Università della Calabria

Facoltà di Lettere e Filosofia
Dipartimento di Linguistica

Dottorato di Ricerca in
Psicologia della Programmazione e Intelligenza Artificiale
XXIV Ciclo

Settore scientifico-disciplinare MAT/07

Tesi di Dottorato

DINAMICHE NON LINEARI IN RETI NEURALI CELLULARI

Candidato
Dott. Stefano Vena

Tutor
Prof. Pietro Pantano

Coordinatrice
Prof.ssa Eleonora Bilotta

Anno Accademico 2010/2011

Indice

Capitolo 1

Introduzione

Da von Neumann [19] in poi, molti scienziati si sono posti l'interrogativo sulla questione della modellazione di eventi fisici attraverso un insieme discreto di automi cellulari sia che essi siano rappresentati mediante valori discreti sia che essi siano descritti da valori continui. Molte risposte sono state date a questo interrogativo e molte ipotesi sono state portate avanti. Un importante esempio è dato dagli studi di Stephen Wolfram [80] riassunti successivamente nel famoso libro A new kind of science[81]. Questo problema è stato affrontato per lo più considerando, che la modellazione della fisica attraverso i metodi classici, come ad esempio quelli di Newton, è fondamentalmente diversa da una modellizzazione discreta che fa uso di automi. Infatti, nel caso classico, la fisica è considerata continua, questo perché i materiali e i campi sono considerati continui. Al contrario, nella modellazione fisica attraverso automi, la trattazione della realtà avviene in modo discreto. Gli elementi unitari che la compongono sono dette cellule (o celle), ogni cellula interagisce con l'ambiente circostante secondo una data legge fisica definita in base a considerazioni locali. Tali modelli basati su automi discreti sono stati utilizzati con successo per diverse applicazioni che vanno da sistemi di reazione-diffusione [71] a modelli di simulazione per la prevenzione di incendi boschivi [31], simulazioni di colate laviche [67] e altre applicazioni [15, 78, 76]. Dall'analisi attenta di questi lavori si percepisce la natura degli automi cellulari, ovvero l'emergenza di

un comportamento complesso globale partendo da semplici regole locali. Tali caratteristiche negli ultimi decenni hanno suscitato grande interesse nella comunità scientifica[20].

La modellazione basata su automi cellulari ha cercato di incontrare anche la teoria dei circuiti, questo è stato possibile grazie alle caratteristiche dei sistemi VLSI (Very-Large-Scale Integration) [37]. In questi dispositivi sono integrati un gran numero di processori configurabili, organizzati spazialmente come array di unità di processo analogiche o digitali. In questo campo di ricerca nascono le Reti Neurali Cellulari(CNN) proposte da Chua e Yang nel 1988 [27]. Dal punto di vista topologico, gli automi cellulari e le CNN condividono la base locale delle connessioni, con la differenza che una CNN è un sistema dinamico non lineare a tempo continuo, in cui l'elaborazione delle informazioni avviene attraverso una modalità asincrona e parallela. Oltre che per la scelta del nome le CNN hanno molte similitudini con le reti neurali, tuttavia il concetto chiave che distingue le CNN da altre tipologie di modelli neurali artificiali è la presenza esclusiva di interconnessioni locali (fig.1.1).

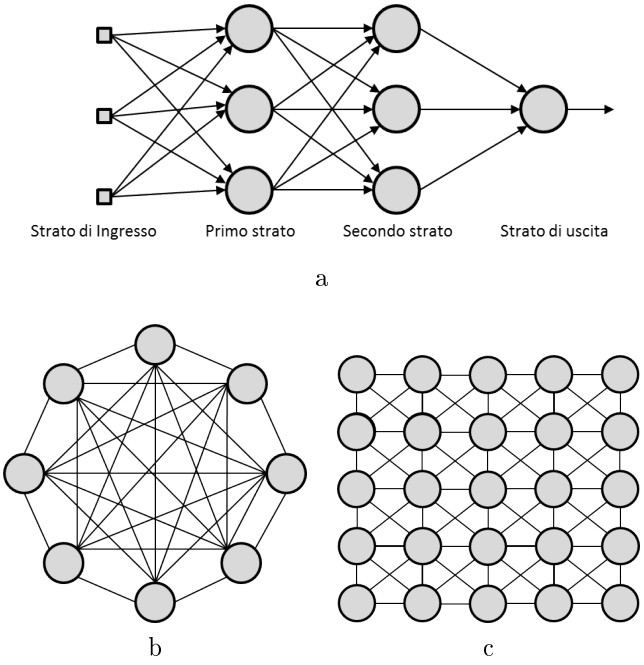


Figura 1.1: Differenti tipologie di reti neurale: Multi livello(a), fortemente connesse(b), cellulari(c)

Una rete neurale cellulare, infatti, è composta da una collezione discreta di processori analogici non lineari, detti celle, disposti a formare una struttura reticolare (griglia), all'interno della quale ciascuna cella è connessa direttamente solo con unità ad essa adiacenti e quindi appartenenti ad una limitata sfera di influenza o ad un vicinato ristretto. Tale strutturazione permette anche interazioni locali non lineari, rendendo, di fatti, le CNN capaci di esibire una dinamica non lineare e di dar luogo alla formazione di dinamiche complesse o fenomeni emergenti. Lo stato di ogni cella, evolve nel tempo in accordo con uno specifico modello dinamico non lineare e, sebbene lo scambio di informazioni avvenga solo a livello locale, l'effetto propagativo proprio della dinamica tempo continua delle celle consente la comunicazione indiretta tra unità non adiacenti. Questa caratteristica conferisce alla rete la capacità di elaborare informazioni a livello globale e di innescare la formazione di pattern nello spazio (spatio-temporal dynamics)[27, 25]. La specifica operazione svolta globalmente da una CNN dipende da un insieme limitato e prestabilito di parametri, che influenzano il reticolo di interconnessione locale, chiamato template. Una volta definito il template, per ciascuna cella è univocamente descritta la legge di evoluzione. Poichè in una CNN, ciascuna cella interagisce fisicamente solo con le unitè circostanti, risulta agevole l'integrazione in circuiti ad alta densità (VLSI) rispetto al modello delle reti neurali tradizionali, in quanto per una CNN è necessario predisporre solo connessioni tra unità di elaborazione vicine. Vista la possibilità di arrangiare nello spazio le celle che compongono una CNN è possibile implementare reti sviluppate lungo una sola dimensione, lungo due dimensioni o lungo tre dimensioni. Poichè le celle, come descritte in [27] sono circuiti analogici, è possibile ottenere, per ciascuna di esse, dimensioni ridotte e bassa dissipazione di potenza, il che ne facilita l'implementazione hardware. Una caratteristica molto importante è legata alla natura analogica delle celle, infatti rende le CNN in grado di processare direttamente i segnali acquisiti da sensori, senza la necessità di effettuare costose conversioni Analogico/Digitale. In particolare, ciascuna sorgente di

dati può essere mappata direttamente su un processore (cella), permettendo che l'elaborazione sia implicitamente legata alla posizione spaziale della sorgente che alimenta la cella stessa. Questa caratteristica trova applicazione nella robotica e in particolare nel controllo di robot con zampe [7, 16, 17]. Dai lavori citati è possibile associare una cella a ciascun sensore che misuri la posizione angolare di un arto robotico, inviando un segnale al relativo attuatore il quale trasmette il comando generato, in uscita dalla cella stessa.

Un campo in cui le CNN trovano largo impiego è il settore dell'image processing e della computer vision [59, 25, 2]. Infatti ciascun pixel che compone un'immagine è instradato e mappato su uno dei processori che costituiscono la CNN, in base alla posizione del corrispondente pixel all'interno dell'immagine 1.2.

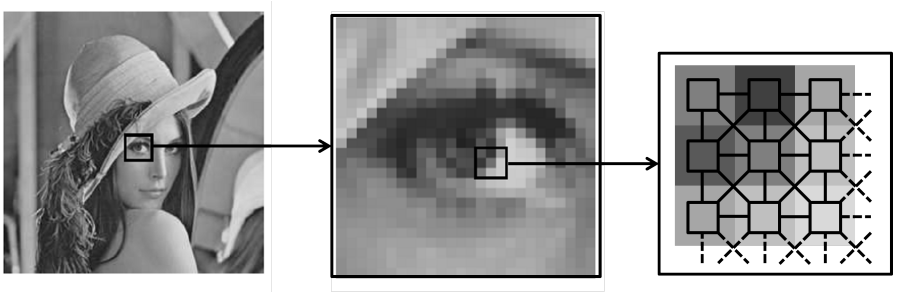


Figura 1.2: Passaggio da immagine a CNN

Nell'image processing basato su CNN ogni pixel viene processato da una cella. Gli array computer basati su CNN realizzano architetture hardware in cui l'elaborazione del flusso di dati in ingresso non avviene attraverso una sequenza di operazioni logiche, ma il risultato emerge dall'evoluzione di dinamiche non lineari analogiche scaturite dall'interazione tra le celle. Le dinamiche spazio-temporali della CNN dipendono dal template assegnato per definire i legami di interconnessione tra gli elementi della rete e rappresentano un'istruzione elementare per un programma analogico [22]. Nel paradigma computazionale fornito dalle reti neurali cellulari, l'esecuzione di un task su un chip CNN consiste nell'applicare, ad un dato flusso di dati in ingresso, una sequenza opportuna di specifici template. I passi intermedi dell' elaborazione vengono memorizzati attraverso gli elementi di logica

digitale di cui è corredato il chip (analog-and-logic computing). Al termine di un periodo di assestamento delle celle, detto *transitorio*, lo stato di ogni elemento di una CNN si assesta su un valore stabile, a questo punto il risultato prodotto dall'elaborazione in cascata dei template è pronto è può essere prelevato ai terminali di uscita della rete [61].

In termini di potenza di calcolo, le architetture CNN sono in grado di offrire, su un singolo chip, performance notevoli e sono capaci di processare flussi continui di dati in real time [61, 11]. L'efficienza dei processori CNN-based dipende dal fatto che l'evoluzione dello stato delle singole cella è definita nel dominio del tempo continuo e, pertanto, a differenza dei processori digitali, non si necessita di un segnale di clock esterno che induca l'evoluzione dello stato del sistema. Grazie a questa caratteristica è possibile ottenere non solo una maggiore efficienza di calcolo, ma anche un minore consumo di potenza, in quanto per un chip digitale l'energia dissipata è proporzionale al quadrato della frequenza di clock. Sebbene le soluzioni basate su architetture digitali offrano maggiore robustezza ed accuratezza rispetto alle CNN, in molte applicazioni pratiche le variazioni dovute al rumore non sono percettibili all'occhio umano, e pertanto la risoluzione offerta dalle reti neurali cellulari risulta sufficientemente elevata in numerosi contesti applicativi [59, 61, 25]. Si parla in questo contesto di CNN-Universal Machine (CNN-UM) ovvero un'architettura di calcolo basata sulle reti neurali cellulari che può essere programmata per eseguire algoritmi memorizzati come sequenze di templates [59]. La caratteristica chiave della CNN-UM è la programmabilita ovvero è possibile scrivere e memorizzare, utilizzando un linguaggio ad alto livello veri e propri programmi che consistono di una o più istruzioni analogiche, le quali possono essere eseguite su un chip CNN. I principi di funzionamento dell'architettura proposta da Chua e Roska, pertanto, sono completamente differenti rispetto a quelli su cui si basano gli elaboratori digitali sequenziali; l'istruzione di calcolo di base nella CNN-UM è implementata dalle dinamiche di una CNN programmabile che risolve un'equazione che descrive

un'onda non lineare, operando su flussi di dati continui [56, 57, 58]. Un'operazione analogica elementare, dunque, corrisponde alla soluzione di un sistema di equazioni differenziali non lineari e determinata dall'evoluzione parallela dello stato di tutte le celle che compongono la rete neurale cellulare, in dipendenza del template assegnato. La CNN-UM combina operazioni logiche, ovvero operazioni tra variabili che possono assumere solo valori binari, con elaborazioni analogiche svolte in parallelo (analog array computing), per le quali il tempo evolve in maniera continua e gli operandi assumono valori reali. Punto di forza di tale paradigma di calcolo duale è la peculiare assenza di convertitori Analogico/Digitale e Digitale/Analogico. I segnali logici sono trattati da dispositivi digitali, mentre le grandezze continue vengono processate da circuiti analogici. Gli elementi che compongono la CNN, nel contesto della CNN Universal Machine, sono identificati come celle estese data la presenza di componenti sia analogici che digitali, non presenti nel modello di rete neurale cellulare originariamente proposto. Come mostrato nello schema in figura 1.3, una cella estesa è composta da un nucleo di calcolo e da un insieme di interruttori (switch), configurabili per selezionare quale sia la memoria a cui la cella necessita di accedere in un dato step dell'elaborazione. Ogni cella estesa è equipaggiata con memorie locali che rendono più efficiente l'esecuzione degli algoritmi. Infatti, lo svolgimento di un task richiede una sequenza di sotto-task, per i quali è necessario memorizzare i risultati intermedi. Quindi la presenza di elementi di local storage in ogni cella consente di limitare il numero di accessi a memorie esterne al chip CNN e, conseguentemente, di migliorare le prestazioni del sistema in termini di velocità di elaborazione. Le celle estese dispongono localmente di una memoria analogica (LAM, Local Analog Memory) per registrare valori reali e di una memoria logica (LLM, Local Logic Memory) per salvare quantità binarie. Oltre a dispositivi di memorizzazione, sono presenti circuiti locali per eseguire operazioni cell-wise logiche (LLU, Local Logic Unit) e analogiche (LAOU, Local Analog Output Unit) sui valori memorizzati. Attraverso una griglia di sensori lo-

cali, generalmente di tipo ottico (OPT, OPTical sensor), è possibile catturare flussi di dati analogici facendo in modo che ogni cella sia alimentata da un rispettivo sensore che trasmette ad essa le informazioni visuali acquisite dall'ambiente esterno. Attraverso la comunicazione tra l'unità di programmazione analogica globale della macchina (GAPU, Global Analogic Programming Unit) e l'unità di controllo locale (LCCU, Local Communication and Control Unit), ciascuna cella riceve informazioni su quale sia l'istruzione che l'intero array CNN deve eseguire. Il paradigma delle CNN-UM è molto importante e largamente usato nel settore della computer vision e dell'image processing [38, 84, 16, 62, 25, 83] .

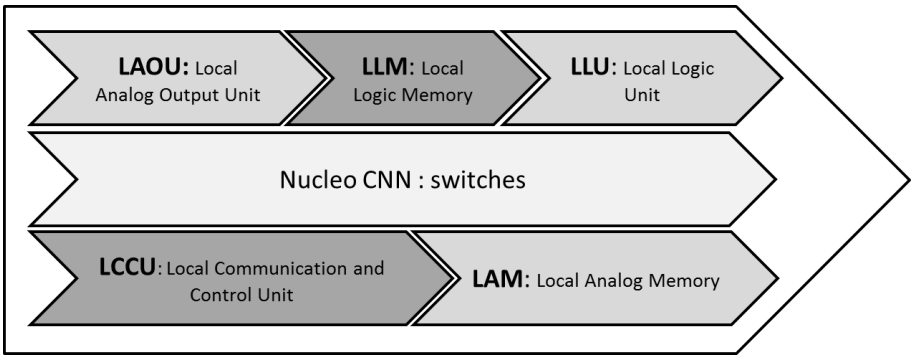


Figura 1.3: Componenti principali di una cella estesa

Le CNN possono essere opportunamente adattate e impiegate come risolutori numerici di equazioni differenziali parziali (PDE), ed in questo senso è stato proposto un approccio alle metodologie per la progettazione e la realizzazione di CNN per una data PDE in un articolo del 1995 dal titolo Simulating nonlinear waves and partial differential equations via CNN. I. Basic techniques [60]. Tale approccio consiste nell'effettuare una discretizzazione spaziale della PDE attraverso uno schema alle differenza finite ottenendo un equazione differenziale ordinaria (ODE) nel tempo che può essere risolta numericamente attraverso una metodologia standard come il Runge-Kutta. Tale metodologia è molto impiegata in questo campo applicativo ed esistono simulator i[47], e anche progetti di sistemi VLSI [63]. Il sistema composto da equazioni differenziali parziali viene implementato usando componenti analogici VLSI, e l'evoluzione temporale del circuito diventa l'evolu-

zione temporale del sistema PDE iniziale. Tuttavia emergono alcune difficoltà in tale approccio. La prima riguarda la stabilità del sistema dinamico CNN. Alcuni studi sulla stabilità delle CNN nella risoluzione di PDE classiche si ritrovano in [64] ma la stabilità deve essere verificata quando si ha a che fare con nuovi problemi specifici come nel caso dello studio delle dinamiche in reattore nucleare [40]. La seconda tipologia di difficoltà che emerge nella trasformazione di una PDE in CNN è strettamente correlata alla corretta espressione della PDE di partenza attraverso una CNN, poichè i metodi a disposizione sono più di tipo euristico che non una formale derivazione della CNN a partire da una generica PDE [12]. Va aggiunto che le caratteristiche della CNN non possono essere facilmente associate ai parametri fisici coinvolti nella PDE originale. Attualmente non esistono ancora metodi formali generali che forniscano gli strumenti adatti alla modellazione di CNN partendo da una PDE generica. La modellazione avviene in due fasi propedeutici, viene progettata la CNN che descrive la PDE e successivamente vengono tarati i parametri. La modulazione successiva dei valori dei parametri può essere svolta attraverso un processo di addestramento supervisionato [47, 12, 82]. Quando si è a conoscenza di una soluzione analitica della PDE è possibile effettuare verifiche a-posteriori come in [64], o come nel caso delle onde viaggianti o dei solitoni [60, 44]. In tutti gli altri casi è possibile effettuare solo un validazione con criteri qualitativi.

