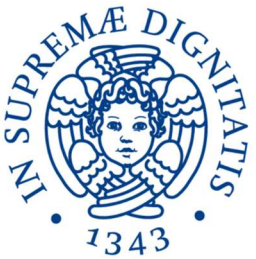


UNIVERSITÀ DI PISA

SCUOLA DI INGEGNERIA

MASTER OF SCIENCE IN COMPUTER ENGINEERING

RELAZIONE DI PROGETTO

INTELLIGENT SYSTEMS

Progetto e realizzazione di un sistema neuro-fuzzy per il confronto di colori

Antonio Di Tecco

1. Introduzione

Considerando un processo industriale il quale core business è stampare copie colore, riteniamo per ipotesi che la prima stampa sia il *master color*. Più accurato è il processo di stampa, più ogni altra stampa successiva è simile al master.
La somiglianza è valutata mediante verifiche visive, in cui le copie vengono confrontate con il master. Questi controlli sono influenzati dall'imprecisione e dalla soggettività dell'osservatore.

2. Scopo

Sviluppo e realizzazione di un sistema basato su rete neurale artificiale.

La rete deve realizzare una funzione di fitting e confrontare un master color e la sua copia, producendo in uscita la loro somiglianza, come mostrato in Fig. 1; per farlo, prende in ingresso le caratteristiche estratte e poi selezionate dallo spettro master e dalla copia, e realizza l'output calcolando la norma Euclidea tra i due colori secondo lo standard Delta-E CIE 76.

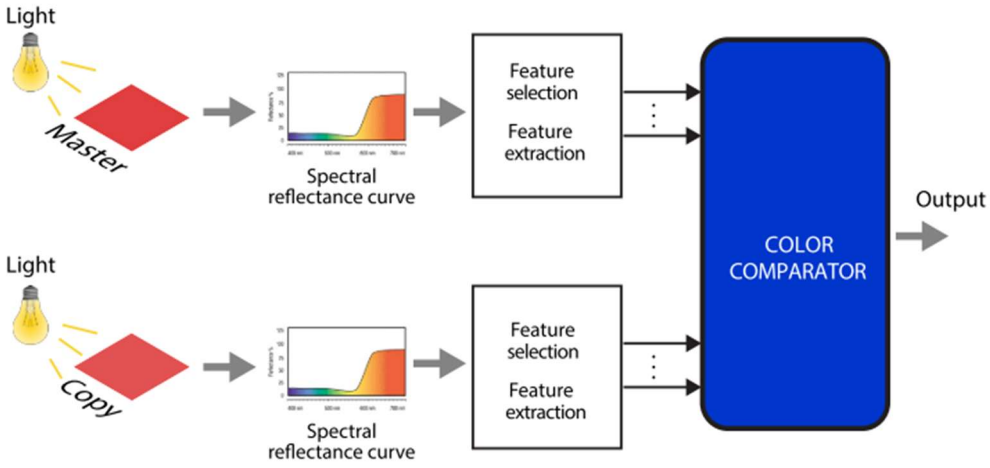


Figura 1: Sistema rete neurale artificiale.

Sviluppo e realizzazione di un sistema basato su logica fuzzy.

Il sistema di inferenza fuzzy di tipo Mamdani deve correggere il Delta-E CIE 76 usato per addestrare la rete neurale del punto precedente, tenendo in considerazione dell'esistenza di regioni nello spazio colore dove il Delta-E calcolato non è valido.
Il sistema prende in ingresso un'appropriata modellazione linguistica delle coordinate (L*, C*, h) del master color, che indicano una regione specifica dello spazio colore, e il Delta-E calcolato con una sola copia, come mostrato in Fig. 2.

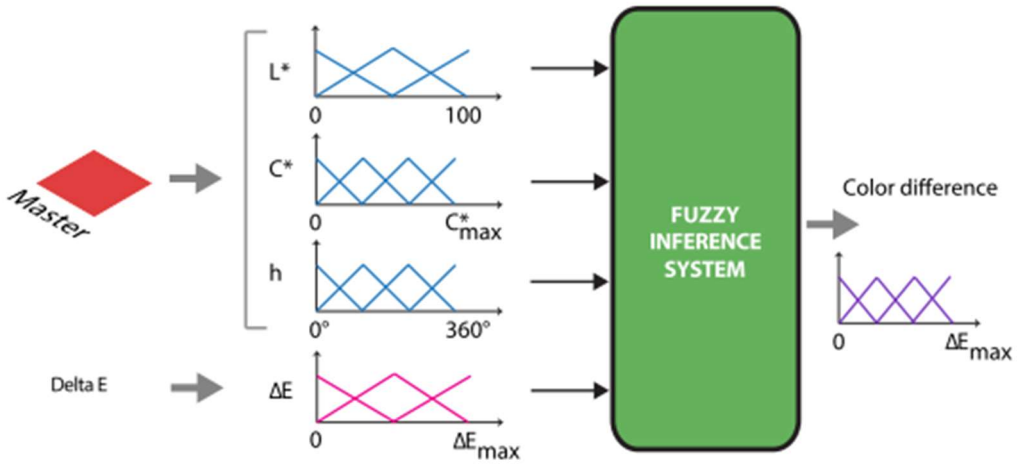


Figura 2: Sistema Fuzzy Mamdani.

3. Risoluzione

3.1 Generazione caratteristiche

In questa sezione si spiega come processare i master, generare le copie colore, estrarre e selezionare le caratteristiche.

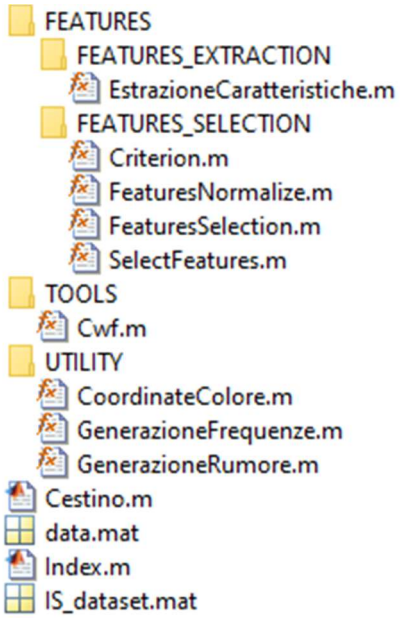


Figura 3: File del modulo CREATE_FEATURES per realizzare la feature selection.

Il file *IS_dataset.m* contiene le strutture dati:

- spectra: matrice di dimensione 421×1269 dove ogni colonna corrisponde ad un colore ed ogni riga ad uno dei 421 campioni della curva di riflettanza del colore;
- coordinates: matrice di dimensione 6×1269 dove ogni Colonna corrisponde ad un colore. Le righe da 1 a 3 corrispondono alle coordinate in ordine RGB, mentre le righe da 4 a 6 corrispondono alle coordinate in ordine L* a* b*. Le coordinate colore sono state calcolate con una sorgente luminosa D65.

Il file *Index.m* processa i dati, genera le copie colore ed esegue estrazione e selezione delle caratteristiche.

Index.m

```
% Percorsi interni
addpath('./FEATURES/FEATURES_EXTRACTION/');
addpath('./FEATURES/FEATURES_SELECTION/');
addpath('./TOOLS/');
addpath('./UTILITY/');

% Reset
clear;
clc;
close all;

% Importazione strutture
load('./IS_dataset.mat');
disp('Importazione strutture da ''./IS_dataset.mat'' completata');

% Inizializzazione strutture
[CoordinateRGB, CoordinateLAB] = CoordinateColore(coordinates);
clearvars coordinates;

% Curva di riflettanza
ReflectanceCurve = spectra';
clearvars spectra;

% Generazione coordinate colore
D65cwf = Cwf();
CoordinateLABMaster = roo2lab(ReflectanceCurve(:,1:end) .*
100,D65cwf,380:800);

% Estrazione caratteristiche master
RAWFeaturesMaster = EstrazioneCaratteristiche(ReflectanceCurve);

% Generazione copia
ReflectanceCurveCopy = GenerazioneRumore(ReflectanceCurve);

% Generazione coordinate LAB della copia
CoordinateLABCopy = roo2lab(ReflectanceCurveCopy(:,1:end) .*
100,D65cwf,380:800);

% Generazione caratteristiche copia
RAWFeaturesCopy = EstrazioneCaratteristiche(ReflectanceCurveCopy);
```

```
% Generazione vettore INPUT/OUTPUT ANN
RAWFeatures = FeaturesNormalize([RAWFeaturesMaster RAWFea-
turesCopy]);
target = de(CoordinateLABMaster,CoordinateLABCopy);

% Delta-E CIE 2000
D20 = de2000(CoordinateLABMaster,CoordinateLABCopy);

% Feature selection
Features = FeaturesSelection(RAWFeatures,target);
input = RAWFeatures(:,Features)';
target = target';

% Salvataggio dati
save ./data.mat input target CoordinateLABMaster D20 RAWFeatures
Features;
save ../CREATE_NN/data.mat input target D20 RAWFeatures Features;
save ../CREATE_FUZZY_SYSTEM/data.mat input target …
         CoordinateLABMaster D20 RAWFeatures Features;
```

La funzione *roo2lab()*[1] è utilizzata per convertire la curva di riflettanza di ciascun colore nelle corrispondenti coordinate L*a*b*. Per la conversione è utilizzata una color weighting function (*cwf*) con illuminante D65 e osservatore 2 generata con *Cwf()*.

```
function R = Cwf()
    R = 'D65/2';
end
```

Inizialmente si era utilizzata una *cwf* con osservatore 10, poiché in letteratura risultasse molto usata dalle aziende topografiche[2], però si è notata una discordanza nella generazione delle *CoordinateLABMaster* con le *CoordinateLAB* originali, motivo per il quale si è scelto osservatore 2 per la progettazione.
La funzione *GenerazioneRumore()* aggiunge rumore casuale e lineare alle principali frequenze spettrali ricavate da *GenerazioneFrequenze()*, per realizzare copie colore.

```
function R = GenerazioneRumore(Spettro) % Vi,B,Ve,G,A,R
    [Vi,B,Ve,G,A,R] = GenerazioneFrequenze(Spettro);
    % Creazione rumore
    [row,col] = size(Vi);
    Vi = Vi+rand(row,1)*0.001+0.01;
    B = B+rand(row,1)*0.001+0.01;
    Ve = Ve+rand(row,1)*0.001+0.01;
    G = G+rand(row,1)*0.001+0.01;
    A = A+rand(row,1)*0.001+0.01;
    R = R+rand(row,1)*0.001+0.01;
    R = [Vi,B,Ve,G,A,R];
    disp('Generazione copia completata');
end
```

[1] Le funzioni di colorimetria utilizzate nel progetto sono installate nel pacchetto OptProp di MATLAB.
[2] https://www.iso.org/standard/67155.html.

Con *EstrazioneCaratteristiche()* si estraggono dodici caratteristiche per ciascun colore dalle matrici *ReflectanceCurve* e *ReflectanceCurve-Copy*. Le caratteristiche estratte sono: minimo, massimo, media, mediana, skewness, kurtosis, moda, mean abs, varianza, deviazione standard, ampiezza, quartile; il risultato sono due matrici *RAWFeaturesMaster* e *RAWFeaturesCopy*, poi normalizzate con *FeaturesNormalize()* e concatenate in *RAWFeatures* matrice 1269 × 24.

```
function R = EstrazioneCaratteristiche(Spettro)
    NumeroCaratteristiche = 12;
    [r,c] = size(Spettro);  % 1269x421
    C = zeros(r,NumeroCaratteristiche);
    for i = 1:r
        C(i,:) = [
                    min(Spettro(i,:)),
                    max(Spettro(i,:)),
                    mean(Spettro(i,:)),
                    median(Spettro(i,:)),
                    skewness(Spettro(i,:)),
                    kurtosis(Spettro(i,:)),
                    meanabs(Spettro(i,:)),
                    mode(Spettro(i,:)),
                    var(Spettro(i,:)),
                    std(Spettro(i,:)),
                    range(Spettro(i,:)),
                    iqr(Spettro(i,:)),
                ];
    end
    R = C;
    disp('Estrazione caratteristiche completata');
end
```

```
function R = FeaturesNormalize(M)
    R = normalize(M);
end
```

Con *de()* si calcola il Delta-E CIE 76 tra master e copia, il *target* della rete neurale, e con *de2000()* il Delta-E CIE (20)00 utilizzato per verificare in un secondo momento il sistema fuzzy.
Si utilizza *SelectFeatures()* per avviare la procedura di selezione delle feature. Questa chiama la *sequentialfs()*, che a sua volta chiama la funzione *Criterion()*, la quale addestra più volte la rete neurale *fitnet* provando tutte le possibili combinazioni delle colonne della matrice *RAWfeatures*, finché non individua il criterio minore (l'errore più basso) con il subset ottimale di feature, da utilizzare come input per la rete neurale.

```
function R = SelectFeatures(input,target,hiddenLayerSize)

    hidden = zeros(size(input,1),size(input,2));
    hidden(1,1) = hiddenLayerSize;
    opts = statset('display','iter');

```

```
    [fs,history] = sequentialfs(@Criterion, input, target, …
    hidden, 'cv', 'none', 'direction', 'forward', 'options', opts);

    R = fs;
end
```

```
function R = Criterion(input,target,hidden)

    input = input';
    target = target';

    net = fitnet(hidden(1,1),'trainscg');

    net.divideParam.trainRatio = 70/100;
    net.divideParam.testRatio = 30/100;
    net.divideParam.valRatio = 0;
    net.trainParam.showWindow = 0;

    [net,tr] = train(net,input,target);

    output = net(input);

    testInd = tr.testInd;
    testTarget = target(:,testInd);
    testOutput = output(:,testInd);

    R = perform(net,testTarget,testOutput);

End
```

Le caratteristiche selezionate dall'algoritmo sono: minimo, media e mean abs del master, e la deviazione standard della copia colore.

3.2 Rete Neurale

In questo paragrafo si illustra come costruire, addestrare e validare una rete neurale artificiale.

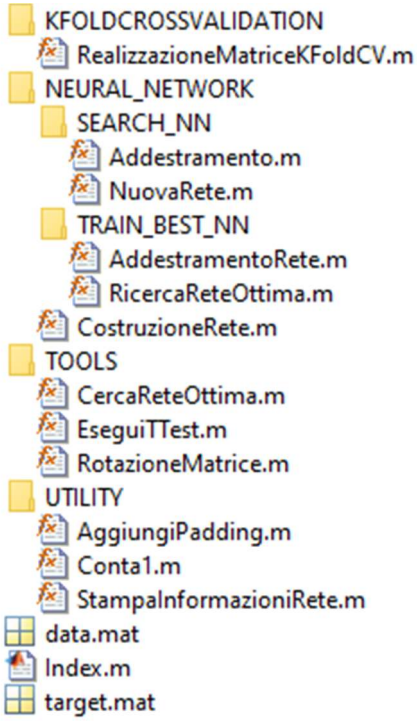


Figura 4: Elementi del modulo CREATE_NN per realizzare una rete neurale.

Il file *data.mat* contiene la matrice input e target della rete neurale realizzate in § 3.1.

Il file *target.mat* sarà trattato in § 3.4.

Il file *index.m* avvia la ricerca di una rete neurale ottima utilizzando le feature individuate nella feature selection.

