante tutto questo i sistemi di ML e DL e l'intelligenza artificiale tutta rappresentano il campo del futuro, le potenzialità racchiuse in questo settore e i margini di crescita sono enormi. Di giorno in giorno trovano applicazioni in sempre più ambiti, rimane dunque da vedere quali saranno le conquiste future. Nel seguito di questa tesi ci concentremo su un lavoro portato avanti per studiare le potenzialità e limiti di questi sistemi in ambito ingegneristico, in

particolar modo nello studio e nella misura delle deformazioni registrate nelle micrografie di materiali metallici.

Capitolo 2

Modello VGG16

2.1 Obbiettivi dello studio

Nel capitolo precedente abbiamo reso una spiegazione di quelli che sono gli strumenti usati nel corso di questo studio, abbiamo quindi parlato di Machine learning e Deep learning soffermandoci particolarmente sul modello supervisionato, abbiamo poi parlato di reti neurali e visto più da vicino le CNN (reti neurali convoluzionali) descrivendone i componenti e il funzionamento, infine abbiamo aggiunto qualche appunto su quelli che possono essere i problemi riscontrabili durante l'uso di questi strumenti.

Tutti gli elementi su cui abbiamo posto attenzione sono stati il centro di questo studio il cui obiettivo è quello di saggiare le potenzialità di tali strumenti nello studio delle deformazioni di materiali mettalici.

Fino ad oggi per studiare e valutare le deformazioni di materiale metallico a livello microstrutturale i metodi usati comportavano un costo sopratutto in termini di tempo elevato. Le lavorazioni di deformazione plastica sui provini e il conseguente studio al microscopio con tanto di confronto con una sturttura non deformata al fine di poter associare un valore alla deformazione erano e sono tutt'ora passaggi che richiedono tempo e risorse, basti pensare che nel corso di tutti questi passaggi un operatore deve essere presente sempre, e non come supervisore, ma deve partecipare attivamente ad ognuno dei passaggi sopra elencati per poi alla fine stimare lui stesso la deformazione a livello numerico.

L'utilizzo degli strumenti come il Machine learning può aiutare a snellire tale procedura, intanto perchè il sistema ha sì bisogno di un operatore che faccia da supervisore ma lavora comunque autonomamente facendo risparmiare tempo e risorse, inoltre, e questo è al centro di questo studio, se il sistema è stato adeguatamente addestrato la precisione dei calcoli dovrebbe fornire un risultato più accurato e per di più in tempi brevi.

La nostra sperimentazione si basa quindi sull' addestramento di una rete neurale convoluzionale, modello noto come $VGG16$, il cui input consistente in una serie di immagini al microscopio di materiale metallico con microstruttura ordinata e non deformate virtualmente da un software e usate come dati

di input per la rete. I relativi valori di deformazione, che per semplicità abbiamo ridotto a solo tre valori appartenenti ad uno stato dimensionale piano, rappresentano l'output e riguardano due deformazioni longitudinali e una deformazione angolare. I risulati ottenuti ci forniscono un'idea sull'affidabilità di tali strumenti in questo ambito.

In questo capitolo spiegheremo passo dopo passo le varie fasi che contraddistinguono il nostro lavoro soffermandoci per ognuna sugli elementi in gioco:

- fase di generazione delle immagini deformate, le immagini in gioco costituiscono i set di dati di training, validatione e test
- allenamento della rete neurale VGG16
- raccolta e analisi dei risultati che tratteremo nel terzo capitolo

2.2 Generazione di immagini deformate

Prima ancora di generare immagini deformate dobbiamo acquisire le immagini microstutturali dei metalli, Possiamo quindi utilizzare immagini ricavate al microscopio o immagini prese da precedenti sperimentazioni; nella reltà per addestrare più velocemente la rete, potremmo generare immagini virtuali regolari e deformarle.

Le immagini generate virtualmente possono risultare dei puntini neri su uno sfondo bianco; le immagini invece ricavate da studi condotti al microscopio su materiali metallici risulteranno avere la tipica microsttura dei metalli in cui si possono distinguere i grani e i bordi dei grani.

Per vedere l'effetto che l'ordine di una microstruttura ha sull'accuratezza dei risultati in uscita dalla rete, abbiamo portato avanti lo studio su metallografie ordinate e su microstrutture caotiche.

Una volta prese le immagini si passa a deformarle. Per questa operazione abbiamo sfruttato come ambiente di lavoro MatLab la cui defizione sta nel nome stesso. *Matrix Laboratory* è un ambiente di programmazione che ci permettere di manipolare e visualizzare dati operando anche complessi calcoli su di essi.

In MatLab abbiamo sfruttato un software di generazione delle immagini deformate in cui caricavamo le immagini acquisite al microscopio e, a partire da quelle, imprimevamo virtualmente delle condizioni di deformazione.

Il software per la generazione di immagini è costituito da più script (sequenza di istruzioni o comandi costituita da una o più righe di codice). Ogni script ha un ruolo preciso e all' interno di ognuno troviamo precise righe di codice che impartiscono al programma le azioni da svolgere. Per prima cosa impostiamo il

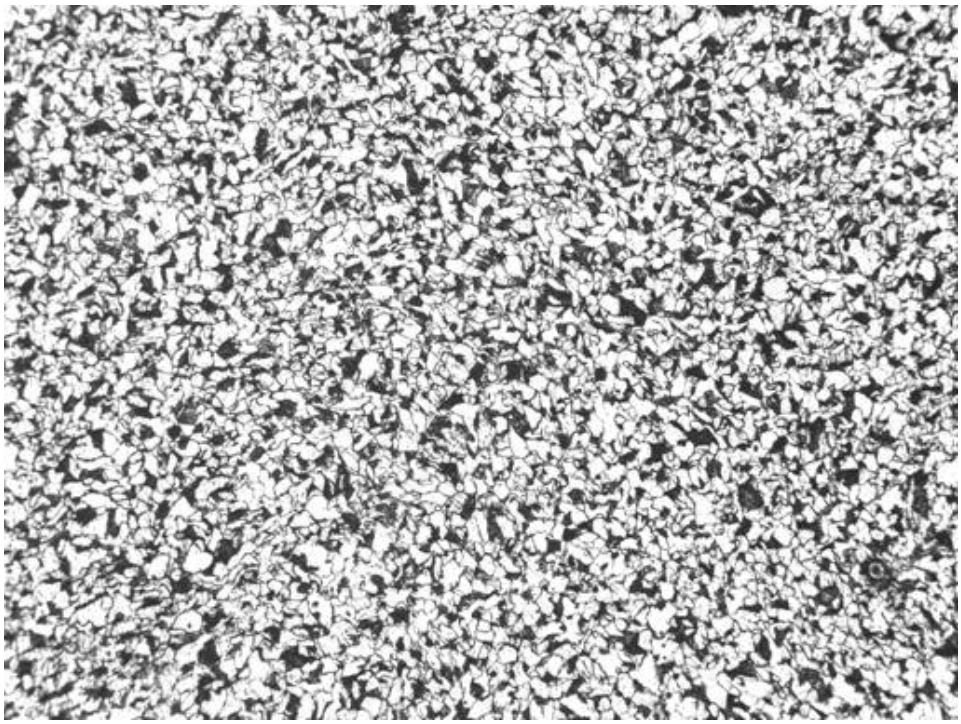

Figura 2.1: metallografia ordinata in bianco e nero

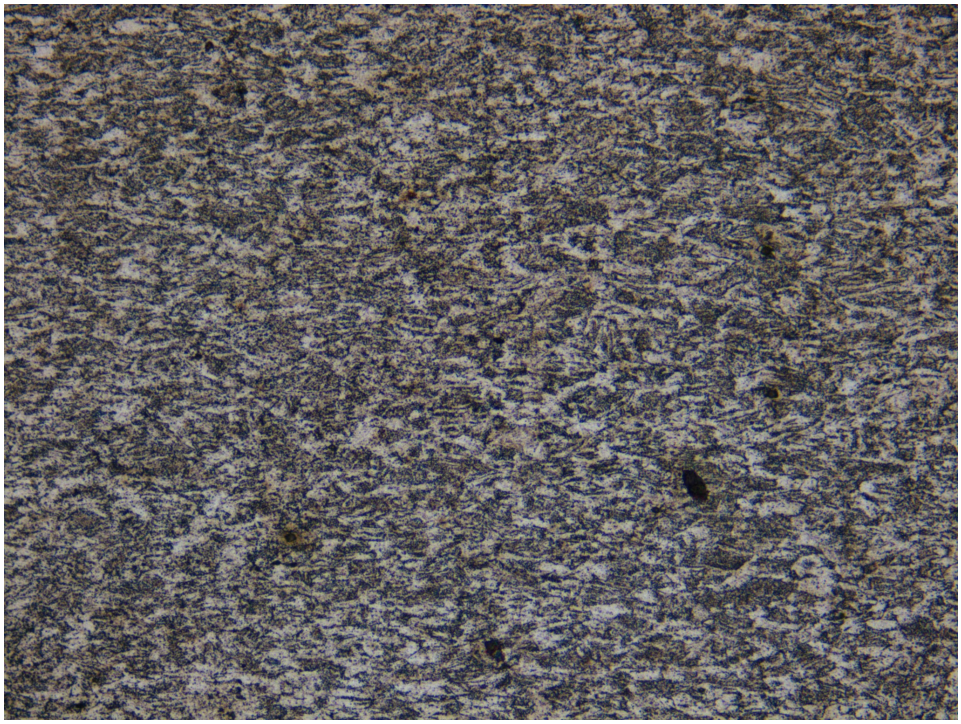

Figura 2.2: metallografia disordinata o caotica a colori

numero di immagini campioni da generare, in altre parole il numero di dati di input ovvero 7000. Inoltre diamo precise istruzioni sulla suddivisione dei dataset in dati di training (5000), validation (1000) e test (1000), questo significa che il 70% dei dati è destinato alla fase di addestramento della rete, il 15% alla validazione e l'altro 15% al test.

- Numero di campioni generati: 7000
- Numero di campioni destinati al dataset di training: 5000
- Numero di campioni destinati al dataset di validation: 1000
- Numero di campioni destinati al dataset di test: 1000

Naturalmente possiamo variare il numero di immagini generate, aumentarlo o diminuirlo, sempre tenendo a mente che un numero di campioni troppo piccolo potrebbe non addestrare adeguatamente la rete, troppo grande potrebbe aumentare eccessivamente la complessità dell'analisi da svolgere. Per quanto comunque varia il numero di campioni la percentuale di ripartizione nei dataset rimane costante.

Una volta caricata la metallografia impostiamo le dimensioni in termini di grandezza *subsize* dell'immagine deformata a partire da quella originale.

Per spiegarla in termini più semplici è come impostare la grandezza dello zoom da raggiungere, il termine *subsize* indica proprio le dimensione della sottoimmagine ottenuta zommando la metallografia originale. Questo passaggio ci permette, partendo da una singola immagine, di generare il numero di sottoimmagini deformate che vogliamo, in questo caso 7000. E' facilmente intuibile che modificando la subsize variano le dimensioni degli elementi su cui la rete svolge l'addestramento e quindi anche la complessità del problema, vedremo poi nel terzo capitolo con lo studio dei risultati quanto influisce tale fattore sulla qualità predittiva del modello.

Successivamente avviene la vera e propria generazione di immagini deformate. Tale deformazione può assumere valori che sono compresi in un intervallo da noi pre-impostato:

Tabella 2.1: intervalli di deformazione

def	min	max
e_1	-0.8	0.0
e_2	0.0	1.5
or	$-\pi/5$	$\pi/5$

dove e_1 è la deformazione lungo la direzione principale 1, e_2 lungo la direzione principale 2 e *or* è l'orientazione.

Come abbiamo già detto per semplificare le cose abbiamo considerato un deformazione bi-dimensionale. I valori di deformazione riportati rappresentano gli intervalli entro cui la deformazione applicata può muoversi.

