Consiglio Nazionale delle Ricerche

Nota Interna

Analisi di Immagini Mediante Reti Neurali ad Architettura Gerarchica

Gabriele Pieri

B4-46
nov-2001

I.E.I.
ISTITUTO DI
ELABORAZIONE DELLA
INFORMAZIONE

Pisa

S.T.A.R.
Servizio Tecnografico Area della Ricerca del CNR – Pisa
Responsabile Salvatore La Polla
dicembre '01 283,86

COVER DESIGN: silvia rigobon

Consiglio Nazionale delle Ricerche

Analisi di Immagini Mediante Reti Neurali ad Architettura Gerarchica

Gabriele Pieri

B4-46
nov-2001

Analisi di Immagini Mediante Reti Neurali ad Architettura Gerarchica

Gabriele Pieri

Istituto di Elaborazione dell'Informazione – CNR
Via G. Moruzzi 1, 56124 PISA
Email: pieri@iei.pi.cnr.it

Indice

1. Introduzione

La classificazione tramite reti neurali è un campo in forte espansione, e data la gran varietà di possibili applicazioni, e la vasta gamma di diversi algoritmi studiati, o utilizzati per la realizzazione di tali reti, sembra esserci spazio per la ricerca di tecniche che possano portare dei miglioramenti, in termini sia di performance sia di velocità.

In generale, le applicazioni che ne fanno uso, basano la loro efficacia sulla capacità che queste hanno di riuscire ad estrarre caratteristiche comuni sia quando le informazioni in ingresso sono molto complesse, sia in situazioni in cui il dato da analizzare sia corrotto o affetto da rumore. Questa capacità di "apprendere" ed "astrarre" rende le reti neurali uno strumento molto potente rispetto ad altre tecniche classiche di classificazione statistica.

Tra i vari algoritmi utilizzati per l'apprendimento delle reti neurali artificiali (ANN), non si riesce a trovarne nessuno che abbia una supremazia netta sugli altri, ogni tecnica (algoritmo) trova sempre il suo utilizzo per un particolare compito, per il quale si rivela più adatto di un altro algoritmo.

Due sono i punti sui quali orientare principalmente la ricerca:

- Miglioramento degli algoritmi esistenti o ricerca di un algoritmo con prestazioni superiori agli altri
- Studio e realizzazione di un'architettura complessa che faccia uso degli algoritmi già esistenti, per sommare i benefici ed eliminare i punti negativi dei singoli algoritmi

Data anche la varietà ed il numero di algoritmi esistenti attualmente si è scelto di spostare l'attenzione su questo secondo punto, ed in questa nota saranno presentati alcuni sviluppi in questa direzione. Utilizzando algoritmi già esistenti sarà mostrata un'architettura che cerca di sfruttare e mettere assieme, combinandole, le capacità di vari algoritmi ed allo stesso tempo di limitare i punti deboli di ciascuno. Tutto questo rimanendo sempre più competitiva possibile in termini di tempi di elaborazione.

Tale architettura prende il nome di *Architettura Gerarchica*.

2. Classificare in modo gerarchico

La classificazione, tramite l'estrazione di caratteristiche (*features*) peculiari, di un'informazione, può essere affrontata in modo diretto con un unico modulo di

classificazione, oppure con uno più complesso, composto a sua volta da sotto-classificatori, questa seconda tecnica fa uso di un'architettura gerarchica [3].

Un classificatore progettato in questo modo, grazie alla continua crescente velocità di elaborazione raggiunta dalla tecnologia, può migliorare molto le prestazioni di riconoscimento rispetto ad un singolo classificatore senza rallentare le prestazioni in maniera eccessiva.

Il classificatore con un'architettura gerarchica può coniugare, infatti, gli aspetti positivi di differenti tecniche di classificazione, cercando allo stesso tempo di ridurre o nascondere gli eventuali difetti di ogni singola tecnica.

3. Struttura dell'Architettura Gerarchica

Un'architettura di questo tipo consiste generalmente di due livelli di classificatori disposti gerarchicamente tra loro (vedi anche [4]). Le features estratte dai dati da elaborare sono utilizzate come ingresso per il livello gerarchico più basso (Livello 0 in Figura 1), questo, elabora tali caratteristiche e invia i suoi risultati al livello di

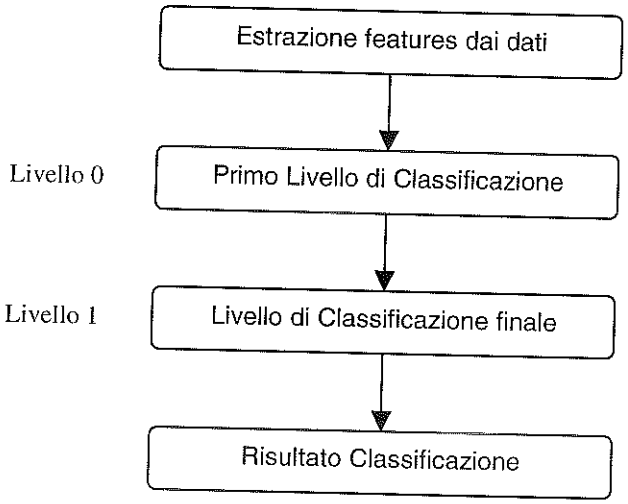

Figura 1

classificazione più alto (Livello 1 in Figura 1), il quale basa la sua classificazione su quest'ultimo risultato e restituisce in uscita la classificazione finale dell'architettura (Figura 1).

In generale, il primo livello di classificazione (*livello di specializzazione*) può essere composto da diverse copie dello stesso tipo di classificatore, ogni singolo classificatore

può essere inteso come specializzato per la classificazione di una particolare caratteristica (*feature*) dei dati elaborati, quindi le features estratte non sono date in input ripetute a tutti i classificatori di primo livello, ma sono suddivise sui singoli classificatori.

Un'altra possibilità, per il primo livello, è quella di avere invece un numero di classificatori diversi (es. diverse reti neurali basate su differenti algoritmi, tecniche di Intelligenza Artificiale...), ad i quali vengono però forniti in input tutte le features estratte.

Il livello finale consiste invece generalmente di un unico classificatore (*livello decisionale*), il cui scopo è di "raccogliere" le classificazioni assegnate all'input dai singoli elementi del livello inferiore, per estrarre a sua volta un risultato da presentare in uscita.

Data la struttura di quest'architettura ed il tipo di dati che si devono elaborare si può pensare di avere dei classificatori più veloci per il livello di elaborazione inferiore e un classificatore, anche meno veloce, ma più accurato, per il livello finale.