Index.m

```
1   % Percorsi interni
2   addpath('./KFOLDCROSSVALIDATION/');
3   addpath('./NEURAL_NETWORK/');
4   addpath('./NEURAL_NETWORK/SEARCH_NN/');
5   addpath('./NEURAL_NETWORK/TRAIN_BEST_NN/');
6   addpath('./TOOLS/');
7   addpath('./UTILITY/');
8
9   % Reset
10  clear;
11  clc;
12  close all;
13
14  %%%%%%%%%
15  load('data.mat');
16  % load('target.mat'); % Output fixato con fuzzy
```

```
f = size(input,1);
%%%%%%%%%

%%%%% Variabili da configurare
    MinHiddenNeurons = 2;      % Numero min di neuroni della rete
    MaxHiddenNeurons = 2 * f;  % Numero max di neuroni della rete
    dim = MaxHiddenNeurons-MinHiddenNeurons+1;
    AddestramentiPerRete = 20; % Numero di train per ANN
    Tentativi = 10;            % Tentativi di ricerca per la rete col
                               % numero ottimale di neuroni
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[input,target] = AggiungiPadding(input,target);

% Addestra reti con numero di neuroni variabile e registra gli
errori
MatriceErrori = Addestramento(input,target,MinHiddenNeurons,...
                                  MaxHiddenNeurons,Addestramenti-
PerRete);

% Si cerca il numero di neuroni ottimale
Neuroni = CercaReteOttima(MatriceErrori,MinHiddenNeurons,dim);

% Si ricerca la rete che ottiene il miglior risultato
[net,tr] = RicercaReteOttima(input,target,Neuroni, …
                                  AddestramentiPerRete,Tentativi);
```

Per trovare la rete neurale che approssima il Delta-E CIE 76, si è usata la matrice *input* degli ingressi alla rete, contenente tutti i feature vector ottenuti dalla feature selection, e il vettore *target* dei valori di uscita. Lo strato di ingresso ha quattro neuroni e lo strato di uscita ha un solo neurone. Quelli di ingresso sono associati alle quattro feature. Per trovare il numero dei neuroni nello strato nascosto si è fatto ricorso ad alcune tecniche di analisi statistica. E' stata quindi realizzata la funzione *Addestramento()*, la quale addestra un insieme di $2 \times f - 1$ reti neurali chiamando *NuovaRete()*, ciascuno contenente reti aventi stesso numero di neuroni nascosti, variabile tra due e $2 \times f$, dove è f il numero delle feature. Ogni rete è addestrata venti volte e validata con k-fold cross-validation, con $k = 10$.

```
function R = Addestramento(input,target,MinHiddenLayer,...
                               MaxHiddenLayer,AddestramentiPerRete)
    % Pre-allocazione matrice degli errori
    errors = zeros(AddestramentiPerRete,MaxHiddenLayer);

    % Prepara le matrici alla KFoldCV
    samples = size(target,2);
    [input,target] = RealizzazioneMatriceKFoldCV(input,target,sam-
ples);
    disp('Inizio ricerca rete');
    for h = MinHiddenLayer:MaxHiddenLayer
        errors(:,h) = NuovaRete(input,target,samples,h,...
                                 AddestramentiPerRete/2);
    end
    disp('Termine ricerca rete');
    R = errors(:,MinHiddenLayer:MaxHiddenLayer);

end
```

```
function R = NuovaRete(input,target,samples,hiddenNeurons,bucket)

    % Creazione vettore errori (in teoria 20, uno per ogni train)
    errors = zeros(bucket*2,1);
    disp(['Rete ',num2str(hiddenNeurons)]);
    net = CostruzioneRete(hiddenNeurons,samples);
    index = 1;

    % Si attiva la KFoldCrossValidation per due volte di seguito
    for r = 1:2

        % Si modificano le matrici per bene
        [input,target] = RotazioneMatrice(1,input,target);

        for i = 1:bucket
            [net,tr] = train(net,input,target);
            errors(index) = mean(tr.tperf); % Oppure tr.best_tperf

            % Si scombussolano le matrici per il prossimo training
            [input,target] = RotazioneMatrice(0,input,target);
            index = index + 1;
        end
    end

    R = errors;

end
```

Per realizzare la 10-fold cross-validation, si è reso prima multiplo di cento il dataset con *AggiungiPadding()*, così da semplificare il partizionamento, poi è utilizzata *RealizzazioneMatriceKFoldCV()* per permutare i dati in *input* e *target*.

```
function [R1,R2] = AggiungiPadding(input,target)

    samples = size(target,2);
    val = mod(samples,100);


    if val ~= 0  % Se i campioni non sono multipli di 100
        val = 100 - val;
        for i = 1:val
            pad = randi(samples);
            input = input(:,[1:end pad]);
            target = target(:,[1:end pad]);
        end
    end

    R1 = input;
    R2 = target;

end
```

```
function [R1,R2] = RealizzazioneMatriceKFoldCV(input,target, …
                            samples)
    % Splitto il sample set in 10 bucket
    blockDim = samples / 10;

    % Il test set è 30% del bucket, mentre il resto è train set
    testBlock = blockDim * 0.3;
    trainBlock = blockDim - testBlock;
    trainInd = zeros(1,trainBlock*10);
    testInd = zeros(1,testBlock*10);
    startBlock = 1;
    endBlock = blockDim;
    StartTrainBlock = 1;
    StartTestBlock = 1;

    for i = 1:10
        trainInd(:,StartTrainBlock:((blockDim-testBlock)*i)) = …
                startBlock:(endBlock - testBlock);
        testInd(:,StartTestBlock:(blockDim-trainBlock)*i) = …
                (endBlock - testBlock + 1):endBlock;
        StartTrainBlock = (blockDim-testBlock)*i+1;
        StartTestBlock = (blockDim-trainBlock)*i+1;
        startBlock = startBlock + blockDim;
        endBlock = endBlock+blockDim;
    end

    R1 = input(:,[trainInd testInd]);
    R2 = target(:,[trainInd testInd]);

end
```

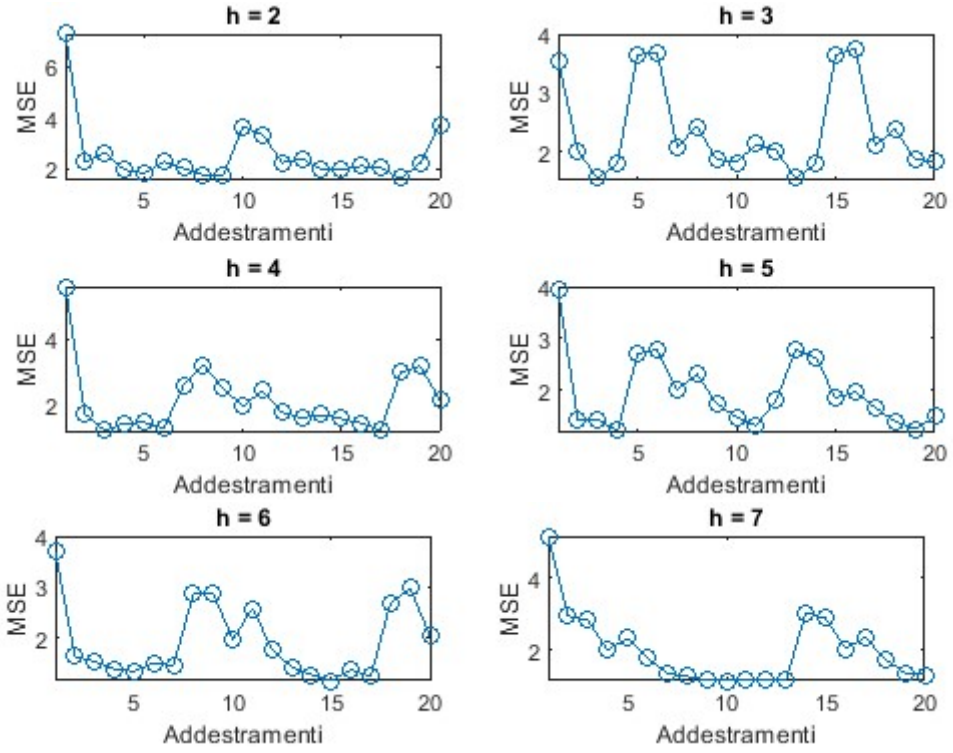
Quindi, è stata costruita la rete neurale con la funzione *CostruzioneRete()* in modo da utilizzare gli indici dei campioni nell'intervallo {1, …, 910} per comporre il training set, e gli indici in {911, …, 1300} per formare il test set. La funzione di attivazione utilizzata nello strato nascosto della ANN è la funzione sigmoidea, mentre l'algoritmo di addestramento è Bayesian regularization.

```
% La funzione setta le impostazioni di default per la rete.
function R = CostruzioneRete(n,samples)

    % Indici per implementare la KFoldCrossValidation
    % Il sample set è diviso in 10 bucket.
    % Tre decimi del bucket è utilizzato per il test set, il resto
    % è per il training set della rete.
    testInd = samples * 0.3;
    trainInd = samples - testInd;

    net = fitnet(n,'trainbr');
    net.trainParam.showWindow = false;
    net.divideFcn = 'divideind';
    net.divideParam.trainInd = 1:trainInd;
    net.divideParam.testInd  = (samples-testInd):samples;
    net.divideParam.valInd   = [];
    net.performFcn = 'mse';
    R = net;

end
```

Per attuare la 10-fold cross-validation, prima di ogni chiamata della funzione `train()` di MATLAB, si effettua una chiamata alla funzione `RotazioneMatrice()`, che è stata opportunatamente realizzata per effettuare uno shift delle colonne delle matrici `input` e `target` di un numero fisso di posizioni.

```
1  % La funzione ruota le matrici input e target, se:
2  % random = 0 le matrici sono shiftate staticamente;
3  % random = 1, le matrici sono shiftate randomicamente per
4  % implementare correttamente la k-fold cross-validation.
5  function [R1,R2] = RotazioneMatrice(random,input,target)
6      [r,c] = size(target);
7      if random == 1
8          j = randi([1 c],1,1);
9      else
10         j = 0;
11     end
12     c = c / 10;
13     R1 = circshift(input,c+j,2);
14     R2 = circshift(target,c+j,2);
15 End
```

A supporto di questa prima fase si è creata la `MatriceErrori` di dimensione $m \times n$, con $m = 20$ numero di reti da addestrare (tramite la funzione `train()` di MATLAB), e $n = 2 \times f$ numero massimo di neuroni dello strato nascosto[3]. Al termine dell'esecuzione della funzione `train()` — cioè al termine dell'addestramento di una delle 20 reti con numero di h neuroni nascosti — il valore di errore medio sul test set viene salvato, ovvero la performance media della rete. Questo valore è salvato nell'elemento (i,n) di `MatriceErrori`, fino ad ottenere venti risultati per ciascun numero di neuroni nello strato nascosto.



[3] http://www.heatonresearch.com/2017/06/01/hidden-layers.html.

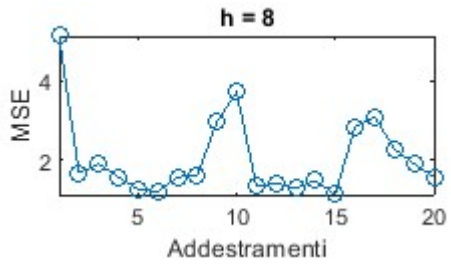


Figura 5: Distribuzione degli errori di ciascuna rete neurale con h neuroni nello strato nascosto, durante i due cicli di 10-fold cross-validation.

A questo punto, la funzione `EseguiTTest()` effettua i test *t* di Student (facendo uso della funzione ttest2() di MATLAB), prendendo come argomento le distribuzioni dell'errore memorizzate in `MatriceErrori`. Il test statistico confronta statisticamente le distribuzioni di errore e consente di determinate il numero ottimo di neuroni nello strato nascosto.

```
1  % La funzione esegue il test di Student sulle colonne della
2  % matrice successo S, assegnando il token (1) alla colonna ME con
3  % probabilità media di successo(errore) maggiore(minore).
4  function R = EseguiTTest(S,ME,dim)
5      disp('Inizio TTest.');
6      M = zeros(dim,dim);
7      for i = 1:1:dim
8          for t = 1:1:dim
9              r = ttest2(S(:,i),S(:,t));
10             if(r == 1)
11                 if(ME(i) < ME(t))
12                     M(t,i) = 1;
13                 end
14             end
15         end
16     end
17     disp('TTest terminato.');
18     R = M;
19 End
```

Il risultato del test appartiene a {0; 1}, restituendo 0 quando l'ipotesi zero non è rigettata, 1 quando questa è rigettata. Il test ha successo se l'ipotesi zero è rigettata. L'ipotesi zero suppone che la differenza fra gli errori medi di due reti sia dovuta al caso. Il risultato ottenuto si memorizza in posizione (*j*,*k*) di `MatriceTTest` (*n* × *n* elementi) se la media di `MatriceErrori(:,j)` è maggiore della media di `MatriceErrori(:,k)`, altrimenti si memorizza in (*k*,*j*). Terminati gli *n* × *n*, confronti si otterrà una matrice binaria antisimmetrica rispetto alla diagonale principale, allora con `Conta1()` si contano i numeri 1 di ciascuna colonna di `MatriceTTest`. La colonna $h^* \in \{2, \ldots, 2 \times f\}$ con il maggior numero di elementi che assumono valore 1 è la rete con numero ottimo di neuroni.

```
% La funzione conta il numero di uno nella matrice M, per colonna
function R = Conta1(M,dim)
    U = zeros(1,dim);
    for i = 1:1:dim
        U(1,i) = sum(M(:,i) == 1);
    end
    R = U;
End
```

Operativamente, trovato il numero *h** di neuroni ottimo, si ripete quanto fatto in precedenza in `RicercaReteOttima()` per addestrare la rete migliore.
Il risultato della ricerca è una rete neurale con singolo strato nascosto composto da 2 neuroni come mostrato in Fig. 6. È stato possibile ottenere un coefficiente di regressione di 0.97 utilizzando come metrica di valutazione l'errore quadratico medio sul test set.

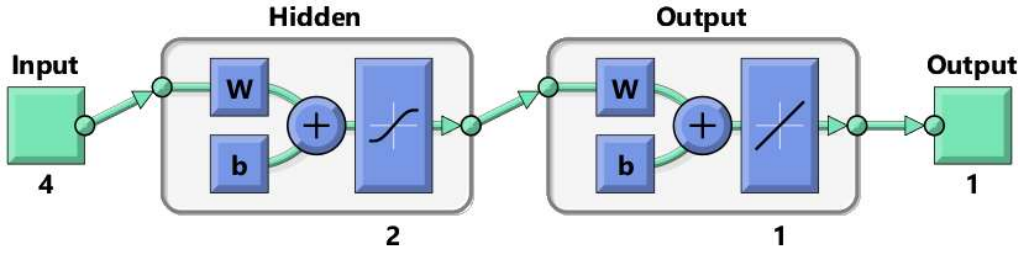


Figura 6: Rete neurale artificiale.

In Figura 7.1, 7.2, 7.3 e 7.4, sono mostrate le curve di regressione e l'istogramma dell'errore.

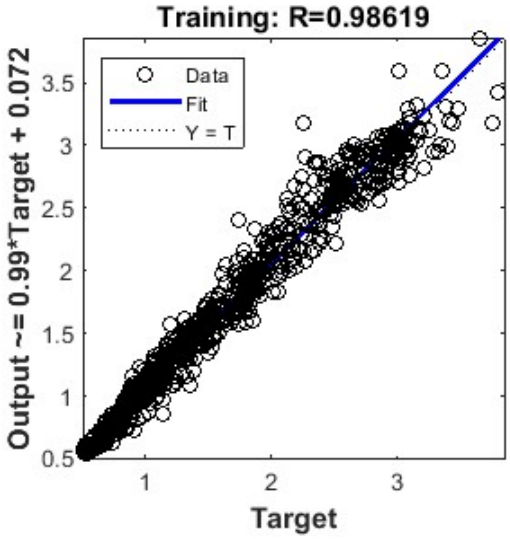


Figura 7.1: Curva di regressione per il training set.

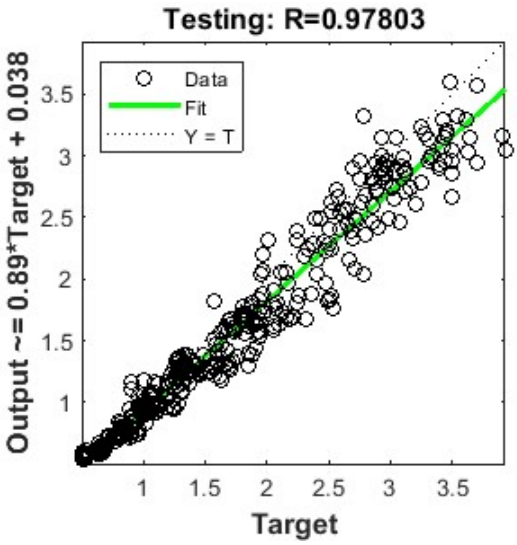


Figura 7.2: Curva di regressione per il test set.

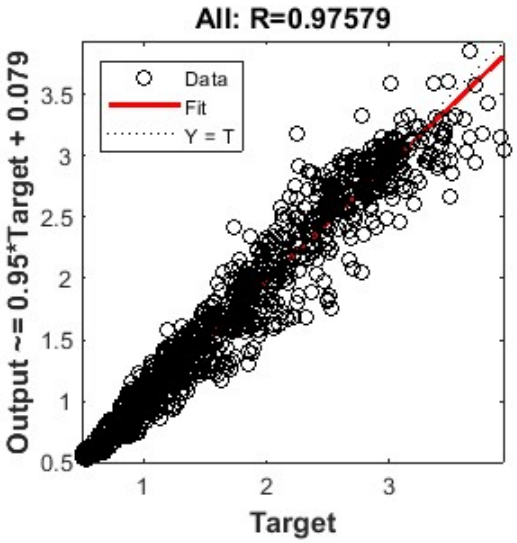


Figura 7.3: Curva di regressione totale.