Una volta generate le immagini deformate per vedere il risultato visivamente possiamo *plottare* ovvero mostrare le prime 12 immagini generate. In questo modo riusciamo a fare un rapido check visivo del lavoro che stiamo svolgendo, insieme poi ad ogni immagine mostrata ci sono i rispettivi valori di deformazione applicati.

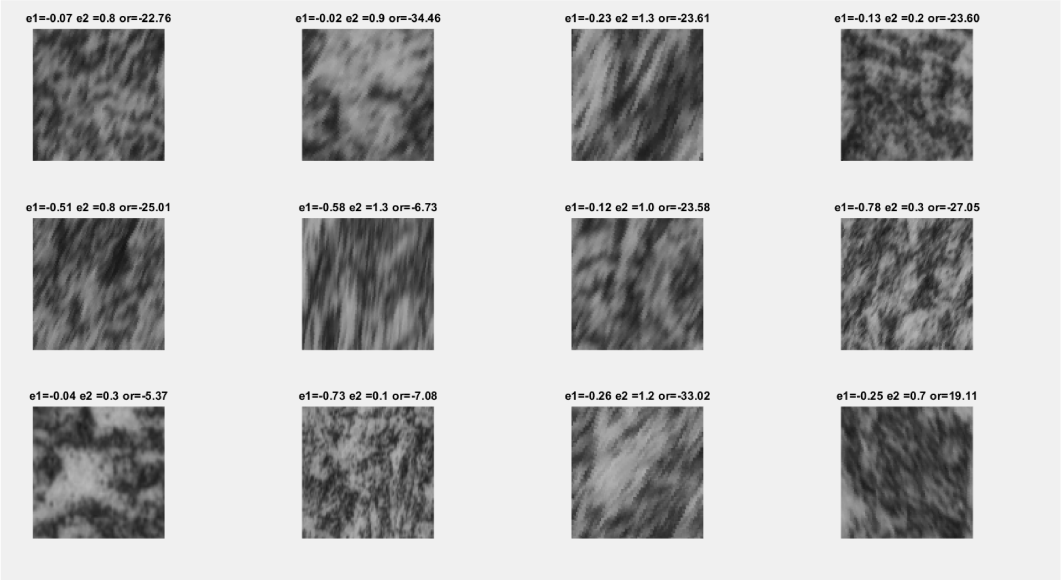

Figura 2.3: immagini plottate con rispettivi valori di deformazione

Infine la parte fondamentale : generare i dataset per il training, la validation e il test. Dando una rapida spiegazione, gli elementi che rappresentano i dati di input, quindi le immagini, sono indicati con la lettera X seguiti dalla macroarea in cui sono stati suddivisi train per il training, valid per la validation e infine quelli per il test. Gli elementi indicati con la Y sono invece gli output cioè i valori di deformazione sempre suddivisi nelle tre aree. Usando le funzioni di MatLab possiamo facilmente leggere le dimensioni dei dataset:

$$Xtrain = (5000; 99; 99; 1) \tag{2.1}$$

dove 5000 indica il numero di sottoimmagini generate usate nella fase di training come input, immagini che hanno una subsize pari a 99 (pixels). L'ultimo parametro (1) indica invece una caratteristica dell' immagine associata al colore di quest'ultima, nello specifico il parametro 1 indica un' immagine in bianco e nero.

Per quanto riguarda il dataset di output proprio della fase di training:

$$Ytrain = (5000; 3) \tag{2.2}$$

dove 5000 indica il numero di immagini di cui vogliamo conoscere l'output, 3 indica invece i valori di deformazione associati ad ognuna di queste immagini.

Per fare un ulteriore esempio se prendessimo i dataset di output associati alla fase di test questi risulterebbero:

$$Ytest = (1000; 3) \tag{2.3}$$

Qui non troveremo più 5000 ma 1000 in quanto le immagini destinate alla fase di test sono solo il 15% di quelle generate. Gli output saranno quindi altrettanti e per ognuno ci sarnno 3 valori di deformazione associati. Ovviamente possiamo notare come nei dati di output contrassegnati con la Y non appaia il parametro legato alla subsize (99), questo perchè l'output non è più una figura bensì un valore numerico.

Una volta generate le immagini passiamo alla fase di addestramento del modello, il fulcro di questa operazione.

2.3 Scelta del modello e addestramento della rete

Il passo successivo alla creazione dei dataset è il caricamento della rete neurale che vogliamo addestrare. Possiamo caricare una rete precedentemente addestrata oppure caricare una rete mai addestrata in precedenza.

La scelta del modello è fondamentale perchè bisogna vagliarne uno adatto al tipo di operazioni e alla quantità di dati che dovrà svolgere e analizzare. Inoltre tale scelta ci permette di prevenire il sorgere di problemi come l'overfitting o l'underfitting e ci permette di arrivare a risultati più precisi.

Esistono vari modelli di reti neurali, quello usato in questa sperimentazione è il modello noto come VGG16.[4]

2.3.1 Modello VGG16

Il modello VGG che prende il nome dal team di scienziati di Oxford che lo ha progettato (Visual Geometry Group) è l'architettura di rete neurale che ha segnato una svolta nel mondo delle CNN.

Fu grazie a questo modello che per la prima volta la rete convolutiva registrava un rate di errore inferiore al 10%. Tale modello fu tra i primi a sfruttare filtri di convoluzione 3x3 ripetuti in sequenza; filtri più piccoli rispetto a quelli usati fino a quel momento ma che riuscivano a colmare i difetti dei filtri più grandi.

Il concetto che rappresentava la svolta su cui i modelli VGG accesero i riflettori era la "profondità di rete". Tra i modelli proposti figuravano il VGG11,

il VGG16 e il VGG19. La differenza tra i vari modelli stava tutta nei livelli di profondità rispettivamente 11, 16 e 19.

Il VGG16 su cui ci concentriamo è quindi costituito da 16 livelli di profondità, 13 dei quali convoluzionali e 3 completamente connessi (Fully-connected). Sarà presente anche la ReLu.

Per spiegare il loro funzionamento facciamo un esempio: prendiamo immagini di input che hanno dimensioni 99x99x3, dove 99 indica il numero di pixels mentre 3 indica il numero di canali (RGB) ovvero la capacità di processare immagini a colori, anche se nel nostro studio le immagini vengo fornite alla rete in bianco e nero. Prendiamo dunque 99x99x1.

Come detto sopra i filtri di convoluzione usati sono i filtri 3x3, i più piccoli se escludiamo i filtri 2x2 e 1x1, quest'ultimo può essere interpretato come una trasformazione lineare dei canali d'input. Andiamo quindi ad applicare all'immagine di input 64 kernels (filtri di convoluzione) 3x3 ottenendo un output di dimensioni pari a 99x99x64 e con una trasformazione di max pooling otteniamo 49x49x64. Aggiungendo altri 64 kernels e ripetendo tale operazione otteniamo una dimensionalità pari a 24x24x128, anche qui preleviamo l'output del layer precedente che sarà l'input del successivo, raddoppiamo i filtri, eseguiamo le convoluzioni ed eseguiamo il pooling ottenendo una dimensionalità 12x12x256 Nel passaggio successivo l'output avrà dimensionalità 6x6x512 e infine 3x3x512, ottenuta tale dimensionalità inizia la rete neurale profonda dove un primo layer Fully-connected e un layer dense, rispettivamente di 25088 e 4096 neuroni artificiali, avranno il compito di appiattire la dimensionalità dell' output al fine di ottenere non più una matrice bensì un vettore con 3 componenti.

In questo modo si è partiti da un'immagine di dimensionalità 99x99x1 e si è arrivati ad ottenere 3 valori che indicano le tre deformazioni.

I filtri di convoluzioni sono serviti a ridurre la dimensionalità dell'immagine riducendo sempre più il numero di pixels man mano che aumentavano i filtri stessi, i layer dense e fully-connected servivano invece per passare da un ordine di grandezza proprio di una matrice 3x3 all' ordine di grandezza di un vettore a 3 componenti.

Come si può notare oltre ai livelli di convoluzione e pooling ci sono anche altri layer nella parte più profonda della rete:

- flatten layer
- dense layer
- activation
- dropout

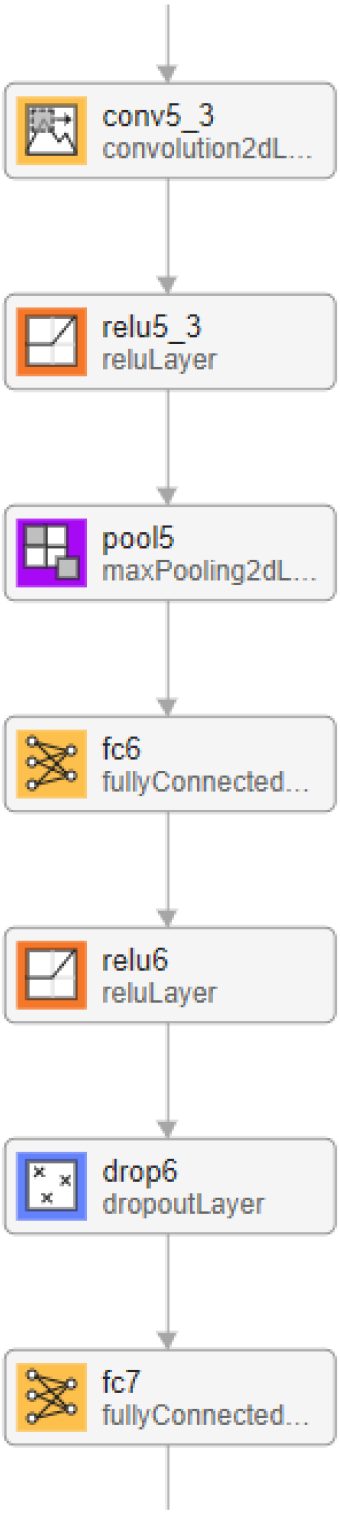

Figura 2.4: architettura parziale del modello VGG16

Layer (type)	Output Shape
input_2 (InputLayer)	[(None, 99, 99, 1)]
block1_conv1 (Conv2D)	(None, 99, 99, 64)
block1_conv2 (Conv2D)	(None, 99, 99, 64)
block1_pool (MaxPooling2D)	(None, 49, 49, 64)
block2_conv1 (Conv2D)	(None, 49, 49, 128)
block2_conv2 (Conv2D)	(None, 49, 49, 128)
block2_pool (MaxPooling2D)	(None, 24, 24, 128)
block3_conv1 (Conv2D)	(None, 24, 24, 256)
block3_conv2 (Conv2D)	(None, 24, 24, 256)
block3_conv3 (Conv2D)	(None, 24, 24, 256)
block3_pool (MaxPooling2D)	(None, 12, 12, 256)
block4_conv1 (Conv2D)	(None, 12, 12, 512)
block4_conv2 (Conv2D)	(None, 12, 12, 512)

Figura 2.5: convoluzioni per la riduzione della dimensionalità

```
block5_conv3 (Conv2D)       (None, 6, 6, 512)
block5_pool (MaxPooling2D)  (None, 3, 3, 512)
flatten_1 (Flatten)         (None, 4608)
dense_3 (Dense)             (None, 4096)
activation_2 (Activation)   (None, 4096)
dropout_2 (Dropout)         (None, 4096)
dense_4 (Dense)             (None, 4096)
activation_3 (Activation)   (None, 4096)
dropout_3 (Dropout)         (None, 4096)
dense_5 (Dense)             (None, 3)
```

Figura 2.6: passaggio da matrice a vettore

2.3.2 Flatten layer

Il layer viene inserito subito dopo l'ultimo MaxPooling, quindi dopo aver ridotto la dimensionalità dell'immagine e dopo aver aggiunto il massimo numero di filtri convoluzionali (nel esempio 512). Il suo scopo è quello appunto di appiattire cioé mettere tutti i dati in fila in un unico array monodimensionale, un vettore. Ovviamente questo passaggio di "appiattimento" avviene una volta sola. Sarà perciò sufficente aggiungere un solo flatten layer. Il flatten layer non appare inoltre nell'architettura della rete presente su MatLab ma è presente nel modello VGG16 usato

2.3.3 Dense layer

Per quanto riguarda il dense layer ne troviamo più di uno, dotati di 4096 neuroni fungono da intermediari dal flatten a quelli successivi che sono di vera e propria classificazione. Anche questo layer non è presente nella struttura del modello inserito in MatLab.