Uno dei metodi più interessanti impiegati per la progettazione di CNN e per la loro taratura è rappresentato dall'addestramento mediante Algoritimi Genetici. Tale metodologia è stata applicata con successo nella realizzazione di robot [16, 17] oppure nella definizione di template per l'analisi di immagini mediche [13, 14].

Le CNN possono essere applicate con successo anche per lo sviluppo di controllori di basso livelli per sistemi robotici [16]. Nel contesto del controllo di robot mediante reti neurali cellulari trova ampia diffusione una classe di CNN a due livelli, che modella dinamiche non lineari di reazione-diffusione [22]. Le Reaction-Diffusion CNN (RD-CNN) rappresentano modelli matematici in grado di descrivere fenome-

ni complessi riscontrati in biologia, chimica, neuroscienze, ed in altri contesti[22]. Numerosi pattern di locomozione bioinspirati possono essere generati ricavando soluzioni oscillanti di RD-CNN [6]. In un articolo in particolare Evolving Robot's Behavior by Using CNNs [16] sono stati adottati metodi evolutivi per progettare un sistema di controllo basato su CNN per un robot esapode. Il template che governa la CNN è stato determinato impiegando algoritmi genetici. Numerose CNN vengono fatte evolvere fino ad individuare il controllore con parametri che ricadono all'interno di una specifica regione dello spazio dei parametri e che consente di ottenere prestazioni soddisfacenti per il movimento del robot. Sempre nel campo della robotica le CNN sono state impiegate per pianificare in tempo reale la traiettoria di riferimento per robot su ruote che navigano in ambienti non strutturati in presenza di ostacoli [3]. Tale approccio si pone come valida alternativa al metodo dei campi potenziali, il quale comporta un onere computazionale poco adatto al caso in cui la presenza di ostacoli mobili richieda il calcolo in tempo reale della traiettoria da seguire per raggiungere l'obiettivo. La strategia proposta in [3] consiste nel considerare l'ambiente in cui il robot si muove come un mezzo attivo, sede di fenomeni non-lineari che permettono la propagazione di autoonde, ossia onde autonome che si propagano senza forzante nel mezzo attivo, mantenendo costanti l'ampiezza e la forma. Scegliendo i parametri della rete in modo che le celle siano localmente attive [22], le RD-CNN sono in grado di comportarsi come un mezzo attivo in cui, in base alle condizioni iniziali, emergono e si propagano onde non lineari. In particolare, gli ostacoli e gli obiettivi rappresentano le sorgenti per generare, rispettivamente, autoonde repulsive ed attrattive; i fronti delle onde che si propagano nella CNN forniscono al robot le informazioni per giungere a destinazione evitando gli ostacoli. L'immagine dell'area in cui si muove il robot è assegnata come stato iniziale della RD-CNN, la quale è in grado di generare auto-onde a partire dai pixel rappresentanti gli ostacoli. Il robot è rappresentato da un'immagine in cui quattro pixel ne descrivono posizione ed orientamento.

Nel campo della computer vision l'impiego delle CNN si impone in un progetto denominato Bio-inspired (Bi-i) Cellular Vision System [84] Il sistema Bio-inspired (Bi-i) Cellular Vision System è una piattaforma di computazione composta da sensoristica, array sensing processing, e processamento di segnali digitali ovviamente basata sul paradgima delle CNN in grado di catturare immagini dall'ambiente e processarle in tempo reale.

In questo lavoro di tesi, verranno formulati dei modelli di simulazione di CNN che verranno implementati e sperimentati. Questo percorso avviene partendo da modelli già esistenti in letteratura e formulandone dei nuovi su simulatori software sviluppati contestualmente al mio percorso dottorale.

Capitolo 2

Le Reti Neurali Cellulari

2.1 Introduzione

Le Reti Neurali Cellulari (CNN - dall'inglese *Cellular Neural Networks*) rappresentano un paradigma di calcolo parallelo simile alle reti neurali e agli automi cellulari, con la differenza che la comunicazione e l'interazione tra celle è consentita solo tra le unità più prossime. Il concetto di CNN è stato introdotto da Leon Chua e Lin Yang nel lavoro del 1988 suddiviso in due parti : Cellular Neural Networks: Theory e Cellular Neural Networks: Applications [27, 26]. Il modello matematico descritto dagli scienziati dimostra, per una specifica implementazione della CNN, che se gli ingressi sono statici e limitati, le unità di elaborazione convergeranno, e tali dispositivi possono essere, quindi, utilizzati per eseguire calcoli o svolgere altre attività. I due scienziati hanno inoltre suggerito alcune delle prime applicazioni dei processori CNN: l'elaborazione delle immagini e il pattern recognition, che sono tuttora le applicazioni più diffuse. Un altro articolo chiave è l'articolo di Tamas Roska e Leon Chua del 1993 [59], The CNN universal machine: an analogic array computer, dove è stato introdotto il primo processore CNN analogico algoritmicamente programmabile. Le ricerche presentate nell'articolo citato sono state finanziate da Office of Naval Research, da National Science Foundation, e dall'Accademia Ungherese delle Scienze,e condotte presso l'Accademia Ungherese

delle Scienze e l'Università della California. Questo articolo ha dimostrato che i processori CNN sono realizzabili, e ha fornito ai ricercatori una piattaforma fisica per testare le loro teorie sulle CNN.

2.2 Definizione

Una CNN è definita da due assunti matematici :

- Un insieme spazialmente discreto di sistemi dinamici non lineari chiamati celle, dove le informazioni che descrivono le celle possono essere espresse mediante tre variabili indipendenti chiamate *input*, *threshold*, e *initial state*
- Una legge di accopiamento che correla una o piú variabili rilevanti di ciascuna cella C_{ij} a tutte le celle vicine C_{kl} localizzate dentro una prestabilita sfera di influenza $S_{ij}(r)$ di raggio r, centrata in Cij.

Ogni cella contiene elementi circuitali di tipo lineare e non lineare. Tipicamente le tipologie di elementi impiegate sono : capacitori lineari, resistori lineari, genetori controllati lineari e non lineari e sorgenti indipendenti. La struttura di una CNN è simile a quella di un automa cellulare, infatti ogni cella in una rete nonlineare cellulare è connessa esclusivamente con le celle ad essa vicina. Le celle adiacenti interagiscono tra loro direttamente. Le celle non connesse direttamente possono influenzarsi in modo indiretto grazie alla propagazione degli effetti della dinamica nel tempo. [27] L'organizzazione spaziale delle CNN può essere in una, due o tre dimensioni, parleremo quindi di CNN monodimensionali, bidimensionali e tridimensionali. Quando si parla di CNN Standard si fa riferimento ad una CNN bi-dimensionale come mostrata in figura 2.1.

Consideriamo una CNN MxN, composta da MxN celle disposte in M righe e N colonne. La cella identificata dalla i-esima riga e j-esima colonna è chiamata $cella(i, j)$ ed è identificata con $C(i, j)$ come in figura 2.1. Ora andiamo a definire cosa vuol dire sfera di influenza di una cella $C(i, j)$. La sfera di influenza di $S_r(i, j)$

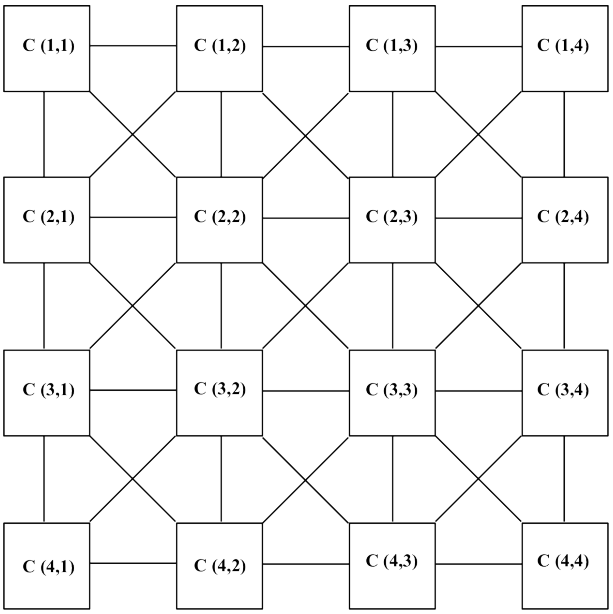


Figura 2.1: L'architettura di una CNN Standard consiste in una matrice MxN di celle C_{ij} con coordinate cartesiane (i, j) con $i \in [1, M]$ e $j \in [1, N]$

di raggio r (N_r) di una cella $C(i, j)$, in una CNN è definita come l'insieme di celle che soddisfano la seguente proprietà

$$S_r(i, j) = C(k, l)|max|k - i|, |1 - j| \leq r, 1 \leq k \leq M; 1 \leq l \leq N \qquad (2.1)$$

dove r è un numero positivo. Rispetto alla sfera di incidenza $S_r(i, j)$ una cella $C(i, j)$ è detta *regolare* se e solo se tutte le celle del vicinato $C(k, l) \in S_r(i, j)$ esistono, altrimenti tale cella è detta di *confine*[27, 26].

In figura 2.2 sono rappresentati tre vicinati della stessa cella(situata al centro e ombreggiata) con raggi rispettivamente $r = 1, 2$ e 3. Il vicinato con raggio $r = 1$ è detto "vicinato $3x3$" con raggio $r = 2$ "vicinato $5x5$" e con raggio $r = 3$ "vicinato $7x7$"

2.3 Modello matematico

Come proposto da Chua e Yang nel lavoro del 1988 : *"Cellular Neural Networks: Theory"* [27] un possibile circuito impiegabile per l'implementazione di una cella

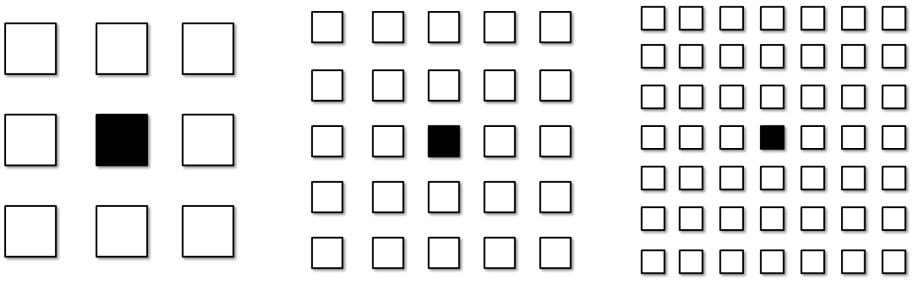


Figura 2.2: Il vicinato della cella $C(i,j)$ definito da (2.1) rispettivamente per $r = 1$, $r = 2$ e $r = 3$.

$C(i,j)$ è mostrato in figura 2.3. I suffissi u, x e y denotano rispettivamente *input*, *stato* e *output* della cella. La tensione nodale v_{xij} di $C(i,j)$ è detta "stato" della cella e la sua *condizione iniziale* deve assumere ampiezza minore uguale a 1. La tensione nodale v_{uij} è detta *input* di $C(i,j)$ e viene assunta come valore costante di ampiezza minore uguale a 1. La tensione nodale v_{yij} è detta output.

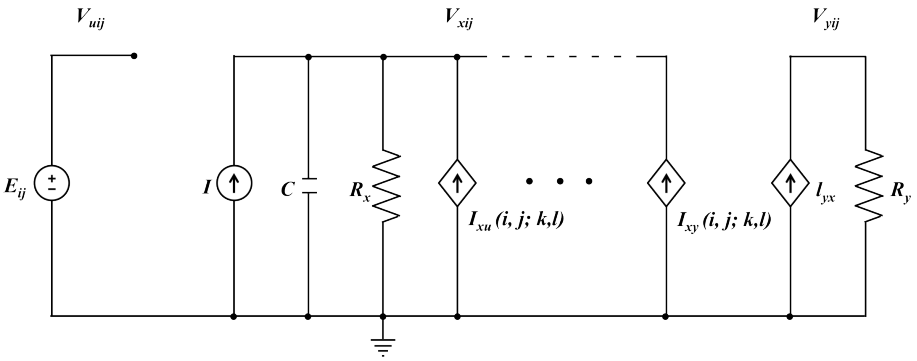


Figura 2.3: Tipico circuito che implementa una cella della CNN

Come mostrato in figura 2.3, ogni cella $C(i,j)$ contiene un generatore di tensione indipendente E_{ij}, un generatore indipendente di corrente I, un condensatore lineare C, due resistenze lineari R_x e R_y e al più $2m$ generatori di corrente pilotati in tensione lineari accoppiati con le celle vicine attraverso il generatore di tensione *vukl* e il *feedback* proveniente dalla tensione di uscita v_{ykl} di ogni vicino $C(k,l)$, dove m è uguale al numero delle celle che compongono il vicinato.

In particolare, $I_{xy}(i,j;k,l)$ e $I_{xu}(i,j;k,l)$ sono generatori di corrente pilotati in

tensione con caratteristica come descritta in (2.2)

$$\begin{aligned} I_{xy}(i,j;k,l) &= A(i,j;k,l)v_{ykl} \\ I_{xu}(i,j;k,l) &= B(i,j;k,l)v_{ukl} \\ \forall C(k,l) &\in S_r(i,j) \end{aligned} \tag{2.2}$$

dove i parametri di accoppiamento $A(i;j;k;l)$ (operatore di feedback) e $B(i;j;k;l)$ (operatore di feedforward o di controllo) possono essere utilizzati per controllare l'intensitá delle interazioni tra le celle, amplificando o attenuando i segnali scambiati lungo le connessioni. La dinamica di una CNN quindi, presenta meccanismi sia di feedback che di controllo; in particolare il feedback sull'uscita é influenzato dall'operatore $A(i;j;k;l)$ mentre il controllo sull'ingresso puó essere gestito intervenendo opportunamente mediante l'operatore di feedforward $B(i;j;k;l)$. L'ingresso della cella corrisponde alla tensione erogata dal generatore di tensione continua E_{ij}. L'unico elemento non lineare presente in ogni cella è il generatore di corrente pilotato in tensione I_{xy} la cui caratteristica è definita dalla relazione:

$$I_{xy} = (\frac{l}{Ry})f(v_{xij}) \tag{2.3}$$

Nelle CNN standard la $f(\cdot)$ assume generalmente un andamento lineare a tratti come mostrato in figura 2.4. Assumendo che la funzione f in (2.3) corrisponda alla nonlinearità standard mostrata in figura 2.4, noto il valore della tensione v_{xij} al generico istante di tempo t, il valore $v_{yij}(t)$, uscita di $C(i,j)$ è regolato dalla seguente espressione:

$$\begin{aligned} v_{yij} = f(v_{xij}(t)) = \frac{1}{2}[|v_{xij}(t)+1| - |v_{xij}(t)-1|] \\ 1 \leq i \leq M, \quad 1 \leq j \leq N \end{aligned} \tag{2.4}$$

In generale la funzione caratteristica $f(\cdot)$ puó essere una qualsiasi funzione monotona non decrescente nell'intervallo $[-1;1]$, quale ad esempio una sigmoide,