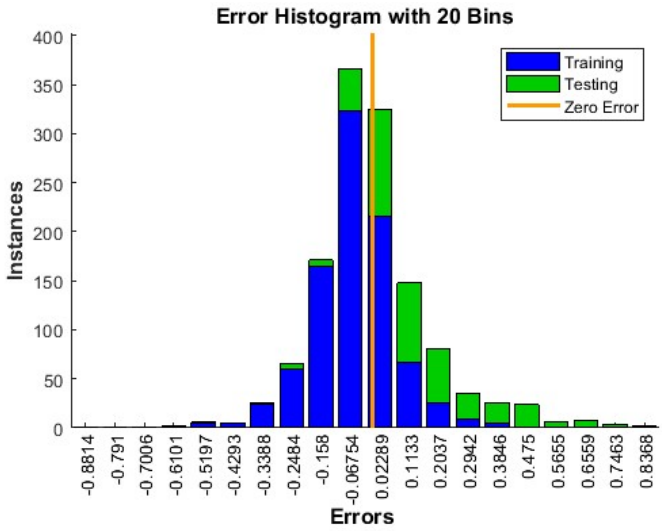


Figura 7.4: Istogramma dell'errore.

3.3 Sistema di Inferenza Fuzzy

Esistono regioni nello spazio CIE, mostrato in Fig. 8, dove la differenza all'osservatore, anche se molto elevata, non è percepibile[4], alcuni esempi sono la regione blu-violetto, la regione dei colori molto chiari e dei molto scuri.

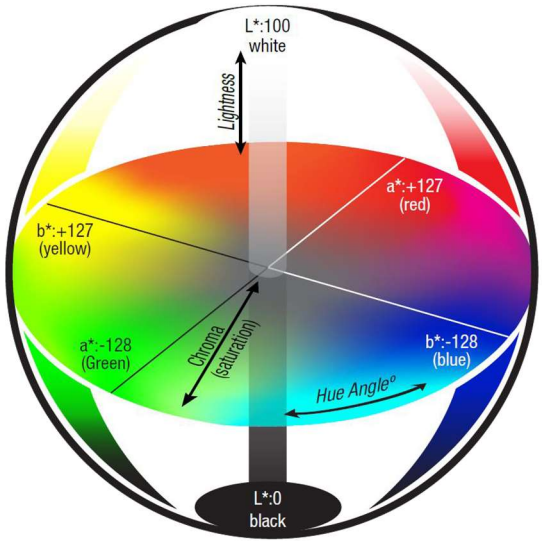


Figura 8: Spazio colore CIE.

Quindi, l'intenzione è di progettare un sistema di inferenza fuzzy (FIS), come mostrato in Fig. 9, per correggere l'errore Delta-E CIE 76 generato in § 3.1.

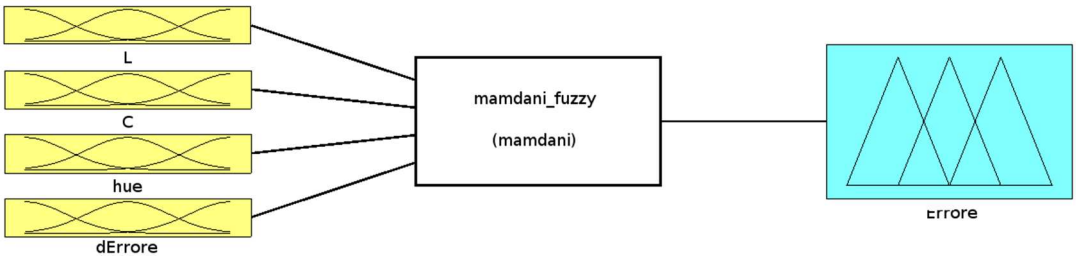


Figura 9: Sistema di Inferenza Fuzzy Mandami.

Il sistema di inferenza fuzzy tipo Mamdani che si è progettato ha cinque variabili linguistiche. Quattro sono di ingresso, una è di uscita. Delle variabili di input, le prime tre sono relative alle tre coordinate nello spazio CIE L*C*h del master, la quarta al Delta-E CIE 76 calcolata con la copia colore in § 3.1.
La variabile di output `Errore` è la correzione generata dal FIS.
Le funzioni di appartenenza delle variabili fuzzy sono state realizzate sapendo che la luminosità (L) ∈ [0, 100], la saturazione (C) varia nello spazio CIE ∈ [-127, 128], la tonalità (*hue*) ∈ [0, 360][5] e dErrore ∈ [0, 5]. Però, come mostrato in Fig. 10, lo spazio colore CIE è asimmetrico nel verso orizzontale, motivo per cui non si è scelto di esprimere la saturazione con l'universo fuzzy ∈ [-127, 128]. In Fig. 11 si può

[4] https://www.hdm-stuttgart.de/international_circle/circular/issues/13_01/ICJ_06_2013_02_069.pdf.
[5] https://www.colourphil.co.uk/lab_lch_colour_space.shtml.

vedere che i colori equidistanti radialmente dall'asse dei grigi si trovano sullo stesso livello *C* di saturazione, quindi è stato deciso di esprimere l'universo di *C* attraverso una funzione radiale con codominio ∈ [0,100].

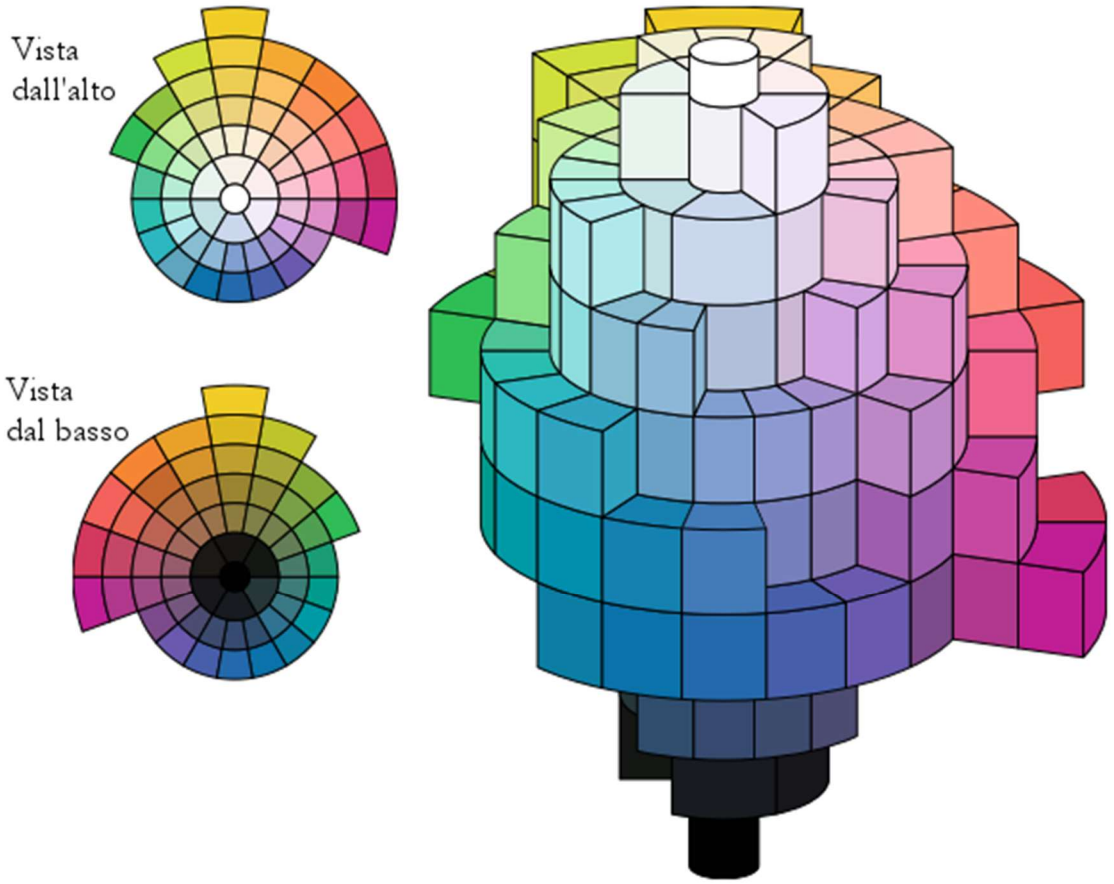


Figura 10: Spazio colore CIE.

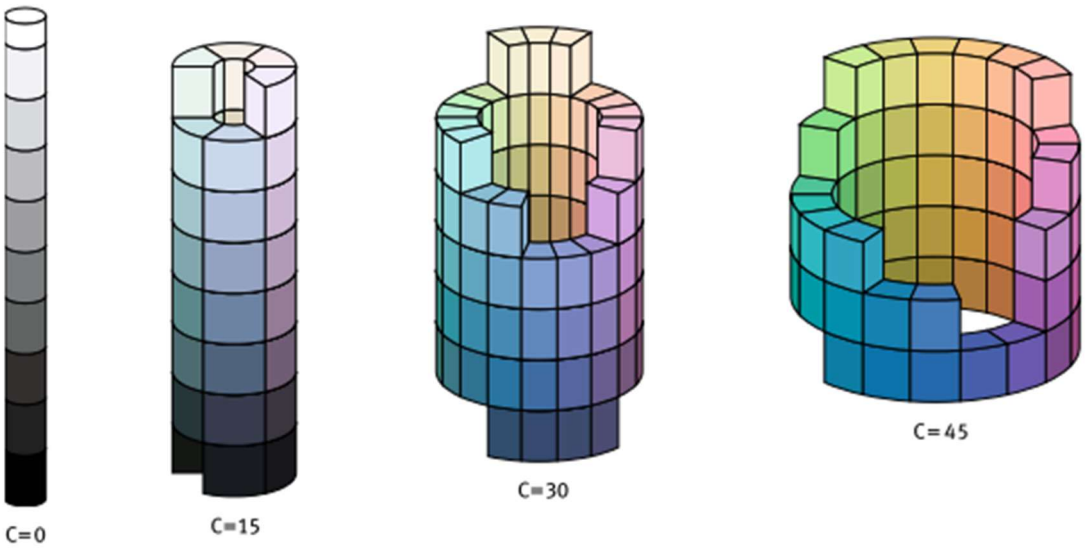


Figura 11: Livelli di saturazione *C*.

Le funzioni di appartenenza per ogni variabile linguistica sono mostrate in Fig. 12.1, 12.2, 12.3, 12.4 e 12.5.

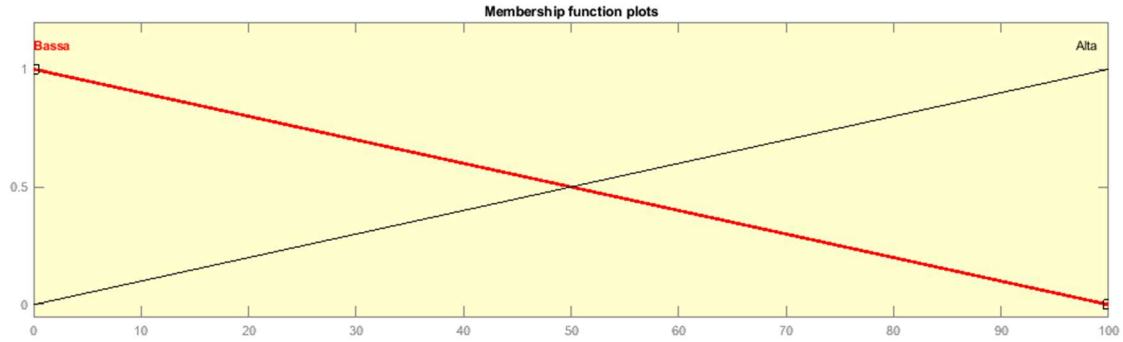


Figura 12.1: Variabile linguistica di input *L*.

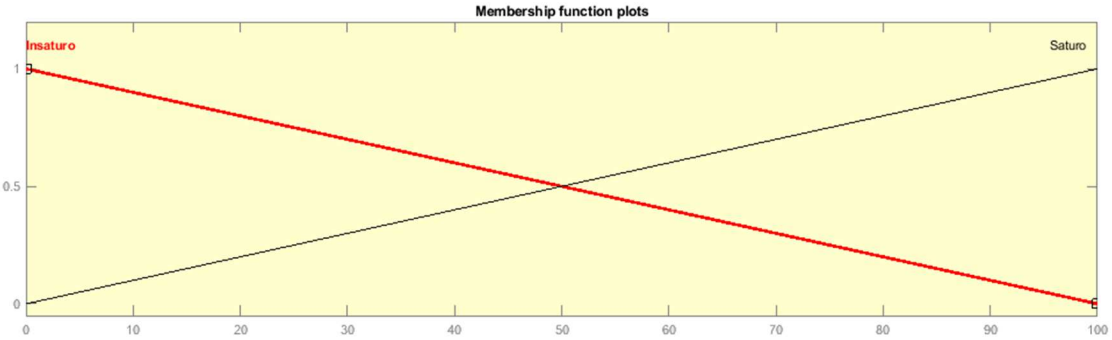


Figura 12.2: Variabile linguistica di input *C*.

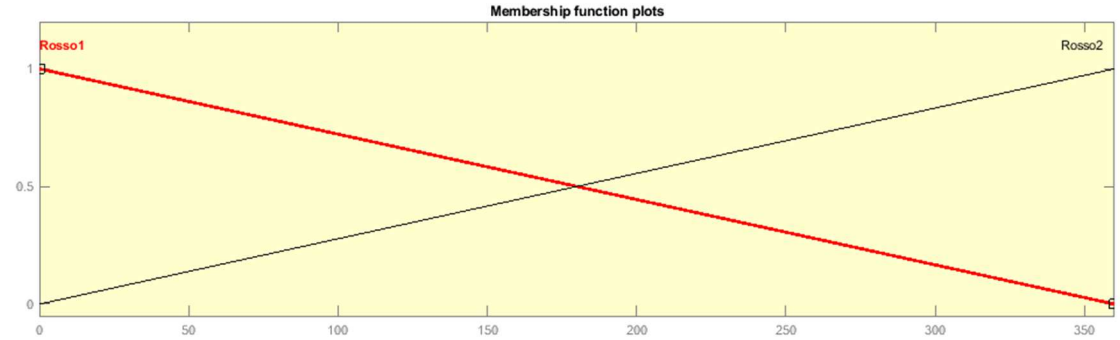


Figura 12.3: Variabile linguistica di input *hue*.

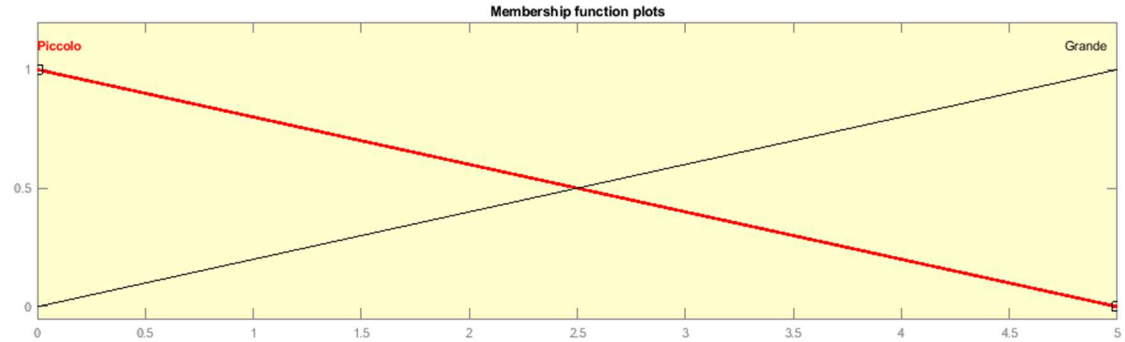


Figura 12.4: Variabile linguistica di input *dErrore*.

