Università degli Studi di Firenze
Facoltà di Ingegneria

Corso di Laurea in
Ingegneria Informatica

# Classificatore di veicoli a partire da immagini

Dicembre 2010

Tesi di Laurea di:

Alberto Bini

Relatori:

Prof. Giovanni Soda
Ing. Angelo Frosini

Anno Accademico 2009/2010

*ad Anna e Luca*

# Indice

# Capitolo 1

# Introduzione

*"Niente è più importante che poter osservare le fonti dell'invenzione, che sono, a mio parere, più interessanti delle invenzioni stesse." (Gottfried Wilhelm Leibniz).* Spesso osservare e cercare di capire quello che ci sta intorno non è facile come sembra. Nel mondo della tecnologia questo è un fatto assai comune e per questo tendiamo a farci aiutare dalle nuove invenzioni che le industrie ci procurano, siano esse di tipo Software, Hardware o di entrambi i tipi.

Già da diversi anni la *Società Autostrade per l'Italia* sta sperimentando sistemi di monitoraggio traffico basato su elaborazione delle immagini, utilizzando sia prodotti di mercato che prodotti realizzati ad hoc. Uno dei prodotti in corso di sperimentazione è stato realizzato dalla ditta *IsTech s.r.l.*. Un prototipo è installato da circa due anni lungo l'autostrada A11 Firenze-Mare in direzione mare alcune centinaia di metri dopo la barriera di Firenze Ovest. Il prototipo è realizzato con una telecamera intelligente Sony (Smart Cam Sony XCI-V3 con sistema operativo LINUX a bordo) installata su un portale e collegata su un server LINUX installato alla base del portale. Il funzionamento del sistema è descritto in appendice A. Fra le varie funzioni richieste al sistema vi è quella di produrre delle statistiche di traffico ( Fig. 1.1) suddivise per categoria di veicoli. Attualmente le categorie vengono determinate stimando le dimensioni del veicolo

rilevato e suddivise grossolanamente in tre categorie:

**classe 1** : moto.

**classe 2** : auto.

**classe 3** : mezzi pesanti (camion, autoarticolati, etc.).

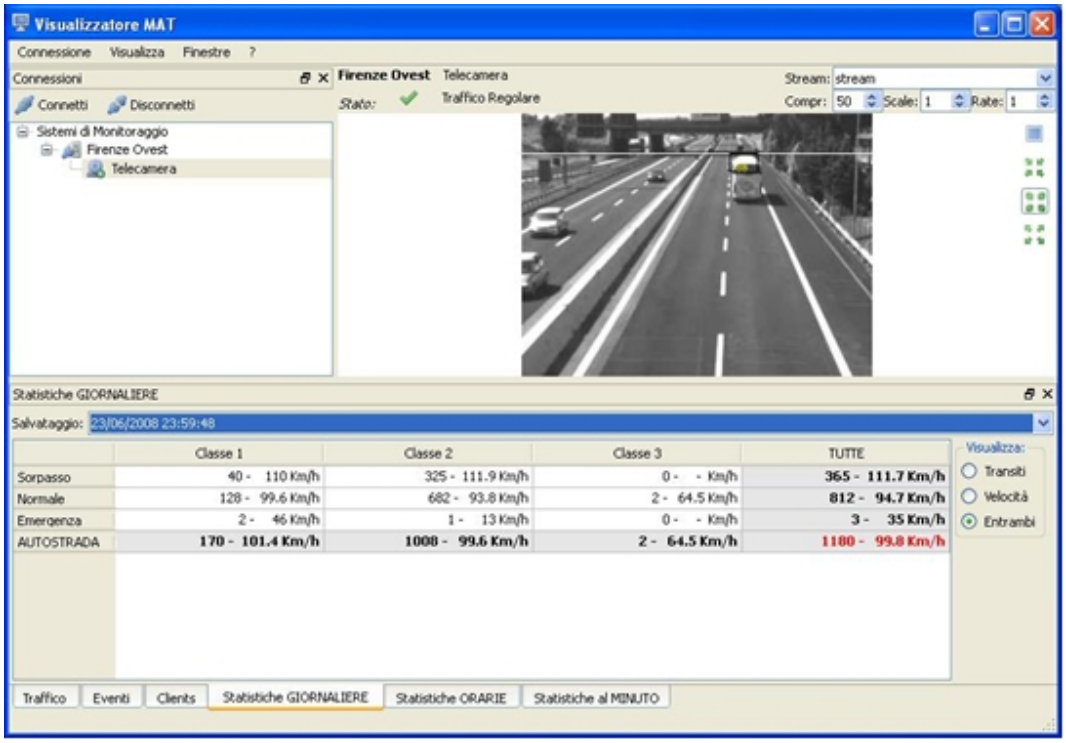


Figura 1.1: Schermata di rilevamento delle statistiche

## 1.1 Obiettivo

Quello che la ditta *Is Tech* ha richiesto è stato un modulo da aggiungere al programma IsTraffic, che svolgesse il compito di categorizzare i veicoli. La decisione di ricorrere ad un modulo aggiuntivo per questa operazione è da ricercarsi nell'attuale funzione del programma adibita a questo scopo. Il

programma di Monitoraggio utilizza un metodo basato sull'altezza e sulla larghezza in pixel del veicolo presente nell'immagine acquisita e, sulla base di confronti con delle soglie stabilite a priori, categorizza i veicoli nelle tre classi precedentemente descritte. Una distinzione del genere risulta un po' troppo grossolona dal punto di vista delle statistiche che il programma fornisce e rispetto al numero ridotto di informazioni sui veicoli che arrivano ad *Autostrade*. Lo scopo di questa tesi è quello di realizzare un apposito modulo in grado di migliorare tale distinzione. Infatti le possibili classi individuabili, definite da una normativa di *Autostrade*, sono:

**Classe A** : automobili, fino a nove posti e fino a tre tonnellate e mezzo di massa complessiva a pieno carico.

**Classe B** : Classe A con rimorchio.

**Classe C** : autocarri per trasporto merci con massa complessiva a pieno carico superiore alle tre tonnellate e mezzo.

**Classe D** : autotreni e autoarticolati.

**Classe E** : autobus con massa complessiva a pieno carico superiore alle tre tonnellate e mezzo.

**Classe F** : classe indeterminata/nessuna delle precedenti.

**Classe G** : moto e motocicli.

Rispetto alle sei classi (la classe indeterminata non viene considerata) precedentemente introdotte, solo cinque sono state prese in considerazione per la distinzione. La classe B non è stata considerata, in quanto dalle immagini che provengono dal programma sarebbe risultato molto difficile distinguere una roulotte o un rimorchio da una macchina. La prima idea presa in considerazione è stata quella di usare un classificatore basato sulle Reti Neurali (cap. 4). Questo

tipo di tecnologia è molto robusta quando si tratta di lavorare con dati continui e molto rumorosi. Successivamente sono stati utilizzati gli Alberi di Decisione (cap. 3) e le Deep Network (in un unico caso) come metodi alternativi di appredimento. L'uso di strumenti del genere è motivato dall'obiettivo di cercare di riconoscere e distinguere le varie classi, avendo una bassa risoluzione delle foto, al fine di massimizzare i risultati. Le Reti Neurali sono state usate in modo massiccio per lo sviluppo di applicazioni di riconoscimento (facciale, di caretteri, etc..); inoltre queste tecnologie sono ormai in circolo da molti anni e sono molto consolidate. In conclusione, questa tesi si propone di realizzare un apposito plug-in per il programma IsTraffic che permetta di compiere una categorizzazione migliore dei veicoli rispetto a quella attuale. Per fare questo la ditta *IsTech* ha fornito un insieme di immagini sulle quali poter lavorare. Tutta la fase degli addestramenti e dei testing è stata compiuta su queste immagini e non sul programma. Concluse tutte le sperimentazioni, il modulo sarà inserito all'interno del programma.

- Nel capitolo 2 si tratterà tutta la fase di raccolta dei dati. Verranno presentati gli esempi delle cinque classi prese in considerazione per lo svolgimento degli esperimenti, verrà descritta la strategia di campionamento usata e si farà qualche accenno alla *k-cross fold validation*.

- Nel capitolo 3 saranno presentati brevemente gli Alberi di Decisione con qualche riferimento alla teoria, al Concept Learning System ed al programma di apprendimento usato che sfrutta l'algoritmo C4.5.

- Nel capitolo 4 saranno presentatate brevemente, con qualche accenno alla teoria, le Reti Neurali e l'algoritmo Backpropagation. Verrà inoltre presentato il programma di apprendimento sviluppato dall'Università di Siena.

- Nel capitolo 5 saranno illustrati gli schemi di classificazione che sono stati impiegati in tutte le sperimentazioni del modulo. Saranno inoltre presentate

le Deep Networks e le RBM come alternativa agli autoassociatori basati su Reti Neurali.

- Nel capitolo 6 verranno descritti i diagrammi UML e le classi che sono state create per svolgere le funzioni di filtraggio delle immagini e testing.
- Nel capitolo 7 si illustreranno tutte le considerazioni che stanno alla base degli schemi di classificazione; si troveranno inoltre i risultati sperimentali fatti sulla *10-cross fold validation* dei dati ed i risultati relativi ad un'applicazione del modello al traffico reale.
- Nel capitolo 8 si trarranno le conclusioni della tesi.
- In appendice A si troverà la descrizione del programma di monitoraggio IsTraffic che la ditta *IsTech* ha fornito.

# Capitolo 2

# Raccolta dati

Ogni singola immagine di veicolo è stata prelevata dal programma IsTraffic durante più esecuzioni del programma stesso. Sono quindi da considerarsi dei dati significativi ai fini degli esperimenti. Tutte le immagini sono state salvate in formato *Jpeg* con le seguenti caratteristiche:

- larghezza cinquanta pixel;
- altezza ottanta pixel;
- immagini in bianco e nero a otto bit di profondità.

Le immagini sono in bianco e nero perché il sensore della smart cam, sulla quale è installato il programma, riesce a catturare soltanto immagini di questo tipo. Inoltre i valori dei byte dell'immagine sono compresi in un intervallo di $[0, 255]$. Di seguito è riportato in Fig. 2.1 un esempio di formato delle immagini usate.

Come si può facilmente notare, l'immagine è suddivisa in 2 parti:

- una parte in alto a sinistra, evidenziata con una croce, contenente il veicolo;
- il resto dell'immagine di colore nero.

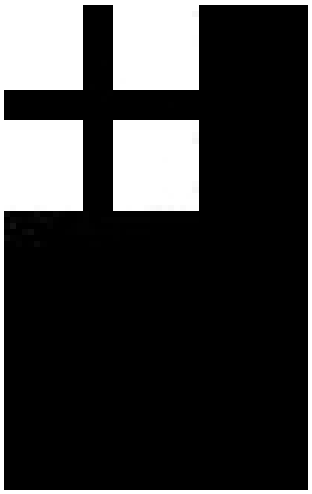

Figura 2.1: Schema di un immagine dati

La posizione relativa del veicolo all'interno dell'immagine nell'angolo in alto a sinistra è una convenzione presa all'atto del salvataggio dei dati provenienti dalla telecamera, ed è una scelta "non motivata" da un punto di vista dell'addestramento. Prima di essere trasmessi al lato client del programma di monitoraggio, per poi essere visualizzati a schermo, i veicoli sono salvati in un buffer di dati e scritti in maniera sequenziale riga per riga. Considerando l'immagine *Jpeg* come un insieme di righe, si può scrivere i pixel partendo dalla prima riga. Così facendo, se per quella deteminata riga si hanno $X$ bytes di dati effettivi, si scrivono nell'immagine; altrimenti si completano i rimanenti cinquanta-$X$ bytes colorandoli di nero. Le immagini così "codificate", prima di venire presentate alla rete per gli apprendimenti (e per la verifica), subiscono un ulteriore campionamento. Ogni immagine è salvata in una più piccola ottenuta prelevando solo i pixel di posizione pari e considerando solo le righe pari. La Fig. 2.2 evidenzia meglio questa situazione.

Quello che si ottiene da una immagine di cinquanta per ottanta è una sottoimmagine di venticinque per quaranta per un totale di mille pixel.

La motivazione di questo campionamento risiede nella volontà di cercare di ridurre il numero di features necessarie per l'apprendimento. Questo perché il

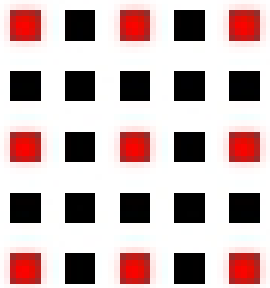

Figura 2.2: Strategia di campionatura utilizzata

Figura 2.3: Immagine dopo il campionamento

modulo dovrà essere eseguito su di una Smart Cam con processore e memoria più limitati rispetto a quelli di un comune PC; inoltre, dato che la funzione di categorizzazione deve essere fatta in Real Time, se si riesce a diminuire il numero di features si migliorano le prestazioni in termini di velocità di esecuzione. Nonostante il campionamento, si è visto che si riescono comunque ad ottenere dei buoni risultati (capitolo 7). Da quest'ultima immagine viene fatta un'ulteriore normalizzione di tipo uniforme per rendere i valori dei pixel dell'immagine compresi tra [0,1]. Le ragioni teoriche di questa operazione saranno discusse nel capitolo 4 sezione 4.2.2.

## 2.1 Categorie di veicoli

É riportato per ogni classe di veicoli un esempio prelevato dal learning set:

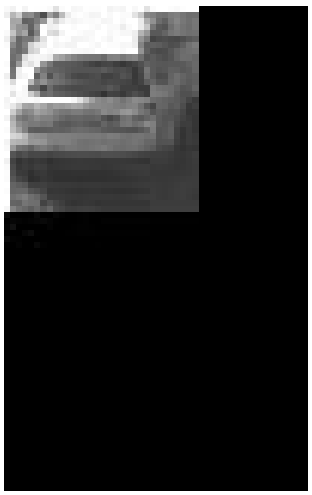

Figura 2.4: Esempio di veicolo appartenente alla Classe A

Figura 2.5: Esempio di veicolo appartenente alla Classe C


Figura 2.6: Esempio di veicolo appartenente alla Classe D

Figura 2.7: Esempio di veicolo appartenente alla Classe E

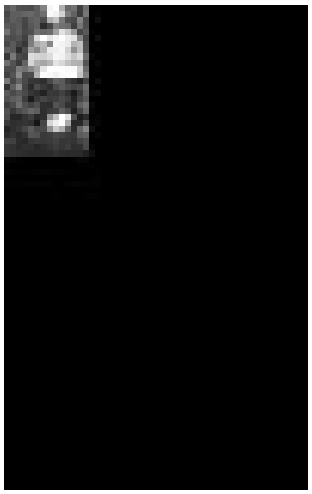

Figura 2.8: Esempio di veicolo appartenente alla Classe G

Prima di partire con ogni tipo di elaborazione sui dati è stato necessario suddividere gli esempi nelle relative classi di appartenenza. Come è facilmente osservabile, la telecamera è posta in una posizione sopraelevata rispetto ai veicoli che percorrono le corsie di marcia. Le autovetture marciano nello stesso senso in cui è posta la telecamera, per cui tutte le immagini sono esclusivamente riprese posteriori di ogni veicolo. Si nota solo con uno sguardo che le Classi A, E e G (rispettivamente Figura 2.4,Figura 2.7 e Figura 2.8) sono riconoscibili ad occhio nudo senza dover utilizzare nessun tipo di criterio e quindi sono facilmente distinguibili tra di loro. Ad esempio per i veicoli di Classe G si può distinguere in maniera agevole il profilo della motovettura, il portapacchi(quando presente) ed il casco del motociclista. Per i veicoli di Classe E si può distinguere le scritte presenti sul retro del pullman e quelle di note ditte di autotrasporti. Questo discorso non vale per le Classi C e D (rispettivamente Figura 2.5 e Figura 2.6) dato che tra di loro presentano molte somiglianze e le riprese posteriori non facilitano un'immediata distinzione tra le due categorie. Il Codice della Strada definisce un'apposita segnaletica visiva posta sul retro del veicolo, utile per poter distinguere le due diverse tipologie di veicolo. É obbligatorio che i veicoli appartenenti alla Classe C espongano sul retro del veicolo delle strisce

alternate gialle e rosse catarifrangenti, poste in posizione orizzontale o verticale. I veicoli di Classe D devono esporre sul retro, sempre in posizione orizzontale o verticale, le stesse strisce catarifrangenti, stavolta formate da un rettangolo di colore giallo, contornato da un rettangolo di colore rosso. Dalle nostre immagini in formato 50x80 è ancora possibile distinguere queste bande.

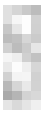

Figura 2.9: Banda catarifrangente Classe C

Figura 2.10: Banda catarifrangente Classe D

## 2.2 Suddivisione dei dati training

Tutte le immagini sono state prelevate durante un arco di tempo che varia dal 15 Luglio 2010 al 15 Ottobre 2010 circa. Per ogni Classe sono presenti duecento immagini diverse. In un primo momento durante una fase di preesperimenti è stata effettuata una divisione in un Learning Set di centocinquanta elementi ed in un Test Set di cinquanta elementi. Per ulteriori approfondimenti rimando al capitolo 7. Successivamente si è usata la strategia della *k-fold cross validation* per testare l'efficacia dei modelli di apprendimento. Nella *k-fold cross validation*, preso in considerazione un insieme di dati, questo viene diviso in $k$ sottoinsiemi di grandezza $\frac{N}{k}$, dove $N$ è il numero degli esempi del mio dataset. Di questi $k$ sottoinsiemi solo uno è usato per effettuare una misura dell'efficienza del modello, mentre gli altri $k-1$ sono utilizzati per la sua costruzione. Questo processo è applicato $k$ volte così che ciascuno dei sottoinsiemi sia usato una sola volta come insieme di test. Dei $k$ risultati si può calcolare una media (o una combinazione)

per produrre una singola stima dei risultati. Il vantaggio di questo metodo è che tutti i dati a dispozione sono usati sia per l'apprendimento sia per il testing. Nel mio caso ho utilizzato una *10-cross fold validation* dove erano presenti per ogni classe un file di training comprendente centottanta esempi ed un validation di venti elementi.