2.3.4 Activation layer

E' un layer non presente nella struttura del modello su MatLab dove invece a svolgere la funzione di attivazione troviamo il layer ReLu, la funzione è appunto la stessa: introdurre la non linearità, essenziale per la risoluzione dei problemi reali.

2.3.5 Dropout layer

Tale layer serve a migliorare la generalizzazione della rete. Durante l'addestramento il layer spegne causalmente un numero di connessioni pari alla percentuale associata facendo in modo che ci siano meno informazioni che raggiungono i layer successivi e questo stimola la rete ad adattarsi e a trovare correlazioni più robuste che superino la mancanza parziale di informazioni.

Una volta scelto e impostato il modello, possedendo i dataset suddivisi in ciascuna fase possiamo far partire l'addestramento della rete.

Ai fini di una maggiore comprensione dell'adeguatezza di questi strumenti nell' ambito di questo studio useremo reti diverse adatte a diverse dimensioni di input nominate: *not_trained_subim*99, *not_trained_subim*199 e *not_trained_subim*64 per immagini di dimensioni rispettivamente di 99,199 e 64 pixels. Lo scopo è quello di vedere quanto varia la precisione del modello al variare delle dimensioni di input, tali differenze verrano evidenziate nel terzo capitolo.

2.4 Training della rete

Nella fasse di training vengono dunque forniti i pattern del training set. Per pattern si intende un campione di dati con informazioni utili. I pattern si dividono in numerici, categorici e sequenziali. I numerici sono valori veri e propri, quelli categorici sono caratteristiche qualitative non esprimibili con i numeri a cui però vengono assegnati dei valori di riconiscimento, gli ultimi invece sono serie di informazioni come una sequenza video o audio in cui la posizione di un'informazione rappresenta essa stessa un' informazione.

Durante il training avviene la parte fondamentale del processo di apprendimento ovvero il *pattern recogniting*, il riconoscimento dei pattern che avviene seguendo tre tecniche:

- classificazione: ad ogni pattern in ingresso è assegnata una classe in base alle sue carattestiche alla fine la classe non è che un insieme di pattern con caratteristiche comuni.

- regressione: ad ogni pattern viene assegnato un valore continuo lo scopo è trovare una funzione che approssimi le coppie input/output fornite, usata per stime e previsioni.

- clustering: individuazione di cluster equivalenti alle classi dalle quali si differenziano per la capacità di trovare similitudini tra i pattern anche molto complesse. Per di più le classi sono note a priori e i cluster vengono formati durante il processo stesso

Come detto all inizio nella fase di addestramento vengono forniti i pattern così che il modello possa cominciare il processo di adattamento dei parametri con l'obiettivo di trovare la funzione otiimale nella lettura dei dati e nella generazione dei risultati. Ovviamente per ottenere risultati migliori sarebbe bene sottoporre il set di dati a più cicli di elaborazione; inoltre la potenza di calcolo è spesso insufficenete per elaborare tutto il set in una volta sola, allora si ricorre a piccoli accorgimenti che però migliorano la qualità del processo.

In primo luogo per evitare di sovraccaricare il modello si cerca di rendere più piccolo il dataset smembrandolo in sottogruppi uniformi detti *batch*, il numero di elementi contenuti in ogni batch è detto *batchsize*.

Si parla invece di *epoch* quando l'intero set di dati è stato sottoposto al modello. Dovendoci essere più cicli di elaborazione viene da sè che per completare un addestramento serviranno più *epochs*. Il termine iterazione invece decrive il numero di batch che servono a completare una epoch quindi facendo un esempio immaginiamo di avere 2000 dati da elaborare divisi in batch da 500 dati ciascuna dunque per completare una epoch serviranno 4 iterazioni.

Inutile dire che il numero di batch e la bacthsize nonché il numero di epochs influiscono direttamente sulla velocità di addestramento e sulla capacità di apprendimento.

Considerando il batchsize non possiamo prenderlo troppo piccolo in quanto ci saranno troppi pochi dati da elaborare. Questa problematica viene accentuta tanto più quando si usufrisce di processori SIMD (single instuction multiple data), che si distinguono per l'efficacia nell' elaborazione di grandi quantità di dati. Se invece prendiamo il batchsize troppo grande potremmo incappare in problemi di overfitting o di esaurimento della memoria (nel corso di questa sperimentazione è accaduto più di una volta). Bacthsize tipici sono 32, 64 e 128; quello usato nel corso di questa sperimentazione è il 64.

Si può inoltre salvare la rete allenata così da poterla ricaricare in un secondo momento e sottoporla ad ulteriori addestramenti.

Durante l'addestramento è possibile seguire l'andamento dell' apprendimento attraverso una trasposizione grafica di quello che sta succedendo, introduciamo quindi la $lossfunction$ e $l'accuracy$.

Per loss function o funzione di costo s'intende una funzione che ha lo scopo di valutare le prestazioni del modello. Qquesta funzione rappresenta una perdita quindi è facilmente intuibile che una buona prestazione del modello si verifica quando la perdita è minimizzata quindi quanto si minimizza la loss. La loss viene calcolata sui dati di training e validation, se i risultati non risultano equivalenti sui dataset di test abbiamo riscontrato il già citato problema dell'overfitting.

L'accuracy invece riguarda la fase in cui i parametri sono già stati appresi e si forniscono al modello i dataset di test. A questo punto vengono conteggiati i campioni correttamente elaborati nella fase di test cioé quei campioni in input a cui è stato associato dalla rete un output corretto. Il rapporto tra i campioni elaborati correttamente e tutti i campioni elaborati fornisce l'accuracy.

Nel caso della regressione viene poi inserito l'$RMSE$ (Root Mean Squared Error), che sarebbe una media degli scostamenti tra valore vero e predetto il tutto sotto radice. Anche di questo avremo un andamento grafico che ovviamente sarebbe bene tendesse a minimizzarsi.

La loss function e l'accuracy sono sostanzialmente degli strumenti che vengono usati per valutare la qualità della risposta del sistema ovvero di quanto si allontana il risultato ottenuto da quello desiderato. Tali strumenti vengono definiti metriche. Ne esistono diverse a seconda della natura del problema. In questo caso come già detto l'obiettivo è minimizzare la loss mentre l'accuracy deve crescere fino a tendere asintoticamente al valore 1. Per quanto riguarda l'RSME anche questo dovrà tendere asintoticamente allo 0.

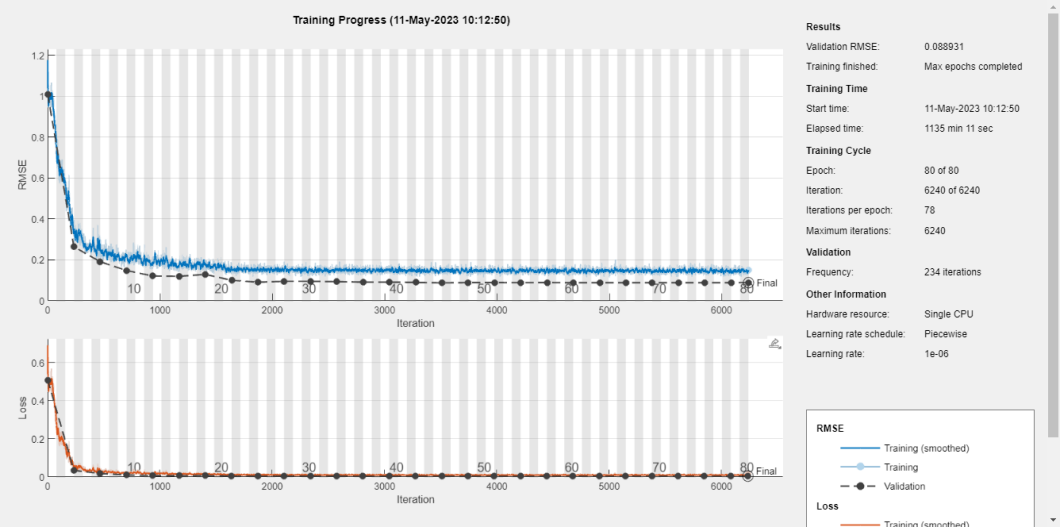

Figura 2.7: RMSE e loss

2.5 Fase di test

Nella successiva fase, quella di test, l'output viene predetto e non più fornito. La rete nella fase di test deve orientarsi solo in base all'input e alla funzione di apprendimento assimilata durante il training. Lo scopo è proprio quello di osservare come se la cava il modello in maniera autonoma senza quindi essere guidato verso un risultato prestabilito.

Il valore predetto, definito $Ypred$ serve anzitutto, paragonandolo al valore di output vero, a vedere se l' addestramento ha dato i suoi frutti e la rete è in grado di generalizzare su dati nuovi, in più serve anche nella ricerca del valore $Rsquared$ che, più avanti lo vedremo meglio, descrive la precisione dei risultati tramite valore numerico.

2.6 Rappresentazione grafica dei risultati

Quando la rete è stata addestrata ed la fase di test è stata svolta allora si "plottano" (mostrano graficamente) i risultati interpolandoli in una retta. I risultati in questione non sono altro che punti che vengono inseriti in un piano cartesiano in cui l'ascissa è data dal target (obiettivo di output) mentre l'ordinata è costituita dai valori predetti. La dispersione dei punti intorno alla retta ci suggerisce visivamente e in maniera molto veloce la precisione del modello: una concentrazione alta di punti vicino alla retta di riferimento indica un modello molto preciso, al contrario una dispersione accentuata è sinonimo di poca precisione. Specifichiamo però che la precisione del modello non dipende solo da come è strutturato quest'ultimo ma anche dalla complessità dei dati.