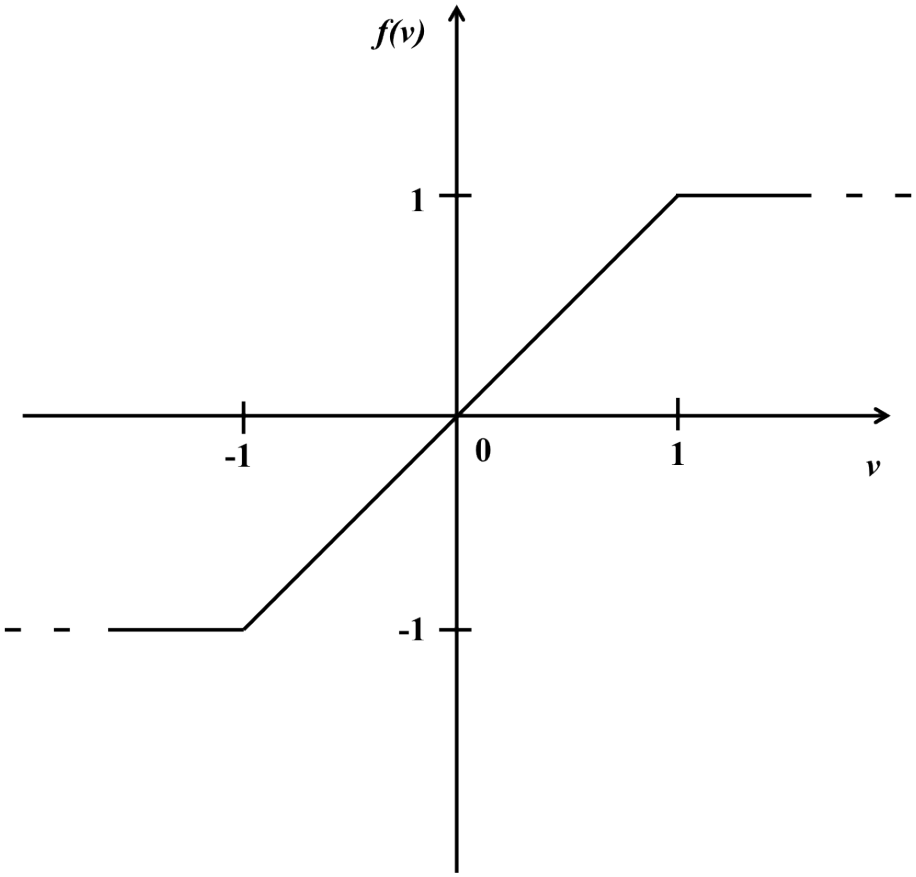


Figura 2.4: Caratteristica del generatore non lineare

tangente iperbolica, etc. Applicando la legge di Kirchhoff delle correnti e la legge di Kirchhoff delle tensioni possiamo derivare l'equazione che descrive il circuito come segue [23] :

- Equazione di Stato

$$\begin{aligned} C\frac{dv_{xij}(t)}{dt} &= -\frac{1}{R_x}v_{xij}(t) \\ &+ \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{ykl}(t) \\ &+ \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{ukl}(t) + I, \\ 1 \le i \le M; 1 \le j \le N. & \end{aligned} \tag{2.5}$$

- Equazione di Uscita

$$v_{yij} = \frac{1}{2}(|v_{xij}(t)+1| - |v_{xij}(t)-1|), \quad 1 \leq i \leq M; 1 \leq j \leq N. \tag{2.6}$$

- Equazione di Ingresso

$$v_{uij} = E_{ij}, \quad 1 \leq i \leq M; 1 \leq j \leq N. \tag{2.7}$$

- Vincoli

$$|v_{xij}(0)| \leq 1, \quad 1 \leq i \leq M; 1 \leq j \leq N. \tag{2.8}$$

$$|v_{uij}| \leq 1, \quad 1 \leq i \leq M; 1 \leq j \leq N. \tag{2.9}$$

(2.9)

Si può dimostrare [26] che il valore dello stato u_{xij} di ogni cella in una CNN è limitato per tutto il tempo $t < 0$ al valore massimo

$$v_{max} = 1 + R_x|I| + R_x \max_{1 \leq i \leq M; 1 \leq j \leq N} \left[\sum_{C(k,l) \in N_r(i,j)} (|A(i,j;k,l)| + |B(i,j;k,l)|) \right] \tag{2.10}$$

Inoltre dopo un periodo si assestamento la CNN si avvicina sempre verso ad uno dei suoi stati di equilibrio [26]

$$\lim_{t \to \infty} v_{xij}(t) = costante, \quad 1 \leq i \leq M; 1 \leq j \leq N \tag{2.11}$$

da questo deriva

$$\lim_{t \to \infty} \frac{dv_{xij}(t)}{dt} = 0, \quad 1 \leq i \leq M; 1 \leq j \leq N \tag{2.12}$$

Un'altra importante proprietà dimostrabile [26] è la seguente: Se i parametri del circuito soddisfano :

$$A(i,j;i,j) > \frac{1}{R_x} \tag{2.13}$$

allora

$$\lim_{t\to\infty} |v_{xij}(t)| \geq 1, \quad 1 \leq i \leq M; 1 \leq j \leq N \tag{2.14}$$

ed in modo equivalente

$$\lim_{t\to\infty} v_{yij}(t) = \pm 1, \quad 1 \leq i \leq M; 1 \leq j \leq N \tag{2.15}$$

I teoremi appena citati sono importanti [26] poichè garantiscono che il comportamento della CNN non diventa ne oscillatorio ne caotico [24], inoltre (2.15) garantisce che la rete ha valori di uscita binari.

In accordo con [22] possiamo andare a definire una formulazione generica di CNN

$$\dot{x}_{ij} = -x_{ij} + \sum_{kl\in S_{ij}(r)} a_{kl}\, y_{kl} + \sum_{kl\in S_{ij}(r)} b_{kl}\, u_{kl} + z_{ij} \tag{2.16}$$

$$y_{ij} = f(x_{ij} = \frac{1}{2}(|x_{ij}+1| - |x_{ij}-1|) \tag{2.17}$$

$$1 \leq i \leq M; 1 \leq j \leq N$$

dove con l'operatore punto $\dot{}$ indichiamo la derivata nel tempo, con x_{ij}, y_{ij}, u_{ij} e z_{ij} sono scalari e rappresentano rispettivamente stato, uscita, ingresso e soglia di una generica cella C_{ij}; a_{ij} e b_{ij} sono anch'essi scalari detti pesi sinaptici e vanno a formare gli elementi delle matrici A e B dette template della CNN. Le $S_{ij}(r)$ sono le regioni di interesse della cella C_ij in funzione del raggio r

2.4 Condizioni al contorno

Come già precedentemente affermato, le celle che hanno una regione di interesse completa, ovvero composta da $(2r+1)^d$ celle dove r è il raggio e d il numero di dimensioni della CNN (1,2 o 3) vengono dette celle regolari le celle che non godono di un vicinato completo sono dette celle di confine.Le celle che appartengono al vicinato di una cella ma non esistono vengono dette *virtuali*. Mentre ciò che succede per le celle interne è chiaro, è necessario approfondire il comportamento delle CNN di frontiera. In particolare possono verificarsi tre tipi di condizioni al contorno [75, 27, 26, 25] :

- Fissa (o di Dirichlet),(fig. 2.5) in questo caso allo stato x_{kl} di ogni cella C_{kl} nell'equazione (2.16) che giace fuori dai confini della CNN è impostato ad un valore fisso; Dal punto di vista circuitale si aggiunge una riga o una colonna lungo il confine della CNN e si forza ogni cella virtuale ad assumere un ingresso fisso e un uscita imposta da una batteria.

- Zero-flusso (o di Neumann),(fig. 2.6) in questo caso gli stati x_{kl} corrispondenti alle celle del vicinato perpendicolari ai bordi sono forzate ad essere uguali.

- Periodica (o toroidale),(fig. 2.7) in questo caso la prima riga e l'ultima (e così anche le colonne) sono identiche, così da formare un toro.

Nel caso di condizione al contorno fissa i valori degli stati delle celle sono regolati dalle seguenti relazioni

$$\textbf{Celle virtuali a sinistra:} \quad y_{i,0} = \alpha 1, \quad u_{i,0} = \beta 1, \quad i = 1, 2, \ldots, M \tag{2.18}$$

$$\textbf{Celle virtuali a destra:} \quad y_{i,N+1} = \alpha 2, \quad u_{i,N+1} = \beta 2, \quad i = 1, 2, \ldots, M \tag{2.19}$$

$$\textbf{Celle virtuali in alto:} \quad y_{0,j} = \alpha 3, \quad u_{0,j} = \beta 3, \quad i = 1, 2, \ldots, M \tag{2.20}$$

$$\textbf{Celle virtuali in basso:} \quad y_{M+1,j} = \alpha 4, \quad u_{M+1,j} = \beta 4, \quad i = 1, 2, \ldots, M \tag{2.21}$$

Dove $\alpha 1$, $\alpha 2$,$\alpha 3$,$\alpha 4$, $\beta 1$,$\beta 2$,$\beta 3$,$\beta 4$ sono costanti.

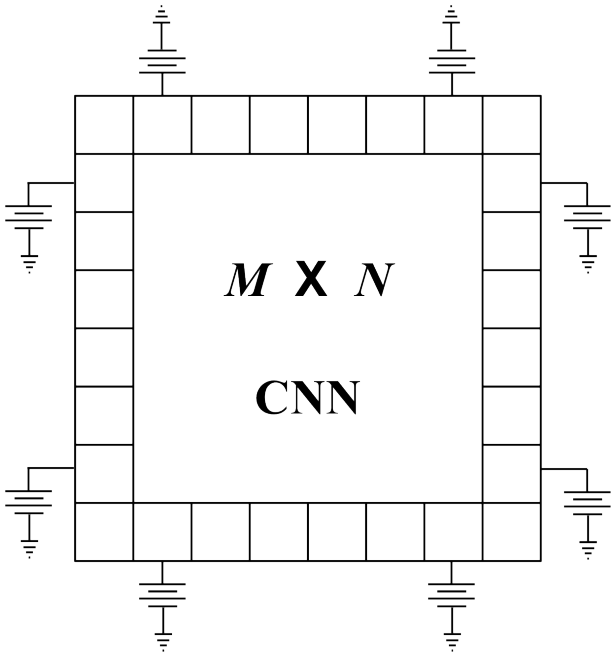


Figura 2.5: Condizioni al contorno Fisse

Nel caso di condizione al contorno Zero-Flusso i valori degli stati delle celle sono regolati dalle seguenti relazioni

$$\textbf{Celle virtuali a sinistra:} \quad y_{i,0} = y_{i,1}, \quad u_{i,0} = u_{i,1}, \quad i = 1, 2, \ldots \tag{2.22}$$

$$\textbf{Celle virtuali a destra:} \quad y_{i,N+1} = y_{i,N}, \quad u_{i,N+1} = u_{i,N}, \quad i = 1, 2, \ldots \tag{2.23}$$

$$\textbf{Celle virtuali in alto:} \quad y_{0,j} = y_{1,j}, \quad u_{0,j} = u_{1,j}, \quad i = 1, 2, \ldots \tag{2.24}$$

$$\textbf{Celle virtuali in basso:} \quad y_{M+1,j} = y_{M,j}, \quad u_{M+1,j} = u_{M,j}, \quad i = 1, 2, \ldots \tag{2.25}$$

Nel caso di condizione al contorno Toroidale i valori degli stati delle celle sono regolati dalle seguenti relazioni

$$\textbf{Celle virtuali a sinistra:} \quad y_{i,0} = y_{i,N}, \quad u_{i,0} = u_{i,N}, \quad i = 1, 2, \ldots, \tag{2.26}$$

$$\textbf{Celle virtuali a destra:} \quad y_{i,N+1} = y_{i,1}, \quad u_{i,N+1} = u_{i,1}, \quad i = 1, 2, \ldots, \tag{2.27}$$

$$\textbf{Celle virtuali in alto:} \quad y_{0,j} = y_{M,j}, \quad u_{0,j} = u_{M,j}, \quad i = 1, 2, \ldots, \tag{2.28}$$

$$\textbf{Celle virtuali in basso:} \quad y_{M+1,j} = y_{1,j}, \quad u_{M+1,j} = u_{1,j}, \quad i = 1, 2, \ldots, \tag{2.29}$$

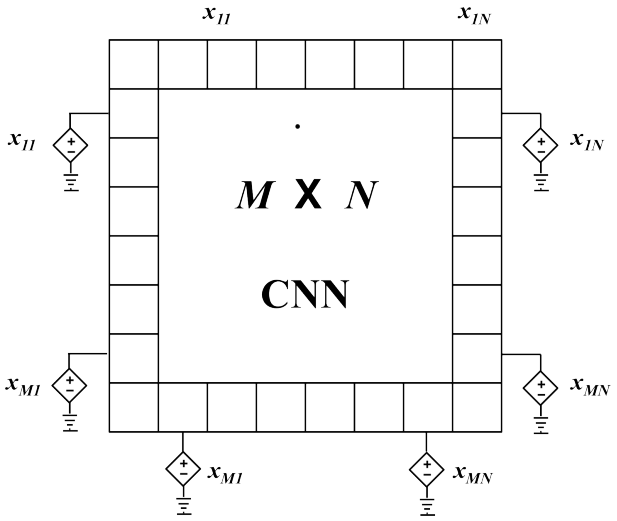


Figura 2.6: Condizioni al contorno di Zero Flusso

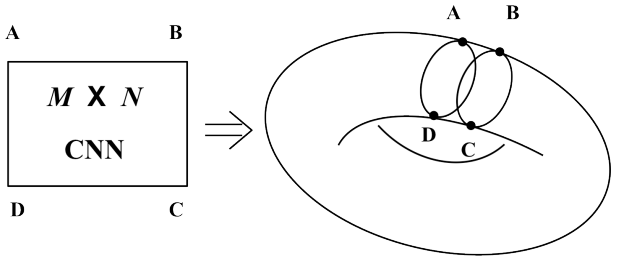


Figura 2.7: Condizioni al contorno Toroidali

2.5 Generalizzazione

La prima formulazione di CNN [27, 26] ha, così come descritto nei paragrafi precedenti, struttura bidimensionale di carattere matriciale. La generalizzazione è ben espressa dall'astrazione di una cella come espresso in figura 2.8 e dalle seguenti espressioni [22]:

- Stato della singola cella :

$$\dot{x_{ij}} = f(x_{ij}, z_{ij}, \mathbf{u}_{kl}, \mathbf{y}_{kl}), \quad kl \in S_{ij}(r). \tag{2.30}$$

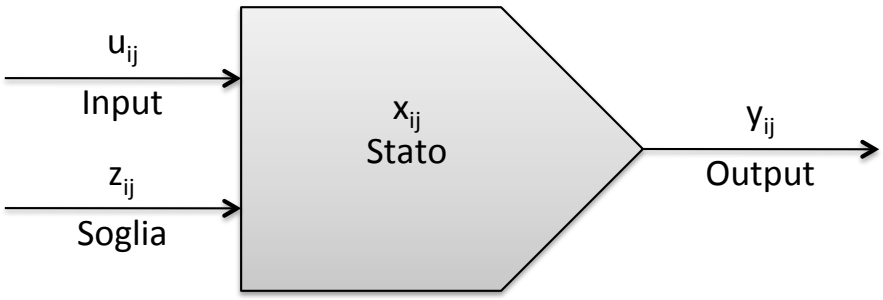


Figura 2.8: Generica cella CNN

dove $\mathbf{u}_{kl}$ e $\mathbf{y}_{kl}$ indicano i vettori le cui componenti u_{kl} e y_{kl} includono tutte le celle vicine $C_{kl} \in S_{ij}(r)$

- Output della singola cella

$$y_{ij} = g(x_{ij}). \tag{2.31}$$

$$1 \leq i \leq M; 1 \leq j \leq N$$

Dove $f(.)$ è un operatore matematico che agisce su $\mathbf{x}_{ij}$, $\mathbf{z}_{ij}$ e $\mathbf{u}_{ij}$ mentre $g(.)$ è una funzione.

Capitolo 3

Strumenti software

3.1 Introduzione

Tra gli obiettivi primari di questo lavoro c'è la volontà di realizzare un nuovo simulatore di CNN in grado di soddisfare sia le esigenze tipiche della ricerca in questo settore, sia fornire le basi per lo sviluppo di soluzioni applicative. Il simulatore è composto da una libreria monolitica denominata libCNN che fornisce tutti gli strumenti necessari a simulare CNN monodimensionali, bidimensionale, e tridimensionali anche a più livelli. Tale libreria inoltre è in grado di essere estesa a più livelli. Infatti essa espone interfaccie di programmazione (API) che permettono all'utente di inserire nuovi modelli di CNN, nuovi algoritmi di integrazione numerica. Impiegando la libreria appena descritta sono stati realizzati tre ambienti applicativi ognuno dedicato ad una tipologia differente di CNN:

- wireExplorer, per l'analisi di CNN che modellano linee di trasmissione;
- cnnExplorer, per l'impiego e l'addestramento di CNN bidimensionali standard;
- cnn3DExplorer, per la simulazione di CNN3D multi livello e la loro visualizzazione.