Questo perché è presumibile che i dati in input all'architettura, vale a dire le features estratte, siano di dimensione maggiore (anche molto maggiore), rispetto a quelli che sono i dati in ingresso al classificatore finale, questi sono semplicemente i valori (etichette) di classificazione del livello inferiore, quindi saranno in numero pari al numero di elementi del primo livello.

4. Caso di Studio su Immagini

Un campo che riveste un interesse molto ampio nella ricerca, per quanto riguarda la classificazione ed il riconoscimento automatico tramite reti neurali artificiali, è quello del trattamento delle immagini. Questo tipo di elaborazione è utilizzata in numerosi campi di studio

- ✓ campo medico
- ✓ riconoscimento di difetti tessutali o di prodotti industriali
- ✓ classificazione automatica delle immagini da tele-rilevamento

ed in generale, per la modellazione di fenomeni complessi ove un certo numero di variabili (solo quantitative oppure sia quantitative che qualitative) influiscono sulla risposta in output del fenomeno (che può essere a sua volta quantitativa o qualitativa).

Un ruolo importante lo assume la scelta delle features significative, da estrarre dalle immagini, su cui allenare la rete. Tale scelta influenza sia la complessità dei dati di input (e quindi la velocità), sia la differenziabilità tra i dati (cioè, in ultimo, la performance sul riconoscimento). La scelta e l'estrazione di queste caratteristiche peculiari non fa parte del processo di allenamento/classificazione delle reti neurali, dato che avviene fuori da questa ed è precedente, ma dato che rappresenta la codifica dell'informazione che la rete classifica, è importante scegliere features che rappresentino il più possibile vari e diversi aspetti dell'informazione.

Nella scelta dei classificatori da utilizzare per i singoli livelli si è cercato di adottare il seguente criterio: il livello più basso, di specializzazione, avendo in output tutti i dati, doveva essere meno pesante, anche se meno accurato del livello più alto (decisionale), il quale invece, utilizzando solo le classificazioni del livello inferiore, quindi un input meno grande, può utilizzare un algoritmo di classificazione più oneroso computazionalmente. Nell'architettura elaborata, gli algoritmi scelti per le reti neurali dei due livelli sono quindi:

- Self-Organising Maps (SOM) per il livello di specializzazione
- Error Back-Propagation (BP) per il livello decisionale

Vediamo brevemente le caratteristiche significative e il metodo di funzionamento di questi due algoritmi.

4.1. Rete Neurale basata sulle SOM

La rete utilizzata per il livello di specializzazione è basata sulle Self-Organising Maps (SOM) [1], un'architettura di tipo auto-organizzante in grado, da una parte di riconoscere le caratteristiche significative dei dati in ingresso e dall'altra di definire un ordine topologico sullo spazio delle caratteristiche d'ingresso in modo del tutto automatico. Una SOM è una rete neurale artificiale monostrato in cui ogni neurone è connesso con tutti quei nodi vicini la cui distanza risulta inferiore ad un valore fissato, detto raggio d'interazione. La procedura di addestramento, segue uno schema senza supervisione, in cui i pesi vengono modificati facendo sì che la rete si specializzi ad individuare determinate caratteristiche delle features estratte. Neuroni vicini risultano eccitati da *pattern* simili e, una volta terminata la fase di apprendimento, sulla rete viene definito un ordine topologico.

In linea generale, la legge di apprendimento per questo tipo di rete neurale, è un algoritmo che fa "muovere" uno dei pesi della rete verso l'elemento che è a lui più simile, tra tutti quelli dell'insieme di apprendimento. Tale *nodo* è detto vincitore della competizione e viene spostato, di una porzione della distanza che lo separa dall'elemento del set di training, verso di lui. La percentuale di tale distanza è determinata dal coefficiente della legge di apprendimento.

Più formalmente, sia data una sequenza di input $\{\mathbf{x}\} \in \Re^n$ e un insieme di vettori di riferimento $\mathbf{m}_i \in \Re^n$ associati a ciascun nodo della rete inizializzati opportunamente. Ad ogni istante t, viene preso un elemento $\mathbf{x}(t)$ dall'insieme d'ingresso e confrontato con i valori di riferimento $\mathbf{m}_i(t)$ utilizzando una misura per la distanza $d(\mathbf{w},\mathbf{v})$ definita su $\Re^n$, con $\mathbf{v},\mathbf{w} \in \Re^n$.

Il valore $\mathbf{m}_i(t)$ che risulta più simile a $\mathbf{x}$ viene aggiornato in modo tale che la somiglianza sia ancora più marcata, ovvero che $\mathbf{m}_i(t)$ sia più vicino ad $\mathbf{x}$, se $\mathbf{x}$ e $\mathbf{m}_i(t)$ risultano appartenenti alla stessa classe. In questo modo i diversi vettori di riferimento si sintonizzano su diversi valori del dominio dello spazio degli input.

La quantità $d(\mathbf{x},\mathbf{m}_c) = \min_i d(\mathbf{x},\mathbf{m}_i), i = 1,2,\ldots,k$, con k numero totale dei nodi della rete, definisce il *mapping* dell'input $\mathbf{x}$ sul nodo il cui peso è $\mathbf{m}_c$. Se si considera la distanza euclidea su $\Re^n$ si ottiene:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \\ \vdots \\ x_n \end{bmatrix}, \mathbf{m}_i = \begin{bmatrix} m_1^i \\ m_2^i \\ \\ \\ m_n^i \end{bmatrix}, \|\mathbf{x} - \mathbf{m}_i\| = \sqrt{\sum_{h=0}^{n} (x_h - m_h^i)^2} \qquad (1)$$

Il parametro $\alpha(t) \in]0,1[$ è detto coefficiente d'apprendimento o guadagno e decresce linearmente in funzione del tempo. Le fasi che regolano l'utilizzo della rete sono regolate principalmente dalla scelta del numero di iterazioni sugli input campione e da $\alpha(t)$.

Ciò che caratterizza l'addestramento delle SOM, invece, è che esso avviene grazie a due meccanismi, detti d'inibizione e d'eccitazione laterale:

Inibizione laterale: i neuroni della rete concorrono fra di loro finché ciascuno si specializza nel riconoscimento di una particolare feature in ingresso:

$$\|\mathbf{x}-\mathbf{m}_c\| = \min_i \|\mathbf{x}-\mathbf{m}_i\|. \qquad (2)$$

Il nodo c, vincitore della competizione, è aggiornato in modo tale che la sua distanza dall'input sia ogni volta minore. In questo modo i diversi vettori di riferimento si sintonizzano su diversi valori del dominio dello spazio degli input.

Eccitazione laterale: ad ogni passo dell'algoritmo di apprendimento vengono aggiornati anche i pesi di quei nodi che si trovano nelle vicinanze di c. In questo modo neuroni adiacenti vengono eccitati da *pattern* simili.