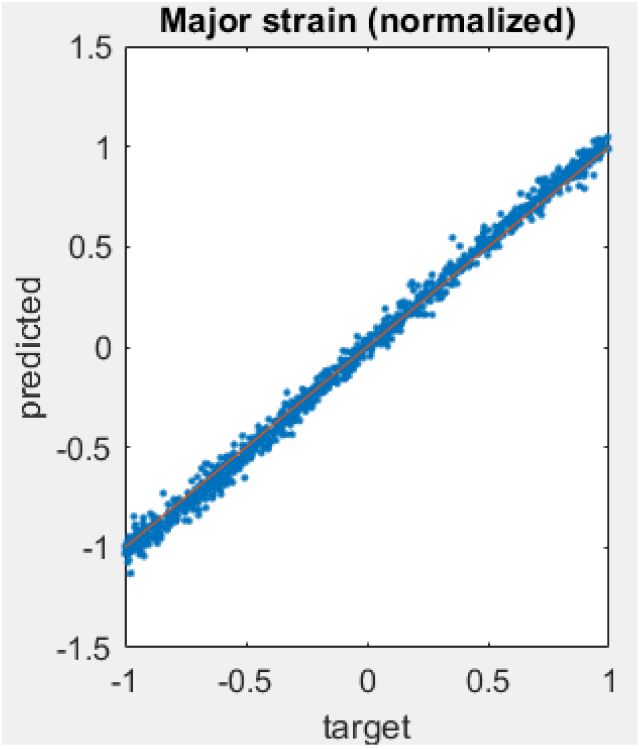

Figura 2.8: plot dei valori di deformazione e_1

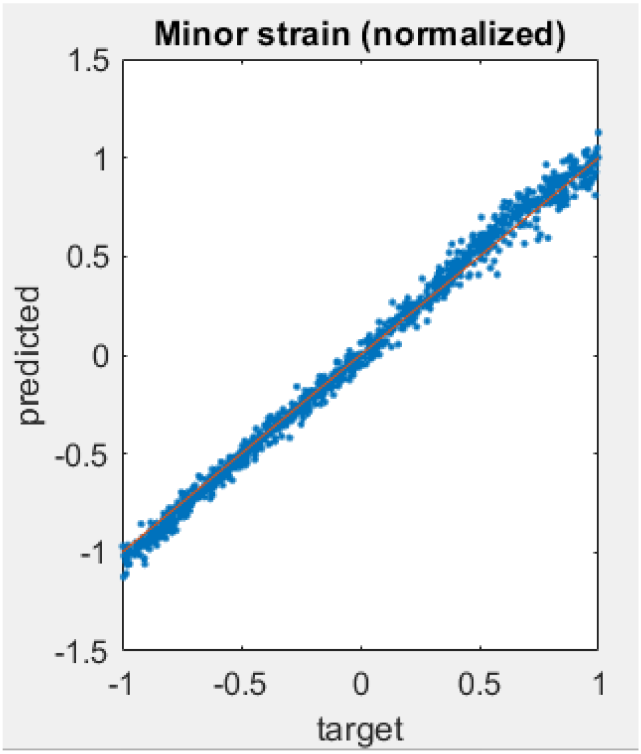

Figura 2.9: plot dei valori di deformazione e_2

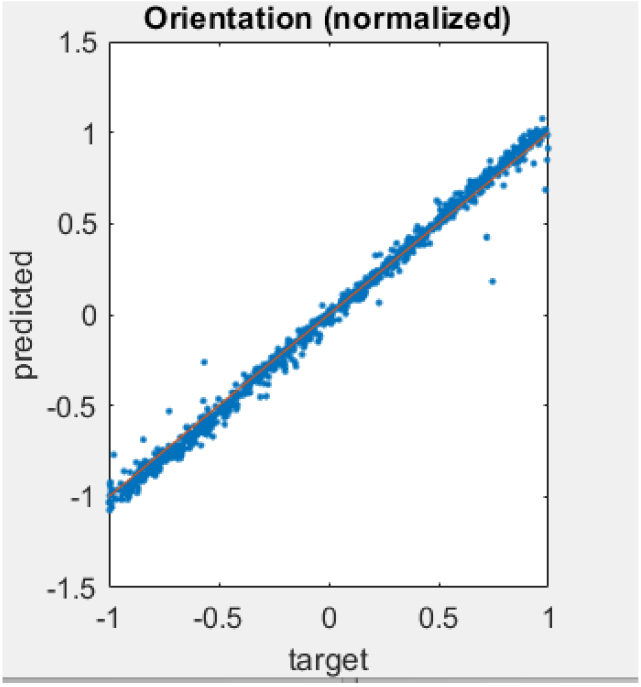

Figura 2.10: plot dei valori di deformazione di orientazione

Ovviamente vengono plottati tutti e tre i valori di deformazione che costituiscono l'output, quindi lunghezza e_1, larghezza e_2 e orientazione che vengono definite rispettivamente *majorstrain*, *minorstrain* e *orientation*.

I risultati in figura sono ottenuti da sperimentazione dove sono state utilizzate come input immagini generate virtualemente, costituite da uno sfondo bianco con presenti dei cerchi neri con bordo preciso. Tali immagini sono proprio per questo motivo molto ordinate anche dopo aver subito la deformazione e non caotiche come potrebbero essere invece delle immagini reali. Per questo motivo nelle figure possiamo notare un' elevata precisione dei risultati grafici: la dispersione dei punti è ridotta e questi sono concentrati tutti nelle prossimità della retta di riferimento.

Per retta di riferimento si intende la retta in un certo senso ottimale ovvero quella ottenuta nel caso in cui la l'output predetto coincida sempre con l'output vero e proprio.

I grafici sono quindi la sovrapposizione, per le tre deformazioni, dei valori predetti con i valori fissati.

2.7 Valutazione numerica della precisione del modello

Oltre ad una rappresentazione grafica dei risultati cerchiamo anche di ottenere una valutazione numerica della precisione del modello. Tale valuta-

zione si esprime con la grandezza $R - Squared$ definita come coefficente di determinazione.

"R-squared (R2) è una misura statistica che rappresenta la proporzione della varianza per una variabile dipendente che viene spiegata da una o più variabili indipendenti in un modello di regressione. " [5] ..., in altre parole indica la misura in cui la varianza di una variabile spiega la varianza della variabile seguente.

Ricordiamo che la varianza è una misura statistica che indica la distanza di un insieme di punti dal loro valore medio. Entriamo però più nel dettaglio e vediamo cos'è, quando si usa e come si calcola.

Lo scopo ultimo del coefficente di determinazione è quello di valutare se un modello di regressione è adatto a svolgere previsioni, quindi se è abbastanza preciso. La precisione dipende da quanto esattamente le sue variabili indipendenti riescano a predire le variabili dipendenti.

Per la valutazione della precisione ci si affida alla valutazione dei cosidetti indici della bontà di adattamento. Il compito di questi indici è quello di stimare la differenza tra i valori dati di output e quelli predetti dal modello. Naturalmente se le differenze in questione sono troppo grandi ci troviamo di fronte ad modello con una scarsa capacità previsionale. Il coefficente di determinazione è l'indice di bontà maggiormente utilizzato sopratutto nei modelli di regressione lineare, unico campo in cui veniva sfruttato fino a qualche anno fa. Ora invece sta trovando applicazione anche nei modelli non lineari.

Ma come dobbiamo interpretarlo?

La corretta interpretazione di questo indice è essenziale e parte addirittura dal simbolo stesso: la R maiuscola. Questa indica che ci troviamo in presenza di una regressione lineare multipla con più di una variabile indipendente. La r minuscola invece indica una regressione semplice con una sola variabile indipendente. R-Squared (R2) sarebbe il quadrato di R che è il coefficente di correlazione multipla, mentre r2 è il quadrato del coefficentedi correlazione bivariato r.

Come già detto R2 indica quanto forte è la relazione tra le variabili indipendenti e quella dipendente, non ne specifica però la positività o negatività.

Essendo una proporzione R2 sarà sempre:

$$0 < R2 < 1 \tag{2.4}$$

Facendo un esempio se R2 tra X e Y è pari a:

$$R2 = 0.39 \tag{2.5}$$

significa che il 39% della varianza di X e Y è condivisa e rappresenta un'intersezione delle varianze.

Figura 2.11: Intersezione varianze

Se R2 è pari a 0 ci troviamo in caso in cui la variabilità dell'output intorno alla sua media non è per nulla spiegata da alcuna variabile predittiva. Se al contrario la R2 è uguale a 1 allora avremmo un modello in cui le variabili predittive spiegano al 100% la variabilità dell'output attorno alla sua media. Se vogliamo paragonare tale situazione al risultato grafico avremmo una dispersione di punti nulla, questi infatti si disporranno tutti sulla retta di riferimento citata prima o retta di regressione dove i valori stimati coincidono con quelli veri. Dunque in linea di massima più è alto il valore di R2 migliore sarà la capacità predittiva del modello perchè i risultati stimati si discosteranno poco da quelli noti [6].

Bisogna precisare che valori di R2 bassi non significano esclusivamente l'aver sbagliato la costruzione del modello, magari la variabile sotto analisi dipende da tantissimi fattori diversi, quindi la capacità predittiva è limitata. Se però l'obiettivo è proprio una capacità predittiva buona allora è condizione necessaria un R2 elevato.

Come si calcola la R-Squared?

Ci sono alcuni passaggi da fare ma prima andiamo a spiegare il significato di ogni termine in figura:

- y_bar sarebbe la media delle $Ypred$ ovvero di tutti i valori di output predetti dal modello
- SSres viene definita devianza ed è la sommatoria dei quadrati della differenza tra i valori predetti e quelli reali (questa differenza si dice residuo, quiesto spiega il res come pedice)

$$SSres = \sum((Ypred - Ytest)^2) \tag{2.6}$$

La somma dei quadrati dei residui (SSres) aiuta a capire la dispersione dei punti perchè appunto misura la varianza nel termine del errore.

- SStot è invece la somma totale dei quadrati data dalla sommatoria dei quadrati della differenza tra i valori predetti (Ypred) e la media degli stessi.

$$SStot = \sum((Ypred - y\_bar)^2) \tag{2.7}$$

Una volta calcolata la media y_bar e le sommatorie SSres e SStot possiamo facilmente ricavare la nostra R-Squared:

$$R - Squared = 1 - (SSres/SStot) \tag{2.8}$$

si potrebbe anche utilizzare la formula:

$$R - Squared = SSexp/SStot \tag{2.9}$$

dove la SSexp sta per somma spiegata dei quadrati e sarebbe nient'altro che la sommatoria dei quadrati della differenza tra i valori di output reali Ytest e la media di quelli predetti y_bar:

$$SSexp = \sum((Ytest - y\_bar)^2) \tag{2.10}$$

quest'ultima formula risulta più diretta tuttavia si preferisce usare l'altra, definita infatti formula corretta. Il probblema con R2 è che questo valore aumenta con l'aggiunta di variabili indipendenti pur non essendo queste ultime esplicative. Questo problema si evita con l'utilizzo della formula corretta con la quale l'aggiunta di una variabile non comporta un aumento numerico di R2 quando non c'è un effettivo miglioramento del modello di regressione.

Per concludere il modello di rete di cui ci serviamo in questa sperimentazione è il VGG16, una rete neurale convoluzionale costituita da 16 livelli di profondità, tale rete viene addestrata con dei dataset generati a partire da immagini deformate di metallografie con subsize modificabile, l'output ricercato è il valore di deformazione (3 valori in realtà) della microstruttura, attraverso il plottaggio dei risultati e il calcolo del coefficente di determinazione possiamo farci un'idea del livello di accuratezza e precisione predittiva del modello sia da un punto di vista grafico sia numerico.

Nel seguente capitolo riporteremo i risultati ottenuti nelle simulazioni, inoltre li confronteremo così da vedere quali sono le condizioni in cui il modello ha prestazioni migliori e quelle in cui invece la capacità predittiva è minore.

Capitolo 3

Risultati ottenuti

3.1 Risultati simulazione n°1

La prima simulazione riguarda delle immagini generate da una metallografia caotica o se vogliamo poco ordinata. Tali immagini hanno una subsize pari a 99 pixels dunque la rete addestrata con queste ultime dovrà essere adatta a ricevere un input di dimensione 99x99 pixels. Come detto in precedenza il sistema ci forniva due reti non allenate, ognuna delle due adatte per l'analisi di immagini di dimensioni differenti, quella usata in questa prima simulazione è la *not_trained_subim*99, ovvero quella adatta per immagini 99x99.