La realizzazione dei tre ambienti rispetta il pattern architetturale Model-View-Controller(MVC) descritto in figura 3.1.

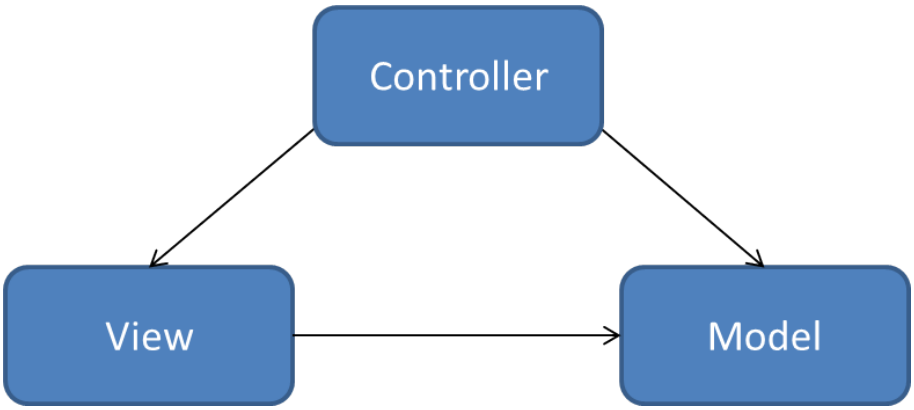


Figura 3.1: Pattern Model-View-Controller, il Controller è associato direttamente alla View e al Model; La View è associata al Modello; Mentre il Modello non ha associazioni dirette con gli altri elementi.

Nel pattern MVC l'utente interagisce con l'interfaccia grafica (GUI) in qualche modo (per esempio, premendo un pulsante del mouse). La GUI implementa la parte di View del pattern. Gli eventi generati dall'interfaccia grafica vengono inviati ad un Controller che li gestisce e li trasforma in azioni appropriate per i Modelli. Il controller interagisce con il modello modificandone lo stato. Una Vista può a sua volta interrogare il modello al fine di generare una interfaccia utente appropriata e ricevere i dati necessari. L'obiettivo principale del pattern MVC è ridurre la complessità della progettazione e aumentare la flessibilità e la gestibilità del codice. [29, 53, 54] Nella figura 3.2 è riportato lo schema che descrive l'organizzazione di massima di ogni applicativo suddiviso nelle tre componenti del pattern MVC. Nel primo livello (Insieme View) si inseriscono le interfacce implementate. Il secondo livello contiene un insieme di moduli, esterni al programma e implementati come plug-in, che caratterizzano il concetto di *Modello* espresso nel paradigma architetturale considerato. Alcuni moduli sono generici e forniti dalla libreria *libCNN* altri, invece, sono legati a particolari problemi e verranno descritti in dettaglio successivamente. Nel terzo livello rientra la libreria libCNN. Essa gestisce la logica generale delle applicazioni, si occupa della simulazione, fornisce gli

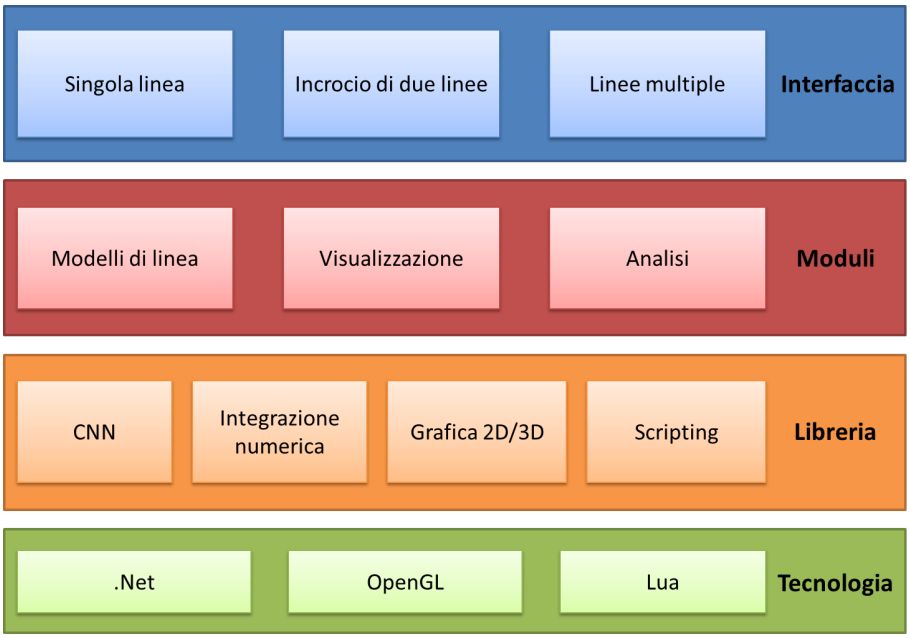


Figura 3.2: Schema della struttura dell'applicazione

strumenti per la realizzazzione di interfacce grafiche per l'interazione con CNN e fornisce l'implementazioni di alcuni *Modelli* di base che saranno trattati in seguito. La libreria libCNN va quindi a inquadrarsi nel paradigma MVC ricoprendo il ruolo di *Controller*. Per completare il quadro descritto nella figura 3.2 si riportano le tecnologie impiegate ovvero *Framework Microsoft .Net* [50] con il linguaggio *C#*, *OpenGL* [74] per la grafica 3D e *Lua* [72] come linguaggio di scripting.

3.2 libCNN

La libreria libCNN rappresenta il cuore pulsante di tutte le applicazioni sviluppate. Tale libreria è stata scritta durante il corso del dottorato e si occupa di descrivere e simulare le CNN secondo il modello introdotto da Chua e Yang, offrendo anche sistemi di supporto alla logica delle applicazioni, il caricamento di moduli esterni e funzioni di ausilio nella rappresentazione grafica di matrici.

Le funzionalità che mette a disposizione la libreria libCNN sono descritte il dettaglio di seguito :

- **CORE**

  - **uniDimensional**
    - CNN1D
    - Template
    - BorderedArray
  - **biDimensional**
    - CNN2D
    - Template
    - BorderedMatrix
  - **triDimensional**
    - CNN3D
    - Template
    - BorderedSpace

- **GUI**
  - TemplateEditor
  - BorderedMatrixViewer
  - BorderedArrayViewer
  - BorderedSpaceIsosurface
  - BorderedSpaceProjection

- **MATH**
  - IIntegration
  - Euler
  - RK4th
  - RK4thCS

- **MVC**

- IViews
- Controller

L'elenco appena proposto presenta in grassetto i namespace contenuti nella libreria e in testo normale i nomi delle classi pubbliche messe a disposizione attraverso le api. *BorderedArray, BorderedMatrix* e *BorderedSpace* sono delle astrazioni del concetto di array, matrice e matrice tridimensionale, forniscono metodi per l'accesso ai dati che contengono e (da qui il suffisso Bordered) permettono di gestire le celle virtuali implementando dei metodi in grado di applicare le condizioni al contorno viste nel paragrafo 2.4. Di seguito è riportata lo schema della classe *BorderedMatrix* con i metodi pubblici esposti, per gli altri due tipi di oggetti omologhi la struttura è analoga, varia esclusimente il numero di dimensioni.

Codice 3.1: BorderedMatrix

```
public class BorderedMatrix
{

        /** Metodi accessori per la lettura di una cella
          x: numero di riga
          y: numero di colonna
        */
        public double get(int x, int y);

        /** Metodi accessori per la scrittura di una cella
          x: numero di riga
          y: numero di colonna
          v: valore da assegnare
        */
        public void set(int x, int y, double v);

        /*
           Metodo per il ridimensionamento della matrice
           c: numero di colonne
           r: numero di righe
        */
        public void resize(int r, int c)
```

```
        public int Rows { get; }
        public int Columns { get;}

        /*Proprieta' che restituisce la tipologia di condizione al contorno
          I valori disponibili sono
          Boundaries.Fixed ( Dirichlet )
          Boundaries.ZeroFlux: ( VonNeumann )
          Boundaries.Periodic: ( Toroidal ) */
        public Boundaries Boundaries { get; set; }

        /* Proprieta' che restituisce il valore che assumono le celle
           virtuali se e' stata scelta  Boundaries.Fixed  come condizione al ↩
               contorno
        */
        public double FixedBoundariesValue { get; set; }

        /* Forza l'aggiornamento dei valori delle celle virtuali in base alla ↩
            tipologia attiva*/
        public void RefreshBoundaries();

}
```

Le varie classi *Template* presenti nei diversi namespace sono dei contenitori per gli array,matrici o matrici 3D che descrivono appunto i Template di una CNN. Sicuramente più interessanti sono le classi *CNN1D*,*CNN2D*,*CNN3D*. Esse forniscono le unità di base per la simulazione di una CNN. Nel listato seguente è mostrato lo schema della classe CNN2D

Codice 3.2: CNN2D

```
public class CNN2D
{
   /*
     Delegato che descrive la funzione che viene invocata alla fine
     di un passo di evoluzione della CNN.
    */
   public delegate void OnCnnEvolutionDelegate( CNN cnn );
```

```
/* Evento generato dopo ogni passo di simulazione della CNN */
public event OnCnnEvolutionDelegate OnCnnEvolution = null;


//Proprieta' che contiene l'oggetto che descrive il template
public Template Template { get; set;};

//Proprieta' che rappresenta il passo di integrazione impiegato nella ↩
    simulazione.
public double Timestep;


/*T0 e T1 sono gli indici dello stato attuale T0, e dello stato successivo T1
  Gli indici sono due 0 e 1 e vengono scambiati ad ogni iterazione.
  questa soluzione garantisce efficenzia computazionale e spaziale.
 */
protected int T0 = 0, T1 = 1;
/* Matrici che descrivono Ingresso (mU), Stato (mX[0] e mX[1]) e Uscita (mY↩
    [0] e mY[1]).
   la classe BorderedMatrix e' un particolare oggetto in grado di gestire le ↩
       celle virtuali
   e le condizioni al contorno in modo trasparente */
protected svutils.types.BorderedMatrix mU;
protected svutils.types.BorderedMatrix[] mX = new svutils.types.↩
    BorderedMatrix[2];
protected svutils.types.BorderedMatrix[] mY = new svutils.types.↩
    BorderedMatrix[2];

/*  Costruttore
    Template t: Classe che descrive il template
    int width : Numero di colonne della CNN
    int height : Numero di righe della CNN
 */
public CNN(Template t, int width, int height)
{
     Timestep = 0.01F;
     this.Template = t;
     Resize(Math.Max(2, width), Math.Max(2, height));
     Converge = false;
 }

/* Funzione che valuta la matrice di Output. */
```

```
public void EvaluateOutput() {

    for (int i = 0; i < mU.Width; i++)
        for (int j = 0; j < mU.Height; j++)
            mY[T0][i, j] = NonLinearity(mX[T0][i, j]);
}

/* Funzione invocata per modificare le dimensioni delle matrici della CNN
    int width : Numero di colonne della CNN
    int height : Numero di righe della CNN
*/
public virtual void Resize(int width, int height)
{
    if (mU == null) mU = new BorderedMatrix(width, height, Template.Radius);
    if( mX[T0] == null) mX[T0] = new BorderedMatrix(width, height, Template.↩
        Radius);
    if (mX[T1] == null) mX[T1] = new BorderedMatrix(width, height, Template.↩
        Radius);

    if (mY[T0] == null) mY[T0] = new BorderedMatrix(width, height, Template.↩
        Radius);
    if (mY[T1] == null) mY[T1] = new BorderedMatrix(width, height, Template.↩
        Radius);

    mU.SetSize(width, height, Template.Radius);
    mX[T0].SetSize(width, height, Template.Radius);
    mX[T1].SetSize(width, height, Template.Radius);
    mY[T0].SetSize(width, height, Template.Radius);
    mY[T1].SetSize(width, height, Template.Radius);
 }

/*Proprieta' che restituisce lo stato di convergenza della CNN*/
public bool Converge { get; set; }

/* Funzione che processa la cnn per la durata di un passo di simulazione. */
public virtual void  Process()
{
     mU.RefreshBoundaries();
 mY[T0].RefreshBoundaries();
 Converge = true;
 for (int x = 0; x < Input.Width; x++)
   for (int y = 0; y < Input.Height; y++)
   {
```

```
            mX[T1][x, y] = Integrate(x,y);
            mY[T1][x, y] =NonLinearity(mX[T1][x, y]);
            if (Converge)
            {
              if (mY[T1][x, y] != mY[T0][x, y]) Converge = false;
            }
          }
    });
    T0 = 1-T0;
    T1 = 1-T1;
  }
   /* Funzione che processa la CNN per un numero di passi pari a time.
      time: Numero di passi di simulazione da iterare.*/
   public void Process(int time) {
       while (time > 0) {
           Process();
           time--;
       }
       if (OnCnnEvolution != null) OnCnnEvolution(this);
   }

   /* Proprieta' che permettono l'accesso alle matrici di stato corrente */
   public svutils.types.BorderedMatrix Input { get { return mU; } }
   public svutils.types.BorderedMatrix State { get { return mX[T0]; } }
   public svutils.types.BorderedMatrix Output { get { return mY[T0]; } }


   /* Proprieta' che impostano la tipologia di condizione al contorno.*/
   public Boundaries InputBoundaries
   {
      get { return mU.Boundaries; }
      set { mU.Boundaries = value; }
   }
   public Boundaries StateBoundaries
   {
       get { return mX[T0].Boundaries; }
       set {
           mX[T0].Boundaries = value;
           mX[T1].Boundaries = value;
       }
   }
   public Boundaries OutputBoundaries
   {
```

```
        get { return mY[T0].Boundaries; }
        set
        {
            mY[T0].Boundaries = value;
            mY[T1].Boundaries = value;
        }
    }


    /* Proprieta' che impostano il valore delle celle virtuali
       Tale proprieta' e' impiegata solo nel caso di condizioni al contorno fisse↩
           .
    */
    public double InputVirtualCellsValue
    {
        get { return mU.FixedBoundariesValue; }
        set { mU.FixedBoundariesValue = value; }
    }
    public double StateVirtualCellsValue
    {
        get { return mX[T0].FixedBoundariesValue; }
        set
        {
            mX[T0].FixedBoundariesValue = value;
            mX[T1].FixedBoundariesValue = value;
        }
    }
    public double OutputVirtualCellsValue
    {
        get { return mY[T0].FixedBoundariesValue; }
        set
        {
            mY[T0].FixedBoundariesValue = value;
            mY[T1].FixedBoundariesValue = value;
        }
    }

    /*
       Equazione che descrive lo stato delle celle della CNN.
       double x : Stato attuale della CNN.
       double [] parameters: Parametri della CNN, l'impiego di un vettore ↩
           garantisce
                           flessibilita' nell'estensione della cnn.
```

```
                        Nel caso della CNN standard e' un vettore con due ↩
                            elementi la sommatoria sulla Matrice A, e la ↩
                            sommatoria sulla Matrice B;
*/
protected  virtual double CNNEquation(double x, double [] parameters) {
      return -x + Template.Z + parameters[0] + parameters[1];
}


 /*Proprieta' che imposta/restituisce un istanza di un oggetto in grado di ↩
     integrare una funzione differenziale */
 public Integrator integrator { get;set;}

 /*
    Funzione che integra le equazioni della CNN.
    int i : riga della Cella da simulare.
    int j : colonna della Cella da simulare.
 */
 protected virtual double Integrate(int i, int j) {
    double[] parameters = { SumA(i, j), SumB(i,j) };
    double x = mX[T0][i, j];
    return integrator.run( Timestep, x, CNNEquation, parameters;
}


/* Funzione che lega lo stato all'uscita.
   Implementa una funzione lineare a tratti.
   double d: Il valore dello stato
 */
protected virtual double NonLinearity(double x)
{
    if (x >= 1) return 1;
    else if (x <= -1) return -1;
    else return x;
}

/*Funzione che valuta la sommatoria dei parametri del template A e dell'↩
    uscita
  int x : riga della Cella da simulare.
  int y : colonna della Cella da simulare.
*/
protected virtual double SumA(int x, int y)
{
```

```
        double s = 0.0F;
        int d = -Template.Radius;
        for (int r = 0; r < Template.W; r++)
        {
            for (int c = 0; c < Template.H; c++)
            {
                s += Template.A[c, r] * mY[T0][x + r + d, y + c + d];
            }
        }

        return s;
    }

    /*Funzione che valuta la sommatoria dei parametri del template B e dell'↩
        ingresso.
      int x : riga della Cella da simulare.
      int y : colonna della Cella da simulare.
    */
    protected virtual double SumB(int x, int y)
    {
        double s = 0.0F;
        int d = -Template.Radius;
        for (int r = 0; r < Template.W; r++)
        {
            for (int c = 0; c < Template.H; c++)
            {
                s += Template.B[c, r] * mU[x + r + d, y + c + d];
            }
        }

        return s;
    }
}; // CNN2D

}
```

Come affermato in precedenza l'implementazione delle altre tipologie di CNN differisce di poco. L'interfaccia *IIntegrator*, mostrata di seguito, fornisce le linee guida allo sviluppo di moduli di integrazione numerica aggiuntivi. Il simulatore di CNN è modellato secondo il modello generale descritto da a (2.30) e (2.31) ed è in grado di simulare CNN monodimensionali, bidimensionali e tridimensionali. Inoltre suppor-

ta le tre principali condizioni al contorno: Fissa, Zero Flusso e Toroidale. Il motore è in grado di processare in modo parallelo grazie all'impiego delle caratteristiche del framework .net [73].