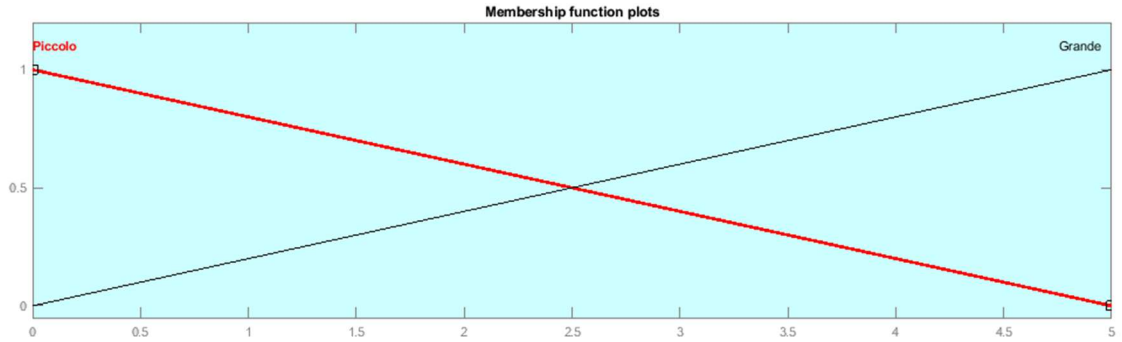


Figura 12.5: Variabile linguistica di output *Errore*.

Le regole sono state realizzate dopo la generazione di molte variabili linguistiche e tentativi[6], basandosi anche in ciò che è scritto in letteratura[7].
Le sedici regole sono mostrate nella tabella di seguito.

1	IF L is Bassa AND C is Insaturo AND hue is Rosso1 AND dErrore is Piccolo THEN Errore is Piccolo
2	IF L is Bassa AND C is Insaturo AND hue is Rosso1 AND dErrore is Grande THEN Errore is Grande
3	IF L is Bassa AND C is Insaturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Piccolo
4	IF L is Bassa AND C is Insaturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Grande
5	IF L is Bassa AND C is Saturo AND hue is Rosso1 AND dErrore is Piccolo THEN Errore is Piccolo
6	IF L is Bassa AND C is Saturo AND hue is Rosso1 AND dErrore is Piccolo THEN Errore is Grande
7	IF L is Bassa AND C is Saturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Piccolo
8	IF L is Bassa AND C is Saturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Grande
9	IF L is Alta AND C is Insaturo AND hue is Rosso1 AND dErrore is Piccolo THEN Errore is Piccolo
10	IF L is Alta AND C is Insaturo AND hue is Rosso1 AND dErrore is Grande THEN Errore is Grande
11	IF L is Alta AND C is Insaturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Piccolo
12	IF L is Alta AND C is Insaturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Grande
13	IF L is Alta AND C is Saturo AND hue is Rosso1 AND dErrore is Piccolo THEN Errore is Piccolo
14	IF L is Alta AND C is Saturo AND hue is Rosso1 AND dErrore is Piccolo THEN Errore is Grande
15	IF L is Alta AND C is Saturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Piccolo
16	IF L is Alta AND C is Saturo AND hue is Rosso2 AND dErrore is Piccolo THEN Errore is Grande

Per generare il vettore `target` modificato è stata utilizzata la funzione `eval-fis()`, che prende in ingresso la matrice `input` e il FIS. `input` è creato con la funzione `CreaInput()`. La matrice ha dimensione $m \times n$, con $m = 1269$ campioni ed $n = 4$. Le prime tre colonne di n sono le coordinate L*C*h convertite dalle coordinate L*a*b* con la funzione `lab2lch()`, la quarta è l'errore Delta-E CIE 76 calcolato in § 3.1.
Il nuovo vettore `target` è stato usato, assieme alla matrice `input` costruita in § 3.1, per addestrare nuove reti neurali come illustrato nel paragrafo precedente.

```
1  function R = CreaInput(LAB,D76)
2      input = lab2lch(LAB);
3      input(:,2) = input(:,2) * 100/128;
4      input(:,4) = D76;
```

[6] Per una spiegazione più dettagliata si rimanda in Approfondimenti § 1 a fine relazione.
[7] https://www.hdm-stuttgart.de/international_circle/circular/issues/13_01/ICJ_06_2013_02_069.pdf.

```
5       R = input;
6   end
7
8   load('./data.mat'); fis = readfis('./mamdani_fuzzy.fis');
9   input = CreaInput(CoordinateLABMaster,target)';
10  target = evalfis(input,fis)';
```

Il risultato migliore è mostrato in Fig. 13.1 e 13.2.

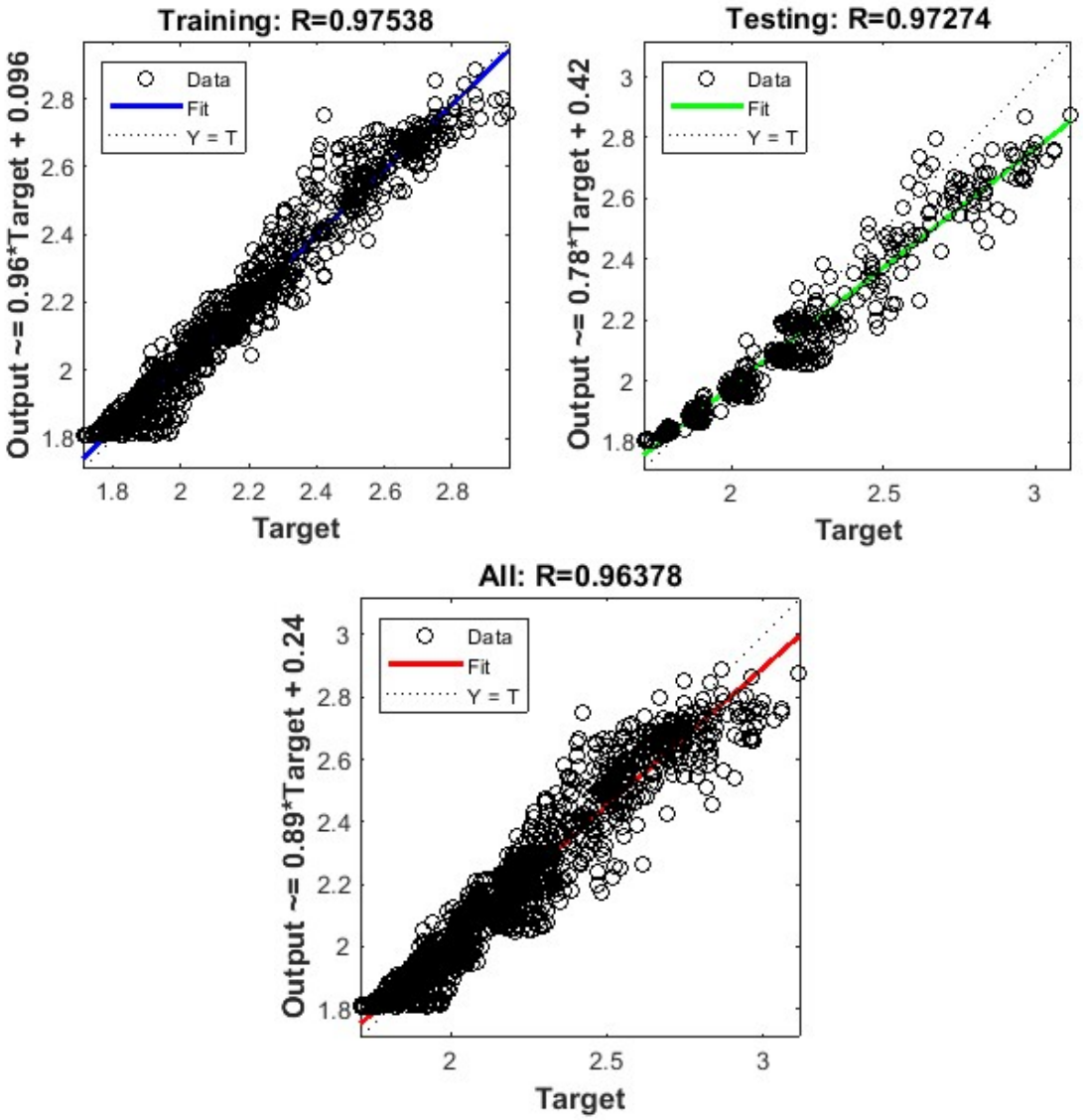


Figura 13.1: Curva di regressione della rete.

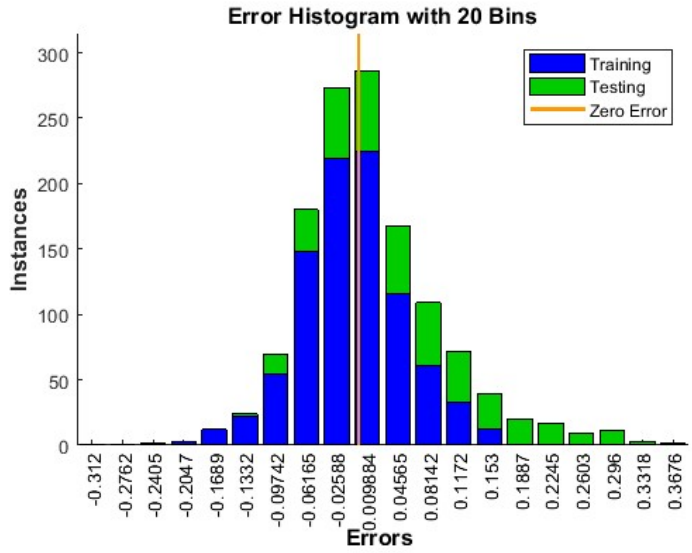


Figura 13.2: Istogramma dell'errore.

La rete neurale, nonostante i risultati ottenuti dalle curve di regressione, non è una buona rete, perché è stata addestrata con un dataset non consistente. Infatti, per ulteriore conferma, sono stati utilizzati alcuni campioni validi ed eterogenei, non presenti nel dataset, da dare in ingresso alla rete. Il risultato è mostrato di seguito.

Figura 13.3: Istogramma per 10 campioni utilizzati per provare la rete. Si nota che l'errore oscilla da qualche unità ad un decimo.

Gli errori D76, che in principio dovevano essere corretti dal sistema fuzzy, sono stati modificati in modo errato.

3.4 Adaptive Neuro – Fuzzy Inference System

In quest'ultimo paragrafo si illustra come migliorare l'uscita prodotta da un sistema fuzzy utilizzando una rete ANFIS.

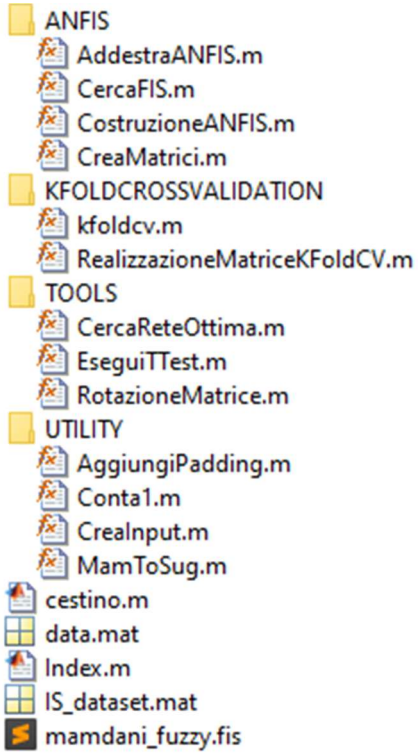


Figura 14: Elementi del modulo CREATE_FUZZY_SYSTEM per realizzare un sistema fuzzy.

I file *IS_dataset*.mat e *data.mat* sono gli stessi di § 3.1.

Il file *index.m* avvia la ricerca e l'addestramento di una rete ANFIS per trovare il FIS ottimo.

Index.m

```
1  % Reset
2  clear;
3  clc;
4  close all;
5
6  addpath('./KFOLDCROSSVALIDATION/');
7  addpath('./TOOLS/');
8  addpath('./UTILITY/');
9  addpath('./ANFIS/');
10
11 target = CercaFIS();
12
13 save ../CREATE_NN/target.mat target;
```

Per trovare il FIS che corregge il Delta-E CIE 76, si chiama la funzione *Cerca-FIS()*.

```
function R = CercaFIS()

    % Importazione strutture
    load('./data.mat');
    disp('Importazione strutture da ''./data.mat'' completata');

    % Inizializzazione strutture
    MatriceErrore = zeros(20,10);
    D76 = target;

    for e = 1:10
        disp(['ANFIS N.',num2str(e)]);
        [fisStruct.fis(e),MatriceErrore(:,e)] = AddestraANFIS( …
                            CoordinateLABMaster,target,D20);
    end

    fis = fisStruct.fis(CercaReteOttima(MatriceErrore,1,10));
    R = evalfis(CreaInput(CoordinateLABMaster,D76)',fis)';

End
```

CercaFIS() si comporta similmente alla funzione *Addestramento()* mostrata in § 3.2. Vengono addestrate dieci reti ANFIS tutte con stesso FIS di partenza chiamando *AddestraAnfis()*. Ciascun FIS è stato addestrato venti volte e validato con *k*-fold cross-validation, con *k* = 10.

```
function [R1,R2] = AddestraANFIS(LAB,D76,D20)

    [input,target] = CreaMatrici(LAB,D76,D20);
    samples = size(target,1);
    [fis, options, trainInd, testInd] = CostruzioneANFIS(samples);
    errors = zeros(20,1);
    index = 1;

    for r = 1:2
        [input,target,train,test] = kfoldcv(1,input,target, …
                            trainInd,testInd);
        for i = 1:10
            fis = anfis(train,options);
            testError = evalfis(test(:,1:4),fis);
            errors(index) = immse(test(:,5),testError);
            options.InitialFIS = fis;
            [input,target,train,test] = kfoldcv(0,input,target, …
                            trainInd,testInd);
            index = index + 1;
        end
    end

    R1 = fis;
    R2 = errors;

end
```

Anche qui, per realizzare la 10-fold cross-validation, sono state create prima le matrici *input* e *target* con la funzione *CreaMatrici()*, poi è stata costruita la rete ANFIS con *CostruzioneANFIS()*. La matrice *input* è stata realizzata come visto prima con *CreaInput()*. Il vettore *target* è uguale al Delta-E CIE 00 calcolato in § 3.1. Si è pensato di usare il Delta-E CIE 00 perché con questa formula sono stati corretti i principali problemi di visibilità nello spazio colore[8]. Per la rete, sono usati gli indici dei campioni nell'intervallo {1, …, 910} per comporre il training set e gli indici in {911, …, 1300} per formare il test set. Inoltre, usando la funzione *MamToSug()* è stato convertito il sistema fuzzy di § 3.3 in Sugeno, in modo da renderla compatibile per la funzione di addestramento *anfis()*.

```
function [R1,R2] = CreaMatrici(LAB,D76,D20)

    input = CreaInput(LAB,D76);
    target = D20;
    [input,target] = AggiungiPadding(input',target');
    [input,target] = RealizzazioneMatriceKFoldCV(input,target);
    R1 = input;
    R2 = target;

End
```

```
function [R1,R2,R3,R4] = CostruzioneANFIS(samples)

    % Indici per KFoldCV
    testInd = samples * 0.3;
    trainInd = samples - testInd;

    % Converti FIS mamdani a sugeno
    fis = readfis('./mamdani_fuzzy.fis');
    fis = MamToSug(fis);

    % Opzioni ANFIS
    options = anfisOptions;
    options.InitialFIS = fis;
    options.EpochNumber = 1;
    options.DisplayANFISInformation = 0;
    options.DisplayErrorValues = 0;
    options.DisplayStepSize = 0;
    options.DisplayFinalResults = 0;
    options.ValidationData = [];
    R1 = fis;
    R2 = options;
    R3 = trainInd;
    R4 = testInd;

end
```

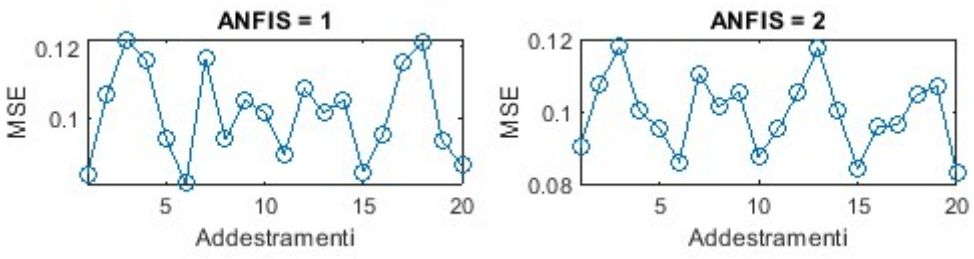
[8] https://pdfs.semanticscholar.org/969b/c38ea067dd22a47a44bcb59c23807037c8d8.pdf

```
function R = MamToSug(fis)

    g = 1;
    for i = 1:2:15
        fis.output.mf(i).name = ['Piccolo',num2str(g)];
        fis.output.mf(i).type = 'trimf';
        fis.output.mf(i).params = [-2 0 3.5];
        fis.output.mf(i+1).name = ['Grande',num2str(g)];
        fis.output.mf(i+1).type = 'trimf';
        fis.output.mf(i+1).params = [3 5 7];
        g = g + 1;
    end

    for i = 1:16
        fis.rule(i).consequent = i;
    end

    R = mam2sug(fis);

end
```

Per completare la *k*-fold cross-validation si è scritta la funzione `kfoldcv()` che, basandosi su `RotazioneMatrice()`, trasla le matrici `input` e `target` e divide il dataset in `train` e `test`, le due matrici con la quale addestrare e testare la rete ANFIS.

```
function [R1,R2,R3,R4] = kfoldcv(ran,input,target,trainInd, …
                                                    testInd)

    samples = size(target,1);

    [input,target] = RotazioneMatrice(ran,input',target');
    input(:,5) = target;
    train = input(1:trainInd,:);
    test = input((samples-testInd):end,:);

    R1 = input;
    R2 = target;
    R3 = train;
    R4 = test;

end
```

Quindi, dopo ogni esecuzione della funzione `anfis()`, si calcola la performance del FIS sul test set e si memorizza il valore in `MatriceErrori` in posizione (*i*,*n*).