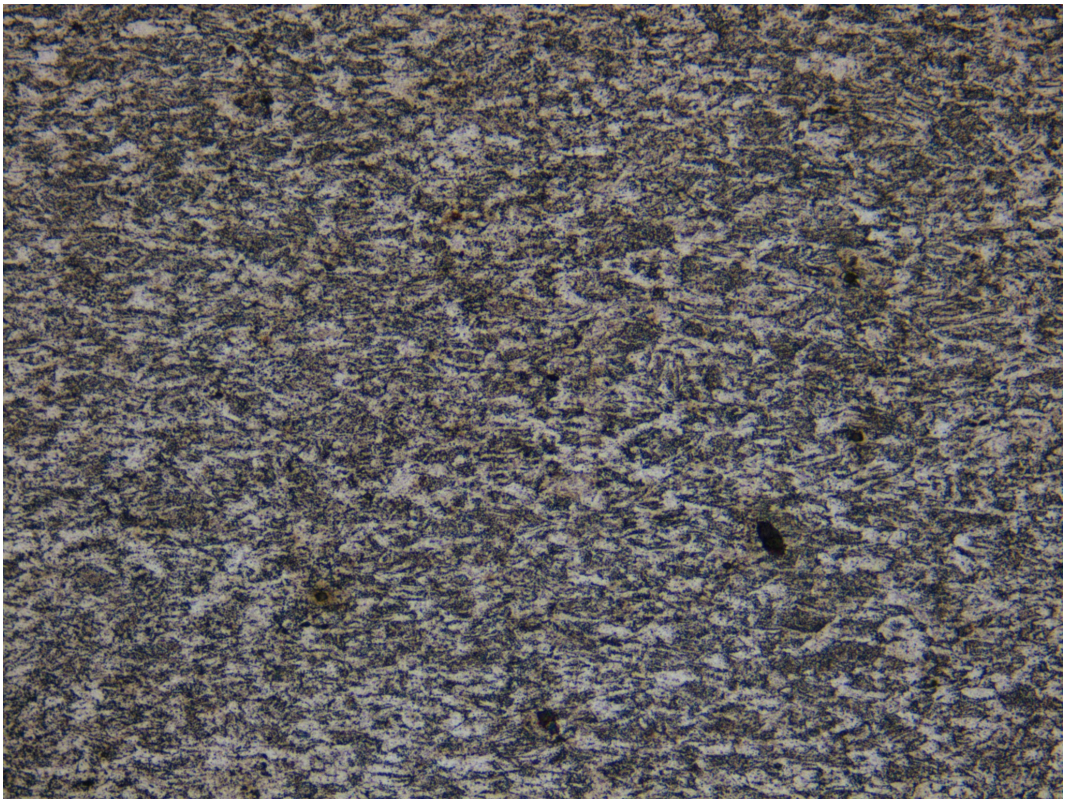

Figura 3.1: immagine della microstruttura di partenza

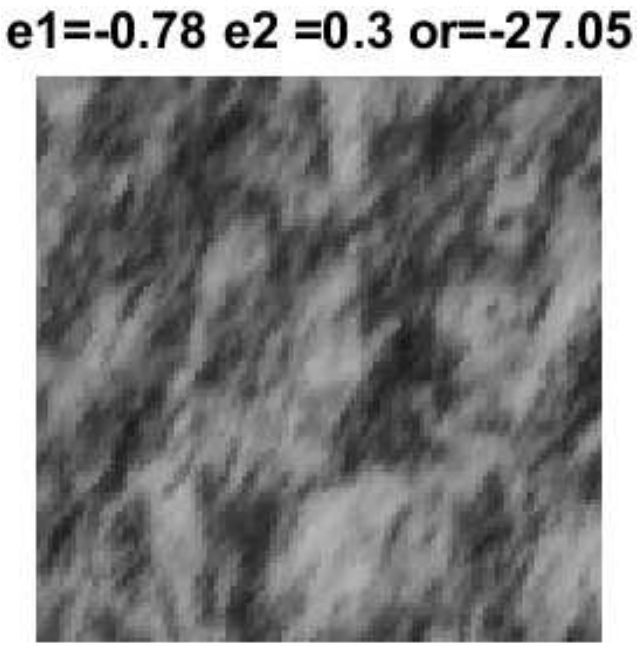

Figura 3.2: esempio di immagine deformata

Le immagini di input generate vengono deformate con dei valori casuali da ricercarsi all'interno dei seguenti intervalli:

Tabella 3.1: intervalli di deformazione

def	min	max
e_1	-0.8	0.0
e_2	0.0	1.5
or	$-\pi/5$	$\pi/5$

dove e_1 è la deformazione nella direzione principale 1, che possiamo intendere come deformazione in lunghezza, e_2 è la deformazione lungo la direzione principale 2 interpretabile come larghezza, per ultimo *or* che sta per orientazione.

Caricato il modello ed inseriti i dataset di training e validation inizia l'addestramento costituito da 80 epoche e 78 iterazioni (mediamente) per epoca. Ciò significa che ci sono 78 batch, ovvero 78 sottogruppi del dataset iniziale ognuno con dimensione 64 (ricordiamo che la dimensione indica il numero di elementi contenuti nella batch stessa), con un rapido calcolo vediamo che le iterazioni totali sono 6240, una volta compiute l'addestramento si può ritenere concluso.

A questo punto vediamo graficamente l'andamento della loss e del RMSE.

come si può vedere la RMSE tende a diminuire entro le prime 20 epoche dopo le quali si stabilizza tendendo asintoticamente al valore 0.2, la minimizzazione limitata è dovuta al fatto che la stessa immagine usata come input è un immagine disordinata, difficile da analizzare.

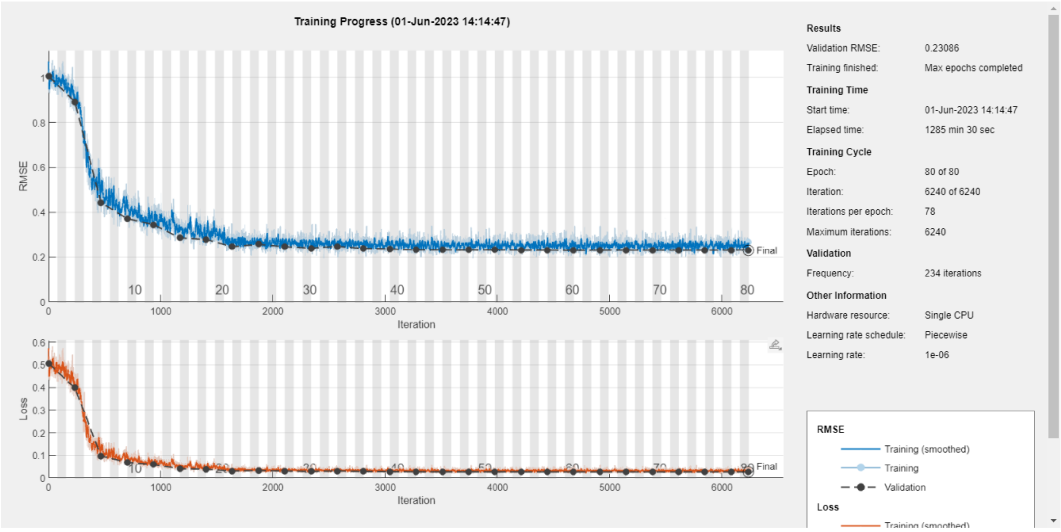

Figura 3.3: loss e RMSE per la 1ˆ simulazione

Per quanto riguarda la dispersione dei risultati, da un punto di vista grafico:

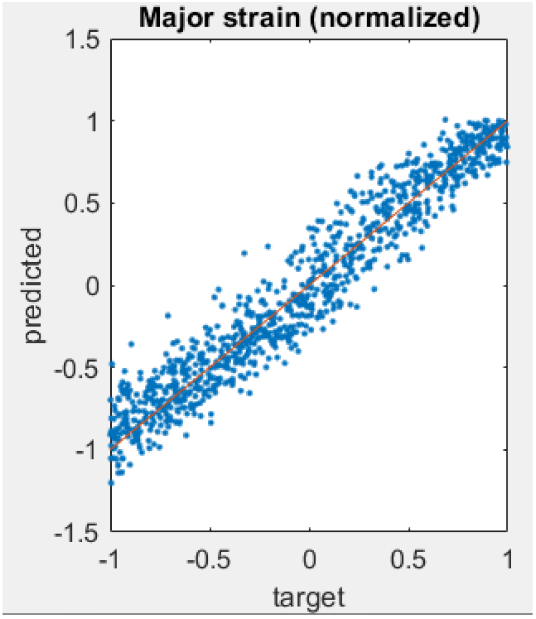

Figura 3.4: deformazione e_1

Già ad un primo sguardo possiamo capire come i risultati sopratutto per quanto riguarda i valori della e_1 non siano molto accurati: la dispersione è visibilmente notevole, la predizione più accurata del modello è quella dei valori di deformazione e_2 ma anche qui non raggiungiamo pienamente la precisione desiderata.

Per quanto riguarda l'accuratezza analizzata numericamente dal coefficente di determinazione R2:

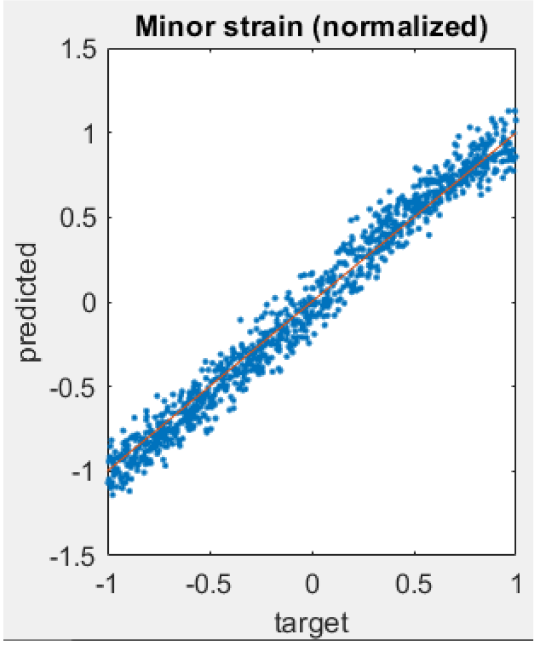

Figura 3.5: deformazione e_2

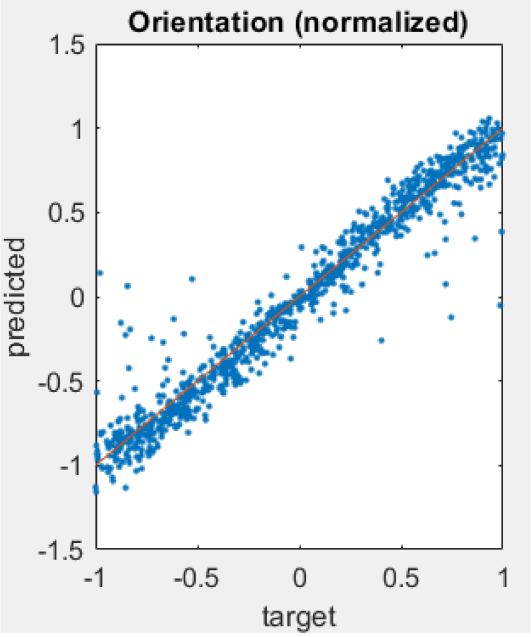

Figura 3.6: orientation

Tabella 3.2: valori di R2 per le tre deformazioni

def	R2
e_1	0.933
e_2	0.97
or	0.94

E' interessante notare come i valori numerici rispecchiano i risultati grafici, il modello infatti predice con scarsa accuratezza la major strain (e_1) a cui è associato il valore R2 più basso (0.933), mentre risulta più accurato, anche se non di molto, nella predizione della minor strain (R2=0.97).

3.2 Risultati simulazione n°2

Per la seconda simulazione abbiamo utilizzato una metallografia più ordinata, mentre gli intervalli di deformazione usati per modificare virtualmente l'immagine erano gli stessi usati nella prima, anche le dimensioni dell'immagine coincidevano (sempre 99x99), di conseguenza la rete non allenata da cui si partiva era anch'essa la medesima, stesso numero di epoche, iterazioni per epoca e totali e stesso numero di batch con uguale batchsize.

Per riepilogare:

- numero di immagini generate: 7000
- dataset invariati training:5000; validation:1000; test:1000
- subsize dell'immagine : 99
- rete caricata : $not\_trained\_subim99$
- epoche : 80
- batch e iterazioni : 78
- batchsize : 64

In definitiva l'unica cosa che cambiava era il tipo di metallografia usata che in questo caso era meno caotica. Ovviamente il maggiore ordine della metallografia è relativo non assoluto. Nella realtà non esistono metallografie perfettamente ordinate, la microstruttura dell'immagine è però più ordinata di quella dell'immagine usata nella 1^ simulazione, tanto basta per confrontarle e vedere come reagisce il modello in termini di miglioramento della qualità.