Codice 3.3: Interfaccia IIntegrator

```
public interface IIntegration {


public delegate double Equation(double x, double [] parameters);
/* Metodo lanciato per eseguire l'algoritimo di integrazione numerica ↩
    implementato
     x: valore attuale della variabile di stato
     step: passo di integrazione
     equation: equazione differenziale del primo ordine da risolvere
     parameters: parametri supplementari da trasmettere all'equazione.
*/
double run( double x, double step, Equation equation, double[] parameters);

}
```

I moduli di integrazione realizzati e impiegati nelle simulazioni sono tre: Metodo di Eulero Implicito, Runge-Kutta del IV ordine classico e il metodo Runge-Kutta (IV ordine) con step adattivo di Cash-Karp[77]. Il namespace **MVC** contiene la classe *Controller* che permette di implementare l'omonimo pattern, il controller verifica se in una specifica directory ci sono assembly (ovvero moduli esterni) [46] che contengono classi derivate da *CNN1D*, *CNN2D*, *CNN3D*, *Integrator* e *IView* e le organizza in modo tale da implementare il workflow descritto in figura 3.1. La ricerca viene effettuata sfruttando la reflection [66] fornita dal linguaggio C# e gli elementi sono caricati dinamicamente dal simulatore come plug-in a runtime.

L'interfaccia per l'implementazioni di visualizzatori (IView) è riportata di seguito:

Codice 3.4: Interfaccia IView

```
public interface IView
{
    /*** Metodo chiamato appena la classe viene istanziata
     *  cnn: Oggetto CNN corrente.
     ***/
    void Init(BorderedMatrix data);
    /*** Metodo chiamato ogni qual volta lo stato della cnn viene modificato
     *  cnn: Oggetto CNN corrente.
     ***/
    void StepDone(BorderedMatrix data);

    /** Metodo chiamato quando e' necessario reimpostare  lo stato del modulo
     *  cnn: Oggetto CNN corrente.
     **/
    void Reset(BorderedMatrix cnn);

    /** Ogni modello potrebbe definire alcuni parametri, quando il valore di un ↩
        parametro cambia allora viene richiamato tale metodo.
     *  cnn: La cnn corrente
     *  param: Il nome del parametro
     *  value: il valore del parametro
     **/
    void ParameterChanged(BorderedMatrix cnn, string param, object value);

}
```

Il namespace **GUI** contiene le classi che implementano i visualizzatori di base delle CNN per le applicazioni;

Il *TemplateEditor* mostra una semplice interfaccia per modificare i valori delle matrici che descrivono i template come mostrato in figura 3.3

Il *BorderedArrayViewer* (figura 3.5) mostra lo stato delle celle di una CNN1D con una curva interpolante. Il *BorderedMatrixViewer* (figura 3.4) visualizza una CNN bidimensionale come una bitmap in scala di grigio. *BorderedSpaceIsosurface* e *BorderedSpaceProjection* visualizzano lo stato delle celle di una CNN3D rispettivamente con una isosurface e con delle proiezioni del volume sui tre piani xy, yz e zx. Le isosurface vengono realizzato mediante l'algoritmo noto con il nome di

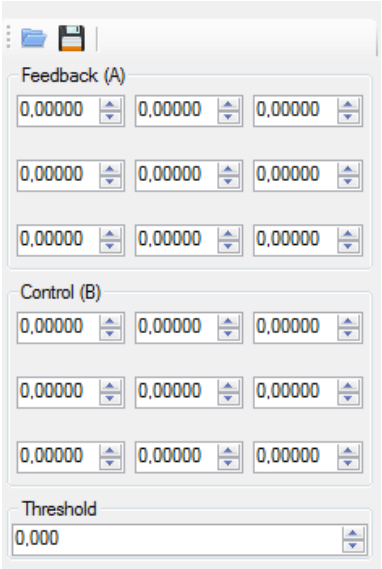


Figura 3.3: Template editor

Figura 3.4: Visualizzatore CNN 2D

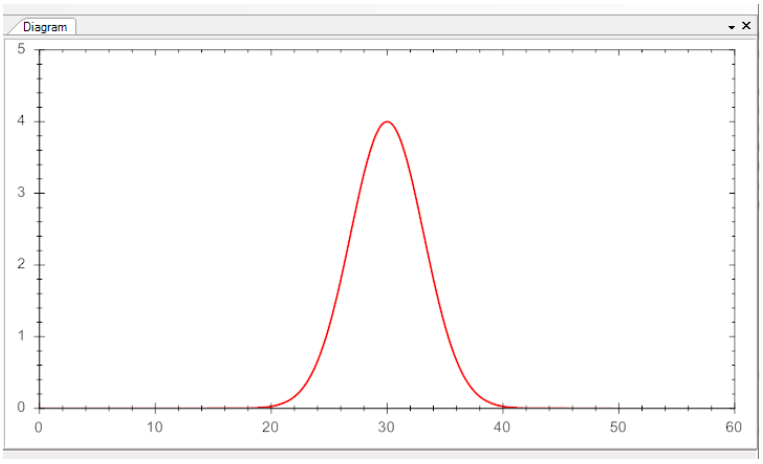


Figura 3.5: Visualizzatore CNN1D

Marching Cubes [48].

3.3 wiresExplorer

Per la realizzazione delle simulazioni riguardanti le CNN1D è stato realizzato un ambiente di simulazione chiamato "wireExplorer" in grado di simulare linee di trasmissione modellate con le CNN di tre differenti tipologie di configurazione : Singola Linea di trasmissione (figura 3.6), incrocio tra due linee (figura 3.7), e

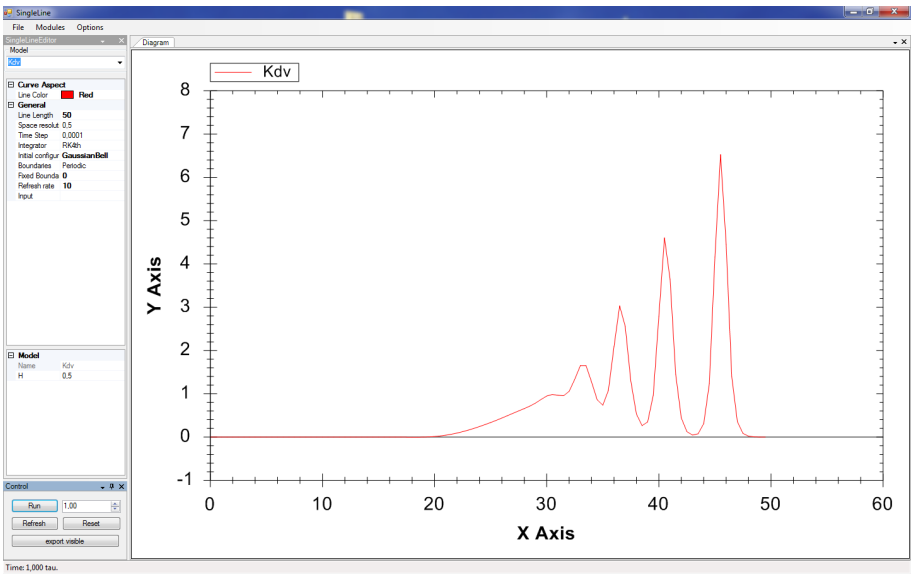


Figura 3.6: Interfaccia analisi singola linea di trasmissione

incrocio tra numerose linee (figura 3.8).

3.3.1 Moduli implementati

Per questo simulatore sono stati implementati alcuni modelli e alcune view raggruppati in due categorie: Modelli Matematici e Analizzatori. Per semplificare il lavoro di sviluppo è stata introdotta una classe astratta denominata *LineModel* che estende la classe *CNN1D* fornendo un modello semplificato per la

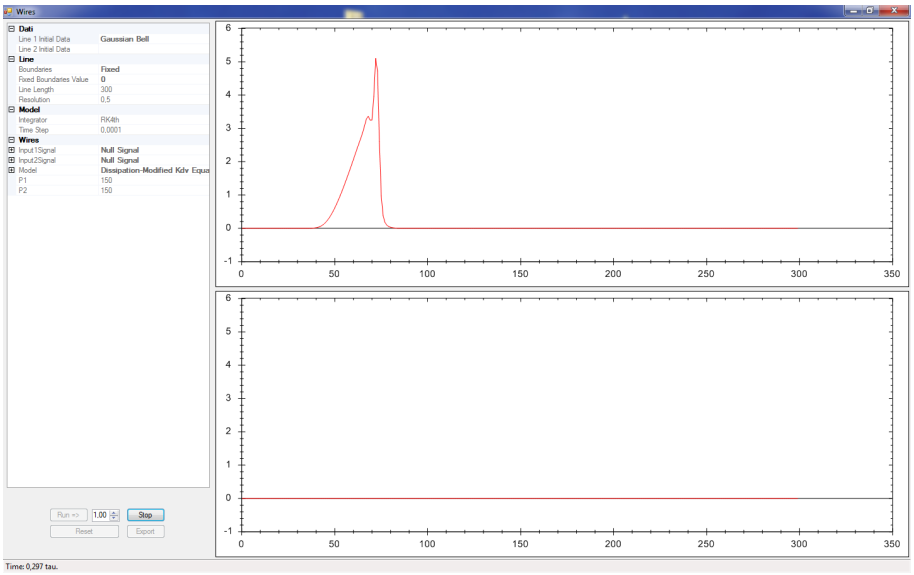


Figura 3.7: Interfaccia analisi doppia linea di trasmissione

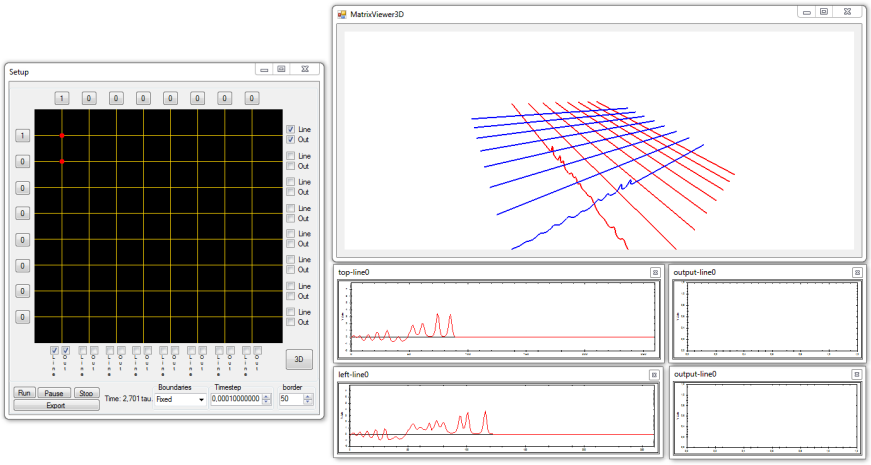


Figura 3.8: Interfaccia analisi incroci tra numerose linee di trasmissione

trattazione di Linee di Trasmissione. Tale classe è mostrata nel listato seguente

Codice 3.5: Interfaccia Modello matematico

```
public abstract class LineModel : CNN1D{
        //Ritorna il nome del modello
        public abstract string Name { get; }

        //Permette di accedere alle varie celle della linea
        public abstract double this[int index ]{ get; set;}

        /*** Inizializza la linea.
             data: La funzione da impiegare per l'inizializzazione della linea
             res: Risoluzione spaziale della rete
        ***/
        public abstract void Init(FunctionProvider data, double res);

        /*** Metodo chiamato per simulare la linea
              i: Integratore numerico
              t: tempo attuale
              dt: delta t di integrazione
        **/
        public abstract void Evaluate(Integrator i, double t, double dt);

        //Lunghezza della linea
        public abstract int Length { get; set; }

        //Ordine matematico del modello impiegato
        public abstract int Order { get; }
    }
```

Per le simulazioni in questo lavoro è stato implementato un solo modello basato sulla KdV. Gli analizzatori implementano l'interfaccia *IView* vista in precedenza. Gli analizzatori realizzati sono quattro: Plotter, CNN Inspector, FFTWindow, Oscilloscope. Il plotter permette di visualizzare lo stato della linea instantaneamente mediante una curva che interpola tutti i livelli delle celle che la compongono; CNNCellViewer(fig. 3.9) mostra i livelli di ogni singola cella impiegando un codice colore che va dal blu al giallo, livelli bassi della cella corrispondono a colori freddi (blue) livelli alti della cella corrispondono a colori caldi (rosso e giallo);

L'oscilloscopio permette di tracciare l'andamento di una singola cella nel tempo. La FFTWindow(fig. 3.10 mostra lo spettro di fourier istantaneo di tutta la linea oppure traccia lo spettro nel tempo del segnale che attraversa una data cella. Il

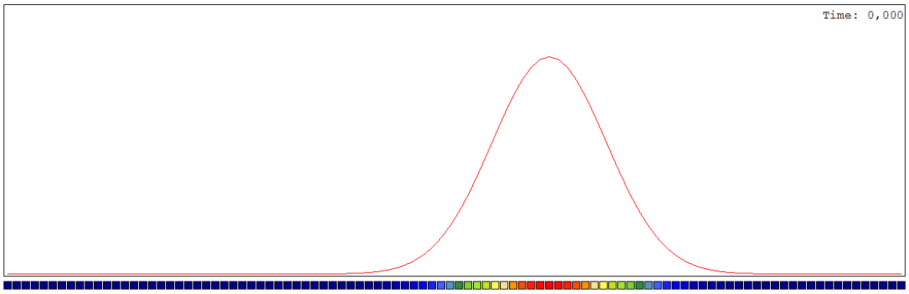


Figura 3.9: CNNCellViewer

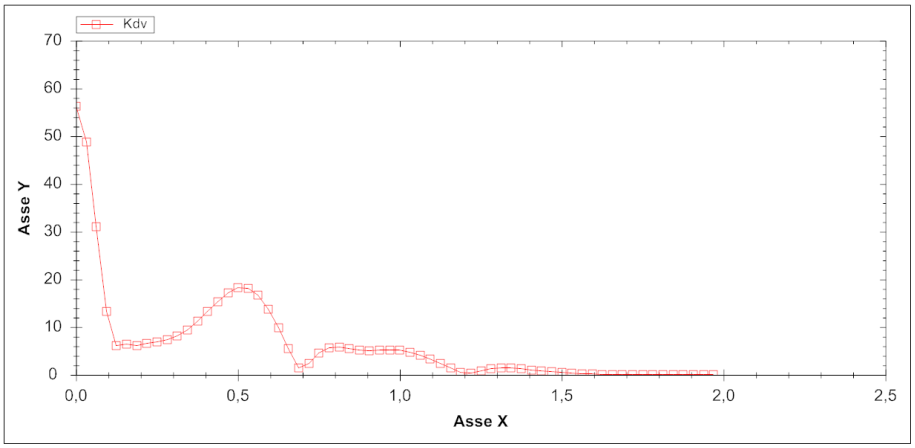


Figura 3.10: FFT Window

gestore della grafica si occupa di transcodificare i dati prodotti dalla linea simulata in informazioni di carattere grafico. La grafica 2D è gestita nativamente dal framework .net impiegato, la parte di visualizzazione tridimensionale, impiegato nella visualizzazione di più linee conteporaneamente, è un wrapper per le librerie grafiche 3D OpenGL [74]. Per ultimo troviamo il motore di scritping basato sul linguaggio Lua [72]. Compito di questa parte della libreria è fornire un'interfaccia di integrazione con il sistema rapida e snella. L'impiego principale è la formulazione dei segnali di ingresso impiegati per le simulazioni. Ogni tipologia di segnale è codificato in linguaggio Lua e salvato come file di testo ed è compito della libreria eseguirlo quando necessario. Il motore è impiegato anche per la gestione degli au-

tomatismi dell'applicazione ed è possibile effettuare alcune attività, come scegliere il modello, configurarlo, eseguirlo e salvare l'andamento su file direttamente da script.