# Capitolo 3

# Introduzione agli Alberi di Decisione

Gli Alberi di Decisione sono uno strumento molto usato per l'apprendimento induttivo e permettono l'approssimazione di funzioni discrete, a partire da dati che possono essere o meno affetti da rumore e che possono anche essere rappresentati come disgiunzione di congiunzioni.

## 3.1 Concetti

Consideriamo un insieme di esempi $T = \{x_i, c(x_i)\}$ con $x_i = v_1, v_2, ..., v_n$ e assumiamo che le variabili $v_i$ siano variabili categoriche (cioè valori numerabili) su domini finiti. Siano $Y^1, Y^2, ..., Y^i$ le classi in cui l'insieme di dati deve essere partizionato; quindi $c(x_i) \in Y^1, Y^2, ..., Y^i$.
*Def.* Un albero di decisione è un albero in cui:

- un nodo esterno (detto foglia) viene etichettato con una classe;
- un nodo interno viene etichettato con l'indice $i$ di un attributo $v_i$ a cui sono connessi $|v_i|$ figli, ognuno dei quali è ancora un albero di decisione.

In pratica:

- ogni nodo interno corrisponde ad un test su di un attributo;
- ogni diramazione corrisponde ad un valore di tale attributo;
- ogni nodo esterno memorizza la classe;
- ogni cammino dalla radice alla foglia corrisponde ad una regola di classificazione.

Da un punto di vista strettamente geometrico, si può considerare un Albero di Decisione come un taglio dello spazio n-dimensionale dei dati. La Fig. 3.1 mostra questo fatto.

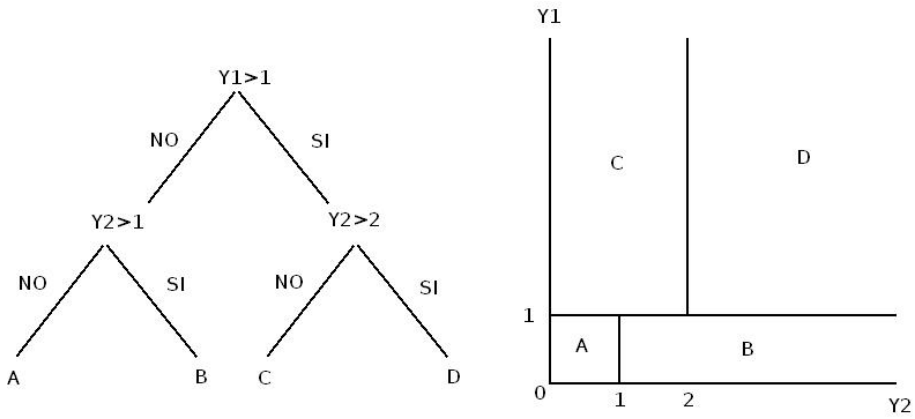


Figura 3.1: Divisione dello spazio secondo l'albero di decisione

## 3.2 Algoritmo Concept Learning System

L'algoritmo successivamente descritto è alla base di tutti i più recenti algortitmi di apprendimento per Alberi di Decisione (tra cui il C4.5 utilizzato negli esperimenti). Lo schema seguente si limita al caso di due sole classi, ma può essere esteso facilmente ad un caso di classificazione n-aria.

1. si associa un Training Set ad una radice $T$ e si crea un albero con nodo $T$.
2. se tutti gli esempi in $T$ hanno valore positivo, si crea un nodo "yes" e si indica $T$ come suo parent e stop.
3. se tutti gli esempi in $T$ hanno valore negativo, si crea un nodo "no" e si indica $T$ come suo parent e stop.
4. si seleziona un attributo $x$ con valori $v_1, v_2, ..., v_b$ e si partiziona $T$ in sottoinsiemi $T_1, T_2, ..., T_b$ in accordo ai loro valori su $x$. Si creano $T_i$ nodi $(i = 1, .., b)$ con $T$ come loro parent e $v_i$ come label del ramo da $T$ a $T_i$.
5. per ogni $T_i$ si pone $T = T_i$ e si torna al passo 2.

Da una prima analisi dello schema, quello che emerge è che tutta la criticità dell'algoritmo si trova nel punto 4, relativo alla scelta dell'attributo sul quale effettuare la divisione dei dati. Viene introdotta un'euristica basata sull'entropia. L'entropia è utilizzata come criterio per caratterizzare il grado di omogeneità di un insieme di esempi. L'euristica si basa su un risultato della teoria dell'informazione. *Shannon* ha affermato che la quantità di informazione in bit $B_m$ trasmessa da un messaggio $M$ dipende dalla sua probabilità $P_m$ e può essere misurata come: $B_m = -\log_2 P_m$. L'entropia serve per calcolare l'informazione media necessaria per individuare la classe d'esempio, ovvero l'informazione media trasmessa dal messaggio M al variare della classe $Y^j$. In pratica, un esempio estratto a caso da un insieme $T$ appartenente ad una classe $Y^j$ costituisce il mio "messaggio". L'informazione media è data dalla formula: $I(T) = -\sum_i P(T = Y^j) \log_2(T = Y^j)$. In Fig. 3.2 è riportato il grafico dell'entropia nel caso binario.

Si nota subito che, se gli esempi sono tutti di un singolo tipo, l'entropia sarà minima, visto che l'insieme dei dati è omogeneo; al contrario, se si hanno esattamente metà esempi di un tipo e metà esempi dell'altro, l'entropia sarà massima. In questo caso c'è massima disomogeneità dei dati. Nel caso k-ario il

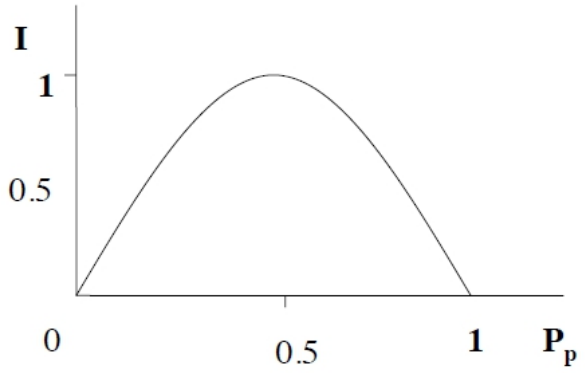


Figura 3.2: Grafico dell'entropia nel caso binario

massimo valore raggiunto dall'entropia è $\log_2 k$. Alla luce di queste considerazioni, il punto 4 dell'algoritmo può essere diviso nei seguenti passi:

1. si calcola $I(T)$ per tutto $T$.
2. per ogni attributo $x$ a valori $v_1, v_2, ..., v_b$ si ripete:
   (a) si considera l'attributo come radice dell'albero di decisione.
   (b) si divide l'insieme dei dati $T$ in $T_1, T_2, ..., T_b$ sottoalberi dove in $T_i$ sono contenuti esempi in $T$ con valori $v_i$ in $x$.
   (c) si calcola l'informazione attesa per tutto l'albero avente radice in $x$. Questa sarà data dalla somma pesata: $EI(x) = \sum_i \frac{k_i}{k} I(k)$.
   (d) per ogni attributo si valuta il guadagno d'entropia calcolato come $G(x) = I(T) - EI(x)$.
   (e) si sceglie l'attributo che massimizza il guadagno d'entropia.

Quello che si tenta di fare effettuando la divisione dell'albero (Fig. 3.3) è aumentare il suo potere decisionale. Un albero con minore entropia ha potere di classificazione maggiore rispeto ad un albero ad entropia alta. Inoltre è lecito sperare che l'informazione attesa sia minore dell'informazione media dell'albero

completo, dato che si parte da una situazione caotica e poi si dividono i dati in gruppi più omogenei.

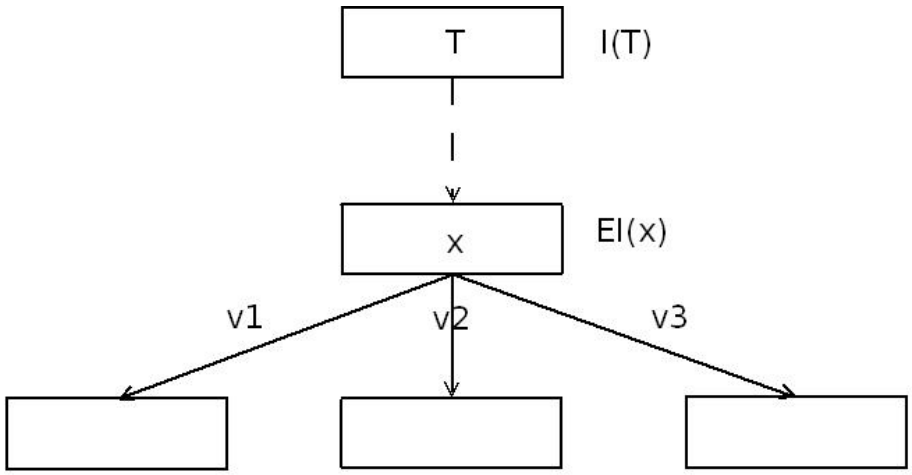


Figura 3.3: Divisione dei dati per un attributo che ha tre valori

Si può osservare che l'altezza massima dell'albero può al limite essere pari al numero di attributi dei dati e che il massimo numero di foglie si ottiene dalla cardinalità dell'insieme degli esempi di training. L'ipotesi finale è data da tutto l'albero, mentre un cammino dalla radice alla foglia è detto regola di classificazione. Il bias induttivo di questo tipo di apprendimento può essere riassunto in modo semplificato dalla seguente affermazione: "Si preferiscono alberi ad altezza bassa". La motivazione di quest'affermazione è che, dato uno spazio di possibili Alberi di Decisione che classificano gli esempi di training, ci sono molti alberi con altezze elevate, ma pochi a bassa altezza. Questi ultimi, essendo meno frequenti, probabilmente si avvicinano di più al concetto da apprendere e saranno meno predisposti al fenomeno dell'overfitting (sezione 3.4). Sfruttando l'euristica basata sull'entropia, si può aspirare a questi risultati, visto che gli attributi con

maggior potere di classificazione sono posti nelle parti più alte dell'albero. Si può osservare che, basando le scelte sul tentativo di massimizzare il guadagno d'entropia, si attua una strategia di ricerca di tipo Steepest Ascent all'interno dello spazio degli Alberi di Decisione. E' facilmente dimostrabile che questo algoritmo non è ammissibile, per cui non si riesce a trovare quell'albero che classifica correttamente i miei dati e che ha altezza minima. L'entropia non è un criterio assoluto, esistono anche dei casi per cui questa euristica sbaglia ed è necessario prendere delle misure di sicurezza. Si consideri un insieme di dati aventi come attributo un codice che identifica univocamente quella particolare istanza. Seguendo il criterio dell'entropia, questo particolare attributo sarà il candidato perfetto per lo splitting dei dati, dato che permette di classificare correttamente tutto il mio insieme di training; tuttavia l'albero che ottengo non ha potere di classificazione rispetto alle istanze non viste (ogni nuovo esempio avrà un codice diverso). Si cerca allora di penalizzare gli attributi che hanno questo tipo di struttura utilizzando l'information splitting $IV(x) = -\sum_i \frac{k_i}{k} \log_2 \frac{k_i}{k}$ e considerando il guadagno di entropia come $G'(x) = \frac{G(x)}{IV(x)}$. Usando questa nuova formula, sono penalizzati gli attributi che provocano una dispersione eccessiva dei dati. Nel caso binario il massimo valore raggiunto sarà pari ad 1 in presenza di un attributo che divide esattamente i dati in due parti uguali.

## 3.3 Trattamento dati

### 3.3.1 Dati continui

Dato che questo algoritmo lavora con dati discreti, nel caso in cui siano presenti dei dati continui, si attua una discretizzazione dell'intervallo dei valori. Il problema fondamentale è quello di determinare per quale soglia è conveniente effettuare le suddivisioni. Un possibile criterio è quello di ordinare gli esempi secondo i valori dell'attributo detti $x_1, x_2, ..., x_n$ e calcolare le soglie $\frac{x_i + x_{i+1}}{2}$ $\forall i$.

Ciascuno di questi valori è una soglia candidata ed a seconda del criterio utilizzato ($G(x)$ o $G'(x)$) si sceglie la soglia più vantaggiosa.

### 3.3.2 Attributi mancanti

La completezza dei dati dipende solo dalla fonte di provenienza. Non è detto che in ogni contesto si riescano ad ottenere istanze complete; talvolta ci si può imbattere in un dataset in cui alcuni attributi sono mancanti. Esistono due possibili metodi di risoluzione di questo problema:

1. se il dataset ha un numero elevato di esempi, si può pensare di scartare questi dati.

2. se il dataset ha piccole dimensioni, bisogna trovare un criterio per trattare i dati.

Ciò che si fa è modificare il test sugli attributi nel modo seguente: $G(x) = F(I(T) - EI(x))$, dove con $F$ si indica la frazione dei casi in $T$ in cui l'attributo è conosciuto. Si introduce un concetto di probabilità da assegnare all'appartenenza di un esempio ad una classe. Se il test sull'attributo $x$ è fatto su di un valore conosciuto, allora si assegna l'esempio ad un sottoalbero con probabilità pari ad 1. Se invece è fatto su di un valore sconosciuto, allora si assegna ad un sottoalbero con probabilità minore di 1.

## 3.4 Overfitting

*Def.* Data $h \in H$ ipotesi e qualche altra ipotesi alternativa $h' \in H$ tale che $h$ commette un errore più piccolo di $h'$ rispetto agli esempi di training, ma $h'$ compie un errore minore di $h$ su tutta la distribuzione dei dati, allora si dirà che $h$ overfitta l'insieme di apprendimento. In altri termini, il sistema impara bene sugli esempi di apprendimento ma non riesce a generalizzare casi non visti.

C'è quindi il rischio che l'albero finale non sia in grado di fornire una buona concettualizzazione per esempi non visti. Si effettua allora un'operazione detta potatura (pruning) per cercare di sfoltire l'albero sperando di migliorare il suo rendimento su istanze non viste. Esistono tre diversi tipi di potatura illustrati di seguito.

### 3.4.1 Pre pruning

Questo tipo di potatura è effettuata durante la costruzione dell'albero. Si fissa una soglia e se un nodo ha un errore al di sotto di questa soglia, si può rinunciare a fare lo splitting.

### 3.4.2 Post pruning

Si effettua ad albero terminato. Si parte dalle foglie e si risale verso la radice potando in maniera ricorsiva. Se si attua la potatura, un nodo è trasformato in una foglia ed etichettato con la classe più frequente. Viene commesso quello che si definisce errore statico $e_s(S) = p(classe \neq c \mid S)$. Se non si effettua la potatura, viene commesso quello che si definisce errore dinamico $E_D(T) = \sum_i p_i E_D(T_i)$, dove i vari $T_i$ sono degli alberi di decisione con radice nel nodo $S$ e $p_i$ è la probabilità che un esempio passi dal nodo $S$ in direzione del nodo $T_i$. Si osserva subito che se $T_i$ è una foglia allora $E_D(T) = e_s(S)$. La potatura è effettuata solo se l'errore statico è minore o uguale all'errore dinamico. Se i dati sono in numero elevato, ci si può riferire ad un criterio basato sull'uso di un Validation Set, un insieme distinto di esempi supervisionati che serve a testare in corso d'opera la validità dell'ipotesi. Altrimenti si può ricorrere a criteri probabilistici come quello dell'm-stima. Il C4.5 implementa un criterio di Post pruning che usa un test statistico come criterio di valutazione dell'errore ed effettua due possibili scelte di sostituzione:

1. si sostituisce un albero con una foglia.

2. si sostituisce un albero con uno dei suoi sottoalberi.

### 3.4.3 Rule pruning

Dall'albero si possono ricavare quelle che vengono dette regole di classificazione, ovvero un percorso che parte dalla radice ed arriva alla foglia del mio albero. Queste regole possono essere scritte in una forma più comprensibile come congiunzione di condizioni e possono essere inserite in sistemi di rappresentazione della conoscenza. In questa prima fase è possibile avere una corrispondenza uno a uno tra regole e albero. Potare una regola significa eliminare una delle condizioni, rendendo la regola più generale. Come detto in precedenza, una regola $R$ può essere vista come un and logico di condizioni e può essere scritta nella seguente forma: $R = R^{-} \wedge x$, dove $x$ rappresenta la condizione che si vuole elidere e $R-$ le condizioni che si vogliono mantenere. *Def.* $\{E_R\}$ estensione della regola $R$ ovvero gli esempi da lei coperti. Segue subito che $\{E_R\} \subseteq \{E_{R^-}\}$ dato che $R^{-}$ è più generale di $R$ e che $\{E_R\} = \{E_{R^-}\} \cup \{E_x\}$. Per trovare le condizioni $x$ da eliminare si attuano i seguenti passi:

1. si considera un congiunto $x$ di $R$.