Andiamo dunque a vedere le differenza per quanto riguarda l'andamento della loss e del RMSE:

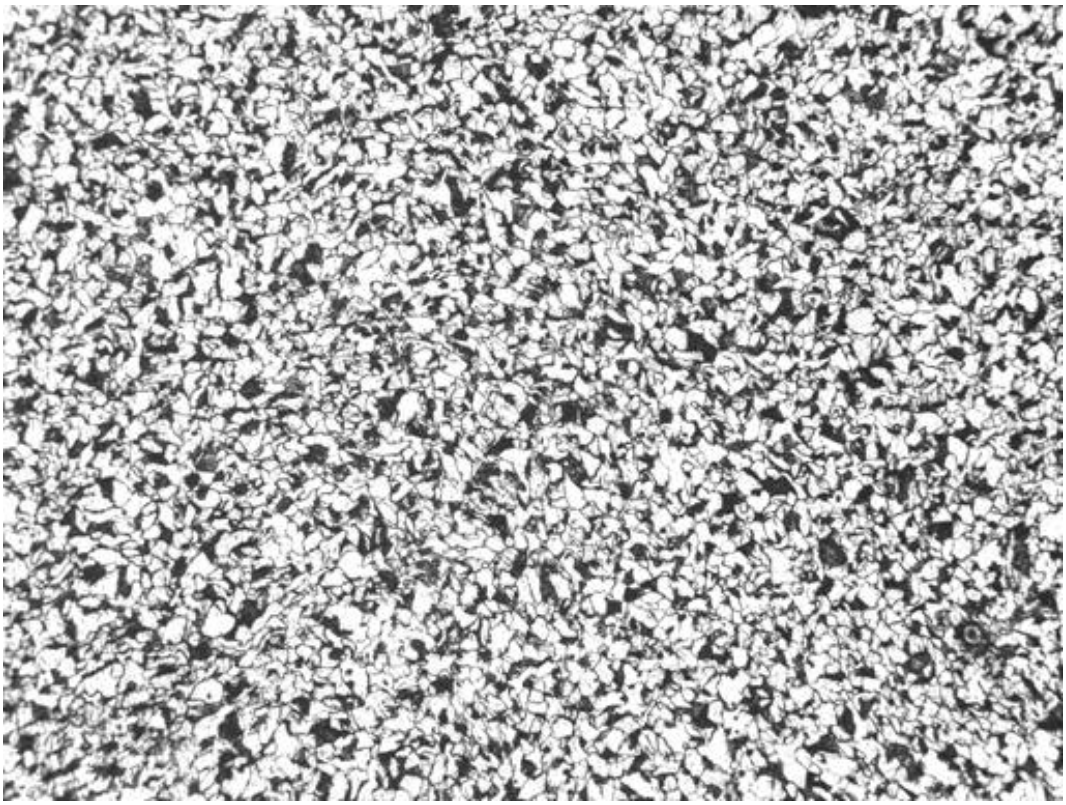

Figura 3.7: metallografia ordinata di partenza

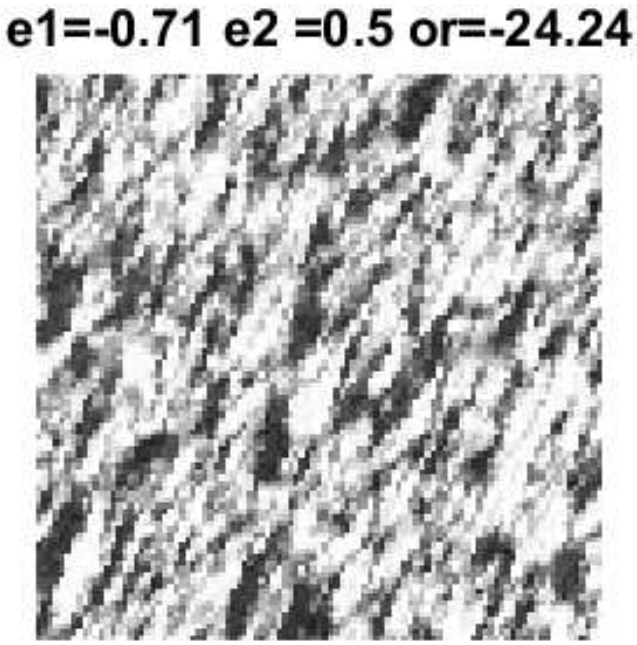

Figura 3.8: esempio di immagine deformata a partire da metallografia ordinata

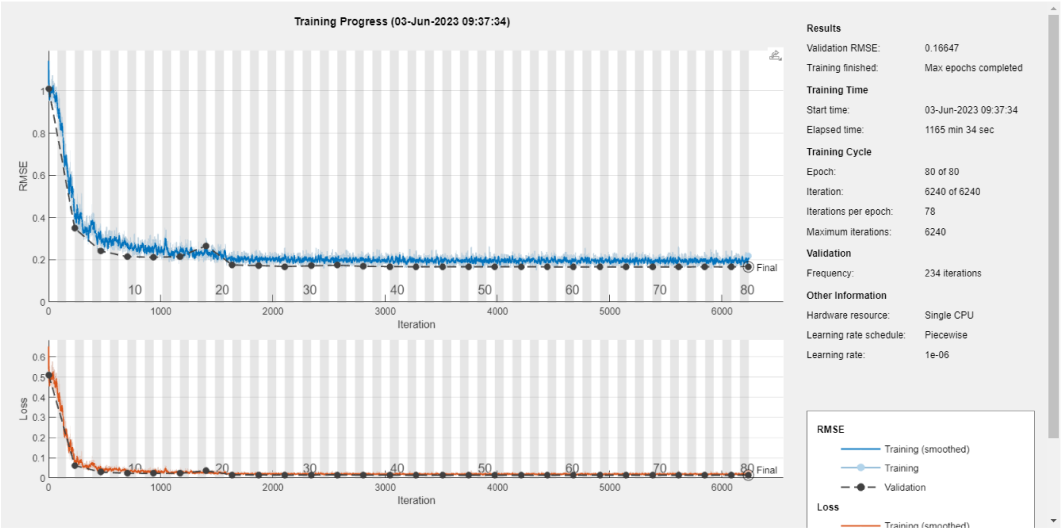

Figura 3.9: loss e RMSE per la 2ˆ simulazione

Come si può vedere da un rapido confronto l'RMSE tende a minimizzare maggiormente, arriva infatti a toccare valori al di sotto dello 0,2 a cui tendeva asintoticamente nella 1ˆ simulazione. Anche in termini di tempo c'è un piccolo miglioramento, l'addestramento svolto a partire da una metallografia ordinata ha richiesto infatti circa 100 minuti in meno, poco più di un'ora, per giungere al termine. Stiamo parlando di simulazioni che possono durare 24h anche di più nel caso siano molto pesanti, guadagnare 1h non è così significativo ma è comunque degno di nota.

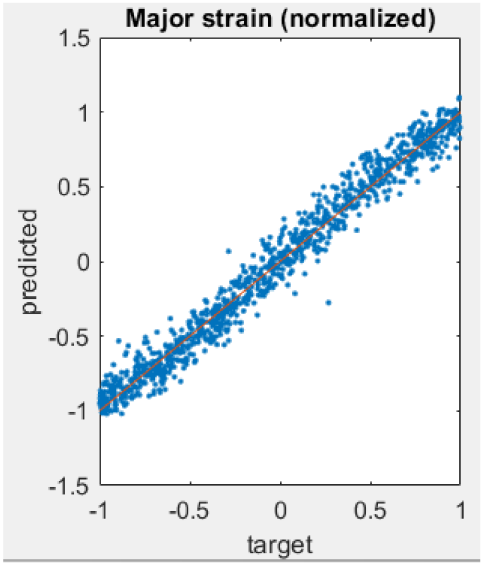

Figura 3.10: deformazione e_1 della 2ˆ simulazione

La differenza a livello di dispersione dei risultati è evidente: nella metallografia ordinata la precisione predittiva del modello è nettamente maggiore, risultato

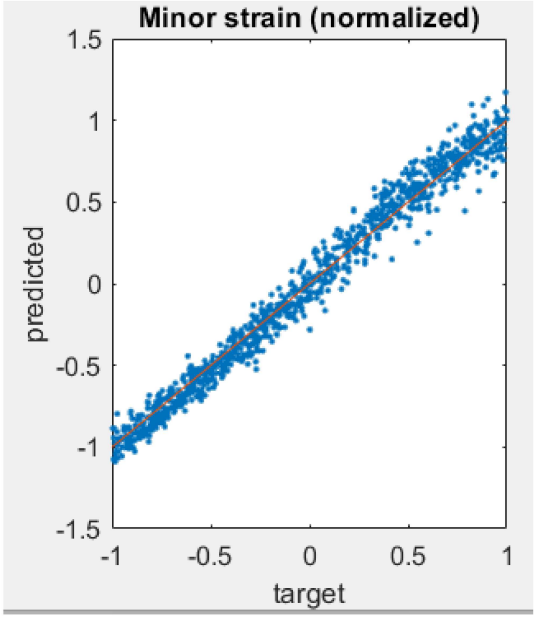

Figura 3.11: deformazione e_2 della 2ˆ simulazione

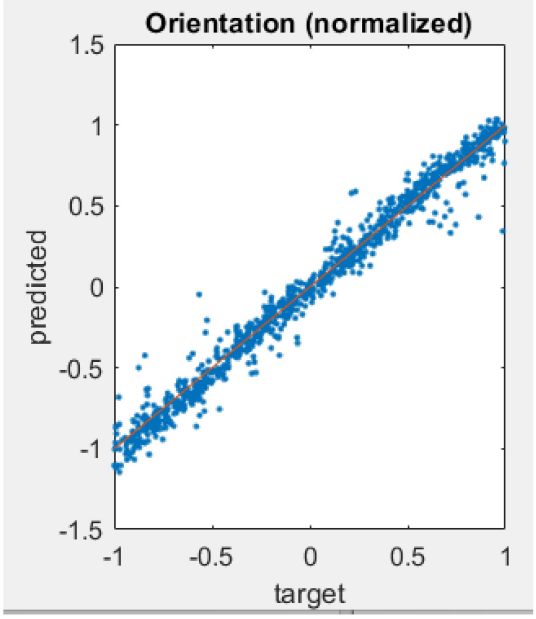

Figura 3.12: orientation della 2ˆ simulazione

che possiamo riscontrare anche numericamente:

Tabella 3.3: valori di R2 per le tre deformazioni della 2^ simulazione

def	R2
e_1	0.972
e_2	0.978
or	0.978

La differenza non è sostanziale per quanto riguarda e_2, nella seconda simulazione possiamo registrare solo un piccolo aumento dell'accuratezza predittiva del modello, mentre per le altre due deformazioni la precisione è sensibilmente migliorata.

Tale miglioramento è da ricercare nel fatto che il modello è avvantaggiato nell'elaborazione di realtà meno complesse, in questo caso un'immagine più ordinata, questo gli permette di svolgere la sua funzione predittiva con minore difficoltà e quindi maggiore precisione.

3.3 Risultati simulazione n°3

Nella terza simulazione siamo tornati ad usare, come immagine preliminare, la microstruttura disordinata usata nella prima simulazione e l'intervallo di valori di deformazione rimane lo stesso:

Tabella 3.4: intervalli di deformazione

def	min	max
e_1	-0.8	0.0
e_2	0.0	1.5
or	$-\pi/5$	$\pi/5$

restano invariate anche le seguenti impostazioni di rete:

- subsize dell'immagine : 99
- rete caricata : $not\_trained\_subim99$
- epoche : 80
- bacthsize: 64

Ciò che varia è invece il numero di campioni. Mentre nelle simulazioni prima il numero di campioni era 7000 in questa 3^ simulazione il numero di campioni è 3500 ripartiti nel seguente modo:

- numero di elementi per il training set: 2500
- numero di campioni per la validation: 500
- numero di campioni per il test: 500

Essendo il numero di elementi inferiore, ma mantenendo il batchsize sempre pari a 64 il numero di batch e di conseguenza di iterazioni è 39, con un totale di 3120 iterazioni.