Fissato un raggio $r(t)$ e preso un intorno del nodo i della rete, si definisce N_i come l'insieme dei nodi "vicini" di i. L'ampiezza del raggio dell'intorno di un nodo $r(t)$ viene fatta variare nel tempo cosicché anche il numero dei nodi dell'intorno $N_i(t)$, risulta dipendente da $r(t)$. All'inizio della computazione si imposta un raggio d'interazione ampio in modo tale da permettere, un primo ordinamento dei pesi; man a mano che la misura del raggio decresce si ha una progressiva sintonizzazione dei pesi verso determinate features.

La legge che regola l'aggiornamento dei pesi dei neuroni per le SOM è la seguente:

$$\mathbf{m}_i(t+1) = \begin{cases} \mathbf{m}_i(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_i(t)] & \text{se} \quad i \in N_i(t) \\ \mathbf{m}_i(t) & \text{se} \quad i \notin N_i(t) \end{cases}, \qquad (3)$$

ed anche in questo caso il parametro $\alpha(t) \in]0,1[$ decresce linearmente in funzione del tempo. L'aggiornamento dei pesi può essere riscritto in forma più generale come:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)], \text{ dove } h_{ci}(t) = h(\|r_c - r_i\|, t),$$

detta funzione di *neighborhood*, è una funzione che decresce all'aumentare della distanza da c (nodo vincitore).

Nella forma appena considerata la funzione h_{c_i} risulta:

$$h_{ci}(t) = \begin{cases} \alpha(t) & \text{se} \quad i \in N_c(t) \\ 0 & \text{se} \quad i \notin N_c(t) \end{cases} \tag{4}$$

mentre, se si considera una Gaussiana, si può riscrivere:

$$h_{ci}(t) = \alpha(t) \exp\left(- \frac{\|r_i(t) - r_c(t)\|^2}{\sigma(t)^2} \right): \tag{5}$$

entrambi i casi sono approssimazioni della funzione biologica che regola il funzionamento delle reti neuronali della corteccia cerebrale sul quale le SOM sono state modellate.

I nodi vengono inizializzati in maniera casuale: all'inizio dell'addestramento il neurone che vince la competizione su un input $\mathbf{x}$ sarà quello che per caso ha una configurazione dei pesi $\mathbf{m}_i$ più simile all'input $\mathbf{x}$ e l'apprendimento è tale da modificare i pesi in modo che $\mathbf{m}_i$ converga verso $\mathbf{x}$. La diversificazione dei pesi, per come è stata descritta in (3), è tale da potenziare l'analogia tra l'input presentato e i pesi del neurone che vince la competizione e anche di quelli topologicamente vicini.

4.2. Rete Neurale basata sulla BP

La rete neurale che compone il classificatore di livello decisionale, è costituita da una rete basata sulla Back-Propagation. Questo algoritmo rappresenta la scelta più diffusa per reti multistrato, ove s'intende reti formate, non solo da unità d'ingresso e da unità d'uscita, ma anche altre unità di elaborazione intermedie, atte a rappresentare lo stato interno della rete [2]. L'algoritmo di apprendimento, al contrario di quello delle SOM, è un algoritmo di tipo supervisionato, cioè la rete ha a disposizione i valori desiderati (*target*) di ogni ingresso, così da poter confrontare la sua uscita con quello che è il valore effettivo ed utilizzare tale indicazione per correggere gli errori.

Il principio basilare dell'algoritmo si basa su due passi fondamentali:

1. L'input è presentato alla rete, propagato in avanti (forward) per ottenere il valore d'uscita. Tale valore d'uscita viene confrontato col valore desiderato (target) per ricavare l'errore delle unità d'uscita.
2. L'errore calcolato al primo passo per le unità d'uscita viene rimandato indietro (back) a ciascuna unità della rete per effettuare i necessari cambiamenti ad i pesi della rete, in modo da orientare i pesi verso quell'input in modo corretto.

Diversamente dall'algoritmo per le SOM, l'algoritmo di Back-Propagation fa muovere tutti i pesi della rete per ogni input, cercando di sfruttare al meglio possibile la conoscenza del valore target.

La struttura di una generica rete neurale basata sulla BP, vede tre diversi tipi di unità: unità d'ingresso, unità nascoste (o strato nascosto) e unità d'uscita. Le unità di uno strato sono totalmente connesse con quelle dello strato superiore, le unità di ingresso ricevono tutte i pattern di input. L'uscita di una rete di questo tipo è composta da *n* valori se *n* sono le unità d'uscita. In figura 2 è mostrato un esempio di rete con uno strato nascosto.