3.4 CNN Explorer

Gli esperimenti riguardanti la simulazione software delle CNN bidimensionali, sono stati condotti impiegando un'applicativo realizzato contestalmente allo studio di tale tipologia di reti, denominato CNN Explorer.

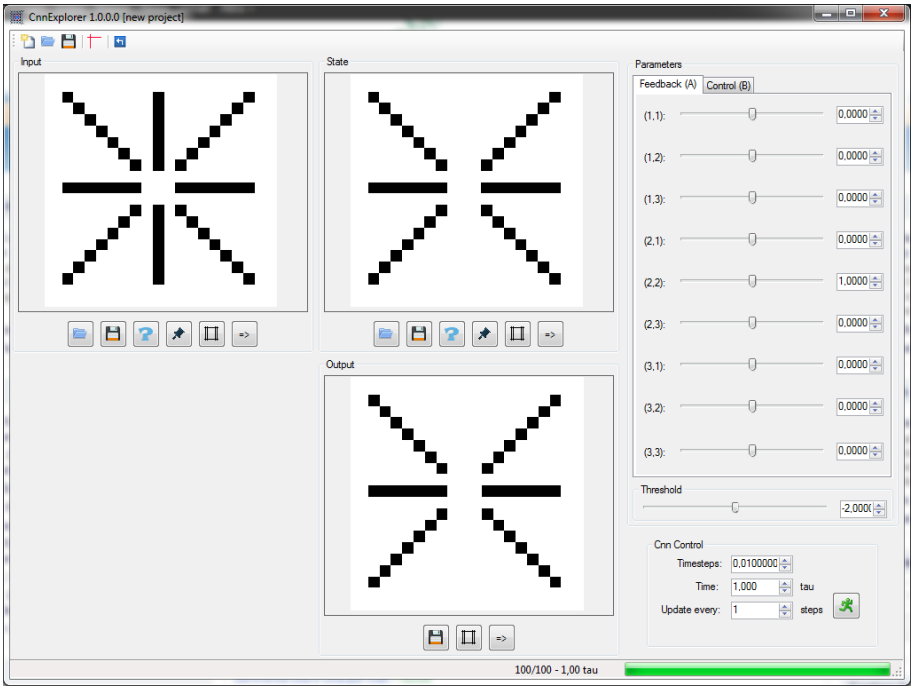


Figura 3.11: Interfaccia principale di cnnExplorer

L'interfaccia principale del software mostra chiaramente le tre matrice che caratterizzano una CNN : Ingresso, Stato Uscita. Le matrici, vengono rappresentate mediante immagini e il colore dei pixel è codificato secondo la (5.3). Per ogni matrice è possibile scegliere il comportamento ai bordi, scegliendo una tra le possibili configurazioni: Fisse, Zero Flusso e Toroidali. É possibile assegnare alle matrici di Stato e di input delle immagini presenti sul computer, oppure è possibile impostarne il valore in modo casuale o con un valore fissato.

Come è possibile modificare il valore delle matrici che descrivono la cnn è possibile anche modificare i valori delle matrici di template. Infine è possibile controllare la simulazione configurando il valore del passo di integrazione e la durata della stessa. Se la CNN dovesse convergere la simulazione viene comunque interrotta.

Una volta terminata la simulazione è ovviamente possibile salvare i risultati sia in formato immagine che in formato testo compatibile con la notazione di Matlab. La compatibilità con Matlab è stata introdotta al fine di impiegare i risultati ottenuti all'interno del famoso software di calcolo numerico per poter effettuare ulteriori analisi.

3.4.1 Moduli implementati

Nella realizzazione di questo applicativo sono state sviluppate due particoli CNN la CNN Standard, già fornita da libCNN e una classe che simula una CNN con l'introduzione del memristor, ottenuta come estensione della CNN standard. Un altra particolare funzionalità di CNN Explorer è l'addrestramento dei template mediante algoritmi genetici. Come mostrato in figura 3.12 il software permette di definire un immagine di ingresso e un'immagine di target da raggiungere. In basso viene mostrato l'andamento della fitness durante l'evoluzione e per ogni nuova popolazione viene mostrato il risultato ottenuto con l'individuo migliore.

3.5 CNN3D Explorer

Gli esperimenti riguardanti la simulazione software di CNN3D, sono stati condotti impiegando un'applicativo realizzato sempre contestualmente al percorso dottorale e denominato CNN3D Explorer. Le funzionalità di questa applicazione vengono erogate completamente attraverso la libreria libCNN, fatta eccezione per la generazione dei dati iniziali. Per garantire una maggiore flessibilità i dati iniziali vengono inseriti mediante script lua. La flessibilità deriva dalla possibilità di poter

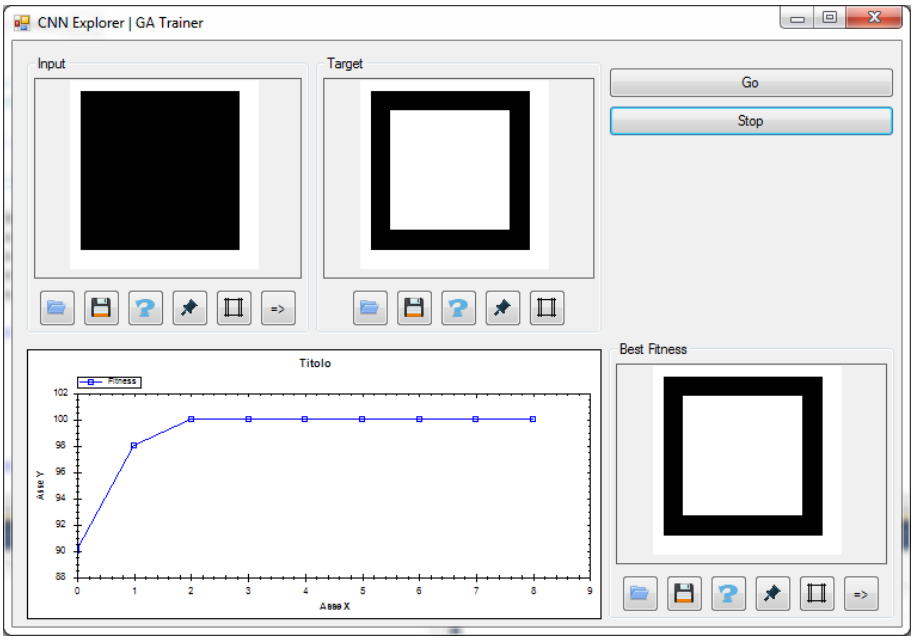


Figura 3.12: Interfaccia di cnnExplorer per l'addestramento dei Template mediante GA.

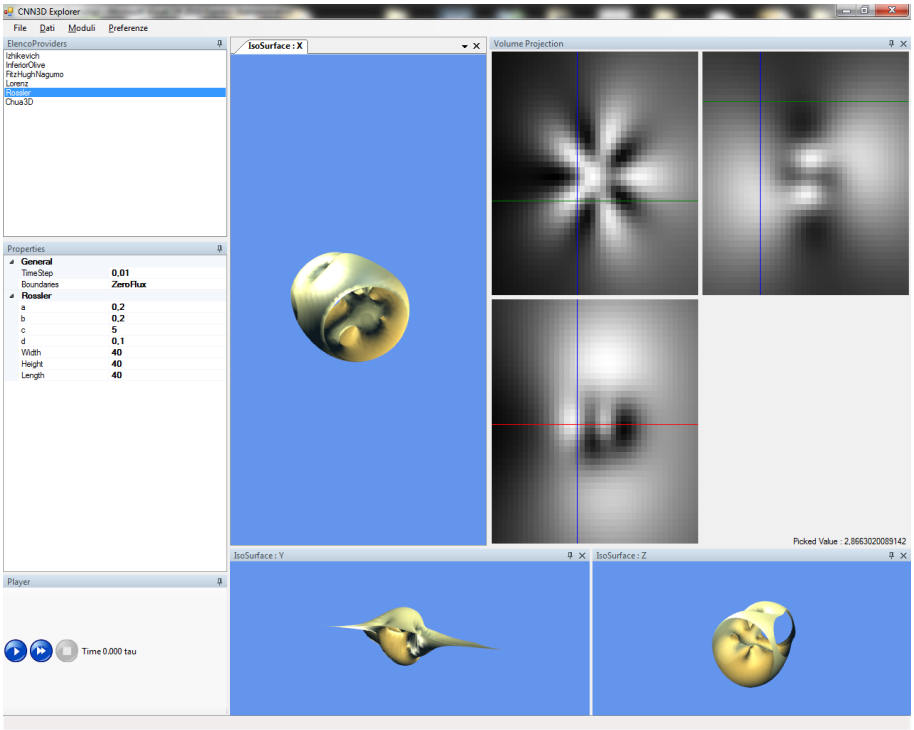


Figura 3.13: Interfaccia principale di CNN3D Explorer

editare gli script lua con un semplice editor di testo senza la necessità di compilazione o riavviare l'applicativo, un esempio di script lua per la definizione di una sfera è mostrato nel listato 3.6

Codice 3.6: Script lua

```
parameters = {
    centre =  Point3D(0,0,0),
    radius = 0.5
};

function getName()
    return "Sfera";
end

function getDimensions()
    return 1;
end

function getComponentName( i )
 return "X";
end

function getValue( x1,y1,z1 )
x = x1 - parameters.centre.X;
y = y1 - parameters.centre.Y;
z = z1 - parameters.centre.Z;
r = parameters.radius;
return (x*x + y*y + z*z - r*r);
end
```

Tutti gli script lua vanno inseriti in una particolare sotto-directory dell'applicativo controllata dall'applicazione e caricati dinamicamente a run-time

Capitolo 4

CNN monodimensionali

4.1 Introduzione

Nell' esplorare le possibilità offerte dal modello computazionale delle CNN nella loro versione mono-dimensionale è stata implementata una linea di trasmissione non lineare.

La linea di trasmissione è stata realizzata, impiegando una CNN monodimensionale non standard implementabile fisicamente mediante un circuito elettronico di cui è mostrato il dettaglio di una singola cella $i-esima$ in figura 4.1. Ogni cella

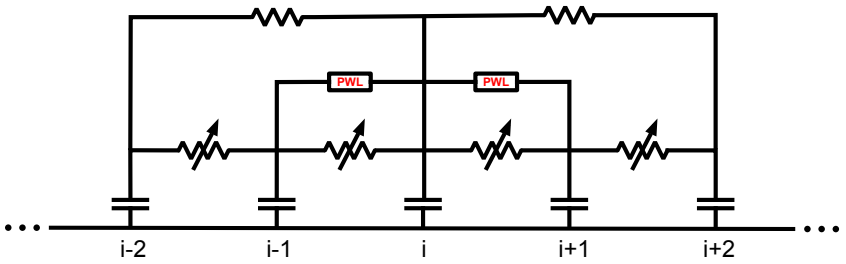


Figura 4.1: Connessioni tra le celle in una NLTL basata su CNN. PWL è un elemento non lineare basato su una funzione lineare a tratti.

è composta da un singolo condensatore C_i ed è collegata alla celle vicine mediante tre differenti connessioni: Un primo collegamento, di natura eccitatoria, è presente tra le celle più vicine [25], un secondo collegamento, di tipo inibitore lineare, mette in contatto i vicini secondi e una terza connessione, di tipo non lineare connette

ancora una volta le celle più prossime. La connessione non lineare può avvenire impiegando generatori che modellano funzioni lineari a tratti (PWL) [36]. Sia u_i lo stato della cella $i-esima$ e la tensione ai capi del condensatore, R il fattore che regola la connessione inibitrice R' il fattore che regola la connessione eccitatoria e R'' il coefficiente del termine non lineare. L'equazione che regola lo stato di una singola cella $i-esima$ è

$$\frac{u_{i-2}-u_i}{R}-\frac{u_{i-1}-u_i}{R'}+\frac{u_{i-1}^2-u_i^2}{R''}=\frac{u_{i+2}-u_i}{R}-\frac{u_{i+1}-u_i}{R'}+\frac{u_{i+1}^2-u_i^2}{R''}+C\frac{du_i}{dt} \quad (4.1)$$

dalla quale otteniamo la seguente

$$-C\frac{du_i}{dt}=\frac{u_{i+2}-u_{i-2}}{R}-\frac{u_{i+1}-u_{i-1}}{R'}+\frac{u_{i+1}^2-u_{i-1}^2}{R''} \quad (4.2)$$

Ponendo $R'=R/2$ otteniamo

$$-\frac{du_i}{dt}=\frac{u_{i+2}-u_{i+1}+u_{i-1}-u_{i-2}}{RC}+\frac{u_{i+1}^2-u_{i-1}^2}{R''C} \quad (4.3)$$

Sostituendo, in fine, $RC=2h^2$ ed $R''C=2h/3$, dove $h=cost.$, otteniamo la seguente equazione

$$\frac{du_i}{dt}=-\frac{1}{2h^3}(u_{i+2}-u_{i+1}+u_{i-1}-u_{i-2})-\frac{3}{2h}(u_{i+1}^2-u_{i-1}^2) \quad (4.4)$$

Quindi, la dinamica di ogni cella è regolata dallo stato delle celle vicine, con un raggio di influenza $r=2$. In particolare le celle adiacenti (distanti raggio $r=1$), interagiscono tra loro sia in modo lineare che non lineare. A questo punto,bisogna indagare il rapporto tra la (4.4) e l'equazione KdV. Per fare questo, consideriamo un gran numero di cellule e impostiamo $h=\triangle x\to 0$, operando le seguenti sostituzioni:

$$\frac{\partial u(x,t)}{\partial t}\to\frac{du_i}{dt}=u_t$$

$$\frac{3}{2h}(u_{i+1}^2 - u_{i-1}^2) \rightarrow$$

$$\rightarrow \quad 3\frac{u^2(x+\triangle x,t) - u^2(x-\triangle x,t)}{2\triangle x} \rightarrow$$

$$\rightarrow \quad 3\frac{\partial u^2(x,t)}{\partial x} = 6uu_x$$

$$\frac{1}{2h^3}(u_{i+2} - u_{i+1} + u_{i-1} - u_{i-2}) \rightarrow$$

$$\rightarrow \frac{u(x+2\triangle x,t) - 2u(x+\triangle x,t) + 2u(x-\triangle x,t) - u(x-2\triangle x,t)}{2(\triangle x)^2} \rightarrow$$

$$\rightarrow \frac{\partial^3 u(x,t)}{\partial x^3} = 6u_{xxx}$$

possiamo riscrivere la (4.3) nel seguente modo:

$$u_t + 6uu_x + u_{xxx} = 0 \tag{4.5}$$

dall'espressione della (4.5), frutto delle sostituzioni precedenti, possiamo dedurre che nel caso continuo la (4.4) approssima la KdV, inoltre possiamo chiamare la (4.4) KdV-CNN [22].

Dopo una prima serie di esprimenti incentrati, principalmente, sulla validazione e l'analisi del modello di KDV-CNN si è proceduto ad estendere il campo di azione indagando il comportamento di questa tipologia di linee nel caso di incroci tra due linee. Le linee che si incrociano, condividono tra loro una cella. Nel punto di intersezione il modello che viene impiegato somma i contributi provenienti da

entrambe le linee. (4.6)

Lo stato della cella C_k posta all'incrocio tra le linee $l1$ e $l2$ nel punto i, j con $i \in l1$ e $j \in l2$

$$\begin{aligned}\frac{du_k}{dt} = &-\frac{1}{2h^3}(u_{i+2} - u_{i+1} + u_{i-1} - u_{i-2}) - \frac{3}{2h}(u_{i+1}^2 - u_{i-1}^2) \\ &-\frac{1}{2h^3}(u_{j+2} - u_{j+1} + u_{j-1} - u_{j-2}) - \frac{3}{2h}(u_{j+1}^2 - u_{j-1}^2)\end{aligned} \tag{4.6}$$

4.2 Esperimenti

Ottenuta l'espressione della Kdv-CNN sono stati condotti alcuni esperimenti. I primi esperimenti sono stati condotti per verificare l'esattezza dei risultati. In particolare sono stati comparati i risultati prodotti dalle simulazioni numeriche della (4.4) con i risultati analitici della (4.5), verificata la correttezza dei risultati si è proceduto con l'estensione delgli esperimenti ai casi in cui siano presenti incroci. Le simulazioni sono state condotte impiegando il software "wireExplorer" realizzato contestualmente a questo lavoro e descritto nel capitolo 3.3. Nelle seguenti simulazioni sono state impiegate CNN composte da 300 celle con condizioni al contorno toroidali. La scelta di tale tipologia di condizione al controno ci permette di simulare linee di trasmissione circolari. Quindi sono state integrate 300 ODE nella forma di (4.4), impiegando come metodo di integrazione il Runge-Kutta del IV ordine e adattando il passo di integrazione in base all'accuratezza richiesta dai singoli esperimenti.