Finito l'addestramento della rete studiamo la minimizzazione della *loss* e del $RMSE$.

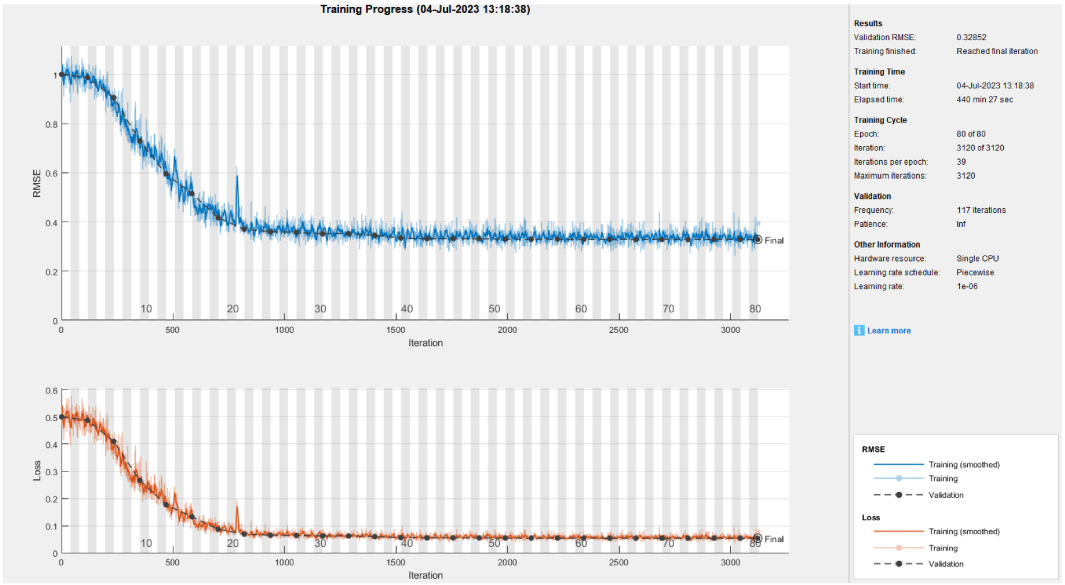

Figura 3.13: loss e RMSE per la 3ˆ simulazione

Possiamo notare come la loss ha un andamento che tende a minimizzare sempre abbastanza, mentre la RMSE tende asintoticamente ad un valore superiore a 0,2 (riscontrato nella 1ˆ simulazione), per la precisione tende a 0,32. Quindi con un minor numero di dati la minimizzazione del RMSE è minore.

Naturalmente anche il tempo impiegato dalla simulazione è minore, essendoci meno dati da elaborare è ovvio che l'addestramento sia più veloce.

Per quanto riguarda l'accuratezza predittiva del modello sarà anche questa minore poiché i dati su cui si è allenata la rete erano in quantità limitata.

Tabella 3.5: valori di R2 per le tre deformazioni nella 3ˆ simulazione

def	R2
e_1	0.902
e_2	0.943
or	0.861

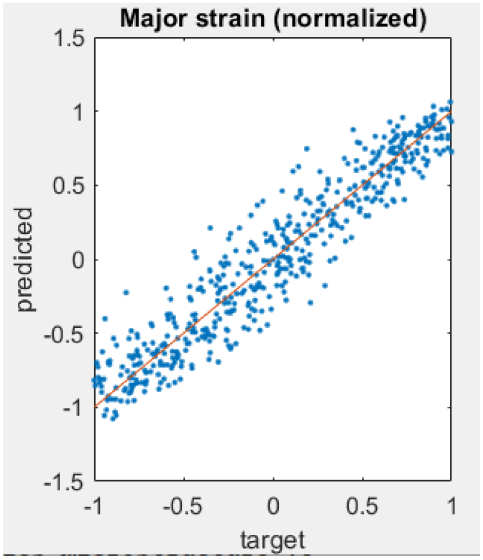

Figura 3.14: deformazione e_1 della 3ˆ simulazione

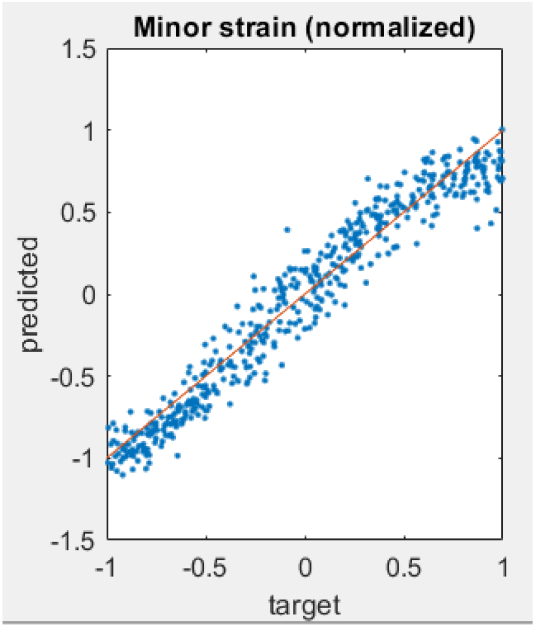

Figura 3.15: deformazione e_2 della 3ˆ simulazione

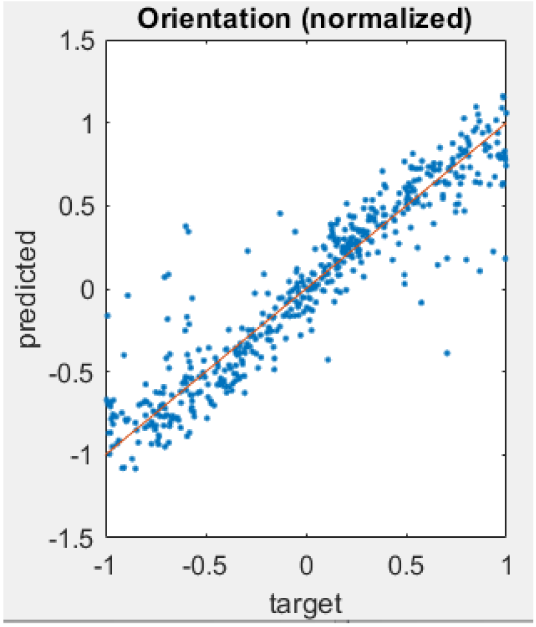

Figura 3.16: orientation della 3^ simulazione

Dalla lettura dei grafici è facilmente intuibile la perdita di capacità predittiva del modello che risente di un allenamento eseguito su pochi dati in input.

L'uso di un minor numero di dati ci consente di svolgere l'addestramento più velocemente e su una realtà meno complessa a discapito però della precisione del modello nel fornire i dati di output. La scarsa accuratezza è evidente e riscontrabile anche nei valori tabellati. In coclusione bisogna dosare bene l'equilibrio tra complessità dei dati da elaborare e accuratezza predittiva del modello.

3.4 Risultati simulazione n°4

In questa simulazione abbiamo riutilizzato l'immagine con microstruttura disordinata, per quanto riguarda i parametri li riassumeremo tutti nel seguente elenco:

- numero di campioni: 7000
- numero di campioni per training: 5000
- numero di campioni per la validation: 1000
- numero di campioni per il test: 1000
- batch e iterazioni: 78
- iterazioni totali: 6240

- batchsize: 64
- epoche: 80
- dimensioni dell' immagine : 64x64x1 (immagini da 64 pixels)
- rete usata: *not_trained_subim*64 (adatte per analisi di input con dimensione 64x64)
- intervallo di deformazione:

Tabella 3.6: intervalli di deformazione

def	min	max
e_1	-0.8	0.0
e_2	0.0	1.5
or	$-\pi/5$	$\pi/5$

Come si può notare la differenza rispetto alla 1^ simulazione sta solo nella dimensione ridotta del input (da 99 a 64), vediamo ora cosa comporta tale differenza.

La minimizzazione della loss è sempre elevata a differenza della minimizzazione del RMSE che non arriva mai a toccare lo 0,2 ma si ferma a valori più simili a quelli del RMSE riscontrabile nella 3^ simulazione (con 3500 campioni invece di 7000). La limitata "discesa" del RMSE è con tutta probabilità dovuta al fatto che le immagini avendo dimensioni ridotte tendono a far risaltare uno stato di maggiore disordine rispetto ad campioni analizzati in scala maggiore.

Spostando l'attenzione sulla capacità predittiva, numericamente otteniamo una dispersione minore di quella ottenuta analizzando 3500 immagini (simulazione n°3) ma comunque maggiore di quella ottenuta analizzando 7000 immagini con dimensione 99x99 (simulazione n°1), graficamente confermiamo tale responso. In conclusione possiamo dire che aver ridotto la dimensione del input ha sortito effetti molto simili (anche se leggermente migliori) alla riduzione del numero di campioni stessi, ovvero un peggioramento della preditivittà del modello.

Tabella 3.7: valori di R2 per le tre deformazioni della 4^ simulazione

def	R2
e_1	0.909
e_2	0.959
or	0.926

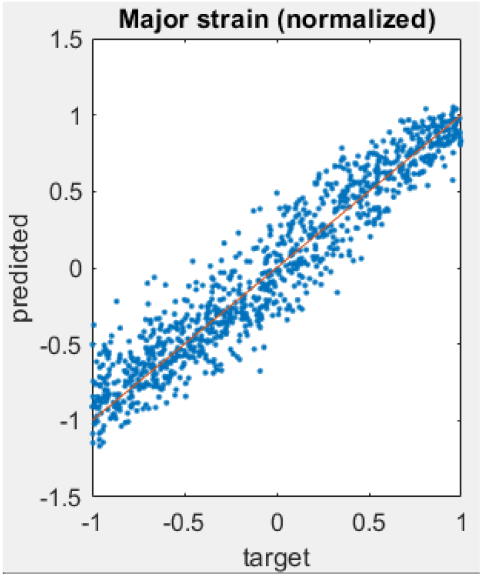

Figura 3.17: deformazione e_1 della 4^ simulazione

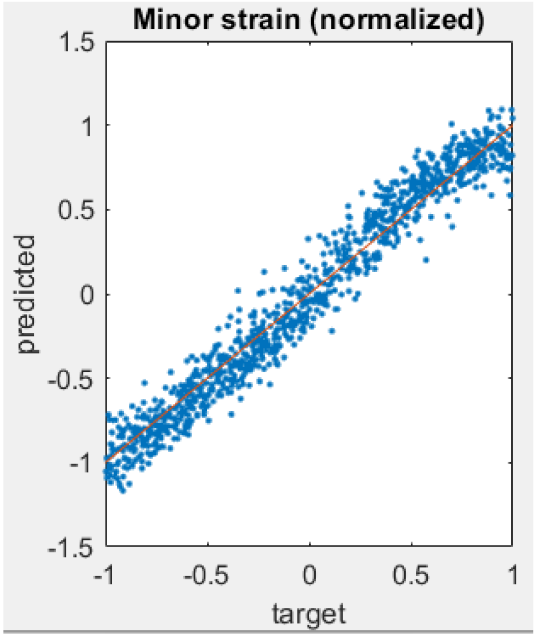

Figura 3.18: deformazione e_2 della 4^ simulazione

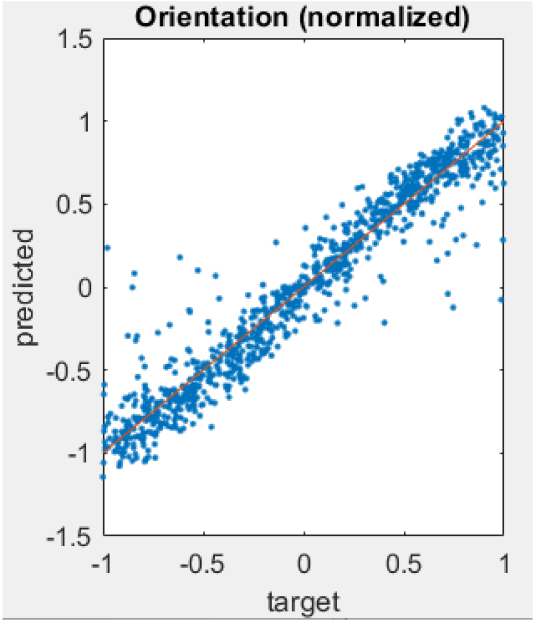

Figura 3.19: orientation della 4ˆ simulazione

3.5 Risultati simulazione n°5

In questa simulazione abbiamo abbinato le modifiche applicate nella terza e nalla quarta simulazione, ovvero abbiamo preso 3500 campioni con dimensioni pari a 64x64 pixels, il resto dei parametri è rimasto invariato.