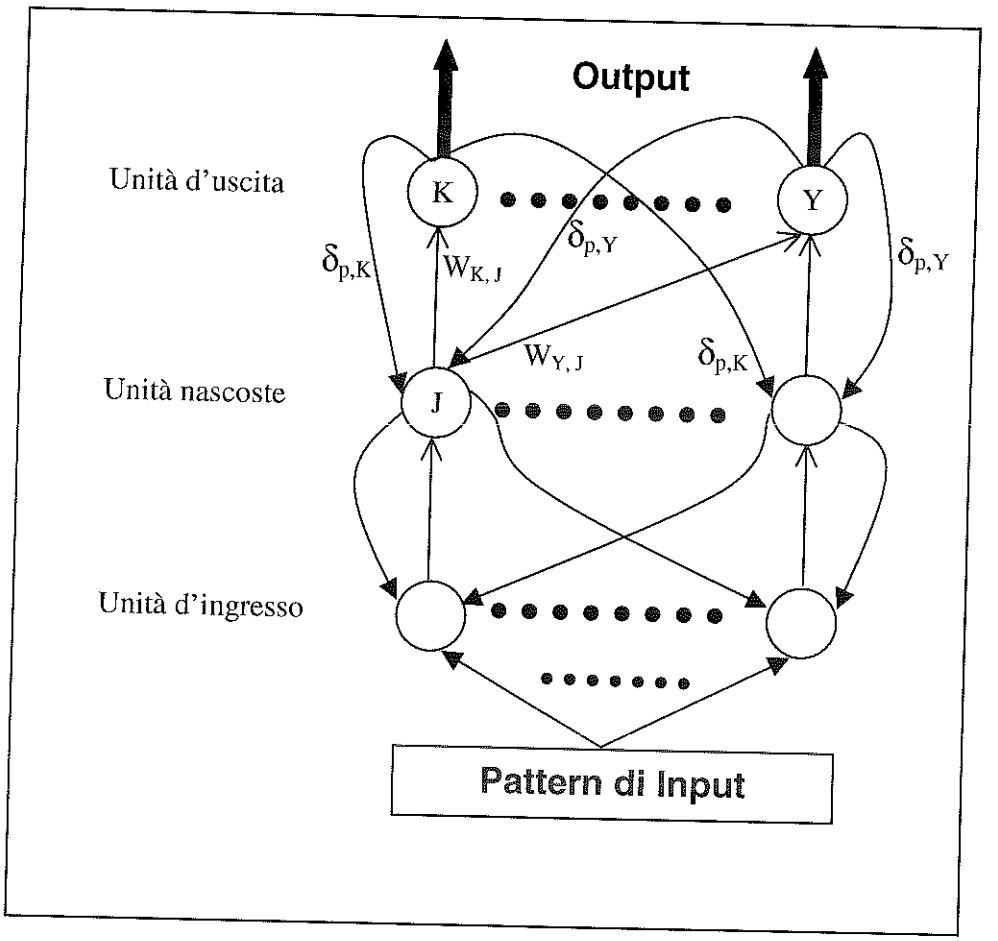

Figura 2

In modo più formale, riferendosi alla fig. 2, dato un *pattern* di input p questo viene presentato a tutte le unità dello strato d'ingresso e propagato in avanti attraverso la rete.

Definendo con $O_{p,I} = \Im(p_I)$ l'output dell'unità di uscita *I* per il pattern d'ingresso *p*, le unità di uscita calcolano il loro valore di output secondo la funzione di attivazione $\Im$, per tale funzione, che deve essere continua e semilineare, può essere scelta la funzione logistica.

$$\Im(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

Utilizzando l'uscita ed il target, viene calcolato l'errore per ogni unità d'uscita, definendo come t_p il target per il pattern *p*, il segnale d'errore $\delta_{p,K}$ per l'unità d'uscita *K* è così calcolato:

$$\delta_{p,K} = (t_p - O_{p,K}) \cdot \Im(p_K) \tag{7}$$

Tale errore viene inviato indietro alle unità dello strato inferiore connesse a l'unità *K* ed è utilizzato dall'unità *K* per aggiornare i pesi delle sue connessioni con le unità del livello inferiore (unità nascoste). Riferendosi alla figura 2, il peso della connessione dall'unità di uscita *K* all'unità nascosta *J* (definito come $W_{K,J}$), viene aggiornato di una quantità $\Delta_p W_{K,J}$ così calcolata:

$$\Delta_p W_{K,J} = \eta \cdot \delta_{p,K} \cdot O_{p,J} \tag{8}$$

Il parametro η (*learning rate*) regola le variazioni dei pesi e può essere dipendente dal tempo, diminuire cioè l'entità delle variazioni mano a mano che la rete si sintonizza sui pattern d'ingresso.

Le unità nascoste, a loro volta, calcolano il segnale d'errore, che tiene però conto anche dell'errore al livello superiore:

$$\delta_{p,J} = \sum_{Y \text{ unità del liv. superiore}} \delta_{p,Y} \cdot \Im(p_J) \tag{9}$$

Quindi a loro volta le unità nascoste modificano i pesi secondo la (8) e passano l'errore $\delta_{p,J}$ indietro alle unità dello strato inferiore (eventualmente d'ingresso). Le unità dello strato d'ingresso non mandano l'errore indietro, ma modificano soltanto i pesi delle connessioni. I pesi delle connessioni sono inizializzati tutti a piccoli valori casuali.

La rete si ferma quando si realizza una delle due condizioni: l'errore sull'insieme di allenamento diminuisce sotto una certa soglia oppure si raggiunge il massimo numero di iterazioni (epoche) fissato a priori come parametro della rete.

5. Implementazione dell'architettura sul caso di studio

Nel seguito verrà mostrata la struttura ed i particolari riguardanti l'implementazione, del modello di architettura proposto per la classificazione di immagini mediche, dalle quali si vogliono estrarre informazioni densitometriche allo scopo di effettuare una ricostruzione volumetrica 3-D di particolari zone d'interesse medico, oppure per la quantificazione statistica di varie misure volumetriche a scopo diagnostico [5, 6].

Caso	Numero sezioni TAC	Sesso	Età	Patologico Normale
1	23	F	70	Patologico
2	21	M	50	Normale
3	18	M	40	Normale
4	15	M	60	Normale

Tabella 1

Le immagini, fornite dal Dipartimento di Neurochirurgia dell'Università di Pisa, riguardano TAC craniche di pazienti normali o affetti da patologie ischemiche (vedi tab. 1).

In collaborazione con esperti neuro-radiologi è stata estrapolata una distribuzione delle aree del cervello in otto possibili classi di densità da attribuire alle zone in esame, da classificare automaticamente (vedi tab. 2).

Con l'aiuto di un esperto neuro-chirurgo è stata scelta una serie di punti campione sulle immagini, per allenare la rete in modo poi da utilizzarla per la classificazione automatica di tutti i punti dell'immagine e di tutte le immagini. E' prevista la realizzazione di un'interfaccia grafica per la scelta e la raccolta automatica dei punti campione da fornire poi in input alla rete neurale.

Per ogni paziente si hanno a disposizione una serie d'immagini che elaborate portano alla creazione del volume del cranio (vedi tab. 1 *Numero sezioni TAC*). Tale ricostruzione volumetrica avviene però soltanto successivamente, dopo cioè che le immagini sono state classificate. Nel seguito si farà comunque riferimento al *voxel*

(volume element) intendendo che le immagini formino nell'insieme il volume. Ogni voxel è quindi identificato dalle tre coordinate cartesiane: <*x*,*y*,*z*> ove *x* e *y* sono le coordinate relative nell'immagine di riferimento *z* (numero della sezione cranica nella relativa TAC).

Numero	Classe
1	Osso
2	Sostanza Grigia
3	Sostanza Bianca
4	Liquor
5	Aria
6	Ipodensità
7	Iperdensità
8	Sangue

Tabella 2

Le immagini sono state trasformate in modo da ottenere le features da poter utilizzare come input per la rete. Le features scelte e le trasformazioni eseguite sulle immagini sono:

- Livello di grigio e posizione del voxel e della relativa sezione
- Valore del livello di grigio medio del voxel, calcolato in un intorno del voxel, tale intorno è stato scelto come una quadrato di nove pixel di lato
- Differenza tra la media di cui sopra ed il valore del voxel stesso
- Gradiente del livello di grigio calcolato nel medesimo intorno. Tale effetto è stato realizzato tramite un algoritmo per l'estrazione dei bordi (*edge finding*).

5.1. Classificazione del livello di specializzazione

La classificazione a questo livello è organizzata con una serie di moduli, ognuno dei quali realizza un modello di SOM. Il numero di moduli corrispondenti al numero di features che devono essere riconosciute, cioè quattro.