2. si valutano tutti gli esempi di apprendimento con $R$, $R = R^{-} \wedge x$.

3. vengono tolte tutte quelle condizioni $x$ per cui $R^{-}$ (regola più generale) ha lo stesso potere di classificazione di $R$ a meno di un errore limitato ma fissato a priori. Si potrebbe validare l'errore con un Validation Set.

Quello che si ottiene è un insieme di regole più semplici di quelle di partenza, che tuttavia non hanno alcuna attinenza con un albero, in quanto non è più possibile riportarle in questa struttura.

## 3.5 Software utilizzato

L'apprendimento degli alberi di decisione si basa su di una versione evoluta dell'algoritmo Concept Learning System (sezione 3.2) chiamato C4.5. Questo programma è Open Source e disponibile sul sito della *University of Regina*[1] del Canada. Il programma è stato scritto utilizzando il linguaggio di programmazione C. Per funzionare, il programma richiede come ingressi:

- Un file .names, dove sono presenti le classi in cui partizionare l'insieme di dati e le descrizioni dei loro attributi;
- Un file .data, contenente i dati supervisionati su cui fare l'apprendimento;
- Un file .test, opzionale, su cui testare l'efficienza dell'albero di decisione.

Tutti questi file devono avere lo stesso prefisso; ad esempio si dovranno avere tree.names, tree.data e tree.test. Il comando per lanciare il training e il successivo testing sui dati è: c4.5 -f suffisso -u, dove l'opzione -f suffisso specifica quale file .names e .data prendere in considerazione, mentre l'opzione -u richiede di eseguire il testing sul file .test con lo stesso suffisso dei due precedenti. Quello che si ottiene alla fine sono 2 diversi file in formato .unpruned e .tree. Il primo rappresenta l'albero ottenuto dall'apprendimento sul dataset, mentre il secondo rapprensenta l'albero quando è stata effettuata la potatura. Come riportato sulla documentazione e come è già stato spiegato per motivi teorici nella sezione 3.4, è preferibile usare a fini pratici l'albero potato rispetto a quello non potato. Come output testuale il programma fornisce le percentuali di errore sull'apprendimento e le percentuali di errore sui casi non visti di entrambi gli alberi.

[1] http://www2.cs.uregina.ca/ dbd/cs831/notes/ml/dtrees/c4.5/tutorial.html

# Capitolo 4

# Introduzione alle Reti Neurali

Lo studio delle Reti Neurali Artificiali è stato ispirato dall'osservazione dei sistemi bioligici di apprendimento. Questi tipi di sistemi sono costituiti da un insieme complesso di neuroni interconnessi. In buona approssimazione, si può considerare che una Rete Neurale Artificiale è un insieme di elementi che chiameremo neuroni, interconnessi gli uni agli altri. Storicamente esistevano due gruppi di ricerca motivati da due differenti obiettivi:

- usare le Reti Neurali per studiare e modellare i sistemi di apprendimento biologico;
- usare le Reti Neurali per ottenere algoritimi di apprendimento efficaci, indipendentemente dal fatto che questi algoritmi mirassero o no a riprodurre un fedele modello biologico.

## 4.1 Modello matematico neurone

Il modello matematico del neurone è rappresentato in figura 4.1.

Come si può notare, il neurone risulta composto di cinque parti essenziali:

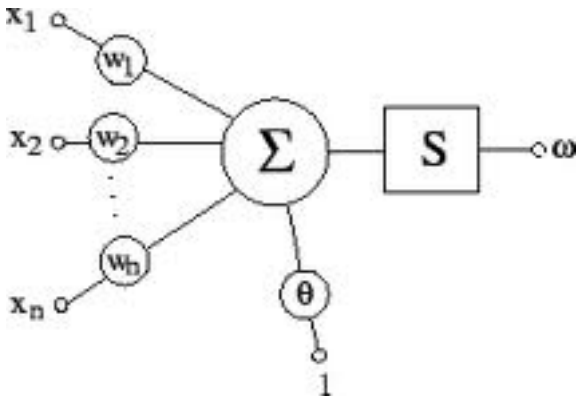


Figura 4.1: Esempio del modello matematico del neurone

- gli input $x_i$ forniti al neurone, provenienti dall'esterno o da output di altri neuroni;
- la soglia $\theta_i$, un valore numerico che determina se il neurone è attivo o no;
- i pesi $w_{ki}$, valori numerici presenti sugli input di un determinato neurone;
- un sommatore, che calcola la somma pesata di un input con il suo relativo peso;
- una funzione di trasferimento $f_i(x_k, w_{ki}, \theta_i[i \neq k])$, che determina lo stato del neurone come funzione di stato dei nodi collegati tramite i link di ingresso, dei pesi dei link di ingresso e della soglia. Si osservi che in questa definizione le reti non hanno dei loop data la condizione $[i \neq k]$.

Il funzionamento del neurone è il seguente:

1. si forniscono alla rete gli input.
2. si calcola la somma pesata $S_i = \sum(w_{ij} * x_j) - \theta_i$.
3. si computa l'output della singola cella (che verrà poi trasmesso ad un'altra) come: $f_i(S_i)$.

Spesso si considera il valore di soglia come un ingresso fittizio della rete avente come valore di input $x_0 = 1$ e come peso $w_0 = \theta$ detto anche *Bias* per semplificare la somma $S_i = \sum(w_{ij} * x_j) - \theta_i$ inglobando la soglia ed ottendo: $S_i = \sum(w_{ij} * x_j)$. Così facendo, si semplificano le operazioni da svolgere durante la fase di apprendimento e si riesce a calcolare in maniera più agevole il valore di tale soglia, considerata come un peso e quindi come output dell'algoritmo (sezione 4.2.4).

## 4.2 Reti Neurali

Per caratterizzare una Rete Neurale, occorre precisare quattro aspetti fondamentali:

1. Topologia della rete.
2. Proprietà della cella.
3. Dinamica della rete.
4. Algoritmo di apprendimento.

### 4.2.1 Topologia della rete

Si possono distinguere all'interno della rete tre diverse tipologie di neuroni:

**Neuroni di input** : sono quei neuroni che ricevono l'input.

**Neuroni di output** : sono quei neuroni che forniscono l'output della rete.

**Neuroni di hidden** : sono quei neuroni che non sono né di input né di output.

La rete neurale può essere infine di due tipologie diverse:

**Rete feedforward** : una rete in cui non sono presenti dei cicli(Fig. 4.2).

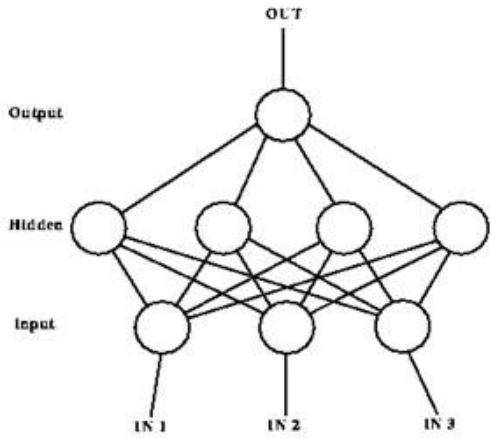


Figura 4.2: Un esempio di rete feedforward

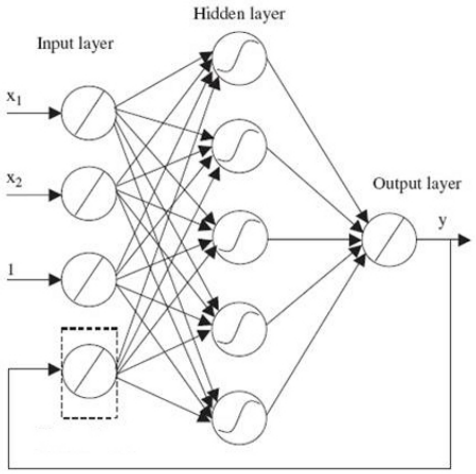


Figura 4.3: Un esempio di rete ricorrente

**Rete ricorrente** : una rete dove sono possibili dei cicli(Fig.4.2.1).

Il modello di rete che si è usato per gli esperimenti è feedforward.

### 4.2.2 Proprietà della cella

I valori di attivazione computati dalle celle sono compresi in intervalli discreti o continui, come ad esempio tra $[0, -1]$ o $[-1, 0]$. La funzione di attivazione che è stata usata è la funzione sigmoide o logistica definita come: $f(x) = \frac{1}{1+e^{-x}}$ la cui uscita varia tra $[0, 1]$ in maniera continua. Una particolarità di questa funzione, che la rende molto utile da un punto di vista di calcolo, è il fatto che la derivata $f'(x)$ può essere scritta nei termini della funzione stessa: $f'(x) = f(x)(1 - f(x))$. In figura 4.4 si può vedere il grafico della funzione logistica.

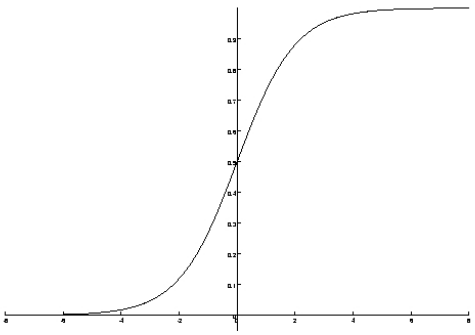

Figura 4.4: Funzione logistica

Dall'osservazione del grafico si può notare che la sigmoide è composta da due parti:

1. una parte lineare tra $[-4, 4]$ circa.
2. una parte costante, detta anche regione di saturazione, tra $[-\infty, -4]$ e tra $[\infty, 4]$.

La presenza della regione lineare induce sui dati e sui pesi iniziali della rete alcune conseguenze:

**Pesi** : i pesi iniziali devono stare tra $[-0.1, 0.1]$ o valori più piccoli per poter sfruttare al massimo la linearità della sigmoide. E' facile intuire che una minima variazione del valore di un peso nella regione lineare può portare ad un cambiamento significativo traducibile in un apprendimento migliore. Se si lavora nella parte di saturazione, anche una grossa variazione sortirà pochissimo effetto e quindi un apprendimento peggiore.

**Esempi di training** : gli esempi di training, come i dati che andranno visionati in futuro dalla rete, devono essere compresi tra $[0, 1]$. Questo sempre per evitare di saturare la mia funzione di attivazione.

Tutte le reti usate hanno avuto pesi iniziali randomici compresi tra $[-0.1, 0.1]$, mentre i dati di training sono stati normalizzati in maniera uniforme utilizzando la formula: $x_{norm} = \frac{x-min}{max-min}$ dove i valori $max$ e $min$ sono pari a $max = 255$ $min = 0$.

### 4.2.3 Dinamica della rete

La dinamica della rete specifica in che modo le celle devono computare il proprio output. Nelle reti feedforward si tende a visitare le celle in un ordine fissato (solitamente dalle celle di input verso le celle di output passando per quelle di hidden), così si ha la garanzia di ottenere uno stato detto stazionario. Nelle reti ricorrenti si può procedere come per le reti feedforward, solo che, data la loro natura, non è possibile dire quando si è raggiunto uno stato stazionario. In genere si introduce l'idea di ritardo di un'unità di tempo, cioè si considera l'attivazione al passo precedente.

### 4.2.4 Algoritmo di apprendimento

L'algorimto di apprendimento è quella determinata serie di azioni che permette di modificare i pesi delle varie connessioni affinchè la rete possa "apprendere" il

concetto. All'algoritmo è fornito un insieme di esempi $e^k$ detto insieme di training. Ogni esempio $e^i$ è un p-vettore tale che ogni componente del vettore è fornita in ingresso ad un neurone di input. Insieme alle p-componenti nei problemi con supervisione è fornito anche un vettore di valutazione da confrontare, componente per componente, con le uscite provenienti dai neuroni di output. Il cambiamento dei pesi delle connessioni varia sulla base del confronto tra l'uscita che genera la rete e il valore di supervisione fornito. Se i due valori sono gli stessi, non è necessario apportare modifiche, mentre se differiscono è necessario applicare una strategia per modificare i pesi (quindi eseguire l'algoritmo) in modo da ottenere i risultati sperati. Il vettore di pesi e di soglie finale, generato dall'algoritmo dopo che ha terminato la sua esecuzione(si veda la sezione 4.4), è l'ipotesi o concetto che la rete ha tentato di approssimare sull'insieme di dati di training.

## 4.3 Algoritmo Backpropagation

Il Backpropagation è l'algoritmo utilizzato per l'apprendimento delle Reti Neurali feedforward. L'algoritmo usa il criterio della discesa del gradiente come schema generale; infatti è presente una funzione d'errore che in questo particolare caso è l'errore quadratico medio definito come: $E(w) = \frac{1}{2}\sum_d (c_d - o_d)^2$ dove $c_d$ è il valore di supervisione assegnato all'esempio, mentre $o_d$ è l'uscita calcolata dalla rete in funzione dell'ingresso e dei pesi. Si deve notare che l'errore quadratico medio non è un indice della robustezza della rete, in quanto è solo un'approssimazione dell'errore reale della rete, che viene misurato come $E = \frac{casi\ nonclassificati\ correttamente}{casi\ totali}$. Il suo uso è però motivato dalla tecnica della discesa del gradiente, dato che come tipologia di errore è una funzione facilmente differenziabile. Lo scopo dell'algoritmo è quello di trovare un vettore di pesi finale: $w^* = w - \eta \nabla E(w)$ con $\eta > 0$ a partire da un vettore di pesi iniziale $w$. Questo particolare algoritmo si applica alle reti feedforward in cui la somma dei singoli

nodi viene considerata come $Si = \sum_j w_{ij}u_j$, mentre la funzione di attivazione è data da $f(S_i) = \frac{1}{1+e^{-S_i}}$. L'algoritmo si sviluppa in due importanti fasi:

**Propagazione in avanti** : effettua una computazione completa per tutte le celle dal basso verso l'alto calcolando per ogni cella $S_i$ e l'attivazione $ui = f(S_i)$.

**Propagazione all'indietro** : questa fase è quella che dà il nome all'algoritmo. A partire dalle celle di output, effettua una computazione dall'alto verso il basso calcolando: $f'(Si) = ui(1 - ui)$ e $\delta_i = (c_i - u_i)f'(S_i)$ se $u_i$ è una cella di output o $\delta_i = \sum_m (w_{mi}\delta_m)f'(S_i)$ per tutte le altre celle.

**Aggiornamento dei pesi** : $w^*_{ij} = w_{ij} + \eta\delta_i u_j$.

L'algoritmo Backpropagation ha segnato una svolta nelle reti neurali per l'introduzione della fase di propagazione all'indietro. Con il passo di aggiornamento dei pesi, dove vengono modificati i pesi $w_{ij}$ nei nuovi valori $w^*_{ij}$ utilizzando i $\delta_i$ precedentemente calcolati, si ottiene una retropropagazione dell'errore. Le reti neurali persero interesse quando si incontrò la difficolta di riuscire ad aggiornare i pesi delle celle degli strati intermedi per architetture a multistrato (input, hidden e output). Per neuroni non vicini all'output l'aggiornamento dei pesi avveniva in maniera molto lenta. Retropropagando l'errore, invece, si riesce a distribuire il contributo di errore che ogni singola cella porta all'errore totale, ottenendo così un miglior aggiornamento dei pesi.

## 4.4 Criteri di terminazione dell'apprendimento

Esistono due possibili modi di terminare l'apprendimento:

1. Fissare un criterio limite, come ad esempio terminare l'apprendimento se l'errore quadratico medio è al di sotto di una determinata soglia o se si è compiuto un numero di epoche di apprendimento che si ritiene sufficiente.

2. Utilizzare un insieme di dati supervisionati, distinti dagli esempi di apprendimento, detto Validation Set, con il quale testare l'efficienza della rete in fase di learning. Questo permette di evitare il fenomeno dell'overfitting 3.4, usando il criterio dell'Early Stopping: si interrompe l'addestramento della rete quando l'errore sul Validation Set inizia ad aumentare.

Per tutti gli esperimenti è stato usato un Test Set coincidente con il Validation Set. Dato che il programma di apprendimento per le Reti Neurali non fornisce un grafico d'errore, si è deciso di attuare l'Early Stopping manualmente. Per ogni epoca di apprendimento si è salvato il corrispondente vettore di pesi. Così facendo si è potuto testare l'errore della rete sugli esempi e scegliere quella che non commette overfitting sul Validation.

## 4.5 Deep Network

L'algoritmo di Backpropagation è stato il primo metodo efficace per riuscire ad addestrare delle Reti Neurali che possedessero uno o più strati di hidden; presenta tuttavia due problemi:

1. è necessario scegliere dei pesi iniziali casuali per tutte le connessioni. Se questi valori sono piccoli, è molto difficile riuscire ad apprendere con reti che possiedono più strati di hidden, perché il gradiente diminuisce in maniera proporzionale tutte le volte che si retropropaga l'errore. Se invece i pesi sono grandi, si sceglie in maniera casuale una porzione dello spazio dei pesi, per cui si rischia di incorrere in minimi locali.

2. l'ammontare di informazione che ogni esempio etichettato fornisce è al massimo pari al logaritmo delle possibili classi. Questo significa che grandi reti necessitano di un numero elevato di esempi etichettati se vogliono classificare correttamente gli esempi di test.