4.2.1 Onda Gaussiana

Il primo esperimento è volto a verificare la formazione di solitoni a partire da una onda gaussiana 4.2. Per prima cosa viene impostato lo stato delle celle C_i in modo tale che la linea simulata assuma al tempo $t = 0$ la configurazione data dalla

seguente espressione:

$$u_i(0) = 4e^{-\frac{1}{20(ih)^2}}. \tag{4.7}$$

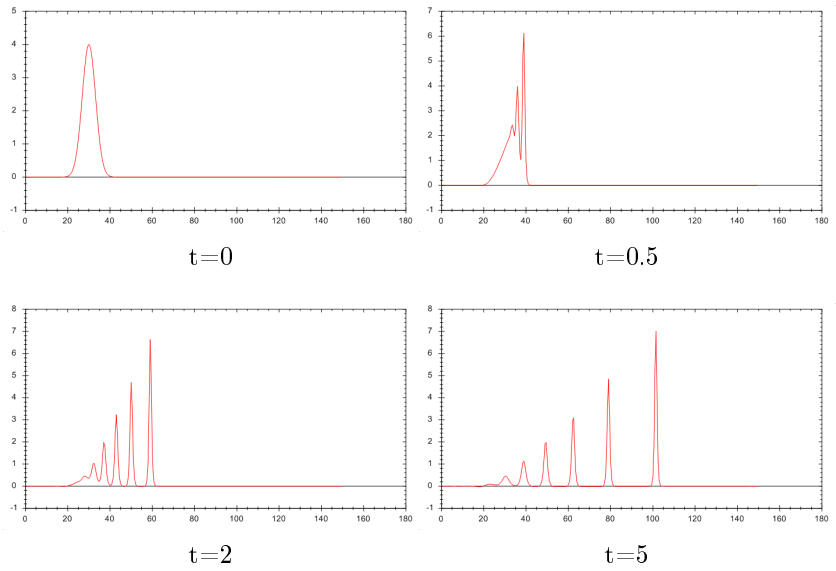


Figura 4.2: Generazione di onde solitoniche a partire da un'onda gaussiana

Successivamente, lo stato delle celle viene fatto evolvere secondo la 4.4 nell'intervallo $t \in [0, 5]$. Ciò che si osserva è la formazione di onde solitoniche. In particolare dopo 5000 passi di integrazione con $t = 0.5$ inizia a formarsi un treno di onde solitoniche (fig. 4.2b), nelle figure 4.2c e 4.2d si nota l'evoluzione individuale delle onde solitoniche generatesi.

4.2.2 Interazione tra solitoni

Successivamente, dai solitoni appena generati, ne sono stati estratti due: uno più veloce (con ampiezza maggiore) e uno più lento (con ampiezza minore).

come ci si aspettava l'interazione tra le due onde impiegate è tipica delle onde solitoniche, l'onda con ampiezza maggiore, ha anche velocità maggiore. Il solitone maggiore incontra il solitone minore e lo sorpassa. Dopo l'interazione la struttura delle onde resta invariata.

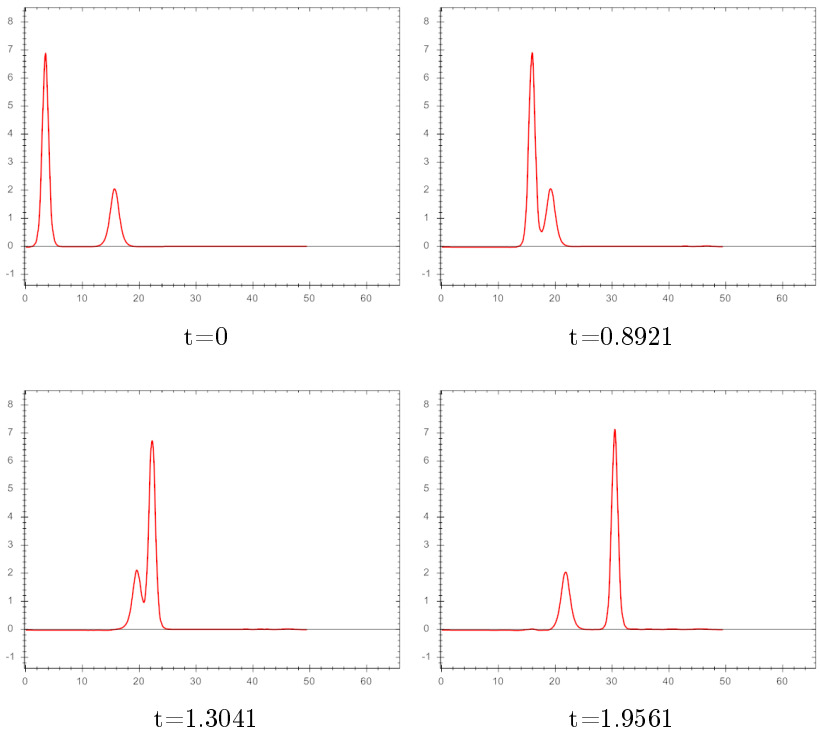


Figura 4.3: Interazione tra due solitoni

4.2.3 Generazione di Solitoni mediante Impulso

Nel terzo esperimento, si è voluto esaminare il comportamento della linea in risposta ad un impulso. L'impulso è stato impresso sulla prima cella della linea con ampiezza 4. Sono stati svolti 3 esperimenti con questa configurazione variando però le condizioni al contorno. In figura 4.4 vengono mostrati i comportamenti in caso di condizioni al contorno toroidali 4.4a, Zero flusso 4.4b e fisse 4.4c.

Ciò che si nota è la formazione di onde solitoniche con concavità verso il basso che si propagano da sinistra verso destra e onde solitoniche con concavità verso l'alto che si mouvono in senso opposto nel caso di condizioni al contorno toroidali. Nel caso di condizioni al contorno Zero Flux, ciò che si evince è la formazione di un treno d'onde che da una prima analisi potrebbero essere considerati solitoni, di fatti essi si propagano come mostrati in figura 4.4 durante tutta la durata dell'esperimento.

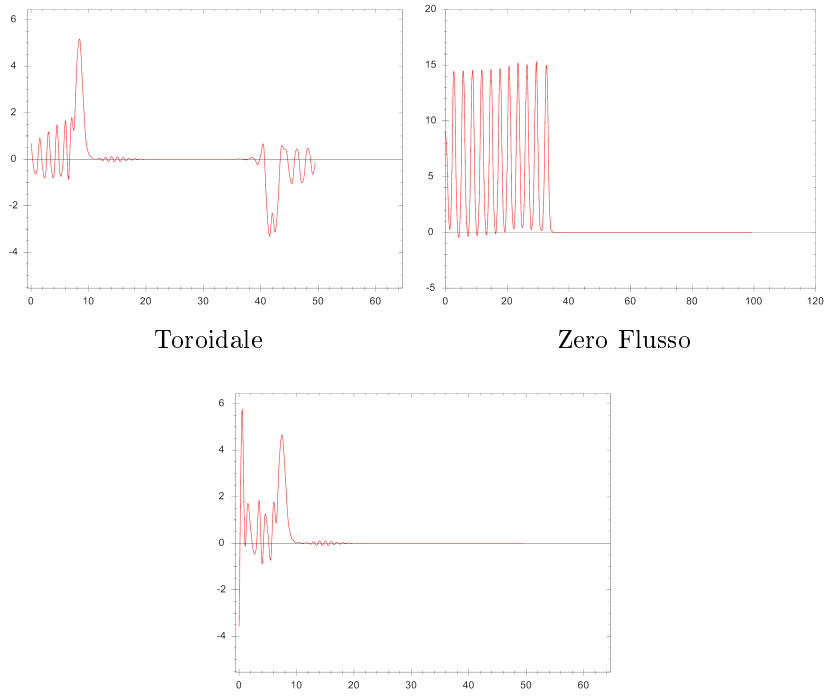


Figura 4.4: Generazione di solitoni dall'impulso con differenti condizioni al contorno

4.2.4 Monitoraggio dello stato di una cella

Nel quarto esperimento è stata monitorata l'evoluzione dello stato di una singola cella. L'esperimento consiste nel visualizzare su una sorta di oscilloscopio l'andamento di una singola cella. In figura 4.5 possiamo notare cosa avviene nella cella 20 al passaggio di un solitone (fig. 4.5b).

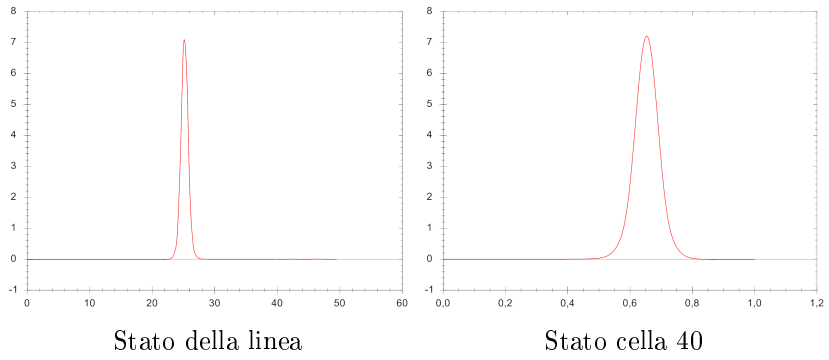


Figura 4.5: Analisi dell'evoluzione di una singola cella

In figura 4.5a vediamo lo stato della linea al tempo t=1.2, come si nota il so-

litone ha già attraversato la cella 20. In figura 4.5b vediamo invece come lo stato della cella 20 cambia mantenendo la forma d'onda del solitone stesso. Nel caso in cui la configurazione della cnn sia di tipo toroidale si ottiene un interessante risultato: lo stato della cella osservata cambia in modo periodico. Ciò che si ottiene è un segnale periodico da impiegare per scopi differenti come adesempio la sincronizzazione. In figura 4.6 è mostrato il segnale rilevato su una singola cella durante un tempo di evoluzione t=2.0. Ovviamente si può agire sulla frequenza

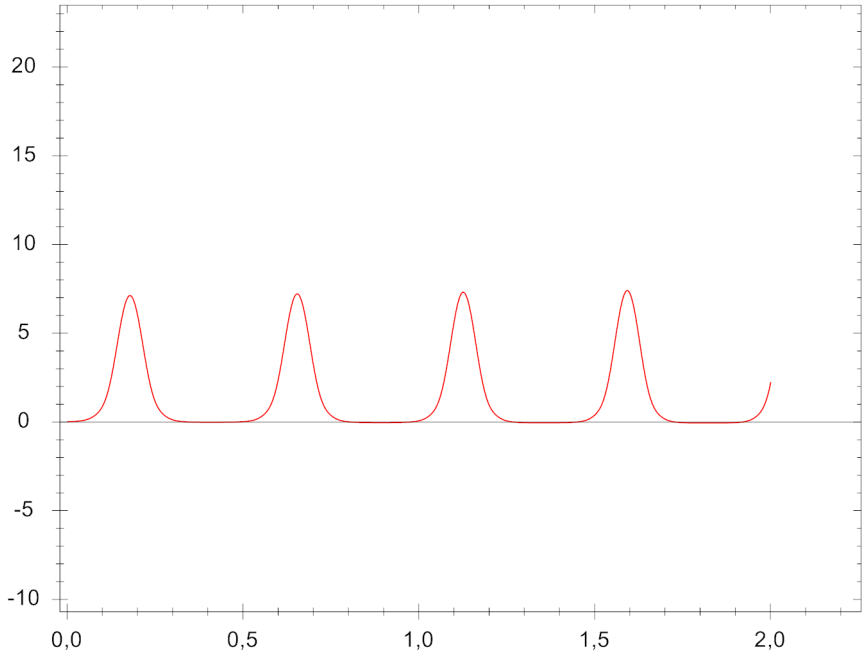


Figura 4.6: Segnale periodico registrato su una singola cella di una cnn toroidale

di tale segnale modificando o la lunghezza della cnn o l'ampiezza del solitone e quindi la velocità di traslazione dello stesso. In figura 4.7 viene mostrato il risultato ottenuto impiegando un solitone con ampiezza doppia rispetto all'ampiezza del solitone nella figura 4.6. Si può notare come il raddoppio della velocità si è tradotto in raddoppio di frequenza. In figura 4.7 si nota come l'ampiezza del solitone aumenta con l'aumentare del tempo di evoluzione. Questo è dovuto ad un effetto dell'errore legato all'integrazione numerica. Infatti durante le sperimentazione, si è verificato che per ampiezze iniziali maggiori del valore 4 si va incontro a fenomeni di instabilità che portano all'overflow numerico.

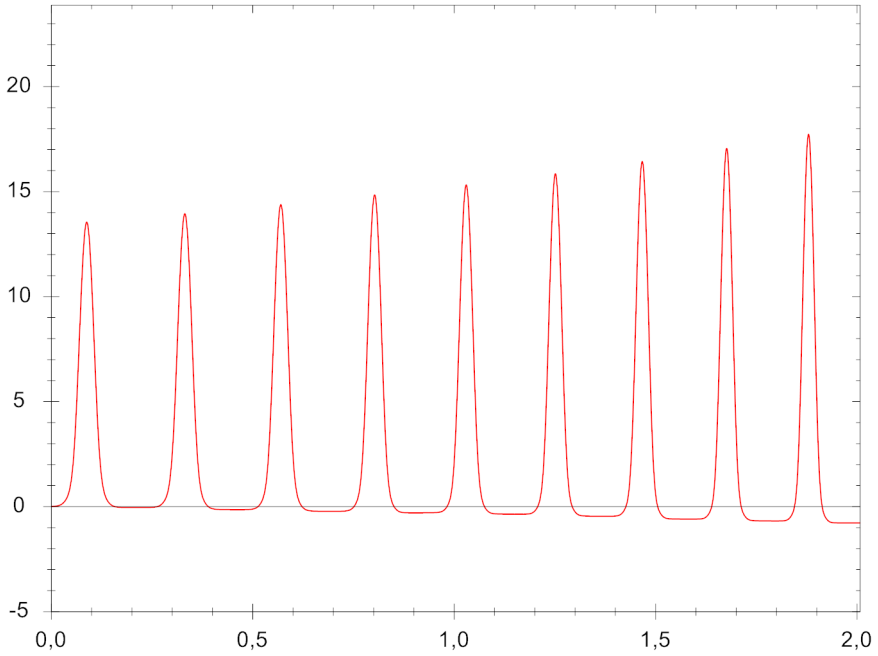


Figura 4.7: Segnale periodico registrato su una singola cella di una cnn toroidale con solitone veloce

4.2.5 Incrocio tra due linee I

Gli esperimenti successivi sono volti ad esaminare il comportamento di linee di trasmissione modellate con la CNN-KDV in presenza di incroci. Le due linee esaminate sono entrambe composte da 100 celle e condividono la cella numero 50. Il primo esperimento, mostrato in figura 4.8, prevede una configurazione iniziale con un solitone sulla prima linea(fig. 4.8a) con ampiezza 6.9 , mentre la seconda linea è impostata a zero (fig. 4.8b).

dopo 6000 iterazioni con t=0.6 il solitone sulla prima linea incrocia la seconda linea superando il punto di incrocio senza subire variazioni (fig. 4.8c). Sulla seconda linea, invece, compare un segnale(fig. 4.8d) che subito si suddivide in due onde solitoniche. La prima onda solitonica viaggia da sinistra verso destra con concavità verso il basso, mentre una seconda onda solitonica viaggia da sinistra verso destra con concavità rivolta verso l'alto (fig. 4.8e e 4.8f). L'ampiezza delle onde solitoni generatesi è 4.7 (-4.7 nel caso con concavità verso l'alto) rispetto al solitone iniziale sulla prima linea.