In totale si hanno quindi quattro reti SOM che agiscono in parallelo nel livello più basso della gerarchia (vedi fig. 3). Ogni rete SOM agisce indipendentemente, cioè da una sua classificazione per ogni voxel dell'insieme di input. I valori di input sono differenziati per ogni modulo (vedi par. 3): quindi ogni modulo è allenato ed utilizzato per

la classificazione di una specifica feature tra le quattro utilizzate. Il formato dell'input, di conseguenza, è una quadrupla per il primo modulo, un singolo valore per il secondo e terzo classificatore ed una tripla per l'ultimo modulo.

Le reti SOM utilizzate qui sono *mono-dimensionali*, cioè l'unico strato è composto da un singolo vettore di neuroni, questo implica che la dimensione del vettore dei pesi è differente tra le SOM secondo la dimensione della feature in ingresso.

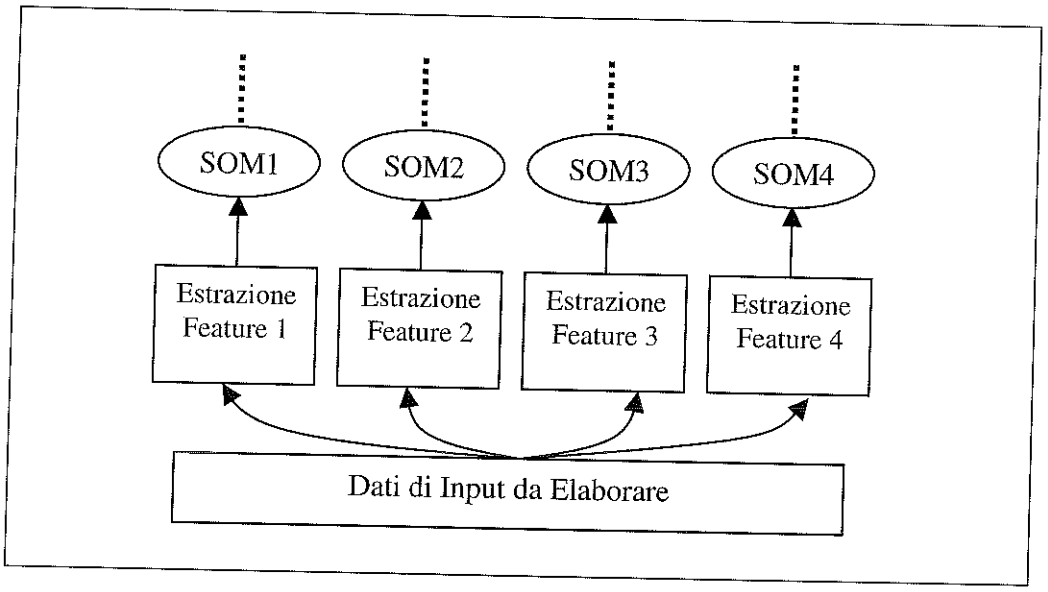

Figura 3

La dimensione della SOM è stata scelta come un vettore di $[1 \times 8]$ neuroni, in questo modo si cerca di specializzare ogni neurone su ciascuna delle possibili classi (vedi fig. 4). Un'altra possibilità allo studio è di avere un numero maggiore di neuroni rispetto alle classi di densità, applicando poi all'uscita un *algoritmo di clustering* basato sulla distanza per ridurre il numero delle classi in output ad otto. L'output della rete è un valore unico che identifica la classe di densità associata a quel input (voxel).

Questo output, insieme con gli altri delle alle tre SOM relative al voxel, viene utilizzato come ingresso per il modulo del livello decisionale.

5.2. Classificazione del livello decisionale

Il livello finale di classificazione, destinato alla raccolta dei risultati del livello inferiore, alla loro combinazione ed alla classificazione finale univoca è composto da un solo modulo.

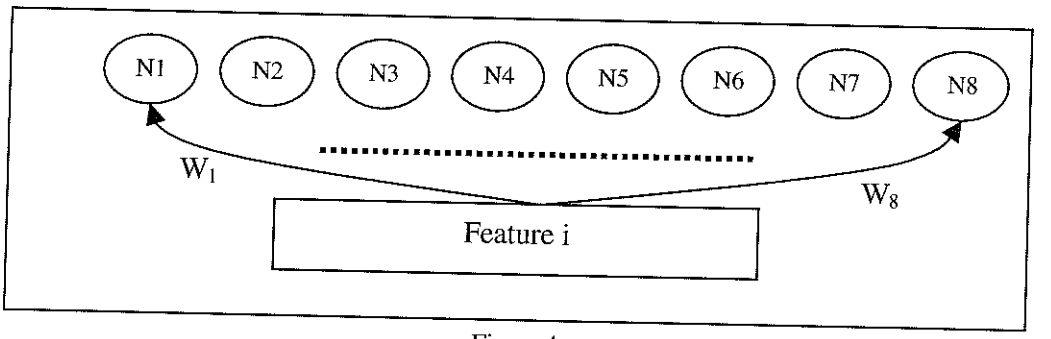

Figura 4

Tale modulo classificatore, per il livello più alto, è una rete neurale basata sull'algoritmo di Error Back-Propagation. I parametri usati dell'algoritmo implementato sono i seguenti:

- La rete è del tipo *feed-forward*
- La funzione di allenamento aggiorna i pesi seguendo l'algoritmo di Back-Propagation di *Levenberg-Marquardt*, implementato dal programma Matlab®
- La rete è formata da uno strato d'ingresso di 25 unità (tutte quante ricevono i quattro input provenienti dal precedente livello) e da 8 unità di uscita (vedi fig. 5)
- Il numero di epoche è fissato a 100