La classificazione di forme geometriche, come il testo scritto a mano, è stato un esperimento che ha mostrato le capacità di riconoscere pattern da parte delle Reti Neurali. Questo, come affermato in precedenza, avviene inizializzando in maniera casuale e con piccoli valori i pesi iniziali della rete. Per prevenire la rete dal modellare delle irregolarità presenti negli esempi, si attua l'Early Stopping oppure, nelle reti ad altà dimensionalità, si decide di imporre delle penalità per l'aggiornamento dei pesi (ad esempio sottraendo una quantità negativa proporzionale al valore del peso stesso al passo precedente di iterazione). Tutte queste strategie servono a migliorare le prestazioni della rete finale rispetto agli esempi di test; tuttavia, secondo uno degli inventori del Backpropagation *Geoffrey Hinton*, non sono nemmeno paragonabili alla ricerca di una strategia più efficace per inizializzare i pesi. Conviene mantenere la semplicità dell'utilizzo della discesa del gradiente per l'aggiornamento dei pesi, ma questi ultimi devono essere aggiornati in modo da massimizzare la probabilità per cui il modello probabilistico creato della rete sia il più possibile vicino a quello che ha effettivamente generato i dati. Per questo scopo *Hinton* ha sviluppato un tipo di rete chiamato Restricted Boltzmann Machine e successivamente le Deep Networks.

### Restricted Boltzmann Machine

Una RBM tenta di replicare il modello generativo dei dati forniti in ingresso usando un modello basato sul concetto di energia e di probabilità. Possiede un solo strato di hidden, le cui unità non sono connesse tra di loro ed hanno delle connessioni simmetriche e bidirezionali con le unità visibili (dette di input). Questo tipo di rete è rappresentato in figura (Fig. 4.5).

Le unità di hidden e quelle visibili sono tra di loro indipendenti (non c'è connessione tra unità dello stesso livello). Ogni possibile configurazione di nodi hidden e nodi visibili possiede un'energia $E(v, h)$ che dipende dai pesi delle connessioni e dai vari bias. In formula: $E(v, h) = -\sum v_i h_j w_{ij}$ con:

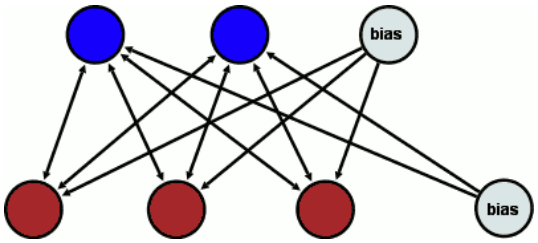


Figura 4.5: Struttura della RBM

- $v$ insieme delle unità visibili;
- $h$ insieme dei nodi di hidden;
- $v_i$ stato binario dell'unità visibile $i$;
- $h_j$ stato binario dell'unità di hidden $j$;
- $w_{ij}$ peso della connessione tra l'unità $i$ e l'unità $j$.

L'energia stessa determina una probabilità $p(v,h) = \frac{e^{-E(v,h)}}{\sum e^{-E(u,g)}}$ dove il membro a denominatore è la funzione di ripartizione che serve a normalizzare tutte le probabilità. La probabilità $p(v)$ delle sole unità visibili è data da $p(v) = \frac{\sum e^{-E(v,h)}}{\sum e^{-E(u,g)}}$. Considerando un ipotetico modello composto da infiniti nodi di hidden, detto *Infinite logistic belief net with tied weights*, in grado di approssimare bene la distribuzione dei dati, le operazioni di aggiornamento dei pesi che si compiono su tale modello sono quelle riportate in Fig. 4.6.

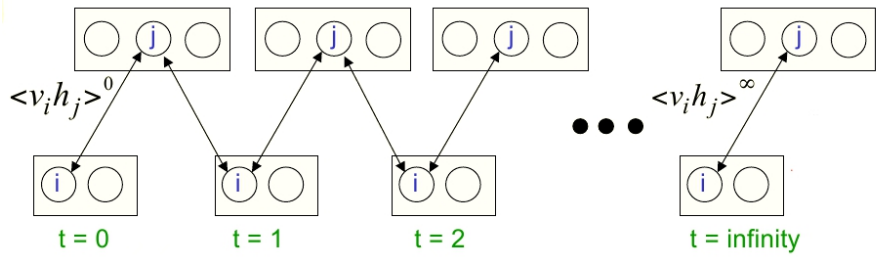


Figura 4.6: Ipotetico apprendimento con infiniti nodi hidden di una coppia input-hidden

Si presenta un esempio all'ingresso della rete e poi si alterna l'aggiornamento in parallelo di tutte le unità di hidden con l'aggiornamento in parallelo di tutte le unità visibili. Utilizzando le formule precedentemente introdotte, si calcola il gradiente del logaritmo della funzione di massima verosimiglianza rispetto al peso $w_{ij}$ e quello che si ottiene è che tutti i valori di aggiornamento intermedi tra $[0, \infty]$ si annullano. In formula: $\frac{\partial \log p(v)}{\partial w_{ij}} =< v_i h_j >^0 - < v_i h_j >^\infty$, dove $< v_i h_j >$ è il valore atteso della distribuzione. In realtà, quello che si utilizza per l'aggiornamento dei pesi è una formula leggermente diversa del logaritmo della massima verosimiglianza cioè: $\Delta w_{ij} = \epsilon(< v_i h_j >^0 - < v_i h_j >^1)$ dove $\epsilon$ è il learning rate. Considerata l'indipendenza delle unità visibili e di quelle di hidden, si può dire che:

- data un'immagine $v$ appartenente al training set, la probabilità che un nodo hidden $h_j$ abbia stato pari a 1 si ottiene dalla formula: $p(h_j = 1|v) = \sigma(b_j + \sum v_i wij)$;
- dato un vettore di hidden $h$, la probabilità che un nodo visibile $v_i$ abbia stato pari a 1 si ottiene dalla formula: $p(v_i = 1|h) = \sigma(a_i + \sum h_j wij)$.

In entrambe le formule, la $\sigma(x)$ è la funzione logistica definita nel capitolo 4 e $a_i, b_j$ sono i bias dei neuroni.
L'apprendimento svolge i seguenti passi:

1. si fornisce un esempio di train alle unità visibili.
2. si aggiornano tutte le unità di hidden in parallelo.
3. si aggiornano tutte le unità visibili in parallelo per ottenere una "ricostruzione" dell'esempio appena processato.
4. si aggiornano nuovamente le unità di hidden.
5. si ripete questo processo per tutti gli esempi di training.

Questa modalità di apprendimento è indicata in Fig. 4.7.

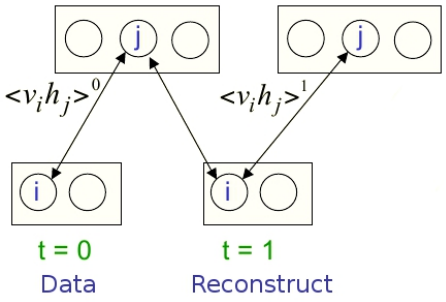


Figura 4.7: Apprendimento di una singola coppia input-hidden

### Deep Network

Il motivo principale per cui si usano le RBM è la possibilità di poter creare le cosiddette Deep Network. Una DN non è altro che una pila di RBM. Supposto di avere un insieme di RBM di cardinalità $N$, le reti sono collegate nel seguente modo:

- la prima RBM riceve in input gli esempi di training;
- le succesive $[1, N]$ reti usano lo strato hidden della precedente come proprio input, mentre come output globale della pila è preso l'hidden della rete $N$-esima.

Per il train della DN si effettuano 2 fasi, dette pretraining e raffinamento. Nel pretraining si addestrano le varie RBM separatamente: ad eccezione della prima alla quale è fornito l'esempio, le altre usano l'output della precedente come proprio input. Dopo questa fase le reti sono unite fino a formare un'unica rete raffinata retropropagando l'errore tra le reti. Il meccanismo è riportato in Fig. 4.8.

Si può usare questo meccanismo per creare un Deep Autoencoder (autoassociatore "profondo") che riesca a categorizzare meglio gli esempi di training.

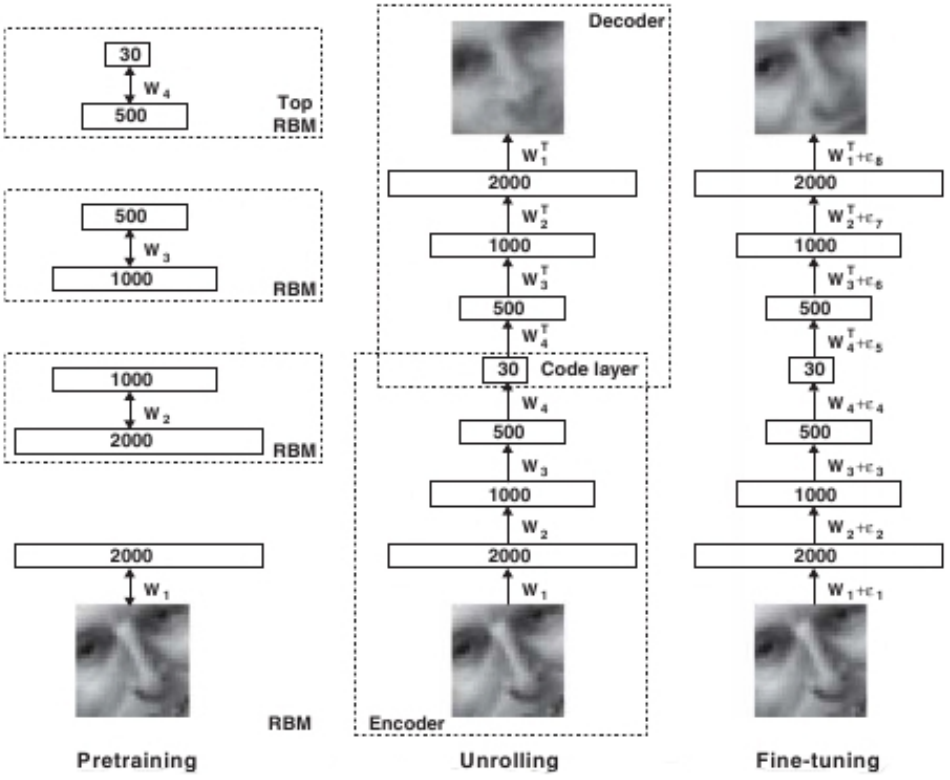


Figura 4.8: Esempio di schema di apprendimento di una Deep Network

## 4.6 Software utilizzato per le Reti Neurali

Per quanto concerne l'apprendimento con le Reti Neurali, si è deciso di usare un programma sviluppato dall'Università di Siena. Il programma è stato interamente sviluppato in linguaggio C (e per questo molto efficiente dal punto di vista della computazione) e richiede in input:

**Un file di configurazione** : dove sono presenti il learning rate, il numero di epoche, la soglia di errore.

**Un file .net** : contentente la struttura della rete.

**Un file di esempi traning** : nel formato .pat.

In output il programma restituisce un file di testo contente i pesi delle singole connessioni e i bias dei singoli neuroni. I criteri di arresto su cui si basa questo algoritmo sono quelli descritti nella sezione 4.4. Per ogni epoca si controlla se l'errore attuale è minore dell'errore minimo scelto o se il numero di epoca attuale è maggiore del limite massimo fissato nel file di configurazione. Se una di queste condizioni è verificata, allora l'apprendimento termina e viene restituito il file dei pesi finale corrispondente a quel dato apprendimento. Ogni 50 epoche viene fatta una copia dei pesi attuali della rete, così da poterli confrontare con quelli della rete finale ed avere un più ampio ventaglio di scelta. I diversi file di pesi generati possono così essere caricati ed utilizzati per il testing verificando l'errore reale della rete. Ciascun file è salvato in formato testuale e comporta due svantaggi:

1. il caricamento dei pesi all'avvio è molto lento.
2. la dimensione fisica del file su disco è elevata.

Questi due fattori possono essere trascurati quando si sta programmando per gli attuali PC fissi o portatili, dove le velocità di caricamento e le dimensioni della

memoria sono molto elevate. Non si può dire lo stesso quando stiamo parlando di sistemi embedded con memoria e potenza di calcolo limitate. Occorre trovare una soluzione alternativa. Si è optato per tradurre il file dei pesi testuali in un file di pesi binario che diminuisce la dimensione fisica e fornisce una lettura sequenziale più veloce di quella testuale. Si pensi che data una stringa di nove caratteri testuali, che rappresenta un valore numerico a virgola mobile, in linguaggio C, questa è salvata su nove byte, mentre in binario è salvata su quattro byte. Il risultato che si ottiene è la diminuzione delle dimensioni del file, pari alla metà del file originale. Per quanto riguarda la lettura usando, la funzione C che permette una scansione sequenziale di un array di dati binari, il tempo di lettura diminuisce di qualche secondo.

## 4.7 Software utilizzato per le Deep Network

Per l'apprendimento con le Deep Network è stata usata una libreria open source per *Matlab* che implementa gli algoritmi descritti da *Hinton*. *AndrejKarpathy* si occupa dello sviluppo di tale libreria, che è scaricabile dal suo sito internet[1]. Per gli apprendimenti sono state usate due funzioni:

1. dbnFit, alla quale sono forniti i dati, la struttura della rete e le etichette dei dati.

2. dbnPredict, che calcola l'accuratezza sulla base dei dati di test e delle loro etichette.

É stato quindi sufficiente far caricare a *Matlab* i dati normalizzati, compiere l'addestramento ed attendere i risultati.

[1] http://code.google.com/p/matrbm/

# Capitolo 5

# Schemi di classificazione

Un classificatore, come si intuisce dal nome stesso, è un'entità capace di associare un'etichetta ad un concetto, definendo così la sua classe di appartenenza. In questo capitolo saranno illustrati gli schemi di classificazione che sono stati usati durante tutta la fase degli apprendimenti e del testing sui dati. Tutti questi modelli sono stati utilizzati sia per le Reti Neurali, sia per gli Alberi di decisione. Le motivazioni del perché sono state costruite queste particolari architetture dipendono dai risultati che sono emersi durante le varie fasi del lavoro riportato nel capitolo 7. I modelli presentati sono:

1. Un classificatore generale, che distingue tutte e cinque le classi.
2. Un modello a cascata, schema generale del modulo, che comprende tre sottomodelli (classificatore grandi e piccoli, classificatore A,G ed un blocco C,D,E).
3. Un classificatore C,D,E, che distingue tra le tre classi.
4. Un classificatore C,D,E con modello campionato composto da tre sottosistemi (C,E, C,D e D,E) ed un votatore.

5. Un classificatore C,D,E composto da tre autoassociatori (uno per classe) ed un votatore.

6. Un classificatore C,D,E con Deep Network.

7. Un classificatore C/D,E.

## 5.1 Classificatore generale

Questo è il primo approccio che viene in genere usato per cercare di comprendere meglio come sono distribuiti i dati e a quali strategie ricorrere per migliorare (eventualmente) la classificazione successiva. Per quanto concerne gli Alberi di Decisione, non si devono attuare strategie mirate, mentre per le Reti Neurali si può utilizzare un adeguato numero di unità di hidden. Dato che la memoria cerebrale risiede sulle sinapsi, che sono i collegamenti tra neurone e neurone, il classificatore che dovrà cercare di etichettare i dati in ingresso sulla base degli esempi di training dovrà avere una "grande memoria". A livello di rete, è riscontrabile l'equivalenza tra i pesi e le sinapsi, ragion per cui maggiore è il numero di pesi, migliore (probabilmente) sarà la categorizzazione finale. Generalmente lo schema che si può adottare per questo scopo è quello risultante da un numero di nodi di hidden pari a $hidden = output * input$. In Fig. 5.1 è riportato lo schema del classificatore.

## 5.2 Modello a cascata

Lo schema generale usato nel modulo è quello riportato in Fig. 5.2.

E' stato addestrato un classificatore che distinguesse tra veicoli etichettati come grandi (Classi C, D, E) e veicoli etichettati come piccoli (Classi A, G). Questo permette di definire una prima partizione dei dati che verranno riconosciuti nei blocchi successivi. Per quanto riguarda le Classi A e G, è stato

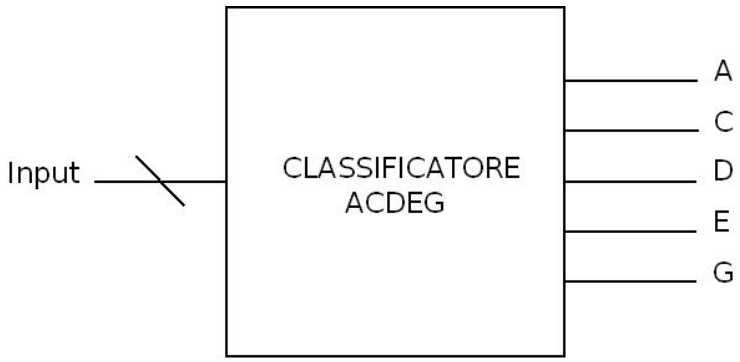


Figura 5.1: Classificatore generale

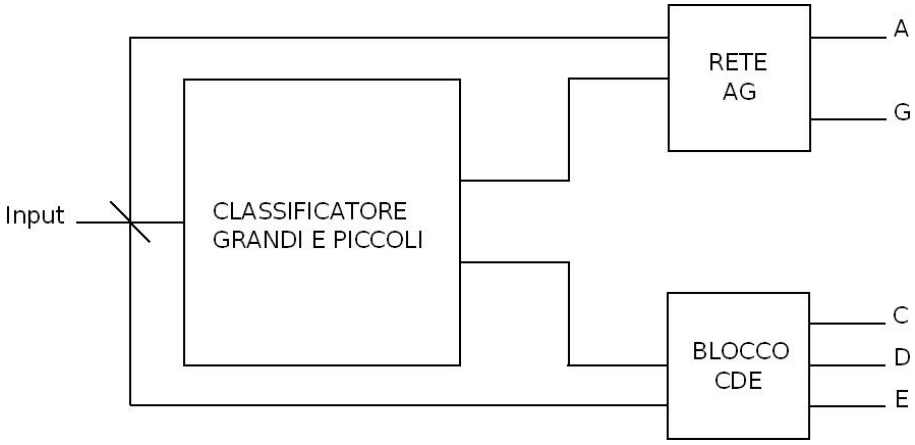


Figura 5.2: Modello a cascata

deciso di usare un classificatore singolo. Per le Classi C, D, E, sono stati creati quattro modelli che saranno illustrati di seguito.