I risultati grafici e numerici hanno evidenziato una combinazione degli effetti peggiorativi registrati nelle simulazioni (3 e 4), in cui i due parametri modificati (numero di campioni analizzati e dimensione dei dati di input) erano considerati singolarmente, tale combinazione di effetti ha portato ad un crollo delle prestazioni del modello.

Tabella 3.8: valori di R2 per le tre deformazioni della 5ˆ simulazione

def	R2
e_1	0.855
e_2	0.894
or	0.828

Mentre nelle precedenti due simulazione si registra una perdita di accuratezza del 10% circa relativamente alla deformazione e_1, nella 5ˆ tale perdita raggiunge il 15%. Una diminuzione del 5% di accuratezza si riscontra anche nel campo della deformazione e_2, l'orientazione resta invece la predizione meno accurata che tocca il suo massimo in termini di perdita ovvero il 18%.

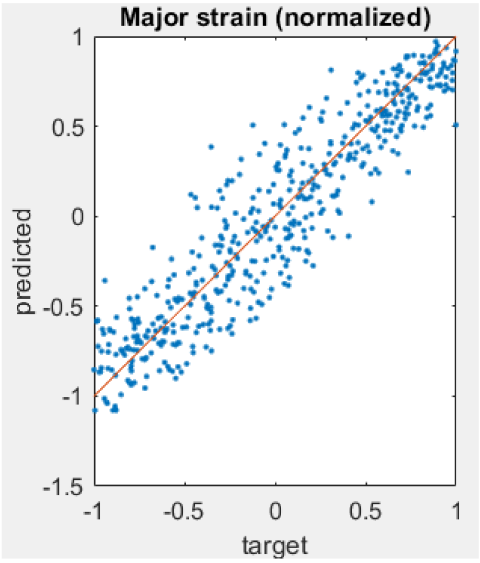

Figura 3.20: deformazione e_1 della 5^ simulazione

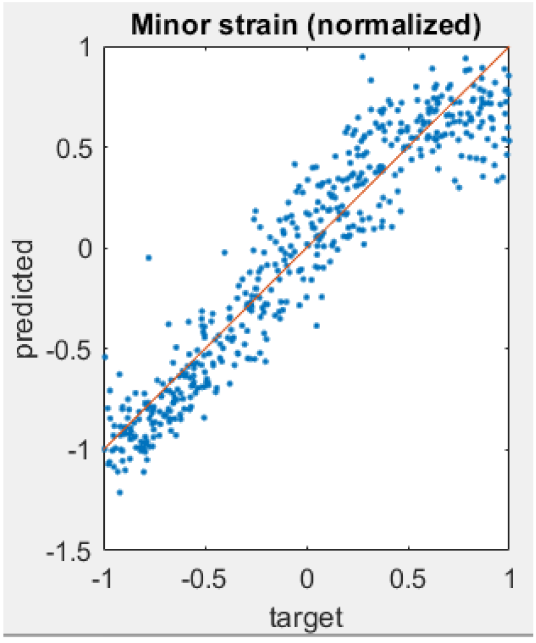

Figura 3.21: deformazione e_2 della 5^ simulazione

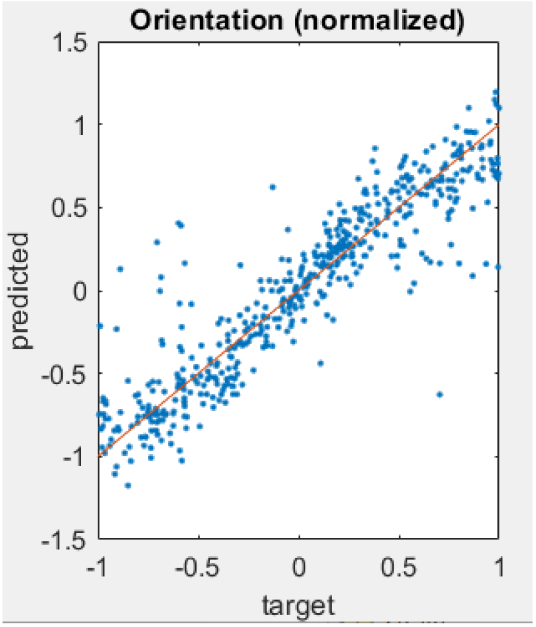

Figura 3.22: orientation della 5ˆ simulazione

3.6 Risultati simulazione n°6

L'ultima simulazione è stata quella maggiormente complessa, non tanto per il numero di campioni elaborato (sempre 7000 con medesima ripartizione nei tre dataset), quanto per le loro dimensioni. Non abbiamo infatti più usato immagini con dimensionamento 99x99 ma 199x199, la complessità dell'elaborazione è cresciuta notevolmente basti pensare che l'addestramento della rete è durato più di 60 ore, più di qualunque altra simulazione fatta in precedenza. Per il resto i parametri sono rimasti invariati e la metallografia usata è quella disordinata.

Analizzando i risultati possiamo notare un leggero miglioramento rispetto alla 1ˆ simulazione, da cui differisce solo per il dimensionamento delle immagini (in questa simulazione maggiorato di 100 pixels), confermando l'ipotesi che usando dimensioni di input più grandi aumenta sì la complessità dell'elaborazione ma anche la precisione dei risultati. La dispersione è infatti diminuita anche se di poco, dai risultati grafici tale miglioramento è appena visibile, riusciamo a notarlo solo analizzando i valori di R2 per le tre deformazioni:

Tabella 3.9: valori di R2 per le tre deformazioni della 6ˆ simulazione

def	R2
e_1	0.951
e_2	0.981
or	0.966

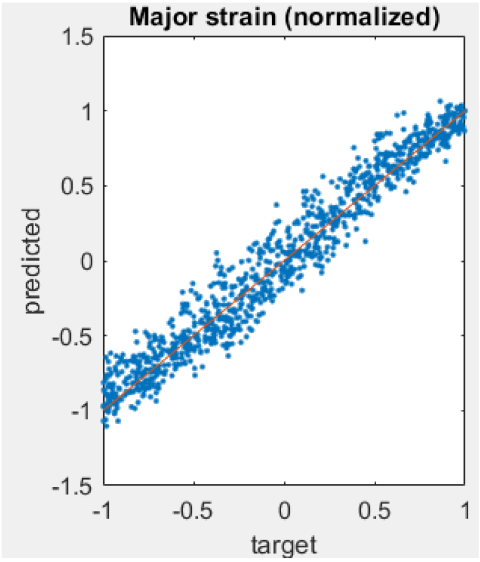

Figura 3.23: deformazione e_1 della 6ˆ simulazione

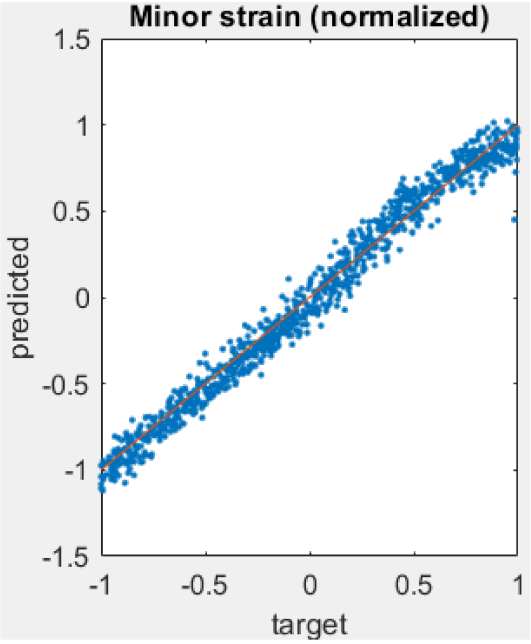

Figura 3.24: deformazione e_2 della 6ˆ simulazione

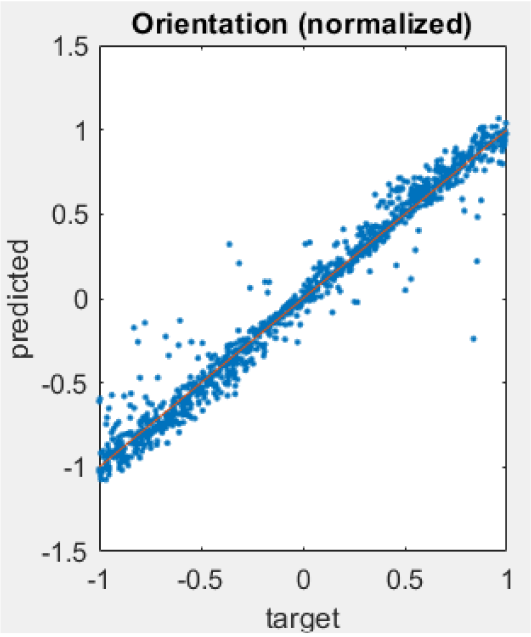

Figura 3.25: orientation della 6^ simulazione

In conclusione abbiamo sperimentato come la variazione della quantità di dati e le dimensioni di questi ultimi influenzino la capacità predittiva del modello. In particolar modo un aumento delle dimensioni o dei campioni stessi comporta un aumento dell'accuratezza del modello mentre una riduzione porta ad una perdita di precisione sensibile. Le tecniche di machine learning nello studio delle deformazioni così come in altri ambiti ingegneristici sono ogni giorno più avanzate, i margini di crescita di questi strumenti sono notevoli e gli utilizzi (anche quelli per ora ipotetici) sono sconfinati, resta dunque solo da vedere quali traguardi riusciremo a raggiungere in futuro.

Bibliografia

[1] Susskind Daniel. *Un mondo senza lavoro*. Saggi Bompiani, 2022.

[2] Carolina Crisci, Badih Ghattas, and Ghattas Perera. A review of supervised machine learning algorithms and their applications to ecological data. *Ecological Modelling*, 240:113–122, 2012.

[3] Po-Hsien Liu, Shun-Feng Su, Ming-Chang Chen, and Chih-Ching Hsiao. Deep learning and its application to general image classification. In *2015 international conference on informative and cybernetics for computational social systems (ICCSS)*, pages 7–10. IEEE, 2015.

[4] Alaa Sabeeh Salim, Mohammed Basil Abdulkareem, Yarub Essam Fadhel, Abdulkarem Basil Abdulkarem, Ahmed Muhi Shantaf, and Ahmed B Abdulkareem. Novel image caption system using deep convolutional neural networks (vgg16). In *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–6. IEEE, 2022.

[5] Data science team. Definizione r-squared. *datascience.eu*, 2020.

[6] Anthony Recchia. R-squared measures for two-level hierarchical linear models using sas. *Journal of Statistical Software*, 32:1–9, 2010.