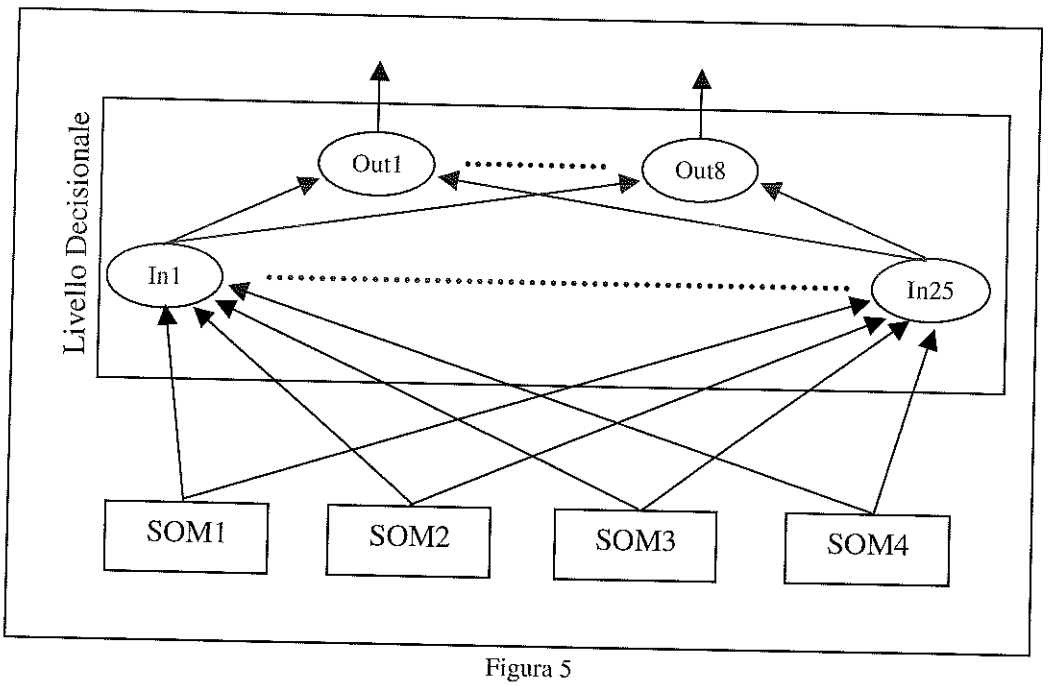

Figura 5

Il formato degli input durante la fase di addestramento della Back-Propagation è quello di una matrice, ove ogni riga è formata da un array contenente i valori delle classificazioni ottenuta dal livello di specializzazione, con aggiunto alla fine il valore rappresentate il valore di classificazione reale per quel voxel, questo proprio perché l'algoritmo di BP è un algoritmo supervisionato (tab. 3).

	SOM_1	SOM_2	SOM_3	SOM_4	Target
$Voxel_1$	$Class_{1,1}$	$Class_{1,2}$	$Class_{1,3}$	$Class_{1,4}$	$Class_{1,T}$
⋮	⋮	⋮	⋮	⋮	⋮
$Voxel_n$	$Class_{n,1}$	$Class_{n,2}$	$Class_{n,3}$	$Class_{n,4}$	$Class_{n,T}$

Tabella 3

Durante la fase di classificazione il formato degli input è invece soltanto un vettore contenente, per ciascun valore, la classificazione assegnata a quel voxel da ognuno dei classificatori del livello di specializzazione (vedi tab. 4).

	SOM_1	SOM_2	SOM_3	SOM_4
$Voxel_i$	$Class_{i,1}$	$Class_{i,2}$	$Class_{i,3}$	$Class_{i,4}$

Tabella 4

Il limitato numero di input e la mancanza di strati nascosti fanno sì che la rete neurale sia addestrabile in tempi brevi. L'output di questo classificatore, rappresenta la classificazione finale assegnata dal sistema al voxel elaborato. La classificazione finale è rappresentata dal nodo, tra gli otto del livello di output, che ha dato la risposta più vicina al target (errore minore).

6. Risultati

Le reti neurali ed il programma di controllo e ricostruzione delle immagini classificate, è stato implementato con il toolbox NEURAL NETWORK® dell'ambiente di elaborazione MATLAB® di "The Math Works Inc.". Matlab (**Mat**rix **Lab**oratory) è un ambiente software di elaborazione matematica specialmente adatto al calcolo con matrici articolato in un modulo principale di base e in strumenti tematici aggiuntivi (Statistics, Financial, Image processing, Fuzzy Logic, Neural Networks, e molte altre).

Nell'ambito dell'applicazione possono essere realizzati programmi personalizzati (*script*) o *funzioni* in linguaggio Matlab (M-file) traducibili automaticamente in C++.

Begin
Load *immagini con feature della sezione da classificare;*
Simulate *SOM_Network dedicata alla feature 1;*
Get *risultati classificazione;*
Simulate *SOM_Network dedicata alla feature 2;*
Get *risultati classificazione;*
Simulate *SOM_Network dedicata alla feature 3;*
Get *risultati classificazione;*
Simulate *SOM_Network dedicata alla feature 4;*
Get *risultati classificazione;*

SOM_To_BP *elabora i risultati per generare l'input per BP;*

Simulate *BP_Network;*
Get *risultati classificazione finale;*

Build *ricostruisce i punti dell'immagine dai risultati della classificazione;*

Create_Image *crea l'immagine con gli otto livelli di grigio e salva il file corrispondente*

Figura 6

In figura 6 è riportata la procedura, scritta successivamente in M-code (vedi appendice), che a partire dalle immagini in input elabora, classifica e ricostruisce le immagini.

In figura 7 sono riportate due immagini: la prima è un'immagine originale di una sezione cranica ottenuta tramite TAC, che fa parte delle immagini fornite dal Dipartimento di Neurochirurgia dell'Università di Pisa; l'immagine a destra è invece il risultato della ricostruzione dell'immagine classificata dalla rete neurale. L'immagine classificata è quindi riportata su otto livelli di grigio equidistanti ognuno equivalente ad una delle classi riconosciute dalla rete neurale.

Nella successiva figura 8, la stessa immagine classificata è stata ricostruita in modo da evidenziare maggiormente tale differenziazione, invece degli otto livelli di grigio sono stati utilizzati otto colori diversi.

La classificazione delle immagini mediche ha portato successivamente alla ricostruzione tridimensionale di volumi cranici, effettuata tramite il software ARIS per la ricostruzione ed il programma di visualizzazione AVS della "Advanced Visual System Inc.". Un immagine relativa al volume ricostruito è in figura 9.