### 5.2.1 Blocco CDE con classificatore

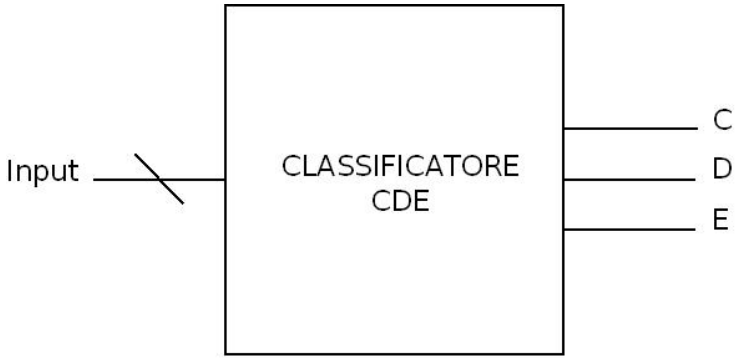


Figura 5.3: Classificatore C D E

Questo è l'approccio più semplice dei quattro. Dopo la prima partizione dei dati, è stato allenato un singolo classificatore a distinguere tra queste tre classi. E' possibile sperare che senza gli esempi delle Classi A e G riesca a trovare una concettualizzazione per le tre categorie rimaste compiendo un errore minore.

### 5.2.2 Blocco CDE con campionato

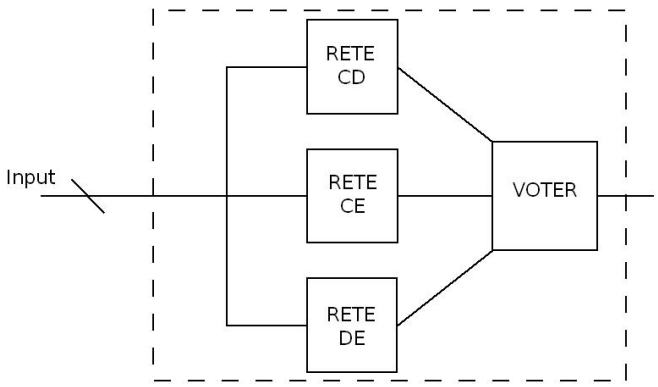


Figura 5.4: Modello campionato

Sono stati creati tre sottomodelli capaci di distinguere tra CD, CE e DE. Ogni singolo esempio è portato in ingresso a tutti e tre, mentre gli output sono inseriti all'interno di un votatore che decide la classe sulla base degli scontri avvenuti. Anche in questo caso si cerca di limitare il più possibile gli errori utilizzando delle codifiche ripetute: ad esempio, si possono trovare due codifiche diverse della classe C nella rete CD e nella CE.

### 5.2.3 Blocco CDE con autoassociatori

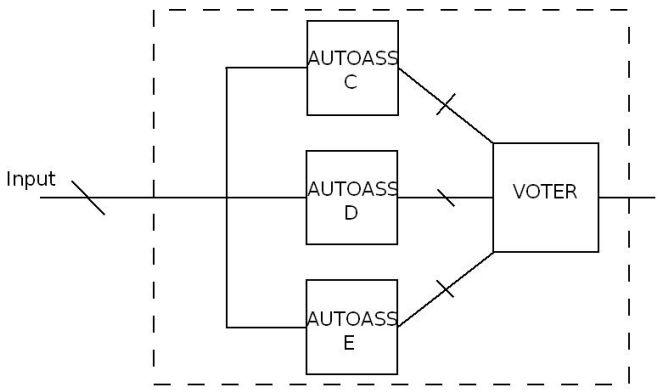


Figura 5.5: Modello autoassociatori

Le reti autoassociatrici sono quelle reti che tentano, dato un input, di replicarlo in output. Queste reti negli strati di hidden forniscono una codifica dell'input. Talvolta sono usate nella codifica/decodifica delle immagini. L'idea di fondo è quella di creare un autoassociatore che per ogni Classe crei per essa una codifica interna. Fatto questo, all'atto dell'esecuzione del programma, ogni esempio è presentato a tutte e tre le reti. Una volta processato l'output, si cerca quella rete che dà la distanza minima tra input e output e l'esempio è etichettato con la classe di quel particolare associatore. La distanza è calcolata come la distanza

euclidea al quadrato: $\sum_i (x_i - y_i)^2$. Questo approccio è stato usato soltanto per le Reti Neurali.

### 5.2.4 Blocco CDE con Deep Network

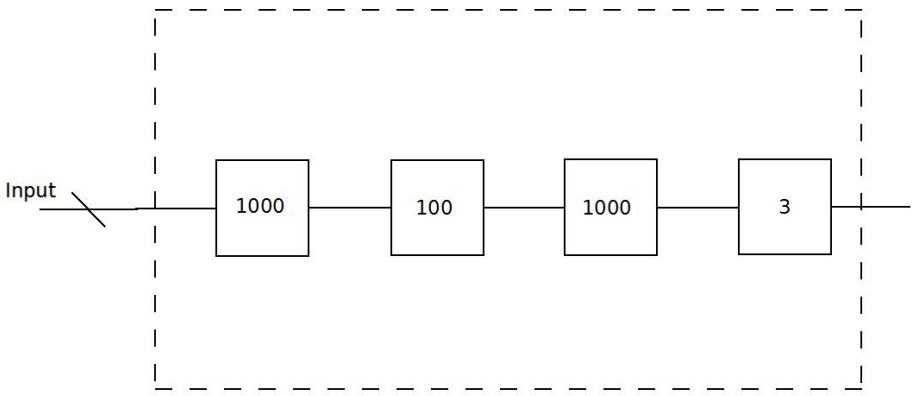


Figura 5.6: Modello con Deep Network

La rete ha una struttura di 1000x100x1000x3 neuroni, dove i primi tre strati si possono considerare come un autoassociatore, mentre l'ultimo è quello che raffina i pesi utilizzando le etichette dei vari esempi. Durante la fase di pretraining, si sono usate tre RBM aventi rispettivamente 1000x100, 100x1000, 1000x3 neuroni. Nella fase di raffinamento le reti sono state rimontate tra di loro ed addestrate nuovamente con i dati di train per migliorare l'efficienza globale della rete con uno schema simile a quello in Fig. 4.8.

### 5.2.5 Blocco CDE con classificatore CD/E

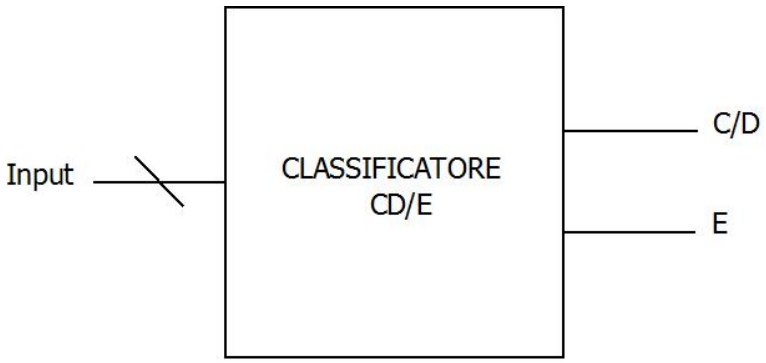


Figura 5.7: Classificatore C/D E

Questo modello non è altro che un classificatore che distingue tra la Classe E e tra le Classi C e D viste come un'unica categoria.

# Capitolo 6

# Software sviluppati

In questa sezione saranno descritti i programmi sviluppati per effettuare tutte le varie trasformazioni relative ai dati. Per la lettura, il salvataggio e il testing sulle immagini fornite è stata utilizzata la libreria Open Source OpenCV (Open Computer Vision), mentre per la gestione dei file di configurazione è stata usata la libreria Open Source TinyXML. Di queste librerie sono state prese in considerazione solo le parti che erano necessarie: il caricamento delle immagini ed il loro eventuale salvataggio (questa azione nella fase di testing dei programmi) ed il caricamento dei file XML. Il ricorso ad un file di configurazione permette di cambiare il programma all'atto dell'esecuzione modificando solamente poche stringhe di testo. Questo fatto è possibile perché nei vari programmi sono stati sfruttati, dove consentito, i concetti di ereditarietà e polimorfismo, come mostrato nei diagrammi UML in Fig. 6.3 e Fig. 6.12.

## 6.1 Parser di filtraggio

Lo schema generale di funzionamento del programma di filtraggio è il seguente: dato un insieme di immagini presenti nella medesima cartella, ciascuna di esse, a turno, è sottoposta alla catena descritta in Fig. 6.1


Figura 6.1: Catena di campionamento

La figura è di facile interpretazione:

- si carica un'immagine;
- l'immagine viene campionata riducendo le sue dimensioni da cinquanta per ottanta a venticinque per quaranta;
- si salvano i singoli pixel dell'immagine.

Alla fine quello che si ottiene è un file che sarà indicato con la sigla p0 che avrà la seguente struttura:

- percorso del file all'interno del filesystem;
- valori dei pixel dell'immagine espressi in numeri interi.

Questa prima tipologia di file è da considerarsi "grezza". La Fig. 6.2 mostra le possibili evoluzioni possibili a partire dal p0.

I dati del file di origine possono essere normalizzati in quattro differenti categorie di file che variano di poco l'una dall'altra. Il file .pat avrà la seguente struttura per ogni riga:

- numero dell'esempio di training;
- dati normalizzati;
- valore di supervisione.

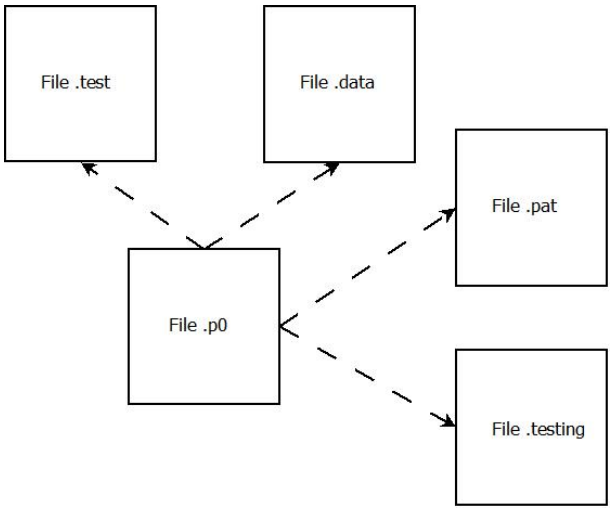


Figura 6.2: Possibili trasformazioni del file .p0

Il file .testing avrà la stessa struttura, solo che al posto del numero dell'esempio sarà presente il percorso dell'immagine di origine all'interno del filesystem. La differenza tra i due è data sopratutto dall'utilità di poter avere un riscontro visuale sugli errori che la rete commette nel testing ed eventualmente prendere opportune decisioni. Per gli Alberi di Decisione il file .test e .data hanno la stessa struttura. Ogni riga di questi file contiene:

- valori degli attributi dell'esempio separati da una virgola;
- valore di supervisione dell'esempio.

### 6.1.1 Schema programma

Di seguito è riportato il diagramma UML del programma in Fig. 6.3 con la descrizione delle singole classi.

Come si può facilmente intuire dalla figura, il programma applicativo crea un oggetto di tipo Parser che a sua volta istanzia due diversi oggetti: un Filter ed un DirIterator. Saranno illustrate più nel dettaglio queste tre classi e le loro derivate.

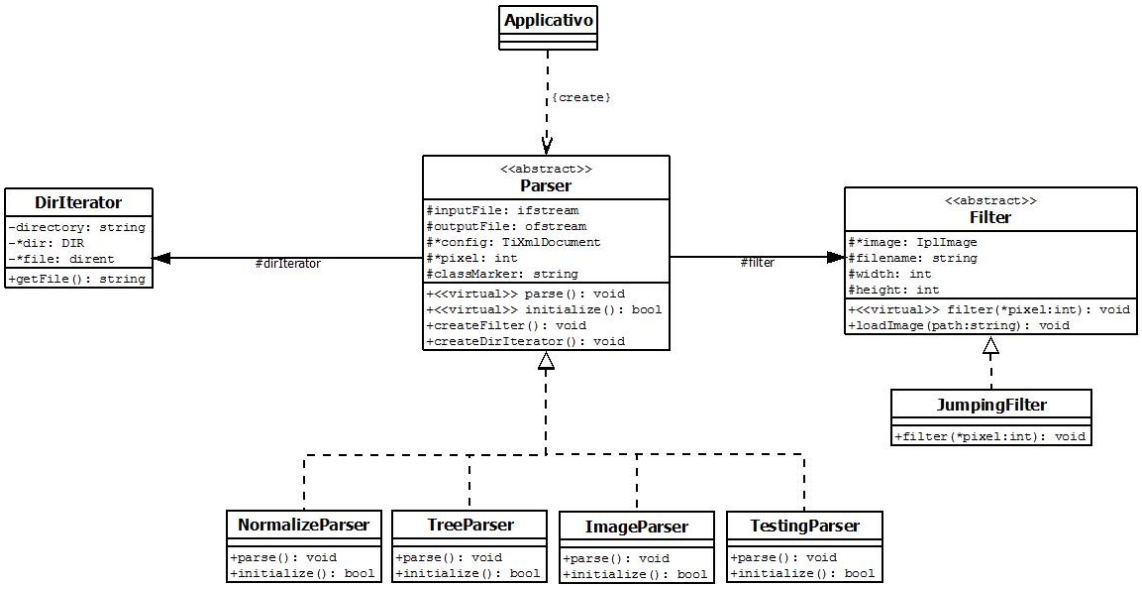


Figura 6.3: Schema UML del programma di Parser

### 6.1.2 Classe Parser

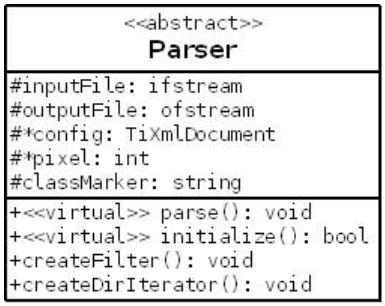


Figura 6.4: Modello UML della classe astratta Parser

Questa è la classe che fornisce tutta la gestione dei differenti tipi di file, come i .p0, i .pat e i .test. La classe ha come attributi:

- Un file di ingresso, nel caso si dovessero fare delle conversioni da un tipo di file ad un altro;
- Un file di uscita, che è l'output generato dalla classe dopo le diverse normalizzazioni;

- Un file di configurazione, che descrive le modalità di funzionamento del Parser;
- Un vettore di pixel, che contiene i dati provenienti dall'immagine letta;
- Un oggetto Filter, che provvede al caricamento ed al salvataggio dei dati dell'immagine nel vettore di pixel;
- Un oggetto DirIterator, che scansiona una cartella contenente le immagini da normalizzare;
- Un marcatore di classe, che è il valore di supervisione che viene aggiunto agli esempi di training.

La classe ha inoltre quattro metodi:

**parse** : ciclo principale del Parser. Questo metodo è implementato nelle classi derivate.

**inizialize** : funzione che svolge tutti i passi necessari per l'inizializzazione di un Parser. Anche questo metodo è implementato nella classi derivate.

**createFilter** : funzione che crea un determinato tipo di filtro specificato nel file di configurazione.

**createDirIterator** : funzione che crea il lettore di immagini.

### 6.1.3 Classe ImageParser

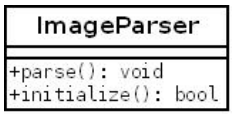


Figura 6.5: Modello UML della classe ImageParser

Derivata della classe Parser, implementa le due funzioni virtuali della classe base:

**parse** : metodo che dato un insieme di immagini crea il file .p0.

**inizialize** : crea il DirIterator,il filter e il file di output a partire dai dati specificati nel file di configurazione.

### 6.1.4 Classe NormalizeParser

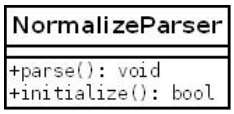


Figura 6.6: Modello UML della classe NormalizeParser

Derivata della classe Parser, implementa le due funzioni virtuali della classe base:

**parse** : metodo che dato un file .p0 normalizza i suoi dati, aggiunge il valore di supervisione e genera il file .pat.

**inizialize** : apre il file di input .p0 in lettura e crea il file di uscita .pat.

### 6.1.5 Classe TreeParser

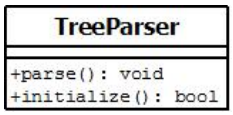


Figura 6.7: Modello UML della classe TreeParser

Derivata della classe Parser, implementa le due funzioni virtuali della classe base:

**parse** : metodo che dato un file .p0 normalizza i suoi dati, aggiunge le virgole come separatori tra i dati, aggiunge il valore di supervisione e genera i file .data o .test.