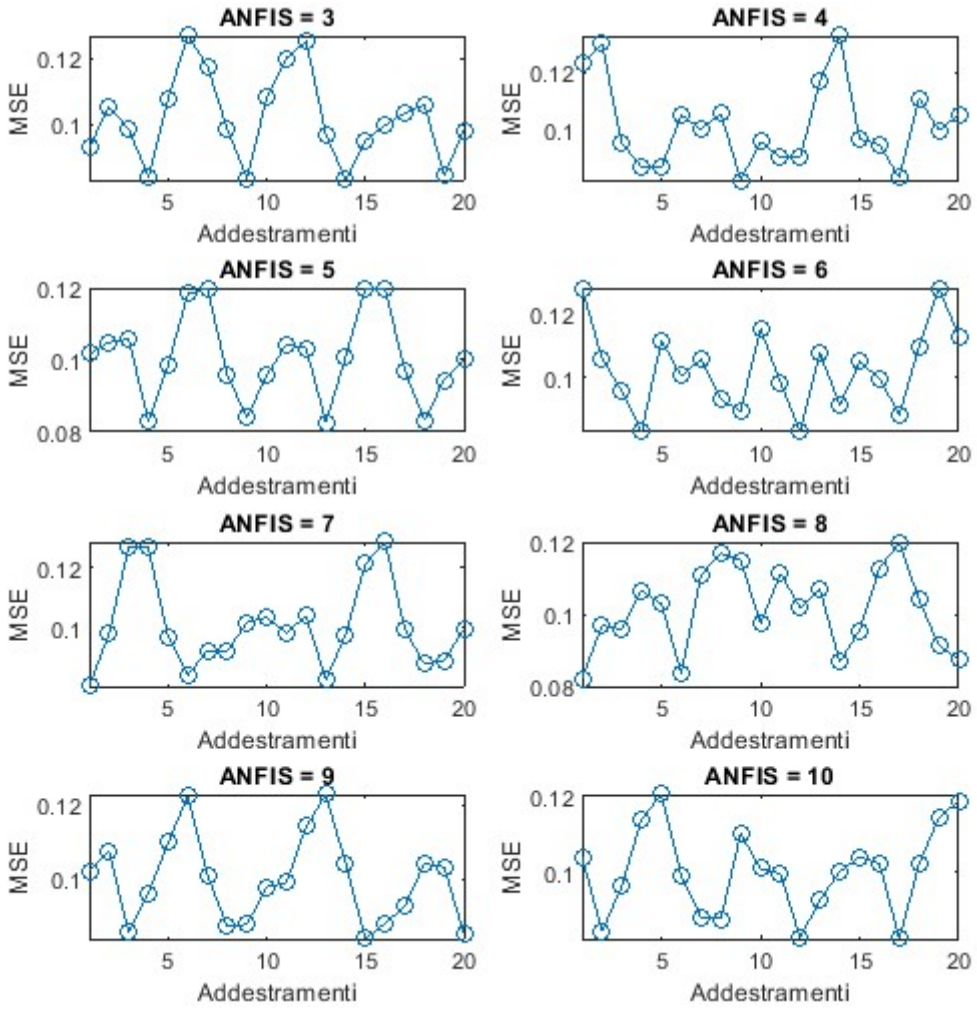


Figura 15: Distribuzione degli errori di ciascuna rete ANFIS, durante i due cicli di 10-fold cross-validation.

Dopo venti esecuzioni della funzione `anfis()`, la FIS addestrata viene salvata nella struttura `fisStruct`. A questo punto, per individuare il sistema fuzzy migliore, viene usato il test *t* di Student come in § 3.2.
La FIS modificata presenta le variabili linguistiche come mostrato in Fig. 16.1, 16.2, 16.3, 16.4 e 16.5.

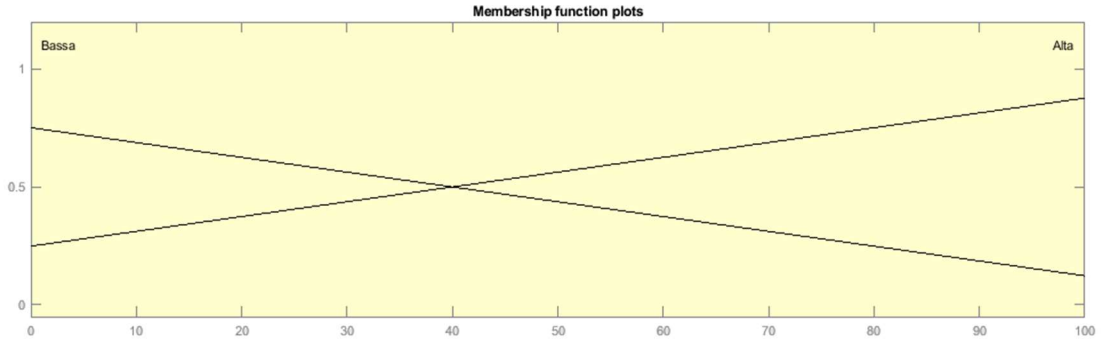


Figura 16.1: Variabile linguistica di input *L*.

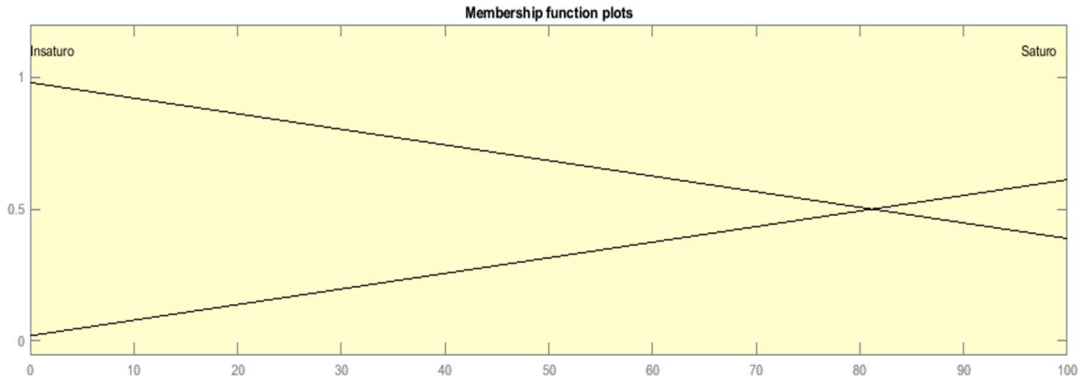


Figura 16.2: Variabile linguistica di input *C*.

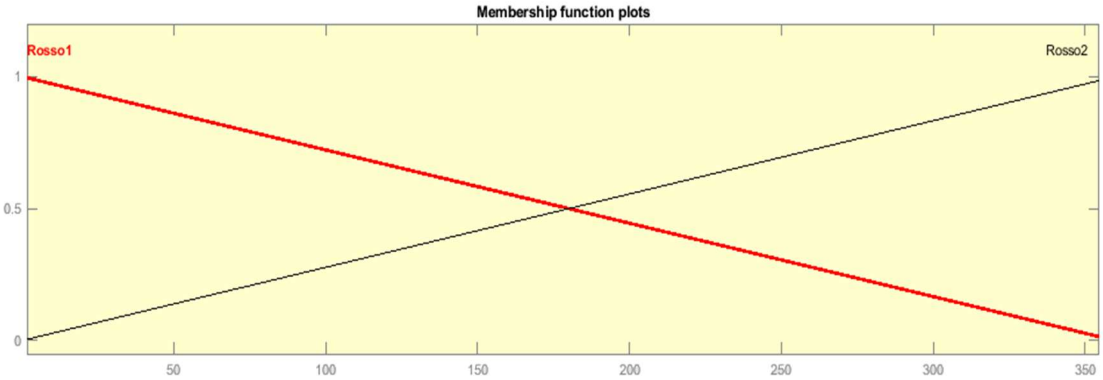


Figura 16.3: Variabile linguistica di input *hue*.

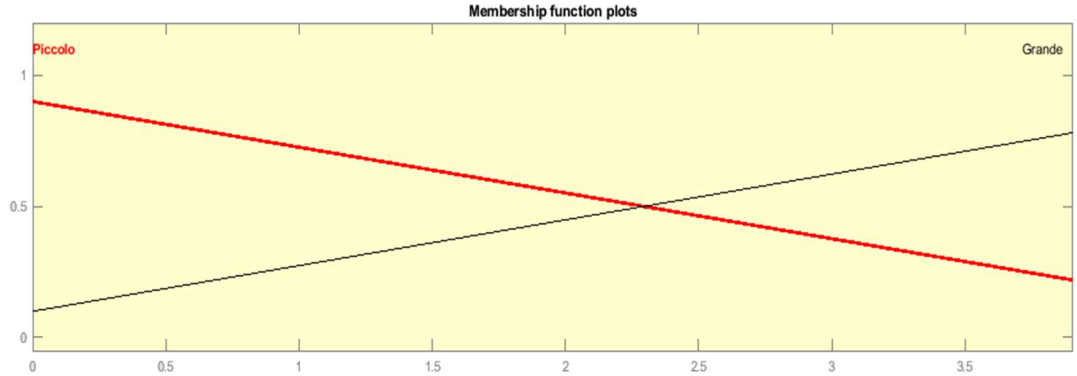


Figura 16.4: Variabile linguistica di input *dErrore*.

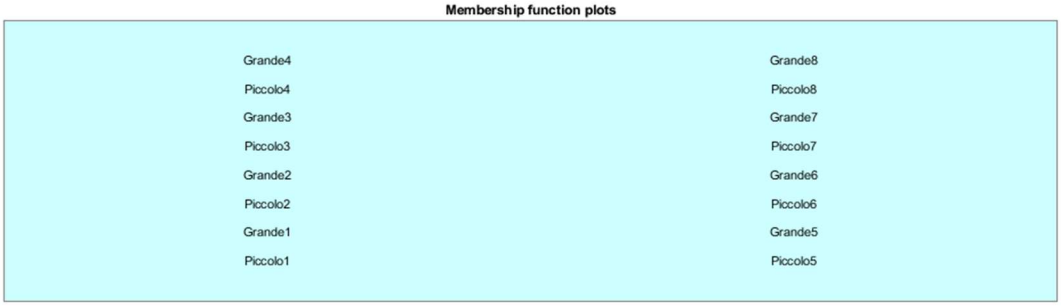


Figura 16.5: Variabile linguistica di output *Errore*.

4. Risultato

Usando il sistema fuzzy ottenuto in § 3.4, è stato generato il vettore `target` modificato come in § 3.3 per addestrare nuove reti neurali con la procedura mostrata in § 3.2. Con il test *t* di Student si è trovato che il numero di ottimo di neuroni nello strato nascosto è 5, Fig. 17.

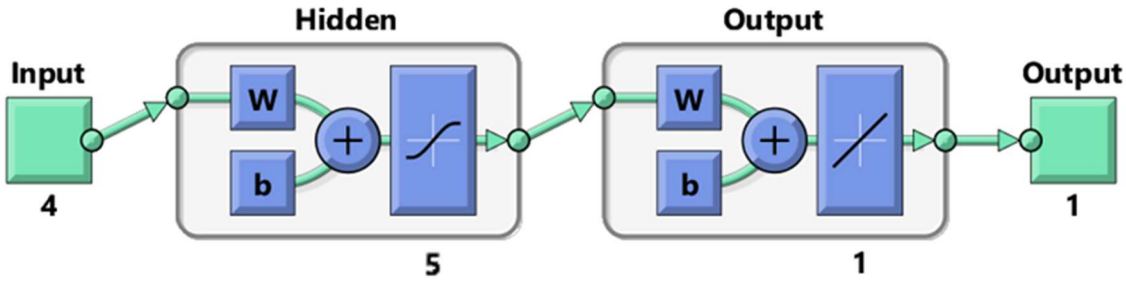


Figura 17: Rete neurale artificiale.

Dopo numerosi addestramenti l'andamento del coefficiente di regressione è mostrato in Fig. 18.1, 18.2 e 18.3. In Fig. 18.4 è visibile l'istogramma dell'errore sul dataset e in Fig. 18.5 l'istogramma dell'errore per dieci campioni esterni al dataset.

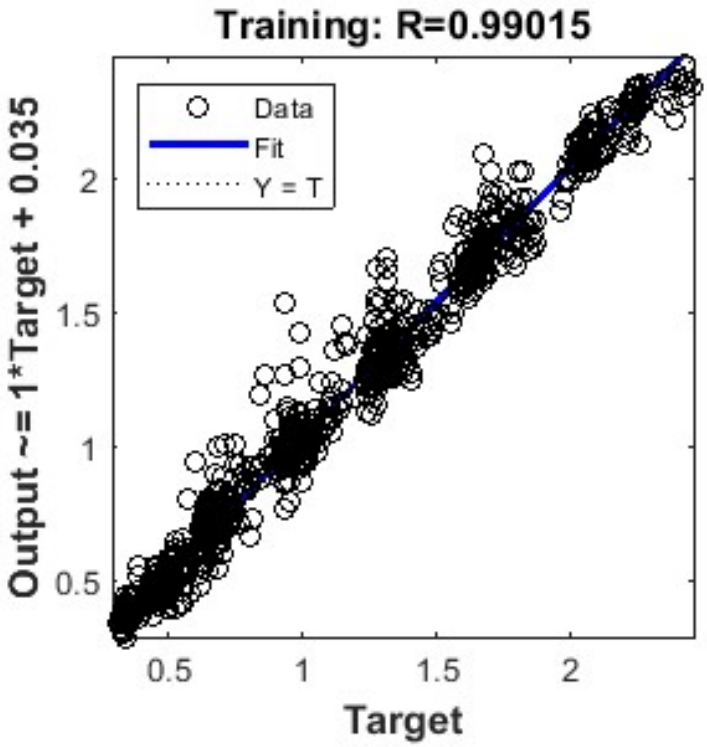


Figura 18.1: Curva di regressione per il training set.

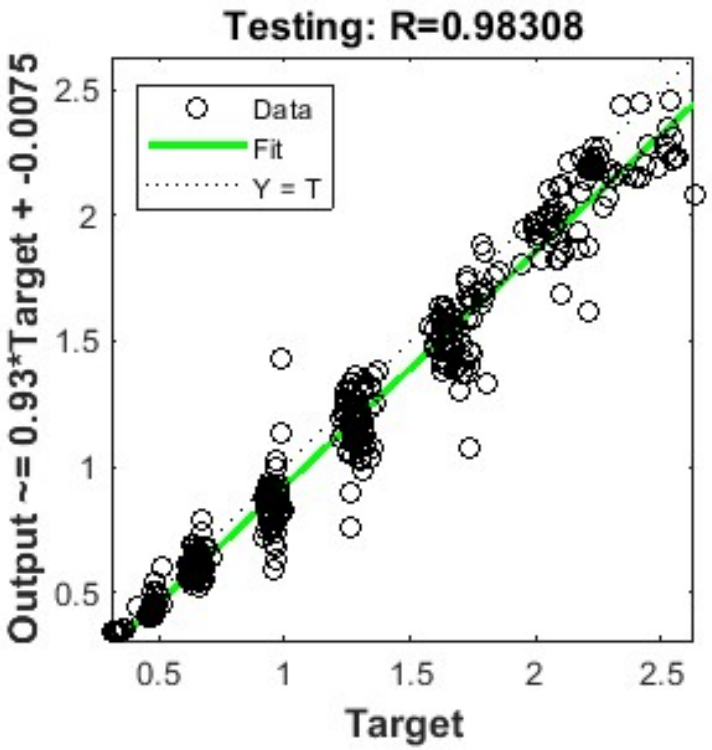


Figura 18.2: Curva di regressione per il test set.