Figura 7

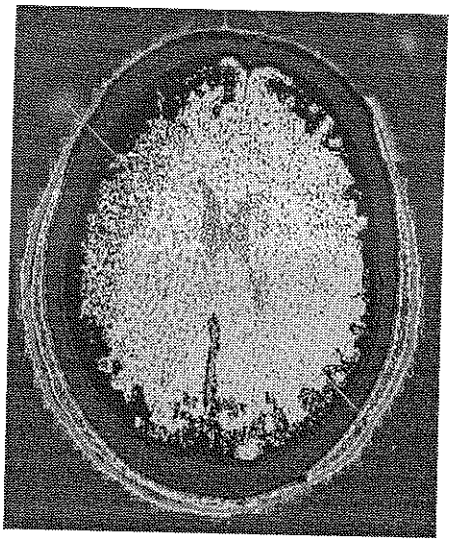

Figura 8

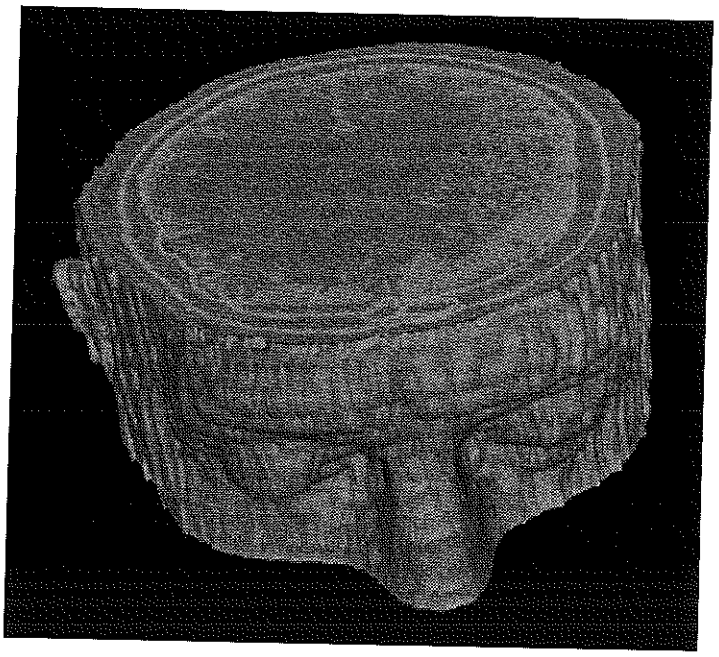

Figura 9

Nella successiva figura 10 è riportata un'immagine di un caso patologico, come prima a sinistra l'immagine TAC rilevata ed a destra la stessa classificata. Nella figura 11 si vede invece una sezione trasversale ed orizzontale del volume realizzata sempre tramite rete AVS.

Figura 10

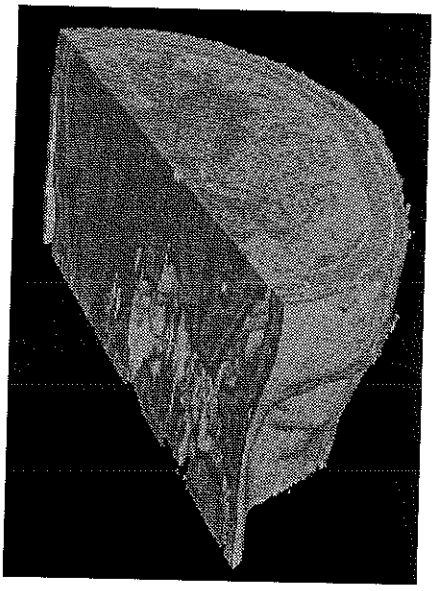

Figura 11

Nelle successive figure 12 e 13 sono riportati i risultati della ricostruzione volumetrica di due diverse classi, singolarmente estratte dal processo di classificazione. Nella prima figura la classe di densità estratta è quella relativa alla "sostanza bianca", mentre nella seconda è la "sostanza grigia". Ognuna delle due immagini contiene soltanto i punti riconosciuti dall'architettura come appartenenti alla relativa classe. La prima è visualizzata in sezione trasversale, la seconda in sezione trasversale e orizzontale

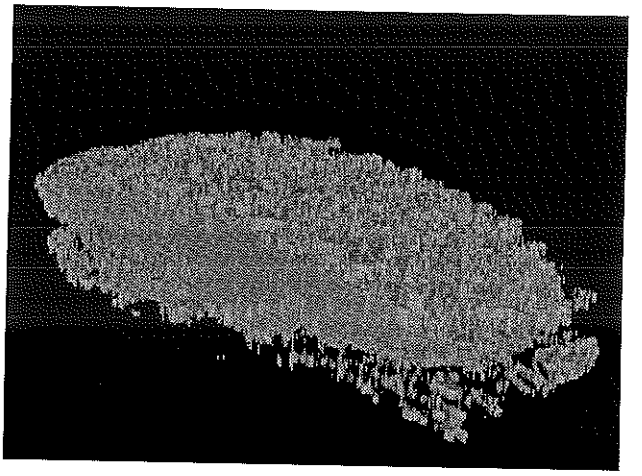

Figura 12

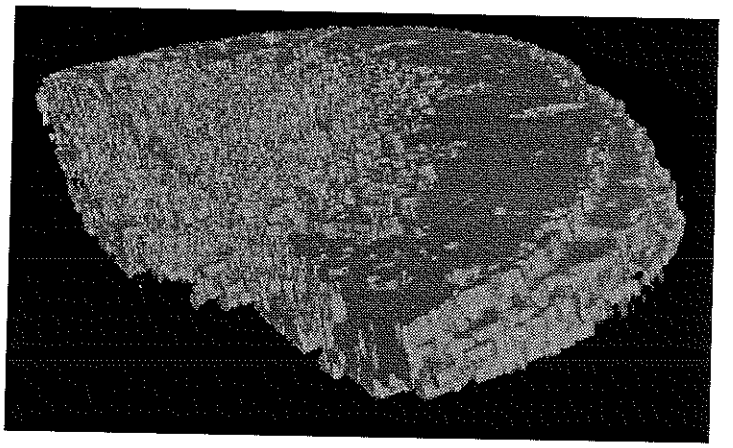

Figura 13

Appendice

In questa sezione si trova il codice elaborato in ambiente Matlab per la classificazione e ricostruzione delle immagini. In figura 14 viene mostrato il codice della funzione di classificazione globale realizzato con Matlab, le reti neurali (*BPNetwork*, *SOM8ClassF1*, *SOM8ClassF2*, *SOM8ClassF3*, *SOM8ClassF4*) utilizzate in questa funzione sono già state create ed allenate precedentemente. Nelle figure successive (figg. 15-19) sono espansi i codici delle sotto-procedure utilizzate all'interno di questa.

```
function classification(filename, index, savefilename);

initTime=fix(clock);

global BPNetwork SOM8ClassF1 SOM8ClassF2 SOM8ClassF3 SOM8ClassF4;
global inputxyzFnorm inputxyzFlmg inputxyzFdiff inputxyzFgrad;
global SOM8Class_outputsF1 SOM8ClassF2_outputs SOM8ClassF3_outputs SOM8ClassF4_outputs;
global BPGlobinput;
global BPOutMatrix;
global BPNetwork_outputs;

%Preparation input to SOMs
LoadInputSom1(filename, index);
LoadInputSom234(filename, index);

%SOMs Execution
SOM8ClassF1_outputs=sim(SOM8ClassF1,inputxyzFnorm);
SOM8ClassF2_outputs=sim(SOM8ClassF2,inputxyzFlmg);
SOM8ClassF3_outputs=sim(SOM8ClassF3,inputxyzFdiff);
SOM8ClassF4_outputs=sim(SOM8ClassF4,inputxyzFgrad);
clear global inputxyzFnorm inputxyzFgrad inputxyzFdiff inputxyzFlmg;

%Preparation of BP inputs from SOMs outputs
SomToBP;
clear global SOM8classF1_outputs SOM8classF2_outputs SOM8classF3_outputs SOM8classF4_outputs;

%BP Execution
BPNetwork_outputs=sim(BPNetwork,BPGlobinput);
clear global BPGlobinput;

%Extraction from BP's output of the image
BuildBPOutput;
clear global BPNetwork_outputs;
creaFig(savefilename);
clear global BPOutMatrix;

execTime=[int2str(etime(fix(clock),initTime)/60) 'min ' int2str(mod(etime(fix(clock),initTime),60)) 'sec']
```