**inizialize** : apre il file di input .p0 in lettura e crea i file di uscita .test o .data.

### 6.1.6 Classe TestingParser

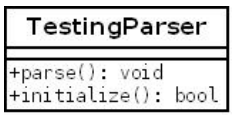


Figura 6.8: Modello UML della classe TestingParser

Derivata della classe Parser, implementa le due funzioni virtuali della classe base:

**parse** : metodo che dato un file .p0 normalizza i suoi dati, aggiunge il valore di supervisione e genera il file .testing.

**inizialize** : apre il file di input .p0 in lettura e crea il file di uscita .testing.

### 6.1.7 Classe Filter

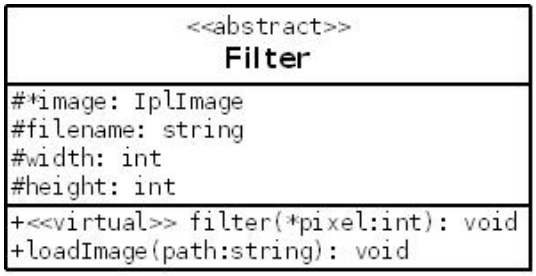


Figura 6.9: Modello UML della classe astratta Filter

Classe che provvede alla lettura dei pixel delle immagini. Ha come attributi:

- Un'immagine che è salvata nella struttura IplImage di OpenCV;
- Il nome dell'immagine;
- Le dimensioni dell'immagine.

Possiede inoltre due metodi:

**filter** : metodo virtuale implementato nelle classi derivate.

**loadImage** : metodo che permette di caricare una nuova immagine.

### 6.1.8 Classe JumpingFilter


Figura 6.10: Modello UML della classe JumpingFilter

Generalizzazione della classe Filter. Implementa l'unico metodo virtuale presente nella classe base:

**filter** : preleva i pixel dell'immagine ricorrendo alla strategia descritta in Fig. 2.2.

### 6.1.9 Classe DirIterator

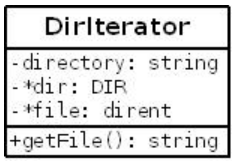


Figura 6.11: Modello UML della classe DirIterator

Classe che permette di leggere i file presenti in una data directory. I suoi attributi sono:

- Una struttura del linguaggio di programmazione C che permette di leggere una cartella chiamata DIR;
- Una struttura del linguaggio C che rende possibile la lettura di un file all'interno di una cartella;

- Il nome del percorso della cartella all'interno del filesystem;

Questa classe possiede solo un metodo:

**getFile** : metodo che restituisce il nome di un file all'interno della cartella.

## 6.2 Programma di testing

Il programma di testing è stato sviluppato prendendo a modello le architetture di rete presenti nel capitolo 5. Questo programma è stato realizzato unicamente per le Reti Neurali, dato che il C4.5 fornisce in automatico la percentuale degli esempi di training sbagliati. Dato un file di testing contenente delle immagini relative alle diverse classi, il programma opera nel seguente modo:

- si caricano i file dei pesi delle singole reti;
- si caricano ad uno ad uno gli esempi test presenti nel file;
- di ogni esempio si controlla il valore di supervisione e si fornisce l'input alla rete;
- si restituise il numero di esempi corretti e sbagliati in percentuale e per quelli sbagliati il nome dell'immagine originale da cui sono stati creati.

Quest'ultima operazione rende possibile un riscontro visivo degli esempi che la rete classifica erroneamente.

### 6.2.1 Schema programma

Di seguito riporto il diagramma UML del programma in Fig. 6.12 con la descrizione delle singole classi.

Questo diagramma rispetto ai precedenti è un po' più complesso e sarà descritto più nel dettaglio. E' presente una classe base chiamata

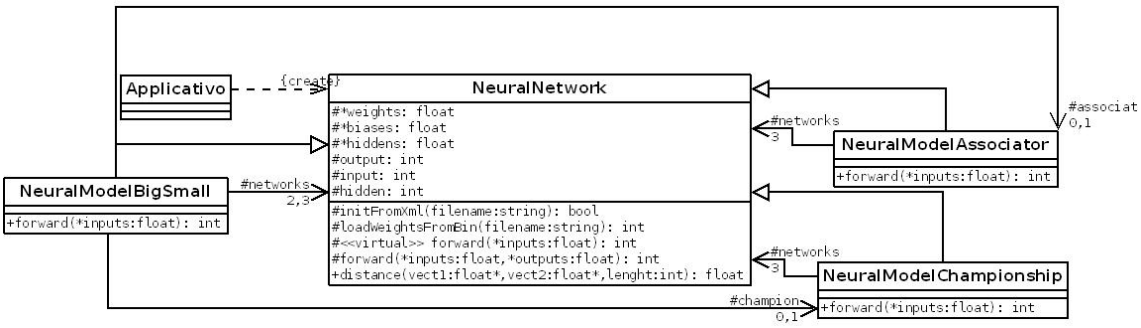


Figura 6.12: Schema UML del programma di Testing

NeuralNetwork, che è una Rete Neurale classica. Di questa classe abbiamo tre derivate: la NeuralModelBigSmall, la NeuralModelAssociator e la NeuralModelChampionship, rispettivamente gli schemi di classificazione: 5.2, 5.2.3 e 5.2.2. Per quanto riguarda il modello C/D,E (sezione 5.2.5) non si è creata una nuova classe, ma si è usata la NeuralNetwork. Da un punto di vista di programmazione, è stato conservato lo stesso macroschema riportato nel capitolo 5. Infatti le reti NeuralModelChampionship, NeuralModelAssociator e NeuralNetwork possono essere agevolmente scambiate tra di loro nel macroblocco di distinzione tra le Classi C, D, E, modificando una sola stringa nel file di configurazione. Di seguito verranno introdotte le parti essenziali di ogni classe.

### 6.2.2 Classe NeuralNetwork

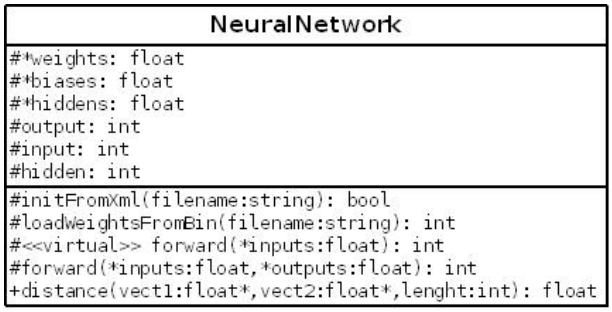


Figura 6.13: Modello UML della classe NeuralNetwork

La classe che implementa una rete neurale a tutti gli effetti. Ha come attributi:

- Un vettore di pesi;
- Un vettore di soglie;
- Un vettore di hidden;
- Il numero di input della rete;
- Il numero di output della rete;
- Il numero di hidden della rete.

I suoi metodi sono:

**initFromXML** : funzione che permette di inizializzare la rete da file di configurazione.

**loadWeightsFromBin** : funzione che permette di caricare i pesi della rete da file. binario

**forward prima variante** : funzione che calcola, dato un vettore di input, l'output della rete.

**forward seconda variante** : funzione che, dati un vettore di input ed uno di output, computa il risultato rispetto all'input e lo salva nell'output.

**distance** : funzione che calcola l'errore quadratico medio di due vettori dati.

### 6.2.3 Classe NeuralModelBigSmall

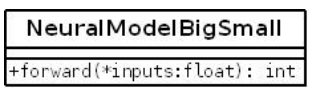


Figura 6.14: Modello UML della classe

La classe che sarà effettivamente usata nel sistema di monitoraggio. I suoi attributi sono:

- Un vettore di due o tre NeuralNetwork;
- Zero o una NeuralModelChampionship;
- Zero o una NeuralModelAssociator.

L'unica parte "fissa" della classe è costituita da NeuralNetwork, che distingue tra veicoli grandi e piccoli e tra Classe A e Classe G. A seconda che si ricorra all'architettura 5.2.1, all'architettura 5.2.3 o all'architettura 5.2.2 si avranno tre possibili situazioni:

1. Tre NeuralNetwork di cui la terza è un classificatore delle Classi C, D, E.
2. Due NeuralNetwork e una NeuralModelAssociator.
3. Due NeuralNetwork e una NeuralModelChampionship.

Tutte queste varianti sono intercambiabili tra di loro modificando solo un parametro nel file di configurazione. L'unica funzione di cui è stato fatto l'override è la forward, che non fa altro che indirizzare i dati ad un classificatore o all'altro sulla base di ciò che ritiene essere un veicolo grande o un veicolo piccolo.

### 6.2.4 Classe NeuralModelChampionship


Figura 6.15: Modello UML della classe NeuralModelChampionship

Questa classe è costituita da:

- Un classificatore CD;
- Un classificatore CE;
- Un classificatore DE.

Il suo metodo forward fornisce l'input a tutti e tre i classificatori e stila una classifica degli "scontri" avvenuti. L'unica situazione sfavorevole che si può ottenere con questo metodo (anche se è molto difficile) è la parità delle tre Classi, dove ognuna è stata battuta ed ha battuto rispettivamente un'altra Classe. In questo caso è stato deciso di considerare tutto come classe C.

### 6.2.5 Classe NeuralModelAssociator

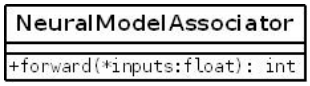


Figura 6.16: Modello UML della classe NeuralModelAssociator

Questa classe è costituita da tre autoassociatori: uno per la Classe C, uno per la D ed uno per la E. Come per il modello a campionato, anche questa generalizzazione di NeuralNetwork implementa la funzione forward. Ad ogni autoassociatore è fornito in ingresso l'esempio da classificare. Le tre reti computano l'output e per ogni rete viene calcolato l'errore tra ingresso ed uscita tramite la funzione distance. La rete che darà errore minimo sarà quella che etichetterà l'esempio.

# Capitolo 7

# Risultati sperimentali

## 7.1 Esperimenti

Una volta conclusa la parte di raccolta dei dati (cap. 2), sono stati svolti gli esperimenti per la costruzione del modulo. Per capire come organizzare tutto il lavoro, è stato compiuto un esperimento iniziale molto semplice: è stato creato un classificatore generale per tutte e cinque le classi. Anche se all'apparenza può sembrare una scelta ovvia e banale, in realtà non lo è. Dall'analisi dei risultati di questo modello, si può notare in che modo i dati sono valutati e se esistono, per così dire, delle "somiglianze" per cui una classe viene confusa con un'altra. Questo primo approccio ha fornito i seguenti risultati:

1. le Classi A e G sono facilmente distinguibili e la rete (o l'albero) non compie un errore eccessivo nel categorizzarli. Queste due classi sono confuse raramente con le altre.

2. le Classi C, D, E vengono confuse tra loro ma non con le altre due Classi.

Questo risultato conferma quanto ipotizzato a priori soltanto a partire dall'osservazione delle immagini; infatti, come precedentemente affermato, è difficile distinguere ad occhio nudo le Classi C, D (si distinguono solo per le

bande). Sulla base di questi risultati, è stato deciso di effettuare una prima distinzione tra i dati suddividendo le cinque classi in due macroclassi: veicoli grandi e veicoli piccoli (schema sez. 5.2). Questa prima divisione fa sì che si possano trattare in modo diverso i veicoli di Classe A,G e quelli di Classe C,D,E, cercando di migliorare il più possibile l'efficienza di categorizzazione dei singoli sottomodelli, specializzandoli nel trattare solo un sottoinsieme dei dati. Sulla base delle osservazioni effettuate nella prima fase, è stato osservato che il sistema riconosceva bene le Classi A e G; per questo si è deciso di usare un singolo categorizzatore per distinguerle, mentre per le altre tre classi si è dovuto ricorrere a schemi più complessi. Le strategie intraprese (in ordine cronologico) per classificare le Classi C,D,E sono:

1. usare un singolo categorizzatore.
2. usare un modello a campionato.
3. usare un modello ad autoassociatori.
4. usare un modello con Deep Network.

La prima scelta risulta ovvia, ma si può sperare che, avendo tolto gli esempi di auto e moto dal learning set, in quanto veicoli piccoli, la rete formuli un'ipotesi migliore per distinguere queste classi. Purtroppo anche in questo caso la rete compie un errore elevato. Questo ci porta agli altri due modelli. Il campionato descritto in Fig. 5.4 non è altro che un sistema composto da tre classificatori (rispettivamente CD,CE,DE) ed un votatore che decide la classe del veicolo sulla base dei tre sottosistemi. Sulla base di osservazioni sperimentali, si è notato che il classificatore che discrimina una corretta etichettatura da una sbagliata è quello CD. Come già affermato nel capitolo 2, le Classi C,D sono le più simili tra loro e per questo è difficile trovare un'ipotesi che permetta una distinzione netta tra le due. In questo particolare modello, il sottosistema CD può essere considerato come un punto di fallimento:

- se il sottosistema categorizza correttamente, il modello 5.4 funziona;
- se il sottosistema non categorizza correttamente, il modello 5.4 non funziona.

Il modello ad autoassociatori in Fig. 5.5 cerca di dare una codifica interna alla classe tramite gli esempi di training. Quando a quest'architettura è fornito un esempio, i tre sottosistemi cercano di riprodurre l'ingresso in uscita. Ovviamente viene classificato l'esempio con la rete che è riuscita a replicare l'output più vicino. Di tutti i modelli possibili per la categorizzazione C,D,E, si è osservato che quest'ultimo è il meno robusto dei tre. Per il modello 5.6 costruito con le Deep Network, l'efficienza è minore rispetto agli autoassociatori convenzionali e questo, almeno in via teorica, non dovrebbe succedere[1]. I medesimi esperimenti sono stati fatti anche con un ultimo modello (Fig. 5.7) in grado di distinguere tra veicoli C/D ed E, dove la C/D è considerata come classe di veicoli pesanti. L'efficienza di questo modello è maggiore rispetto a quella degli altri modelli, anche se si perde la capacità di categorizzare tre classi. Di seguito sono riporate le tabelle relative alla *10-cross fold validation*.

[1]Nei suoi esperimenti *Hinton* ha verificato che i Deep Autoencoder funzionano meglio degli autoassociatori convenzionali. Questo fatto non è stato riscontrato durante gli esperimenti di questa tesi.

| Rete | AG | GP | C/D,E | CDE | CDE Camp. | CDE Auto. | CDE DN |
|---|---|---|---|---|---|---|---|
| Fold 1 | 97,5 | 95,45 | 96,15 | 92,31 | 88,46 | 61,53 | 65 |
| Fold 2 | 100 | 96,97 | 100 | 76,92 | 80,77 | 57,69 | 52,31 |
| Fold 3 | 97,5 | 100 | 92,31 | 80,77 | 80,77 | 61.53 | 59,99 |
| Fold 4 | 100 | 98,48 | 100 | 88,46 | 88,46 | 61,53 | 63,07 |
| Fold 5 | 100 | 100 | 84,62 | 65,38 | 69,23 | 69,23 | 48,84 |
| Fold 6 | 95 | 100 | 96,15 | 69,23 | 69,23 | 50 | 52,69 |
| Fold 7 | 100 | 98,48 | 100 | 84,62 | 76,92 | 57,69 | 52,31 |
| Fold 8 | 100 | 98,48 | 96,15 | 84,62 | 84,62 | 61,53 | 58,46 |
| Fold 9 | 100 | 98,48 | 92,3 | 80,77 | 84,61 | 57,69 | 74,23 |
| Fold 10 | 100 | 98,48 | 92,31 | 80,77 | 84,61 | 57,69 | 49,62 |
| Media | 99 | 98,48 | 95 | 80,38 | 80,77 | 59,61 | 57,65 |
| Deviazione std. | 1,75 | 1,43 | 4,81 | 8,2 | 7,02 | 4,88 | 8,08 |

Tabella 7.1: Risultati 10-cross fold Reti Neurali

| Albero | AG | GP | C/D,E | CDE |
|---|---|---|---|---|
| Fold1 | 100 | 95,5 | 84,6 | 61,5 |
| Fold2 | 100 | 98,5 | 96,2 | 65,4 |
| Fold3 | 100 | 100 | 84,6 | 65,4 |
| Fold4 | 100 | 95,5 | 80,8 | 61,5 |
| Fold5 | 100 | 100 | 80,8 | 65,4 |
| Fold6 | 95 | 98,5 | 80,8 | 69,2 |
| Fold7 | 100 | 97 | 84,6 | 69,2 |
| Fold8 | 100 | 95,5 | 84,6 | 61,5 |
| Fold9 | 100 | 97 | 84,6 | 50 |
| Fold10 | 100 | 95,5 | 92,3 | 73,1 |
| Media | 99,5 | 97,3 | 85,39 | 64,22 |
| Deviazione std. | 1,58 | 1,84 | 5,07 | 6,3 |

Tabella 7.2: Risultati 10-cross fold Alberi di Decisione

Dall'analisi delle tabelle si può notare che le percentuali delle reti e degli alberi sono molto simili tra loro per quanto riguarda la distinzione tra veicoli grandi e piccoli (G,P) e tra auto e moto (A,G). Questo risultato testimonia che il modello proposto (Fig. 5.2 è attendibile e pertanto i blocchi G,P e A,G non verranno ulteriormente modificati. Diverso è il caso del blocco a campionato. Si osserva dalle Tab. 7.3 e Tab. 7.4 che la deviazione standard è abbastanza elevata (come nel caso D,E negli alberi) per quasi tutti gli esperimenti.

| Rete | CD | CE | DE |
|---|---|---|---|
| Fold1 | 95 | 100 | 88,24 |
| Fold2 | 75 | 93,33 | 100 |
| Fold3 | 85 | 86,67 | 88,24 |
| Fold4 | 90 | 100 | 94,12 |
| Fold5 | 60 | 86,67 | 82,35 |
| Fold6 | 65 | 93,33 | 88,24 |
| Fold7 | 70 | 100 | 100 |
| Fold8 | 90 | 93,33 | 94,12 |
| Fold9 | 95 | 73,33 | 88,24 |
| Fold10 | 70 | 86,67 | 100 |
| Media | 79,5 | 91,33 | 92,35 |
| Deviazione std. | 13,01 | 8,34 | 6,23 |

Tabella 7.3: Risultati CDE campionato Reti Neurali

Sempre in base alle tabelle, si può dedurre che per questo tipo di classificazione le Reti Neurali funzionano meglio degli Alberi di Decisione, mentre quando si considerano gli altri due classificatori l'efficienza è più o meno la stessa. Questo risultato è motivato dal modo in cui sono costituiti i dati. Creando delle soglie, gli alberi riescono a capire quando un veicolo è grande o piccolo, basandosi solo sulla dimensione effettiva della ripresa all'interno dell'immagine (Fig. 2.1); lo

| Albero | CD | CE | DE |
|---|---|---|---|
| Fold1 | 80 | 93,3 | 50 |
| Fold2 | 70 | 86,7 | 60 |
| Fold3 | 85 | 80 | 55 |
| Fold4 | 60 | 73,3 | 55 |
| Fold5 | 55 | 80 | 55 |
| Fold6 | 70 | 86,7 | 45 |
| Fold7 | 60 | 80 | 35 |
| Fold8 | 65 | 80 | 70 |
| Fold9 | 95 | 60 | 55 |
| Fold10 | 55 | 73,3 | 65 |
| Media | 69,5 | 79,33 | 54,5 |
| Deviazione std. | 13,43 | 9,14 | 9,85 |

Tabella 7.4: Risultati CDE campionato Alberi di Decisione

stesso vale per le Classi A e G. Per le altre tre classi è difficile creare delle soglie nette di distinzione, visto che le tre categorie sono molto simili tra loro. Un altro problema è dovuto al fatto che tutti gli attributi sono dati continui, mentre gli Alberi di Decisione lavorano meglio su dati discreti. Le Reti Neurali invece sono in grado di cogliere eventuali differenze e sono robuste rispetto a dati rumorosi e provenienti da sensori (come quelli della Smart Cam). Sulla base di queste considerazioni, si è deciso di non svolgere altri esperimenti sulle Classi C,D ed E per gli Alberi di Decisione.