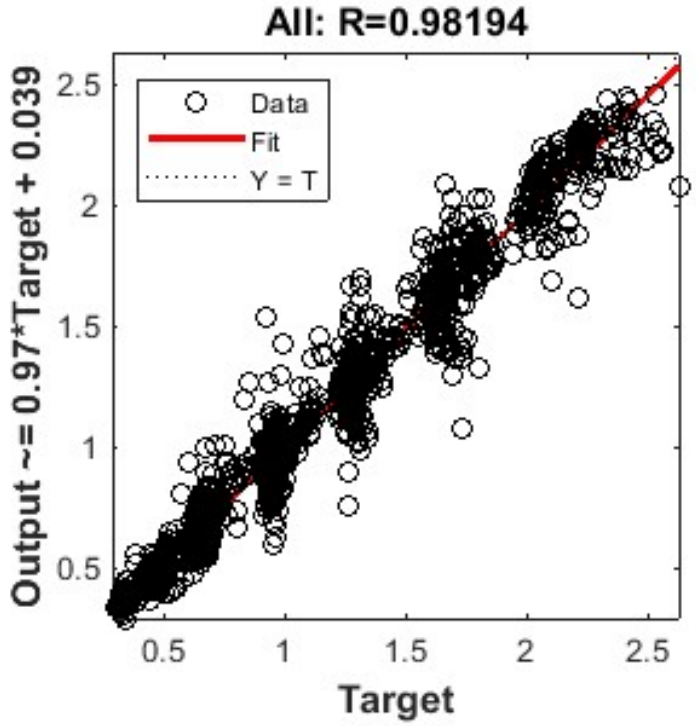


Figura 18.3: Curva di regressione totale.

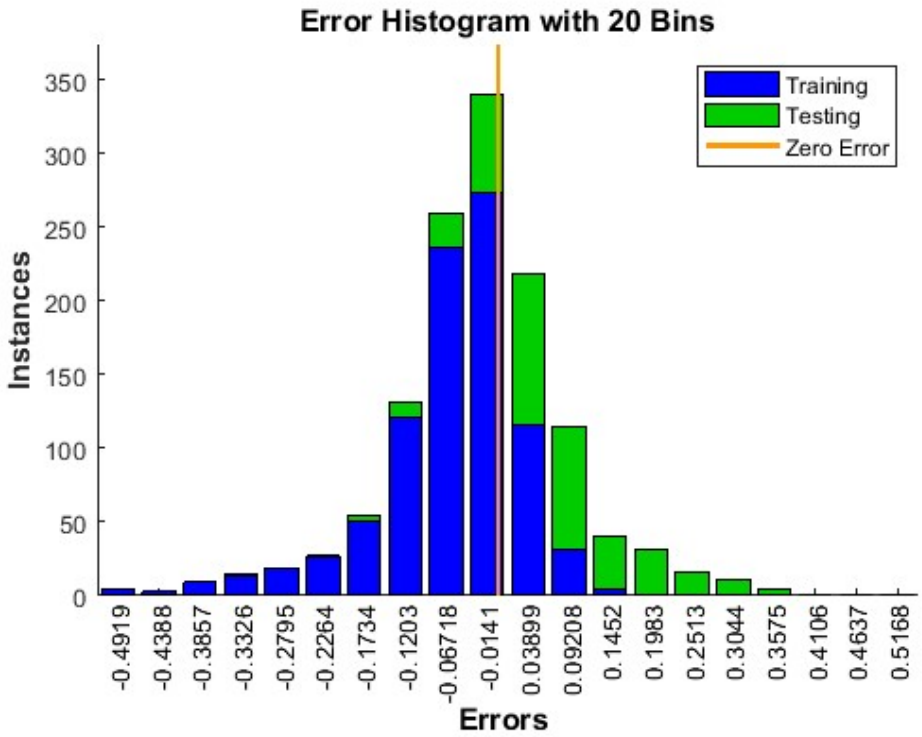


Figura 18.4: Istogramma dell'errore sul dataset.

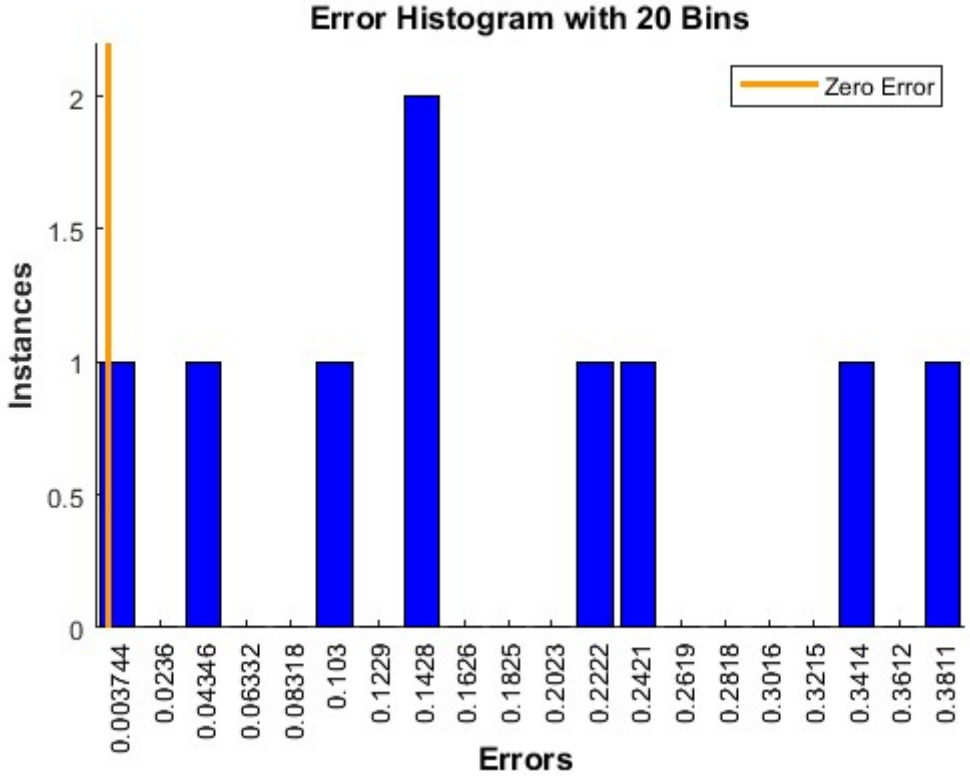


Figura 18.5: Nell'istogramma sono mostrati gli errori di dieci campioni non presenti nel dataset.

La rete di Fig. 17 soffre di asimmetria[9], motivo per il quale se si invertono le feature prendendo minimo, media e mean abs della copia, e la deviazione standard del colore master, il risultato è non coerente con quanto mostrato nei grafici precedenti. In Fig. 18.6 è mostrato il risultato dell'approssimazione e l'istogramma dell'errore.

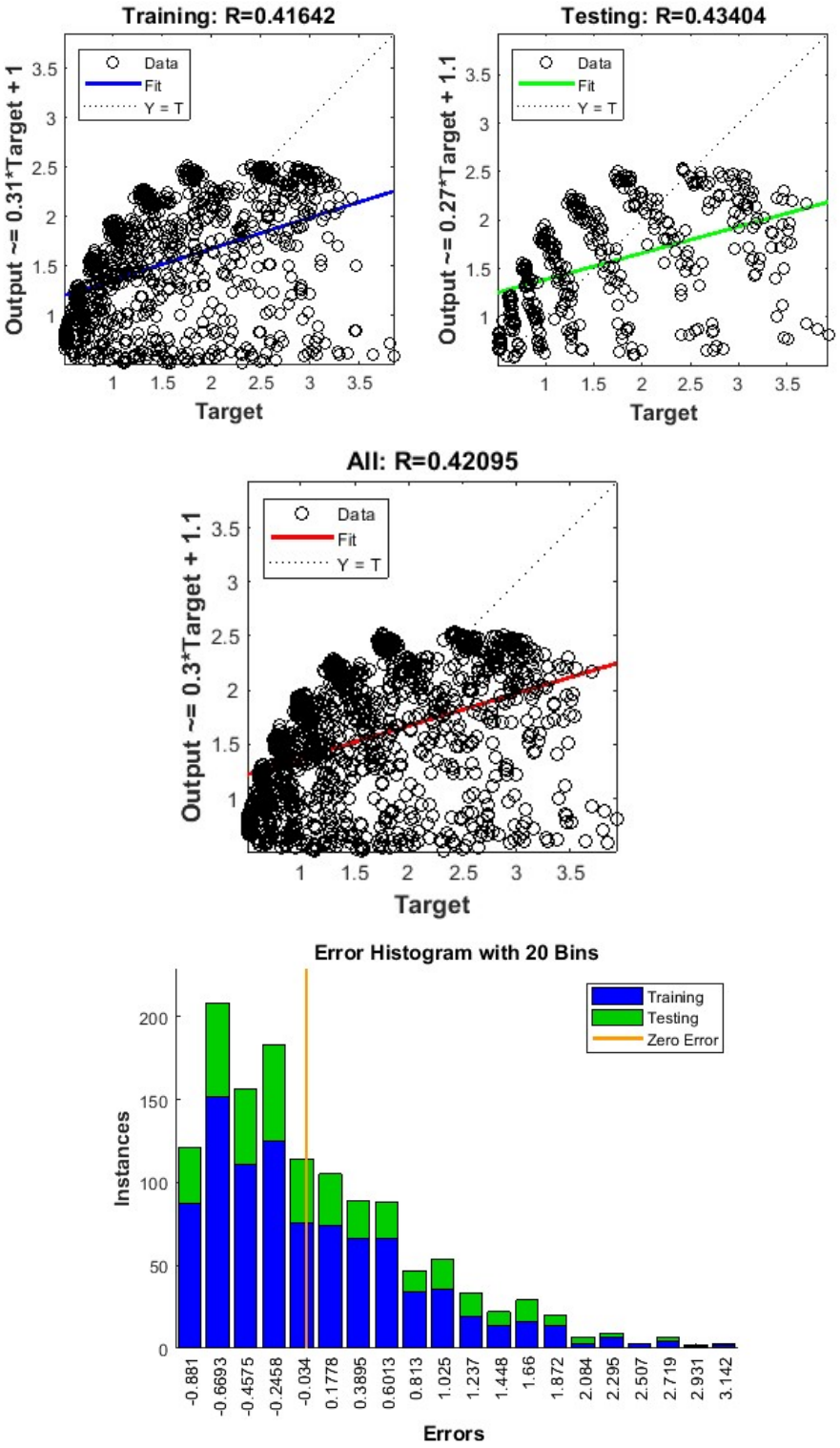


Figura 18.6: Performance della rete invertendo le caratteristiche di master e copia.

[9] Per un approccio simmetrico guardare in Approfondimenti § 2.

1. Realizzazione variabili linguistiche e regole fuzzy

Il sistema fuzzy presentato in § 3.2 è stato ottenuto dopo molti tentativi.
Inizialmente, basandosi anche sul materiale disponibile in letteratura, si era provato un approccio mirato, anziché generico come si è visto in § 3.2. Le funzioni di appartenenza ricoprivano dei range ben precisi, infatti si era ipotizzato di correggere solamente valori `dErrore` di colori etichettati come speciali: i colori molto chiari, molto scuri, i grigi e quelli appartenenti alla regione del blu-violetto.
Le figure 19.1, 19.2, 19.3, 19.4 e 19.5 mostrate di seguito spiegano come erano modellizzate le variabili linguistiche.

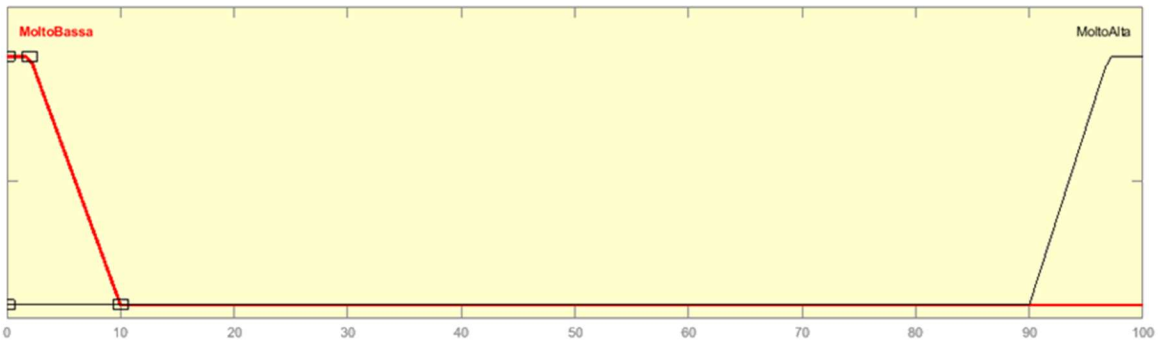


Figura 19.1: Variabile linguistica di input *L*.

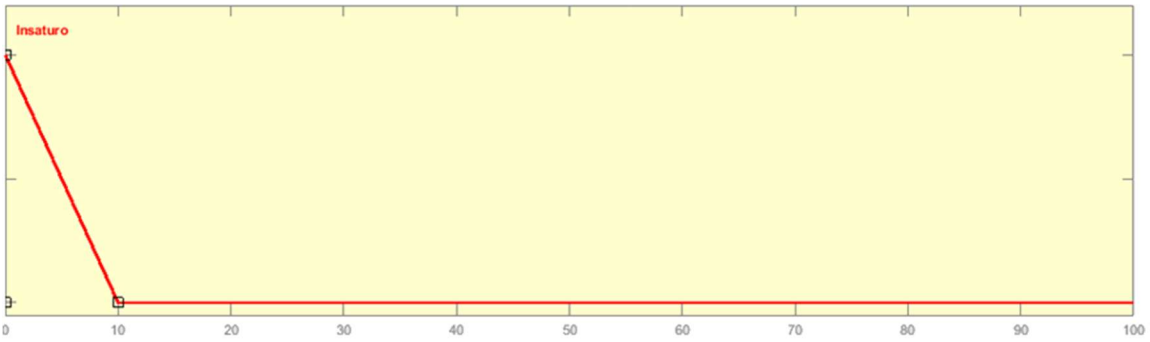


Figura 19.2: Variabile linguistica di input *C*.

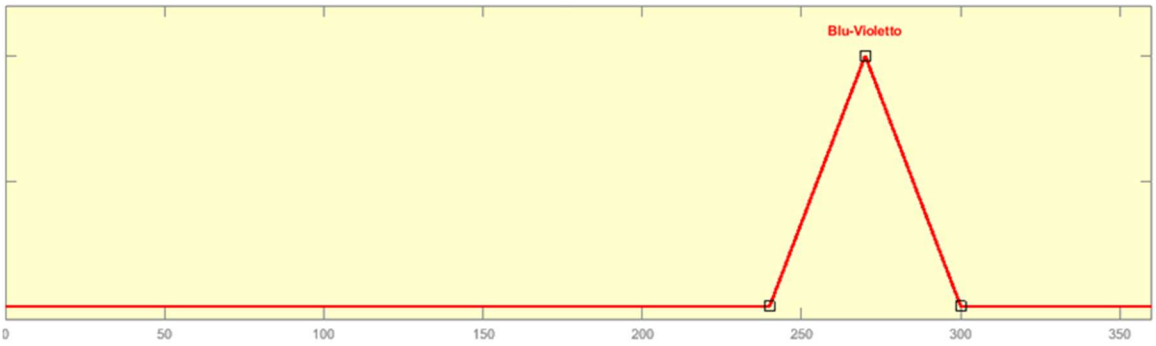


Figura 19.3: Variabile linguistica di input *hue*.

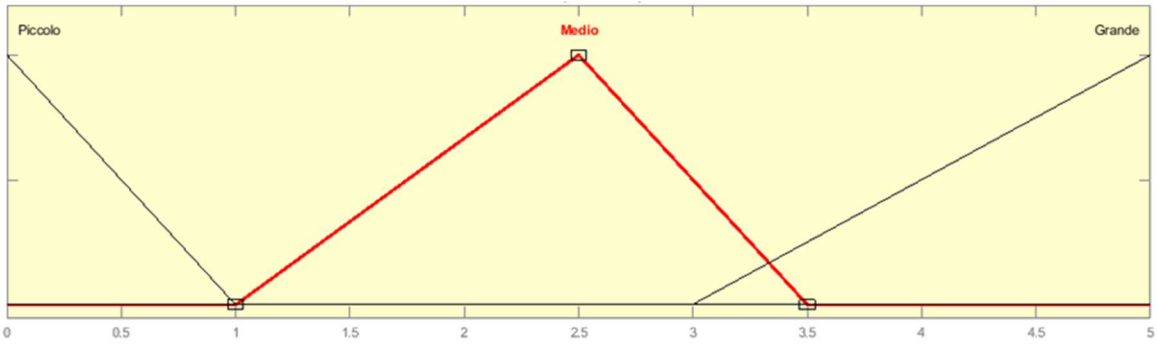


Figura 19.4: Variabile linguistica di input *dErrore*.