Figura 14

```
function LoadInputSom1(nomefile,indice);

%Read the image modified with the extraction of the first feature (that is F1) and prepares the matrix of
%grey levels to be send as input to the network

jpeg=imread(nomefile,'jpg');
image=jpeg(:,:,1);
clear jpeg;
[a b]=size(image);

i=1:a;
j=1:b;
k=1:size(i,2)*size(j,2);
tempj(k)=mod(k,512);
tempi(k)=uint8(k/512+1);
temp4(k)=image(j,i);

input(k,1)=tempj';
input(k,2)=tempi';
input(:,3)=indice;
input(k,4)=temp4';

global inputxyzFnorm;
inputxyzFnorm=input';

clear a b i j image input k tempi tempj temp4;
```

Figura 15

```
%SomToBP Script: takes the SOMs outputs and prepares the matrix
%to be input to the Back-Propagation

file1=SOM8ClassF1_outputs;
file2=SOM8ClassF2_outputs;
file3=SOM8ClassF3_outputs;
file4=SOM8ClassF4_outputs;

[a,b,c]=find(file1);
[a2,b2,c2]=find(file2);
[a3,b3,c3]=find(file3);
[a4,b4,c4]=find(file4);

BPF1input=a';
BPF2input=a2';
BPF3input=a3';
BPF4input=a4';

BPGlobinput=combine(BPF1input,BPF2input,BPF3input,BPF4input);

clear a a2 a3 a4 b b2 b3 b4 c c2 c3 c4 file1 file2 file3 file4;
clear BPF1input BPF2input BPF3input BPF4input;
```

Figura 16

```
function LoadInputSom234(filename,index);

%Reads the images with features F2, F3 and F4 image and builds up the correspondent matrix of gray
%values to be send as input to the network

jpeg2=imread(strcat(filename,'Img'),'jpg');
jpeg3=imread(strcat(filename,'diff'),'jpg');
jpeg4=imread(strcat(filename,'grad'),'jpg');

image2=jpeg2(:,:,1);
image3=jpeg3(:,:,1);
image4=jpeg4(:,:,1);

[a b]=size(image2);

i=1:a;
j=1:b;
k=1:size(i,2)*size(j,2);
input2(k)=image2(j,i);
input3(k)=image3(j,i);
input4(k)=image4(j,i);

global inputxyzFImg inputxyzFdiff inputxyzFgrad;

inputxyzFImg=double(input2);
inputxyzFdiff=double(input3);
inputxyzFgrad=double(input4);

clear a b i j k jpeg2 jpeg3 jpeg4 input2 input3 input4 image2 image2 image3 image4;
```

Figura 17

```
%BuildBPOutput Script: takes the output of the neural network and builds up a squared matrix
%where every cell represents the correspondent pixel of the slice's classification

matrix=BPNetwork_outputs;
[y,w]=max(matrix);
BPOutArray=w;
clear matrix y w;
[a,b]=size(BPOutArray);
rq=sqrt(b);

global BPOutMatrix;

for i=1:rq
   BPOutMatrix(:,i)=BPOutArray(1+((i-1)*rq):i*rq)';
end
clear a b i rq BPOutArray;
```

Figura 18

```
%Create a coloured bmp image saved as nomefile from BPOutMatrix
%which is a matrix where every number is the classification of that specific voxel
%assign a different colour to every different classified class

function creaFig(nomefile);

global BPOutMatrix;
[a b]=size(BPOutMatrix);
for i=1:a
for j=1:b
   switch BPOutMatrix(i,j)
   case 1
      figura(i,j,1)=255;
      figura(i,i,2)=255;
      figura(i,j,3)=255;
   case 2
      figura(i,j,1)=0;
      figura(i,j,2)=255;
      figura(i,j,3)=255;
   case 3
      figura(i,j,1)=255;
      figura(i,j,2)=0;
      figura(i,j,3)=0;
   case 4
      figura(i,j,1)=255;
      figura(i,j,2)=255;
      figura(i,j,3)=0;
   case 5
      figura(i,j,1)=15;
      figura(i,j,2)=15;
      figura(i,j,3)=255;
   case 6
      figura(i,j,1)=0;
      figura(i,j,2)=255;
      figura(i,j,3)=0;
   case 7
      figura(i,j,1)=255;
      figura(i,j,2)=0;
      figura(i,j,3)=255;
   otherwise
      figura(i,j,1)=0;
      figura(i,j,2)=0;
      figura(i,j,3)=0;
   end
end
end

figint=uint8(Figura);
imwrite(figint,nomefile,'bmp');

clear figint Figura a b;
```

Figura 19

Riferimenti Bibliografici

[1] T. Kohonen. *Self-Organising Maps.* In *Springer Series in Information Sciences*, volume 30. Springer, Berlino, Heidelberg, New York, 2nd extended edition, 1997.

[2] D.E. Rumelhart, G.E. Hinton, e R.J. Williams. *Learning Representations by Back-Propagating Errors.* In *Nature*, **323**:533-536,1986.

[3] J. Cao, M. Ahmadi, e M. Shridhar. *A Hierarchical Neural Network Architecture for Handwritten Numeral Recognition.* In *Pattern Recognition*, **30**(2):289-294, 1997.

[4] M. Pfister, S. Behnke, e R. Rojas. *Recognition of Handwritten Zip Codes in a Real-World Non-Standard-Letter Sorting System.* In *Applied Intelligence*, **12**:95-115, 2000.

[5] S. Di Bona, G. Pieri, e O. Salvetti. *A Multilevel Neural Network Model for Density Volumes Classification.* In *Proc. of the IEEE 2nd International Symposium on Image and Signal Processing and Analysis - ISPA'01*, 213—218, 2001.

[6] S. Di Bona, H. Niemann, G. Pieri, e O. Salvetti. *Brain Volumes Characterisation Using Hierarchical Neural Networks.* Submitted to *Artificial Intelligence in Medicine*, 2001.