## 7.2 Sperimentazioni su di un traffico reale

In un secondo momento, è stato fornito un insieme di 1753 nuove immagini, derivate dalla ripresa di un normale traffico autostradale. Tutte queste immagini

sono degli esempi che nessuno dei *fold* ha mai visto e sono divise in:

- 1664 per la Classe A;
- 29 per la Classe C;
- 12 per la Classe D;
- 13 per la Classe E;
- 35 per la Classe G.

Si è potuto così effettuare un nuovo test sui vari schemi presenti e misurare nuovamente la loro efficienza. Il modello che usa le DN non è stato preso in esame, dato che i suoi risultati sono peggiori rispetto a quelli degli altri classificatori. Alla luce delle considerazioni precedenti, si riportano in Tab. 7.5 e Tab. 7.6 i risultati di Reti Neurali ed Alberi di Decisione.

| Rete | AG | GP | C/D,E | CDE | CDE Camp. | CDE Auto. |
|---|---|---|---|---|---|---|
| Fold1 | 98,76 | 98,57 | 87,04 | 66,67 | 74,07 | 66,67 |
| Fold2 | 99,82 | 98,57 | 85,19 | 66,67 | 72,22 | 64,81 |
| Fold3 | 98,76 | 98,4 | 85,19 | 68,52 | 74,07 | 68,52 |
| Fold4 | 98,65 | 98,57 | 83,33 | 68,52 | 72,22 | 66,67 |
| Fold5 | 98,47 | 98,86 | 85,19 | 61,11 | 68,51 | 64,81 |
| Fold6 | 98,71 | 98,46 | 83,33 | 68,52 | 75,93 | 66,67 |
| Fold7 | 98,79 | 98,86 | 81,48 | 70,37 | 72,22 | 66,67 |
| Fold8 | 98,59 | 98,63 | 85,19 | 70,37 | 70,37 | 70,37 |
| Fold9 | 98,71 | 99,03 | 87,03 | 72,22 | 75,93 | 70,37 |
| Fold10 | 98,71 | 98,57 | 85,19 | 72,22 | 68,52 | 72,22 |
| Media | 98,8 | 98,65 | 84,81 | 68,52 | 72,41 | 67,78 |
| Deviazione std. | 0,37 | 0,2 | 1,7 | 3,27 | 2,68 | 2,5 |

Tabella 7.5: Risultati simulazione traffico sulle Reti Neurali e Deep Network

| Albero | AG | GP |
|---|---|---|
| Fold1 | 94 | 98,7 |
| Fold2 | 94 | 99 |
| Fold3 | 94 | 99 |
| Fold4 | 94 | 99 |
| Fold5 | 94 | 99,1 |
| Fold6 | 95,9 | 99,3 |
| Fold7 | 94 | 99,1 |
| Fold8 | 94 | 98,6 |
| Fold9 | 94 | 99,1 |
| Fold10 | 94 | 98,7 |
| Media | 94,19 | 98,96 |
| Deviazione std. | 0,6 | 0,22 |

Tabella 7.6: Risultati simulazione traffico sugli Alberi di Decisione

Visti i precedenti risultati, è stato deciso di creare un modulo strutturato con il seguente schema:

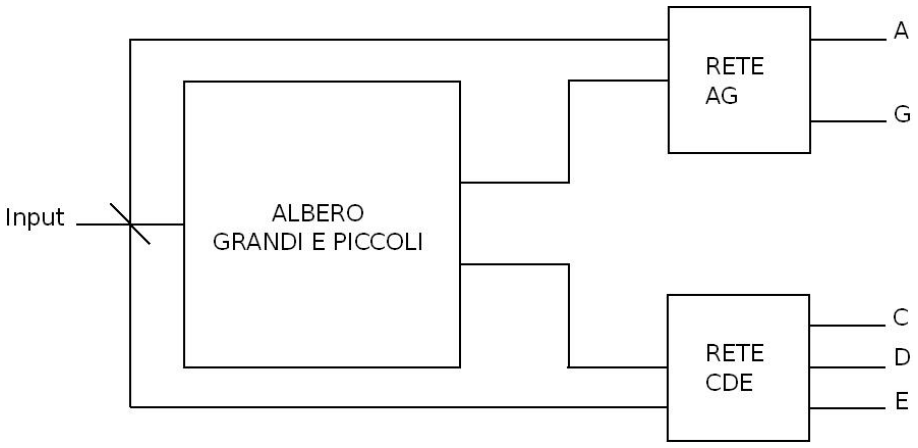


Figura 7.1: Schema complessivo del modulo

Si usa un Albero di Decisione per la distinzione tra veicoli piccoli e veicoli grandi, mentre si ricorre alle Reti Neurali per l'identificazione delle classi C,D,E o C/D,E (a seconda che si ricorra agli schemi 5.2.3,5.2.2,5.2.5 ed A,G. I risultati di quest'ultimo test sono riportati in tab. 7.7.

| Schema finale | Percentuali |
|---|---|
| CDE | 97,14 |
| C/D,E | 97,6 |
| Campionato | 97,26 |
| Autoassociatori | 96,98 |

Tabella 7.7: Risultati modulo

La categorizzazione più efficiente è quella fornita dalla rete C/D,E. Se non si vuole perdere la distinzione di tutte e cinque le classi basta aggiungere in cascata a questa rete un modulo che riconosca le classi C e D (Fig. 7.2).

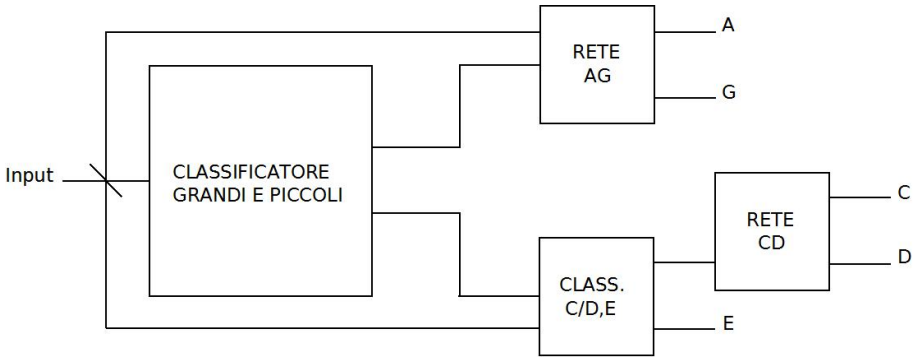


Figura 7.2: Schema finale del modulo

## 7.3 Ulteriori esperimenti

Può sorgere spontaneo domandarsi come sarebbero cambiati i risultati sfruttando anche i 1753 esempi del traffico finale. Dalle tabelle 7.5 e 7.6, si può capire che, per quanto riguarda la categorizzazione di veicoli grandi e piccoli e la distinzione tra Classe A e G, le due reti hanno un'efficienza praticamente identica rispetto a quella riscontrata nel Test Set (Tab. 7.1 e 7.2). Si può pensare che un aumento dei dati di train non modifichi queste percentuali. Invece, per quanto riguarda gli altri modelli, si può ripetere la *10-cross fold validation* e calcolare le nuove efficienze. In conclusione, aumentando il numero di esempi, l'efficienza complessiva dovrebbe essere maggiore. La tabella 7.8 riporta i risultati di quest'ultimo esperimento. Ovviamente sono state testate solo le Reti Neurali perché, come detto in precedenza, gli Alberi di Decisione confondono più spesso le Classi C,D,E.

Si possono osservare i seguenti fatti:

- si sono ottenuti valori uguali o di poco migliori rispetto alla precedente *10-cross fold validation*;
- le varianze di quasi tutte le reti sono diminuite.

Disponendo di ulteriori dati, si potrebbero ottenere dei risultati migliori, ma non si può esserne certi. Questo procedimento potrebbe essere ripetuto all'infinito

| Rete | C/D,E | CDE | Campion. | Autoass. | CDE DN |
|---|---|---|---|---|---|
| Fold1 | 93,75 | 81,25 | 84,38 | 62,5 | 59,37 |
| Fold2 | 93,75 | 84,38 | 84,38 | 81,25 | 65 |
| Fold3 | 90,63 | 71,88 | 81,25 | 71,88 | 60,31 |
| Fold4 | 84,38 | 71,88 | 75 | 68,75 | 53,43 |
| Fold5 | 90,63 | 78,13 | 84,38 | 87,5 | 60,94 |
| Fold6 | 90,63 | 68,75 | 78,13 | 78,13 | 55,62 |
| Fold7 | 96,88 | 71,88 | 78,13 | 71,88 | 51,87 |
| Fold8 | 90,63 | 87,5 | 81,25 | 71,88 | 63,12 |
| Fold9 | 93,75 | 71,88 | 75 | 75 | 51,87 |
| Fold10 | 96,88 | 81,25 | 78,13 | 75 | 59,63 |
| Media | 92,19 | 76,88 | 80 | 74,38 | 58,09 |
| Deviazione std. | 3,68 | 6,45 | 3,67 | 6,88 | 4,65 |

Tabella 7.8: Risultati nuova 10-cross fold

spendendo risorse anche elevate di tempo e denaro, per cui è consigliabile cercare di lavorare al meglio con i dati a disposizione per realizzare un prodotto che sia il più compatibile possibile con le richieste del committente.

# Capitolo 8

# Conclusioni e sviluppi futuri

In questa tesi è stato affrontato un problema di natura pratica. Si è cercato di migliorare una funzione di categorizzazione presente in un software usando delle tecnologie consolidate, come le Reti Neurali e gli Alberi di Decisione, e sperimentando le più recenti Deep Network. Si sono presentati:

- i formati delle immagini usati per l'apprendimento ed i criteri di estrazione di features;
- una breve parte teorica che mostra gli strumenti usati;
- gli schemi riguardanti i diversi apprendimenti e le relative percentuali di riconoscimento.

Quello che si è ottenuto è uno schema generale del modulo che sarà inserito in un futuro prossimo all'interno del programma operante la distinzione di cinque diverse classi.

## 8.1 Sviluppi futuri

Alla fine di tutti gli esperimenti sono emerse le seguenti problematiche:

1. Durante la sperimentazione delle Deep Network si è notato che il loro funzionamento è risultato peggiore rispetto a quello di un normale autoassociatore. *Hinton*, nei suoi articoli, le descrive come una valida alternativa alle Reti Neurali, per cui è molto strano non aver ottenuto almeno gli stessi risultati. Le possibili cause di questo malfunzionamento sono due:

   (a) un errato uso della libreria che gestisce le DN.

   (b) un problema relativo all'aggiornamento dei pesi nella delicata fase di raffinamento.

   La prima opzione è quella che si ritiene più probabile.

2. Il campionamento effettuato per gli esperimenti si limita ad estrarre dalle immagini i valori dei pixel ed a normalizzare tali valori in un intervallo [0,1]. É lecito domandarsi se non esistano altre tecniche di estrazione che migliorino l'accuratezza del modello. Ad esempio, si potrebbe pensare di effettuare una sogliatura delle immagini eliminando la parte nera e lasciando intatto solo il veicolo.

3. Il modello riportato in Fig.7.2 è stato descritto, ma non testato, in quanto non si disponeva di altri esempi di training e di test.

Tutte queste problematiche saranno affrontate successivamente:

- compiendo dei test sul funzionamento della Deep Network e concentrandosi più specificatamente sulle singole RBM che compongono tale rete;
- cercando e sperimentando nuove strategie di campionamento;
- prelevando nuovi esempi di trainig.

# Appendice A

# Is Traffic, Sistema di monitaraggio del traffico

## A.1 Scopo

Lo scopo del sistema è quello di eseguire un'analisi automatica del flusso di traffico in modo da rilevare eventi quali rallentamenti, code, veicoli fermi, ecc. basandosi unicamente sull'elaborazione delle immagini. L'elaborazione è eseguita direttamente a bordo di una telecamera intelligente: il sistema è stato sviluppato utilizzando una Smart Cam Sony XCI-V3 con sistema operativo Linux. Il sistema è stato messo a punto sull'autostrada A11 nei pressi di Firenze.

## A.2 Elaborazioni

Durante l'installazione della telecamera, il sistema viene configurato definendo l'area interessata al flusso del traffico, le corsie esistenti al suo interno ed alcuni riferimenti geometrici necessari alla stima della velocità media del flusso. Alla partenza, il programma determina uno sfondo di riferimento iniziale che successivamente viene aggiornato continuamente. Dopo la determinazione dello

sfondo, ogni immagine viene elaborata per rilevare i veicoli presenti. Ogni minuto vengono aggiornati i seguenti parametri di traffico:

- occupazione delle corsie;
- velocità media del flusso per corsia;
- numero dei veicoli per corsia.

Questi parametri costituiscono l'input di un sistema esperto con un motore inferenziale in logica fuzzy utilizzato per determinare lo stato del traffico sia di ogni singola corsia che dell'intera carreggiata:

- traffico nullo o scarso;
- traffico regolare;
- traffico rallentato;
- coda;

Vengono rilevati inoltre particolari comportamenti del traffico, quali:

- veicolo fermo;
- veicolo ripartito;
- veicolo contromano.

Quando uno di questi comportamenti particolari viene rilevato oppure il traffico cambia il suo stato, viene generato un evento, registrato un corrispondente filmato ed attivato un segnale di allarme. Un'altra caratteristica del sistema è la generazione di un flusso video MPEG-4 per il monitoraggio remoto del tratto di strada inquadrato dalla telecamera.

## A.3 Architettura di sistema

Il sistema è composto da tre sottosistemi, oltre che dalle apparecchiature di rete:

**SC** : Smart Cam, telecamera intelligente con sistema operativo LINUX embedded, per l'acquisizione delle immagini e la loro elaborazione come descritto nella precedente sezione.