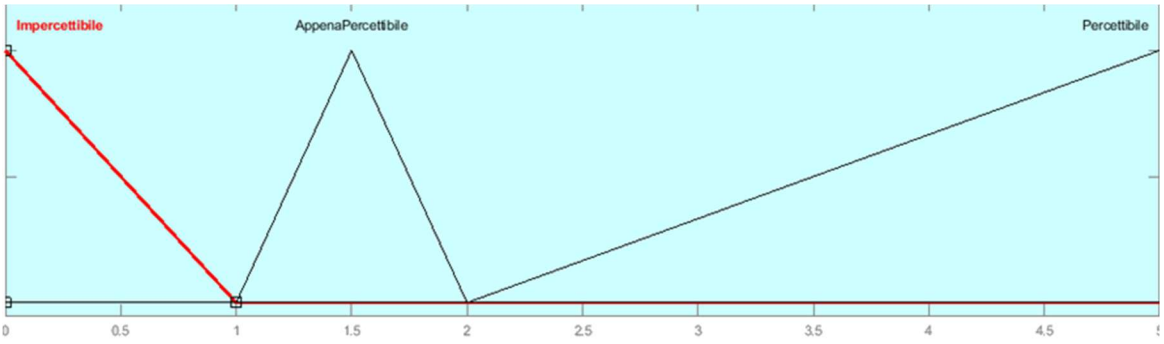


Figura 19.5: Variabile linguistica di output *Errore*.

La tabella di seguito è stata costruita in riferimento al significato assegnato alle etichette linguistiche delle variabili. Infatti, l'intenzione è quella di ammortizzare l'errore troppo "elevato" alla vista dell'osservatore, con l'obiettivo di rimodulare $dErrore \in [1, 3.5]$ nell'intervallo di $Errore \in [0, 1]$, dei colori appartenenti alle regioni troppo chiare, scure, grigie e del blu-violetto.

1	IF hue IS blu-violetto AND dErrore is Medio THEN Errore is Impercettibile
2	IF L is MoltoAlta AND dErrore is Medio THEN Errore is Impercettibile
3	IF L is MoltoBassa AND dErrore is Medio THEN Errore is Impercettibile
4	IF C is Insatura AND errore is medio THEN Errore is Impercettibile
5	IF dErrore is Piccolo THEN dErrore is Impercettibile
6	IF L is not MoltoAlta AND C is not Insatura AND hue is not Blu-Violetto AND dErrore is Medio THEN Errore is AppenaPercettibile
7	IF L is not MoltoBassa AND C is not Insatura AND hue is not Blu-Violetto AND dErrore is Medio THEN Errore is AppenaPercettibile
8	IF L is not MoltoAlta AND C is not Insatura AND hue is not Blu-Violetto AND dErrore is Grande THEN Errore is Percettibile
9	IF L is not MoltoBassa AND C is not Insatura AND hue is not Blu-Violetto AND dErrore is Grande THEN Errore is Percettibile

La rimodulazione con questo modello, e simili, non ha avuto successo. Molti valori, simulati attraverso l'opzione Rules del tool grafico Fuzzy Logic Designer, sono risultati falsi positivi. Per dimostrare l'inefficienza è stato generato il vettore $target$ modificato ed è stato utilizzato per addestrare la rete neurale come alla fine di §

3.3. In Fig. 20 e 21 si ripetono le curve di regressione illustrate in § 3.2 e § 3.3., mentre in Fig. 22 è illustrata la curva di regressione ottenuta con il modello precedente.

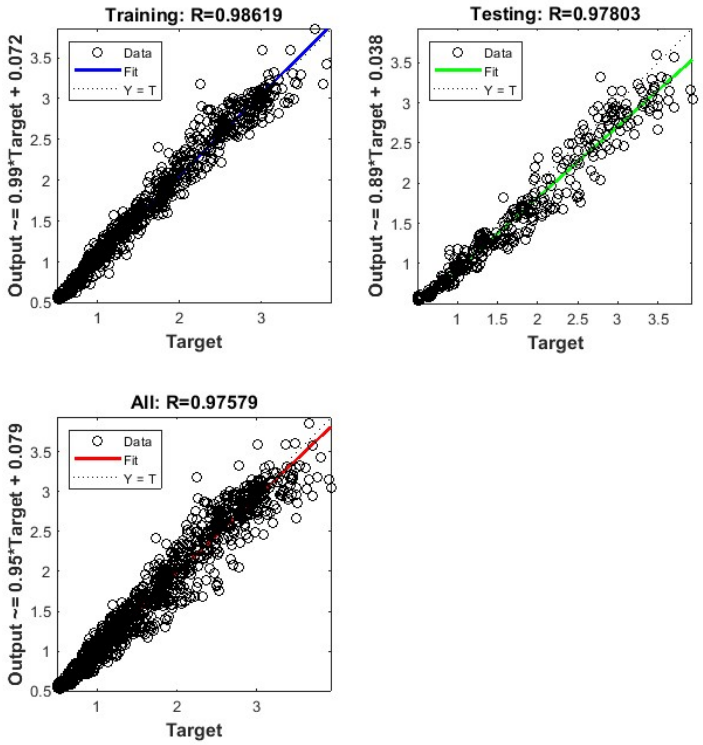


Figura 20: Curve di regressione mostrate in § 3.2.

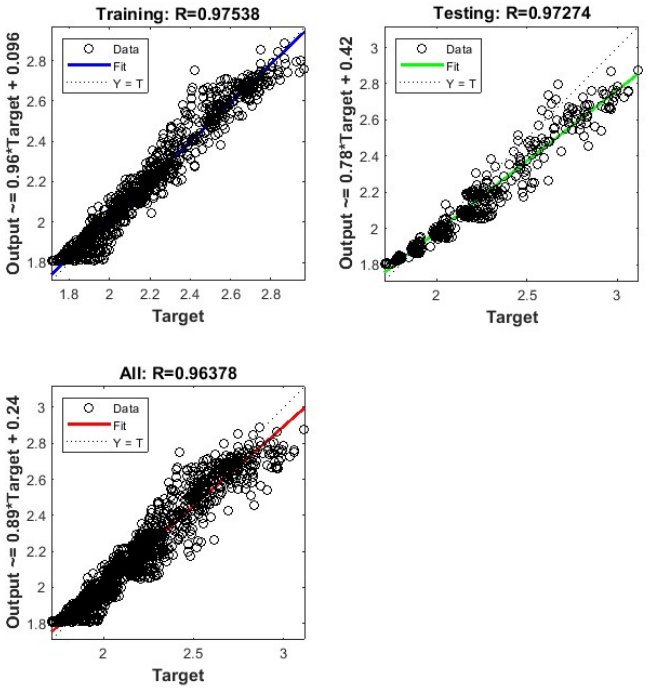


Fig. 21: Curve di regressione mostrate in § 3.3.

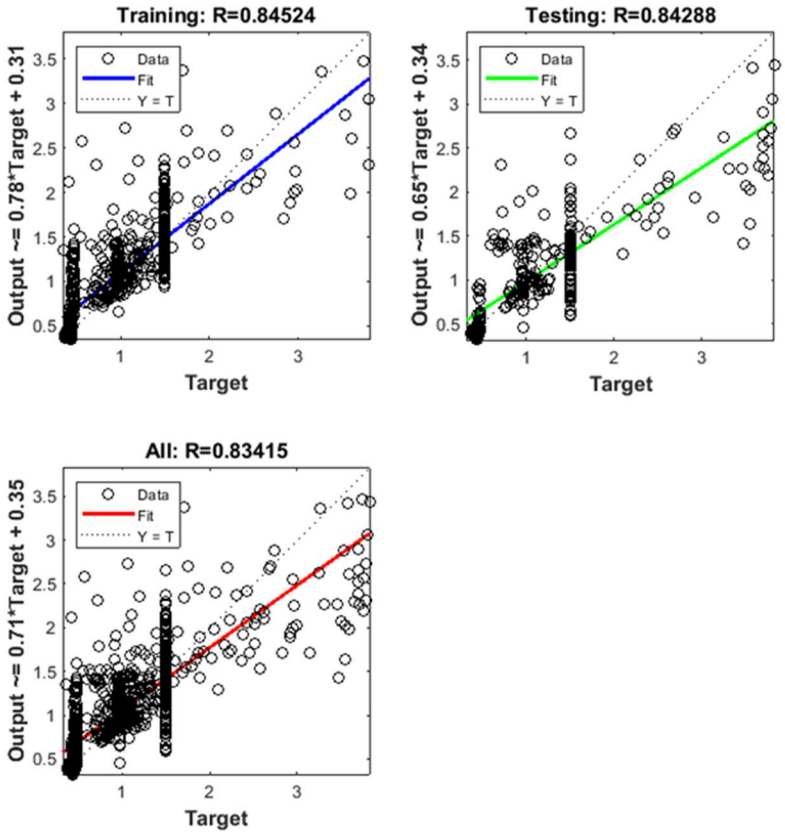


Fig. 22: Curve di regressione dei primi modelli.

A causa di continui insuccessi, le regole dei successivi modelli sono state realizzate con lo scopo di rendere di semplice conversione il sistema Mamdani al sistema Sugeno, così da addestrare il FIS con una rete ANFIS.

2. Twin sequential feature selection

Il sistema complessivo realizzato in § 4 è di tipo asimmetrico: le feature in ingresso sono diverse nella quantità e nel tipo per il master e per la copia colore. Per realizzare un sistema simmetrico si è rivelato necessario sviluppare una nuova forma di sequential feature selection, rinominata twin sequential feature selection (*tsfs*). Si ricorda in § 3.1 che la matrice `RAWFeatures` di dimensione $m \times n$, è composta di m campioni e n caratteristiche, le prime $n / 2$ caratteristiche appartengono all'insieme master, le successive $n / 2$ appartengono all'insieme copia nel medesimo ordine delle feature master. Quindi, quello che realizza la *tsfs* è prendere alla prima iterazione la caratteristica f_1 del master e la caratteristica f_1 della copia, alla seconda il subset $\{f_1, f_2\}$ del master e il subset $\{f_1, f_2\}$ della copia, alla terza $\{f_1, f_3\}$ e $\{f_1, f_3\}$, e così via… generando tutte le disposizioni semplici senza ripetizioni in ordine e prendendo ad ogni iterazione lo stesso insieme di caratteristiche del master e della copia colore.

```
1  % La funzione realizzare una twin sequential forward feature
2  % selection a due matrici di M colonne. La tsfs seleziona ad ogni
3  % iterazione le stesse colonne dell'una e dell'altra matrice.
4  function R = tsfs(input,target)
5
6      disp('Twin sequential features selection iniziata.');
7      features = size(input,2)/2;
8      dis = gendis(features);
9      dis = [dis,dis];
10     rows = size(dis,1);
11
12     [input, target] = AggiungiPadding(input,target);
13     errors = zeros(rows,1);
14
15     for i = 1:rows
16         disp(i)
17         features = find(dis(i,:));
18         errors(i) = kfoldcv(input(:,features)',target');
19     end
20
21     disp('Twin sequential features selection terminata.');
22     R = errors;
23
24
25 end
```

Per realizzare la *tsfs* viene chiamata `gendis()`, che restituisce D disposizioni semplici su f caratteristiche estratte per ciascun segnale, e successivamente viene eseguita la funzione `kfoldcv()`. `kfoldcv()` esegue due 10-fold cross-validation per ciascun subset di caratteristiche selezionate. Al termine la funzione restituisce l'errore medio sul test set, il quale sarà memorizzato in posizione i-esima del vettore `errors`.

```
function R = gendis(n)
    dim = zeros(n,1);
    for i = 1:n
        dim(i) = size(nchoosek(1:n,i),1);
    end
    dis = zeros(sum(dim),n);
    prec = 1;
    for i = 1:n
        dim = nchoosek(1:n,i);
        rows = size(dim,1);
        for r = 1:rows
           bin = zeros(1,n);
           for c = 1:i
               index = dim(r,c);
               bin(index) = 1;
           end
           dis(prec,:) = bin;
           prec = prec + 1;
        end
    end
    R = dis;
end
```

```
function R = kfoldcv(input,target)

    % Pre-allocazione matrice degli errori
    errors = zeros(20,1);
    samples = size(target,2);

    % Prepara la matrice alla KFoldCV
    [input,target] = RealizzazioneMatriceKFoldCV(input,target,sam-
ples);
    net = CostruzioneRete(5,samples);
    index = 1;

    for r = 1:2
        [input,target] = RotazioneMatrice(1,input,target);
        for i = 1:10
            [net,tr] = train(net,input,target);
            errors(index) = mean(tr.tperf); % Oppure tr.best_tperf
            [input,target] = RotazioneMatrice(0,input,target);
            index = index + 1;
        end
    end

    R = mean(errors);

end
```

Estraendo 12 caratteristiche, sono state addestrate 4095 reti neurali con feature differenti. In Fig. 23 è possibile esaminare la distribuzione dell'errore per gruppi di feature aventi stesso numero di caratteristiche.

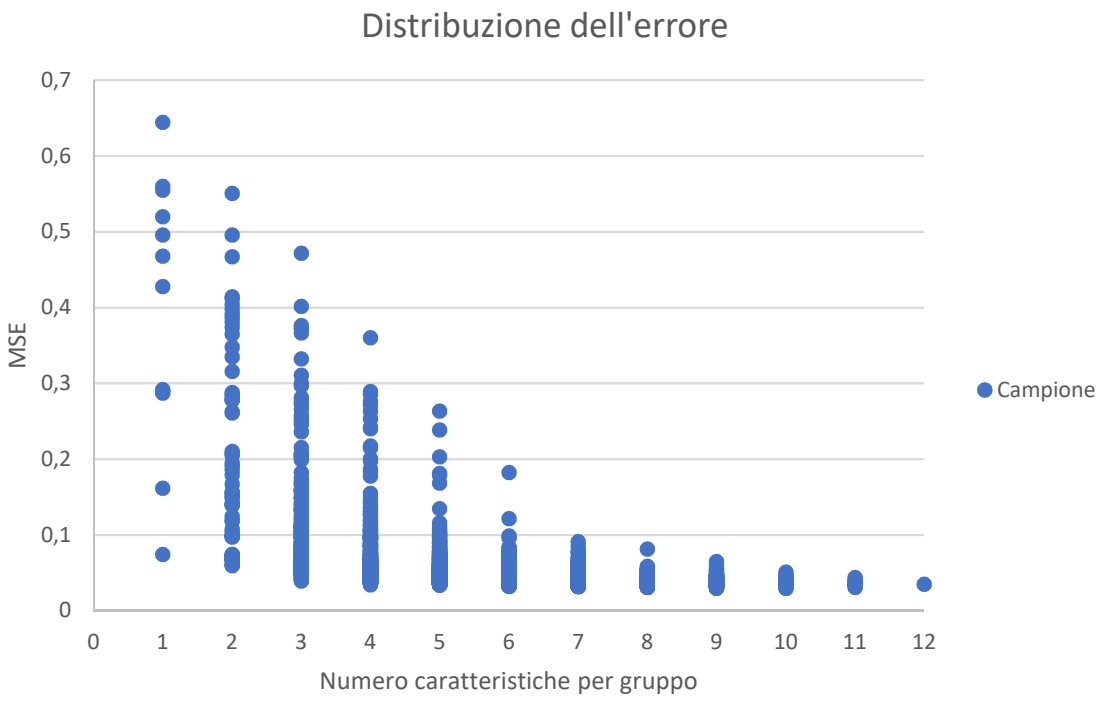
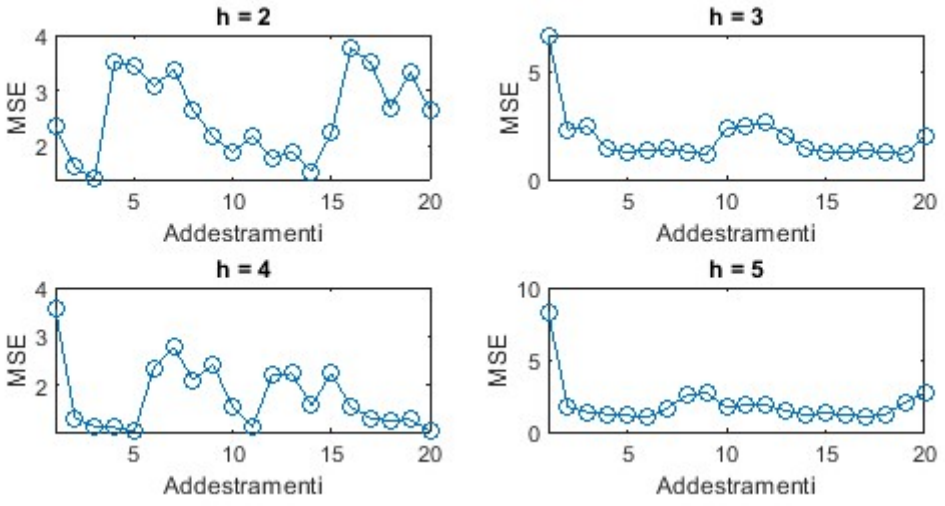
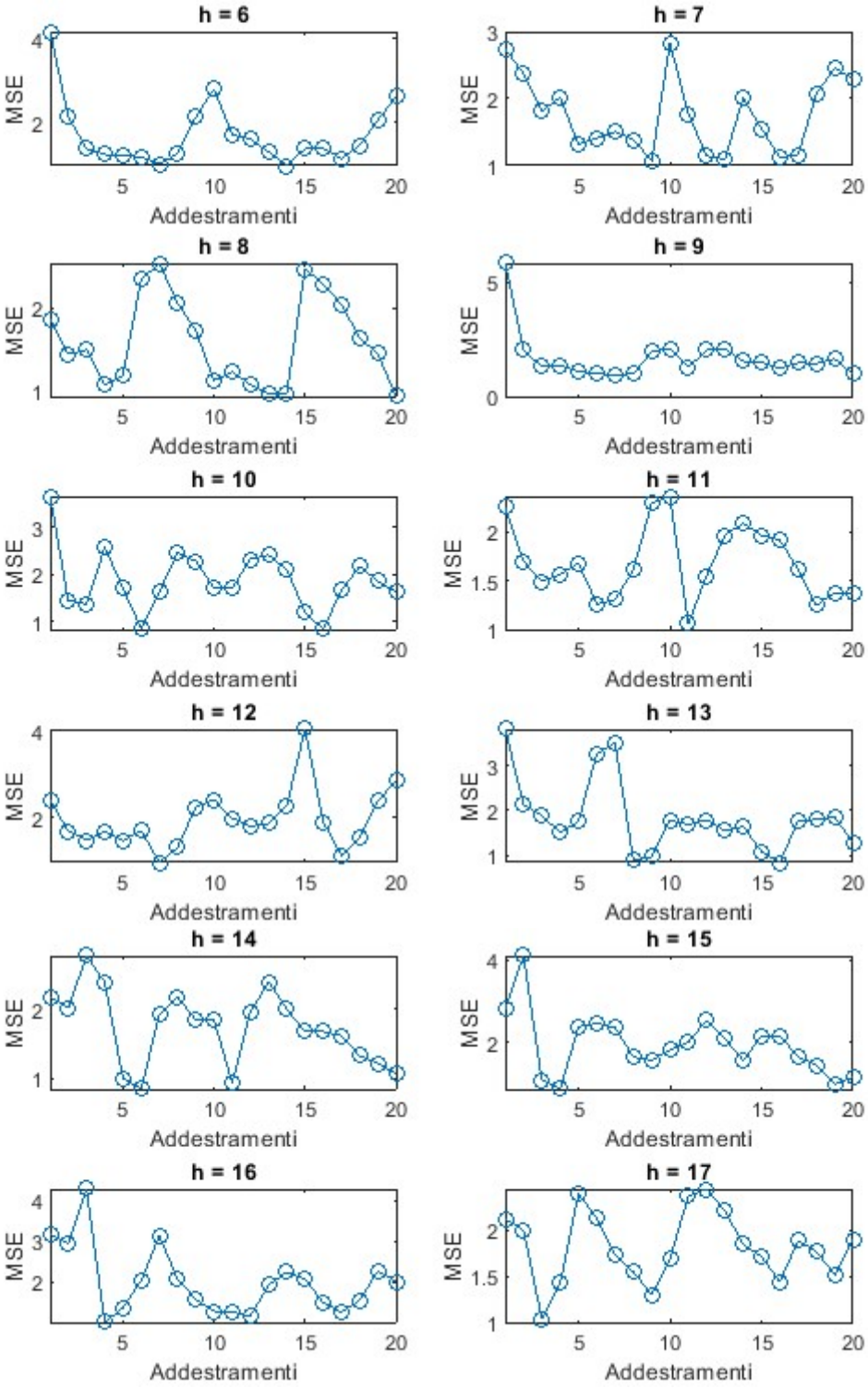


Figura 23: Distribuzione dell'errore per ciascun gruppo di feature eseguendo *tsfs*.

Osservando la distribuzione dell'errore per ciascun gruppo aventi stesso numero di feature, è stato pensato di utilizzare un gruppo con numero di caratteristiche maggiore o uguale a sei per addestrare la rete neurale come in § 3.2. Quindi, si è realizzata una matrice `input` $m \times n$, dove m = 1269 campioni e n = 12 feature, dove le prime sei sono del master, le successive sei della copia.
Le caratteristiche utilizzate sono le sei del gruppo che ha generato l'errore minore: massimo, media, mediana, skewness, ampiezza e quartile.
Nelle prossime pagine sono visibili i grafici della distribuzione dell'errore durante la *k*-fold.




h = 6
h = 7
h = 8
h = 9
h = 10
h = 11
h = 12
h = 13
h = 14
h = 15
h = 16
h = 17
MSE
Addestramenti

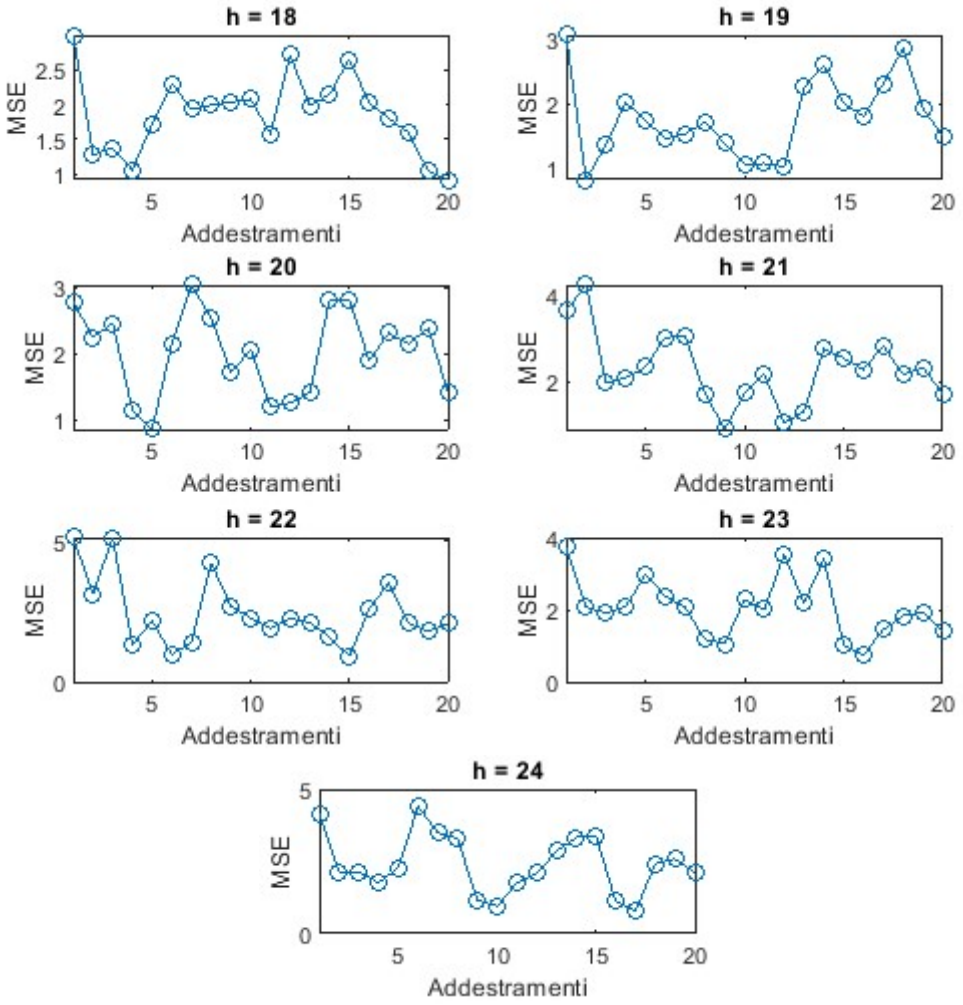


Figura 24: Distribuzione degli errori di ciascuna rete durante i due cicli di 10-fold.

Con il test *t* di Student si è trovato che il numero di neuroni ottimo con cui addestrare la rete è 8.

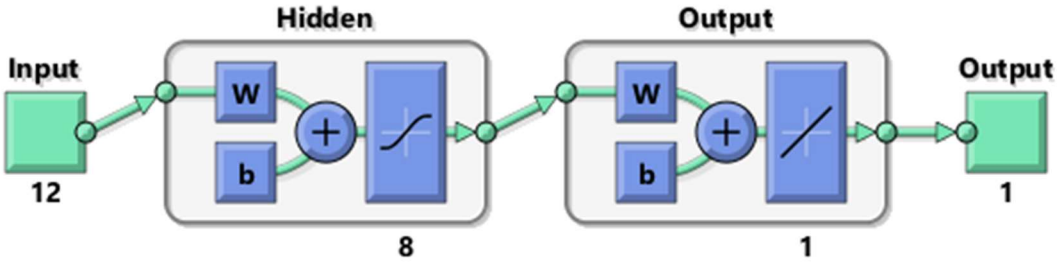


Figura 25: Rete neurale artificiale.

Dopo numerosi addestramenti la rete neurale migliore è stata salvata.
Le curve di regressione e l'istogramma degli errori sono mostrati in Fig. 26.

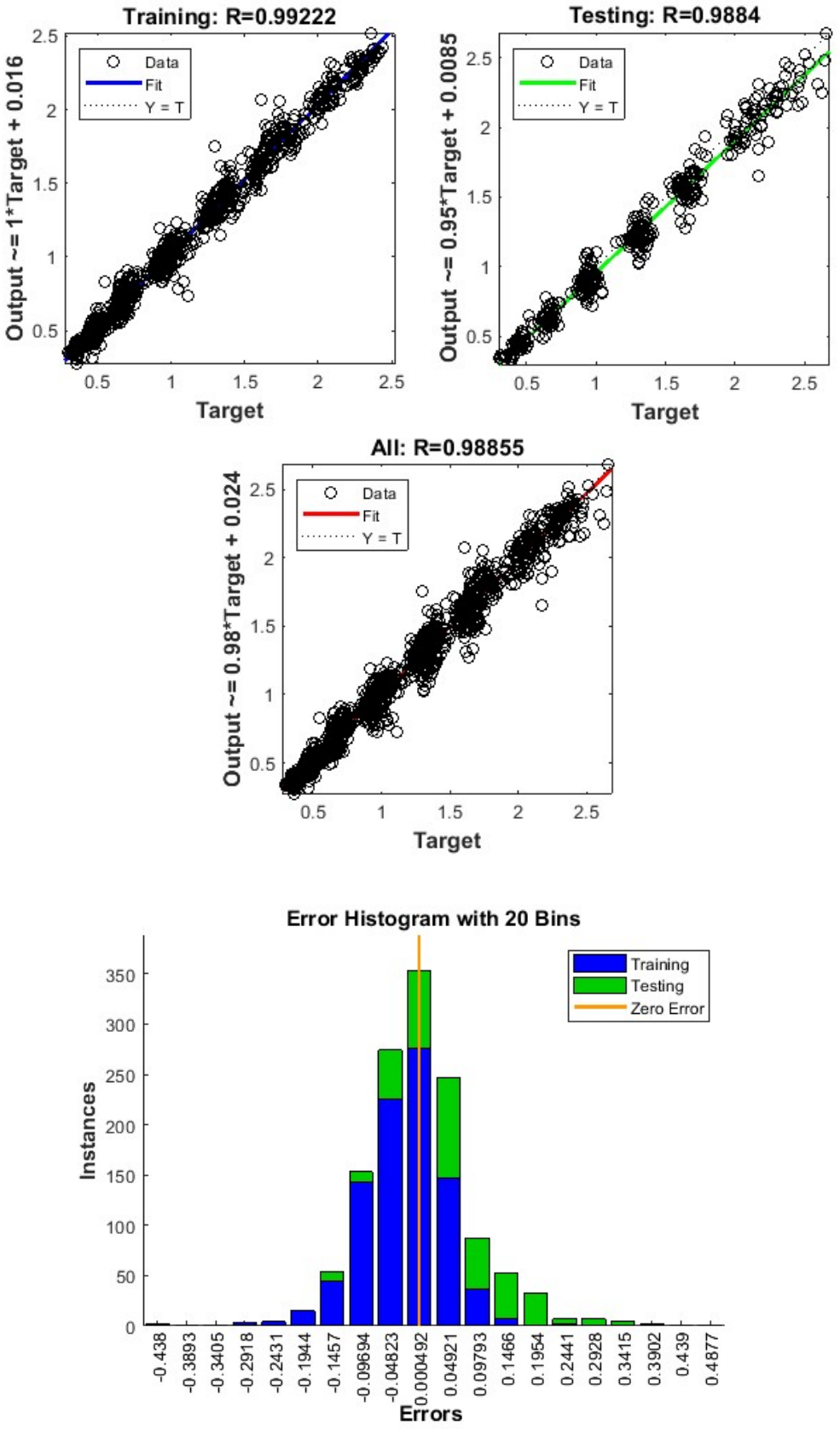


Figura 26: Curve di regressione e istogramma degli errori della rete ottima simmetrica.

Per concludere la sperimentazione, nell'istogramma di Fig. 27, sono mostrati gli errori su dieci campioni non presenti nel dataset usato per addestrare e testare la rete neurale.

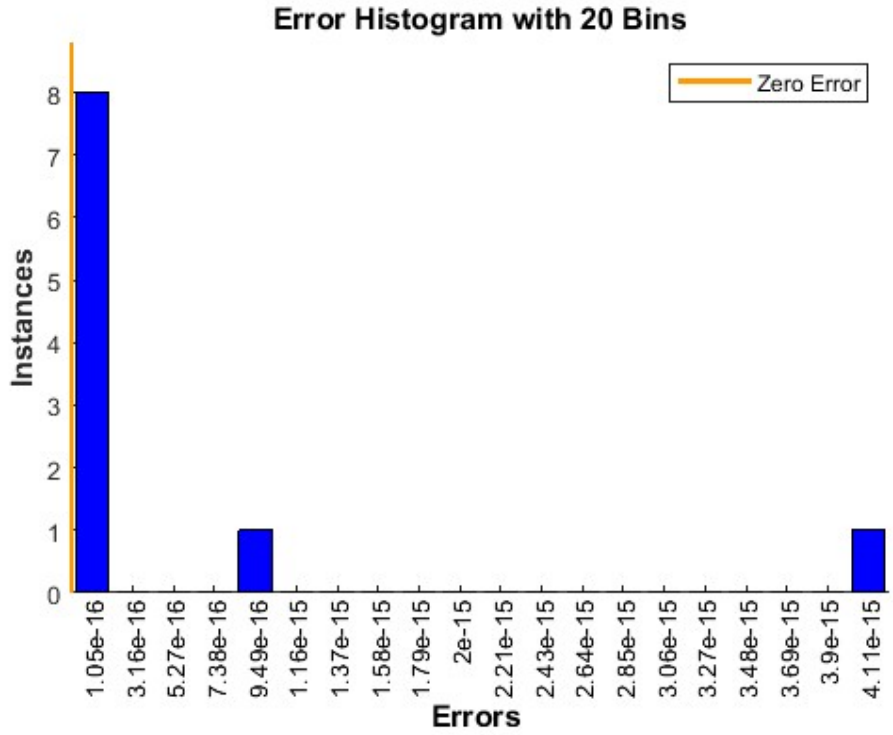


Figura 27: Nell'istogramma sono mostrati gli errori di dieci campioni.