**VS** : Video Server, per la generazione del flusso video MPEG-4 e la registrazione dei filmati relativi agli eventi.

**MU** : Monitoring Unit, per il monitoraggio remoto e la gestione degli eventi.

In questa architettura, l'unità di monitoraggio può connettersi a diversi video server, ognuno dei quali a sua volta può connettersi a diverse telecamere intelligenti. Quando viene abilitata, la Smart Cam genera un flusso continuo di immagini JPEG verso il video server, che lo converte in un flusso MPEG-4 verso l'unità di monitoraggio. La rete limita il numero di flussi video che possono essere monitorati simultaneamente. Il video server è dotato di una memoria di massa per l'archiviazione dei filmati corrispondenti agli eventi rilevati sulla telecamera. L'unità di monitoraggio ha una interfaccia grafica per il monitoraggio del flusso video delle telecamere selezionate, e per avvertire l'operatore ogni volta che viene rilevato un evento da una delle telecamere collegate: l'operatore può allora visualizzare il filmato corrispondente. L'unità di monitoraggio può inoltre comandare un pannello a messaggi variabili per avvertire l'utenza sulla strada. Uno schema a blocchi dell'architettura è rappresentato in Fig. A.1:

Il sistema può essere configurato in una versione semplificata, senza video server, dove è l'unità di monitoraggio a ricevere direttamente dalle telecamere il flusso JPEG ed eseguire l'archiviazione dei filmati. La smart cam genera un flusso

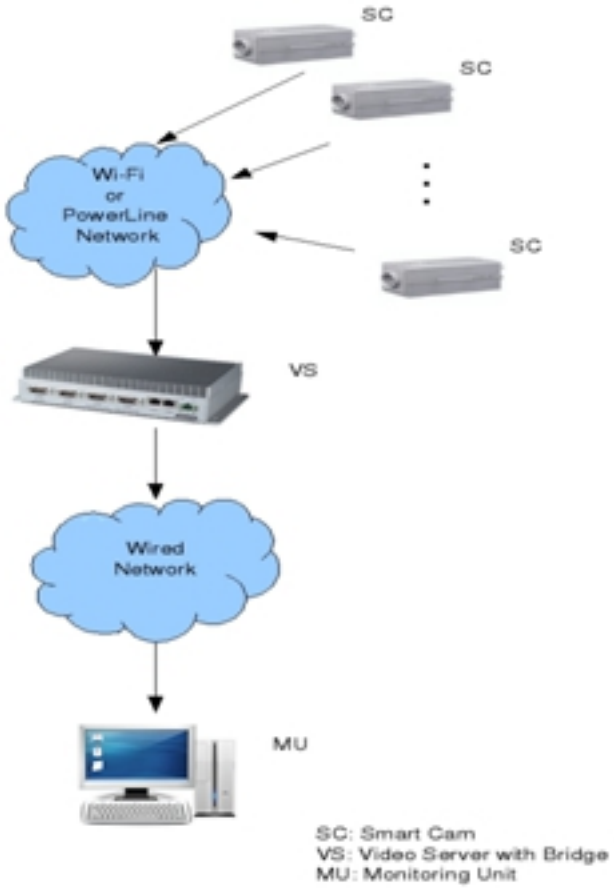


Figura A.1: Architettura con Server

video M-JPEG che può avere problemi con reti lente o con molto traffico. Uno schema a blocchi della configurazione semplificata è rappresentato in Fig. A.2:

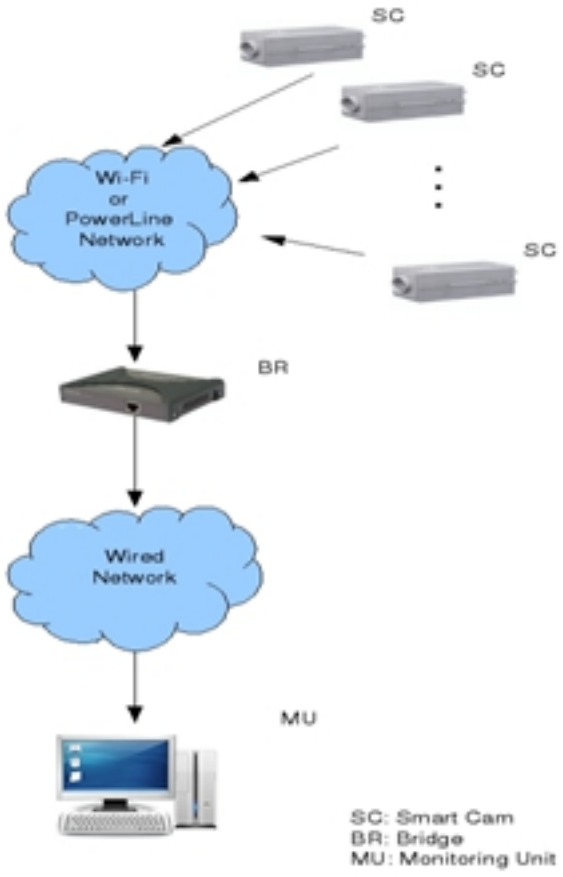


Figura A.2: Architettura senza Server

## A.4 Interfaccia di rete

Generalmente, in un ambiente aperto, le telecamere possono essere connesse al video server tramite una LAN WI-MAX, qualora non esista un rete cablata. Nelle applicazioni in galleria, dove esistono i cablaggi delle line di potenza per

l'illuminazione, può essere usata una LAN powerline. Soluzioni ibride sono possibili.

## A.5 User interface

E' stata sviluppata un'unità di interfaccia utente grafica per l'unità di monitoraggio, che funziona sia con il sistema operativo WINDOWS che con LINUX. L'operatore può effettuare le seguenti operazioni:

- Monitorare in tempo reale la telecamera selezionata;
- Monitorare in tempo reale dello stato del traffico;
- Esaminare i filmati relativi agli eventi;
- Controllare le statistiche del traffico per minuto, ora o giorno.

Alcune schermate dell'interfaccia utente:

## A.6 Sviluppi

Utilizzando telecamere con maggior potenza di elaborazione, quale quelle della nuova serie Smart Cam Sony, si possono effettuare ulteriori elaborazioni a bordo della telecamera, quali:

- effettuare la codifica MPEG-4 direttamente a bordo della telecamera, eliminando la necessità del video server ed ottimizzando l'utilizzo della rete;
- Rilevare nuovi eventi non necessariamente legati al traffico quali "Allarme fumo" o "Allarme incendio" basandosi ancora sulla analisi delle immagini, particolarmente molto importanti nelle installazioni in galleria. Gli algoritmi di rilevazioni di tali allarmi, già testati su PC, non sono ancora

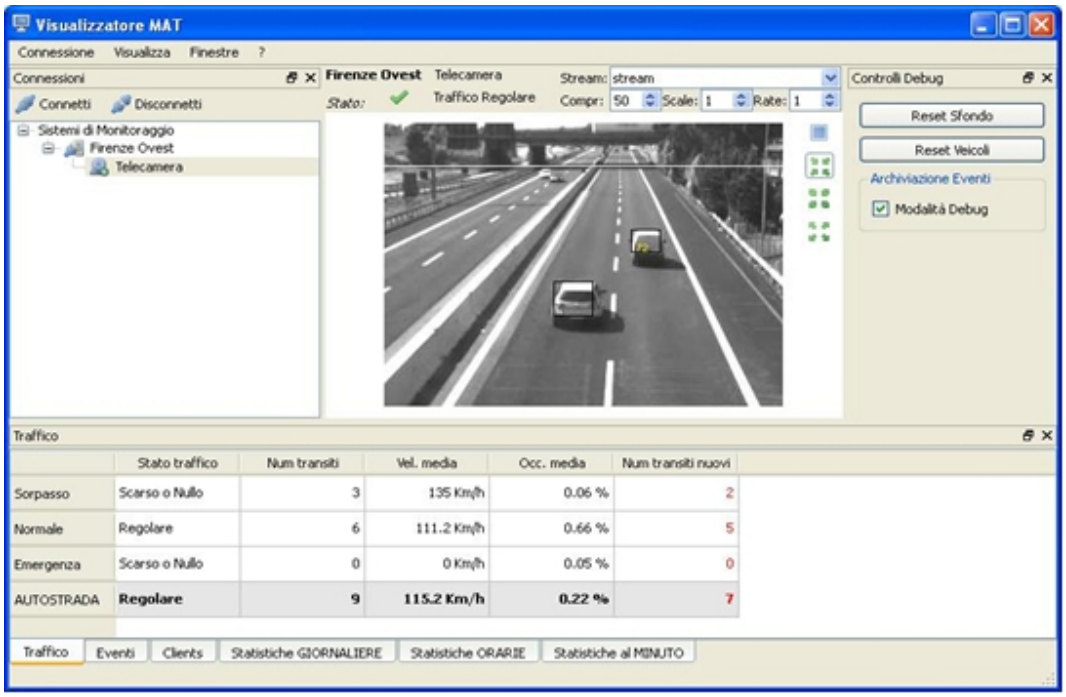


Figura A.3: Monitoraggio dello stato del traffico

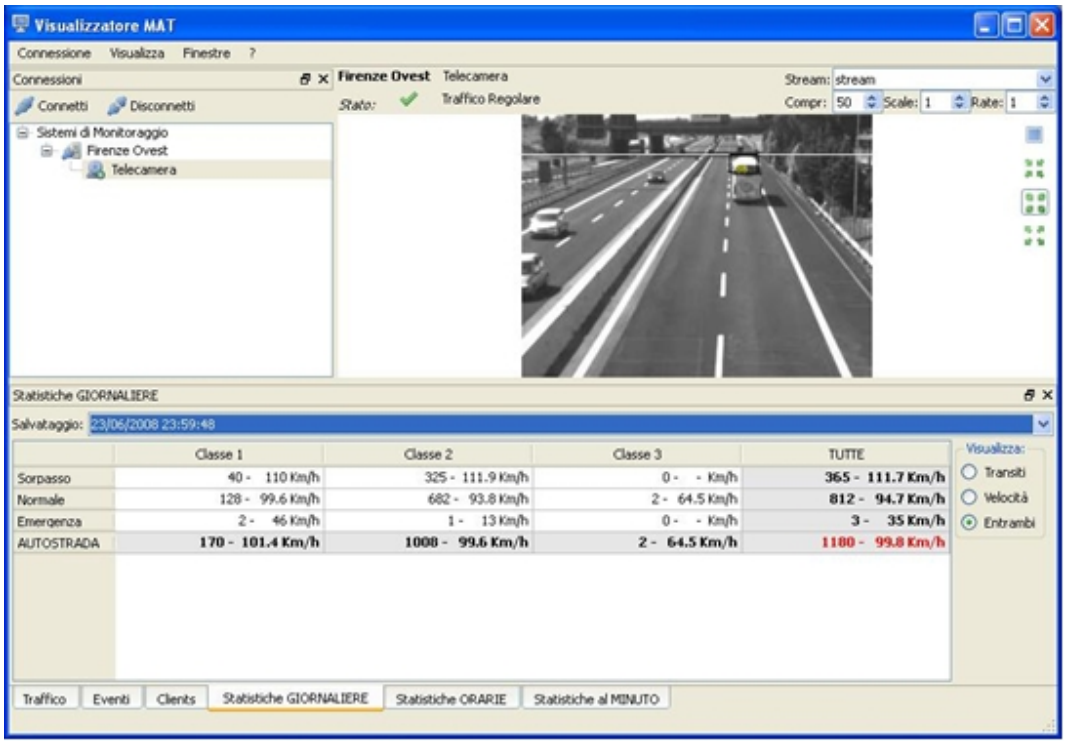


Figura A.4: Monitoraggio delle statistiche

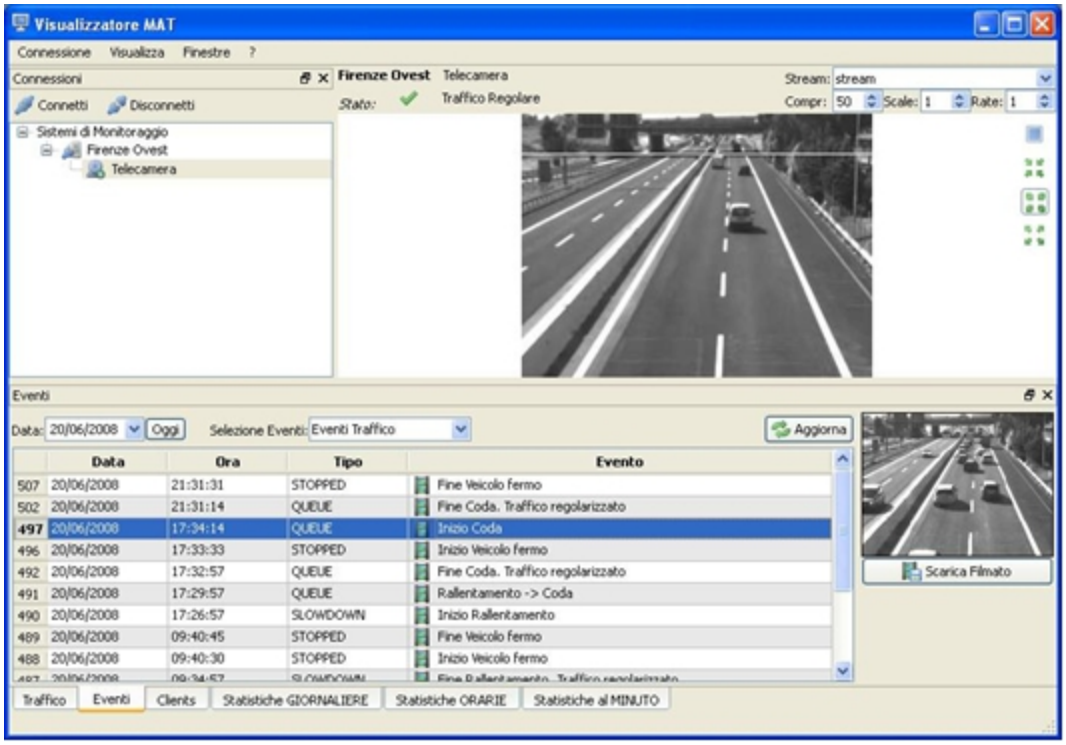


Figura A.5: Monitoraggio degli eventi

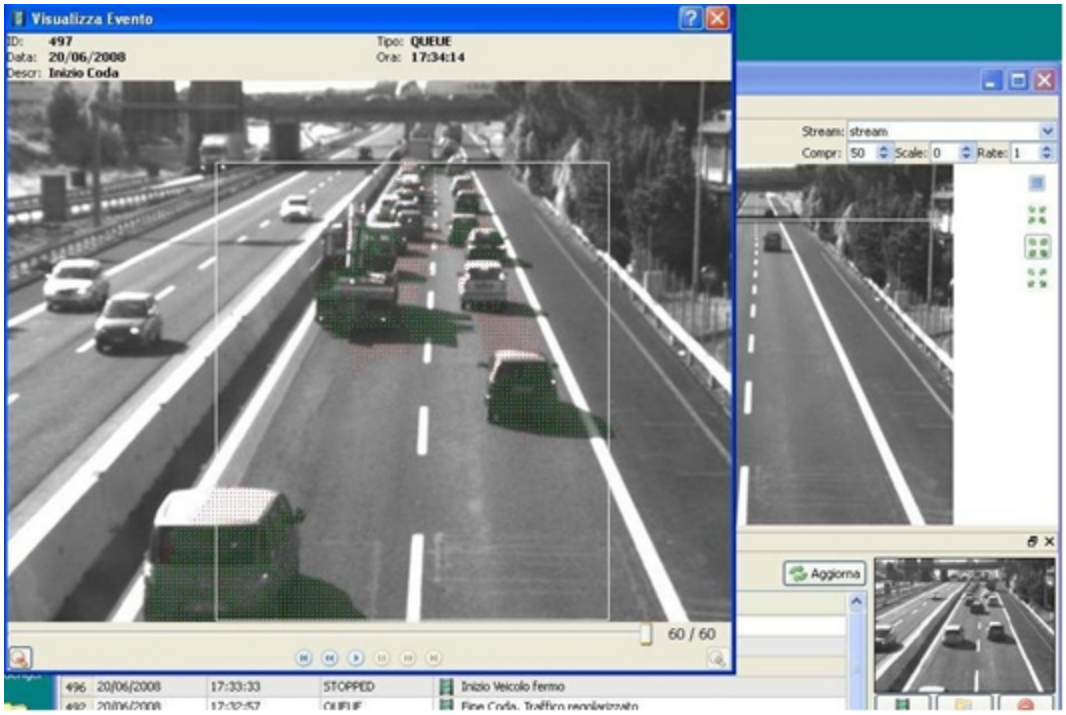


Figura A.6: Monitoraggio dei filmati video relativi agli eventi

stati portati a bordo della telecamera proprio per la limitazione della potenza di elaborazione delle telecamere attualmente in uso.

# Bibliografia


[1] Stuart Russell, Peter Norvig, *Artificial Intelligence, A Modern approach*, Prentice Hall, 2005, vol. 1.

[2] Tom M. Mitchell, *Machine Learning*, Mc Graw-Hill, 1997.

[3] Quinlan, J. R, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.

[4] Hinton, G. E. and Salakhutdinov, R. R, *Reducing the dimensionality of data with neural networks*, Science, Vol. 313. no. 5786, pp. 504 - 507, 28 July 2006.

[5] Hinton, G. E, *Learning Multiple Layers of Representation*, Trends in Cognitive Sciences, Vol. 11, pp 428-434.

[6] IsTech, *Manuale descrittivo del software IsTraffic*.

# Ringraziamenti

*Per prima voglio ringraziare la mia famiglia per avermi dato l'opportunità di intraprendere questo percorso di studi, sostenendomi sempre in ogni mia scelta, universitaria o personale. Un ringraziamento del tutto speciale va alla mia ragazza che ha corretto le innumerovoli bozze di questa tesi e che mi ha sopportato ogni singolo giorno di questi 3 stupendi anni. Voglio ringraziare: Sanjeya,Filippo,Simone,Luca,Alessandro,Giacomo,Lorenzo e Marco, i miei compagni di corso con i quali ho fatto sfacchinate interminabili, annotato quaderni di appunti e sostenuto esami al limite dell'impossibile. Grazie ad Angelo professore di scuola, datore di lavoro e mentore, che mi ha fornito tutto il materiale e l'aiuto necessario per la realizzazione di questa tesi. Grazie infine agli amici d'infanzia, ai nuovi amici del Volley Viaccia, a tutti i membri dell'equipe ed a tutti coloro che mi hanno dato anche solo "una pacca sulla spalla". Tutto questo è anche un po' merito vostro.*