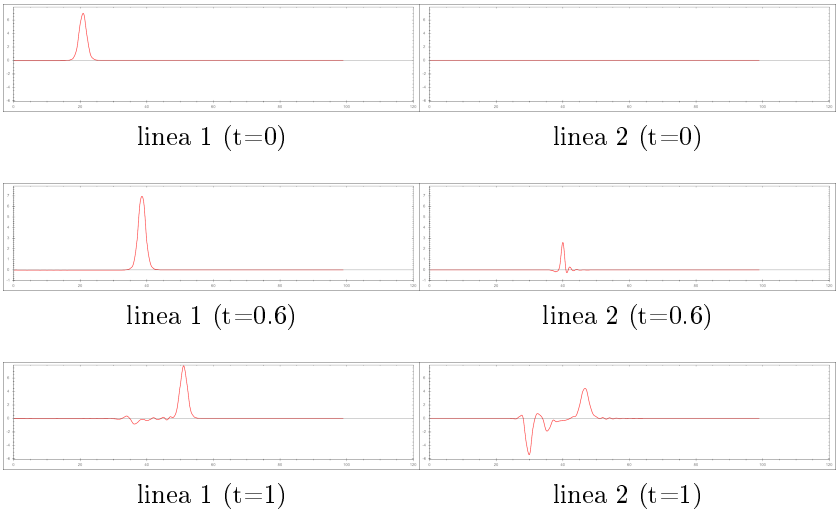


Figura 4.8: Incrocio tra due linee

4.2.6 Incrocio tra due linee II

Nel secondo esperimento, si considerano due linee composte da 100 celle e con punto di intersezione alla cella 30. Entrambe le linee vengono inizializzate con un solitone di ampiezza 6.9.

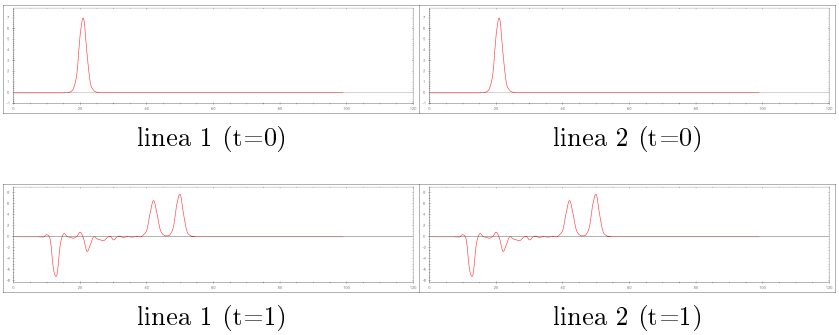
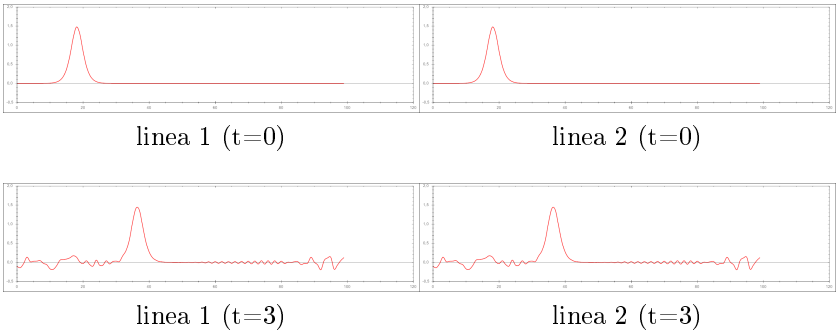


Figura 4.9: Incrocio tra due linee, con solitoni di ampiezza 6.9.

questa volta il risultato è simmetrico, nel senso che su entrambe le linee viene registrato lo stesso comportamento. Dopo il sorpasso della cella di intersezione su entrambe le linee si formano una coppia di solitoni che viaggiano da sinistra verso destra con concavità verso il basso e di ampiezza rispettivamente di 6.2 e 6.7 mentre compaiono due solitoni che viaggiano con concavità e verso inversi. L'ampiezza dei solitoni negativi è di -6.6 e 2.5.

4.2.7 Incrocio tra due linee III

Nel terzo esperimento, si considerano due linee composte da 100 celle e con punto di intersezione alla cella 30. Entrambe le linee vengono inizializzate con un solitone di ampiezza 1.5.



linea 1 (t=0) linea 2 (t=0)

linea 1 (t=3) linea 2 (t=3)

Figura 4.10: Incrocio tra due linee, con solitoni di ampiezza 1.5.

Impiegando solitoni di ampiezza inferiore, il passaggio dal punto di intersezione non genera nuovi solitoni, le ampiezze delle onde descrescono di poco e si forma sulla linea del rumore.

Capitolo 5

CNN bidimensionali

5.1 Introduzione

L'impiego principale di CNN Bi-dimensionale avviene nella computer vision, e nell'image processing così come si evince da numerosi lavori presenti in letteratura [4, 59, 27, 2, 55, 25] tuttavia sono presenti altri esempi di impiego delle CNN, estese alla terza dimensione, impiegate in altri settori e in robotica [16, 5, 9, 35, 52]. E' interessante, anche, osservare che esistono chip in grado si implementare il funzionamento di una CNN, [28, 4, 2, 33, 55, 35]. Tale approccio ci permette di modificare le equazioni che regolano una CNN e di implementarle su un chip dedicato. Come già accennato nel capitolo 2 lo stato x_{ij} di una singola cella $C(i,j)$ appartenente ad una CNN bi-dimensionale di dimensione MxN è espresso dalla seguente equazione:

$$\dot{x_{ij}} = -x_{ij} + \sum_{c(k,l)\in S_r(i,j)} A(i,j;k,l)y_{k,l} + \sum_{c(k,l)\in S_r(i,j)} B(i,j;k,l)u_{k,l} + z_{ij} \qquad (5.1)$$

Mentre l'output è regolato dall'espressione

$$y_{ij} = f(x_{ij}) = \frac{1}{2}|x_{ij}+1| - \frac{1}{2}|X_{ij}-1| \qquad (5.2)$$

Nelle espressioni (5.1) e (5.2) $x_{ij} \in R$, $y_{ij} \in R$, $u_{ij} \in R$ e $z_{ij} \in R$ sono rispettivamente **stato**, **output**, **input** e **soglia** della cella $C(i,j)$. $A(i,j;k,l)$ e $B(i,j;k,l)$ sono chiamati operatori di feedback (A) e di ingresso sinattico(B).

Nei paragrafi successivi verranno proposte le implementazioni di alcuni operatori della computer grafica realizzate con l'impiego delle CNN2D. In particolare verrà mostrato come codificare le immagini in espressione di stato di una CNN e come processarle.

5.2 Codifica delle CNN

Le CNN bidimensionali standard possono essere rappresentate come matrici di valori così come possono essere rappresentate le immagini in generale. Tipicamente, un'immagine digitale viene codifica associando a ciascun pixel un valore. Tale valore rappresenta un colore. Il dominio dei valori assegnati al pixel varia a seconda della codifica impiegata. Nel caso di immagini binarie sono possibili solo due valori: 0 che codifica il nero e 1 che codifica il bianco. Se impieghiamo immagini in toni di grigio il dominio di valori é compreso nell'intervallo $[0;255]$, che rappresenta il livello di grigio dell'immagine in quel punto. Nel caso delle immagini a colori la codifica è abbastanza differente infatti viene impiegata la codifica RGB. Un pixel, e quindi un colore, nel modello di colore RGB è descritto indicando la quantità di rosso (**R**ed), verde (**G**reen) e blu(**B**lu) contenuta. Il colore quindi è espresso come una tripletta RGB(r,g,b), ogni componente può generalmente variare nell'intervallo $[0,255]$ descrivendo così 16777216 colori differenti [70]. Essendo lo stato di ogni cella della CNN unitario, ne deriva che per processi di image processing è possibile impiegare solo immagini binarie o in toni di grigio, mentre le immagini a colori, descritte da tre componenti, non possono essere impiegate senza essere trattate. Tuttavia è possibile convertire una immagine a colori in una immagine in toni di grigio oppure è possibile scomporla in tre differenti immagini in toni di grigio, processando in modo separato le tre componenti del colore e ricompo-

nendole dopo il processo [34, 25]. In una CNN, il valore di ciascun pixel viene normalizzato, senza perdita di generalitá, nell'intervallo $[-1; 1]$; per convenzione, il valore -1 rappresenta un pixel bianco, $+1$ un elemento nero. Attraverso una semplice operazione di scaling, quindi, é possibile effettuare la conversione dalla codifica dell'immagine in scala di grigio alla codifica adottata dalle CNN. Sia U una matrice MxN che rappresenti le celle dell'input della CNN, e sia P la matrice MxN che rappresental'immagine. La trasformazione da pixel a CNN è regolata dalla seguente relazione:

$$U_{ij}(0) = 2 * (1 - \frac{P(i,j)}{255}) - 1 \tag{5.3}$$

Di seguito verranno condotti degli esperimenti volti ad esaminare il comportamento delle CNN. Per realizzare tali esperimenti è stato sviluppato un software che simula il comportamento di una CNN standard. Il software impiega l'algoritmo di integrazione numerico runge-kutta del IV ordine con passo di integrazione $\delta t = 0.01$ per risolvere le equazioni che regolano le celle. Il tempo è espresso dall'unita di misura *fittizia* **tau** multiplo del passo di integrazione [26, 25].

5.3 Edge-Detection CNN

Il primo esperimento tratta la problematica relativa alla detezione dei bordi in una immagine. Definiamo con P di dimensione MxN l'immagine da elaborare e usiamola per impostare i valori dell'input $u_{ij} = P_{ij}$. Lo stato iniziale della CNN lo impostiamo in modo uniforme a 0, ovvero, $x_{ij}(0) = 0$). Per le condizioni al contorno scegliamo i valori scegliamo la tipologia fissa con $u_{ij} = 0$ e $y_{ij} = 0$ per ogni cella virtuale. I template impiegati sono i seguenti [25] :

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad z = -1 \tag{5.4}$$

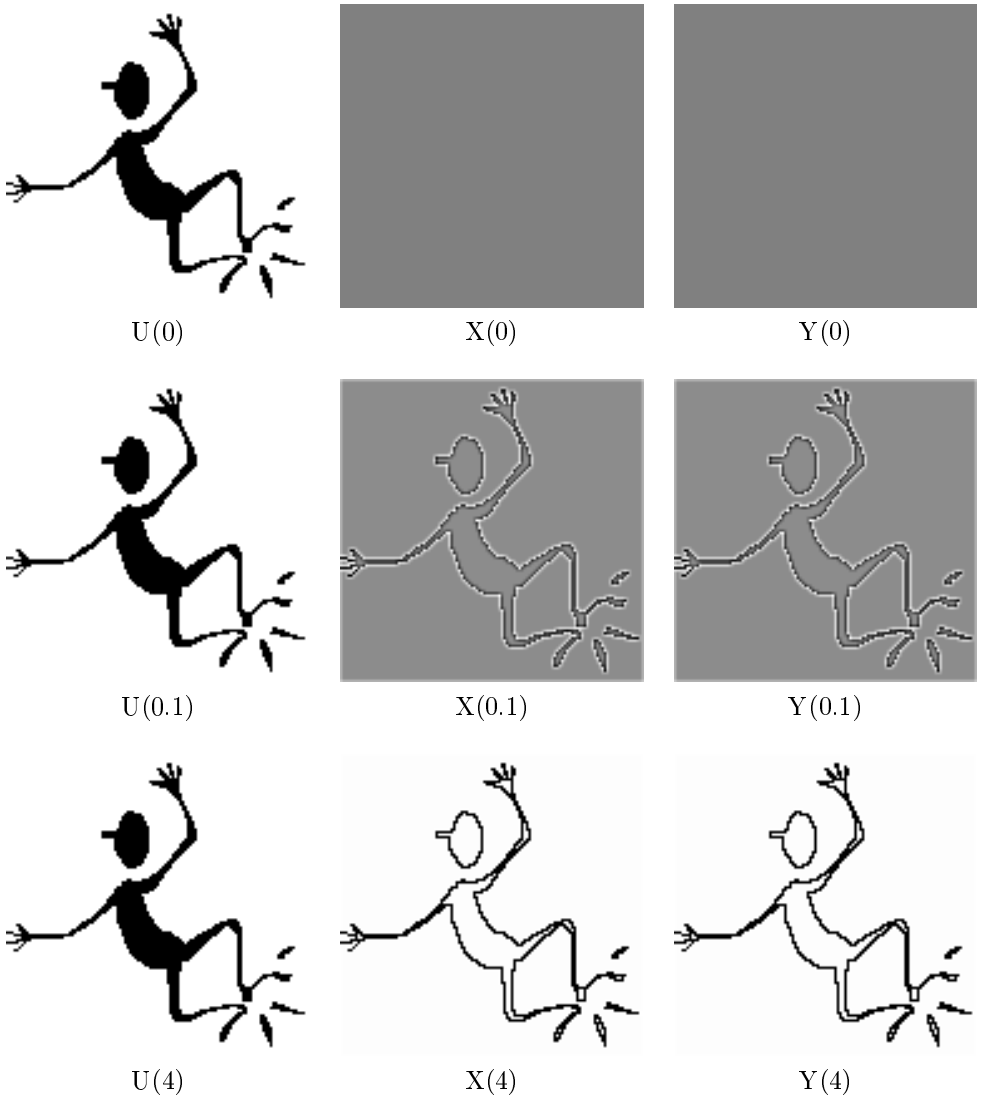
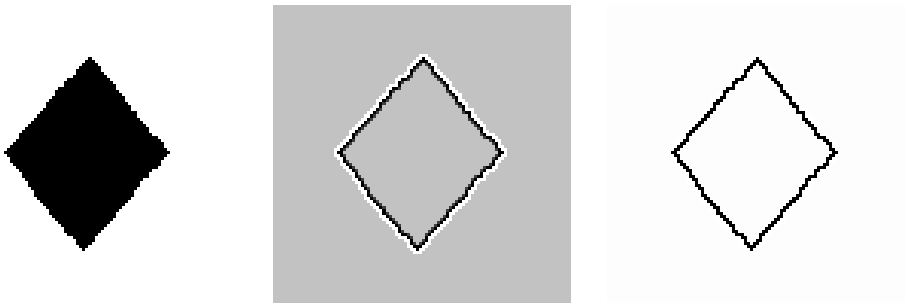


Figura 5.1: Edge Detection: Transizione dallo stato iniziale alla convergenza

Quando la CNN converge avremo i bordi della figura iniziale in nero e il resto dell'immagine si colora di bianco.

Come si osserva nella figura 5.1 l'output converge verso la formazione di una figura con contorni neri dopo 102 passi di evoluzione e un tempo t=1,02 tau. Le lettere $U()$, $X()$, $Y()$ indicano rispettivamente le matrici che determinano input,

stato e uscita della CNN.

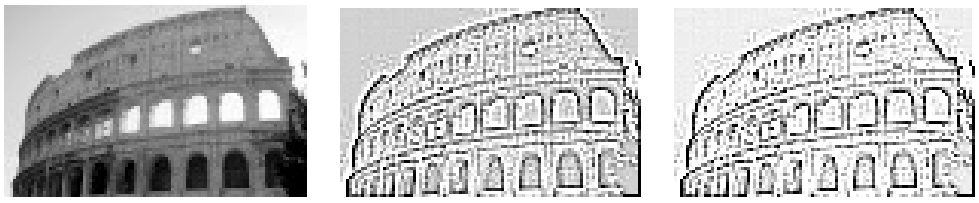


U(0) Y(0.5) Y(5)

Figura 5.2: Edge Detection: ulteriore esempio.

In figura 5.2 viene proposto un ulteriore esempio di scontornamento che richiede però più passi di integrazione prima di convergere.

Il template appena impiegato è limitato all'impiego per le sole figure in bianco e nero. In figura 5.3 viene mostrato il risultato dell'impiego di 5.4 con una immagine in toni di grigio. Dopo 200 passi di simulazione la CNN converge, ma a differenza dell'esperienza precedente notiamo che non c'è una netta separazione tra i contorni e il resto del contenuto visivo.



U(0) Y(1) Y(2)

Figura 5.3: Edge Detection: Transizione dallo stato iniziale alla convergenza di una immagine a toni di grigio.

5.4 Edge-Detection con toni di grigio CNN

Per realizzare la detezione dei bordi in immagini in toni di grigio andiamo a modificare il template in modo tale da riuscire in questo scopo secondo la configurazione proposta in letteratura[25] :

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad z = -0.5 \tag{5.5}$$

Con questo template l'immagine in toni di grigio impiegata viene trasformata in una immagine binaria e scontornata, eventuali imperfezioni dell'immagine iniziale vengono binarizzate. Il risultato è quello atteso, come si deduce dall'esempio in figura 5.4 e dall'esperimento con una immagine più dettagliata in figura 5.5.

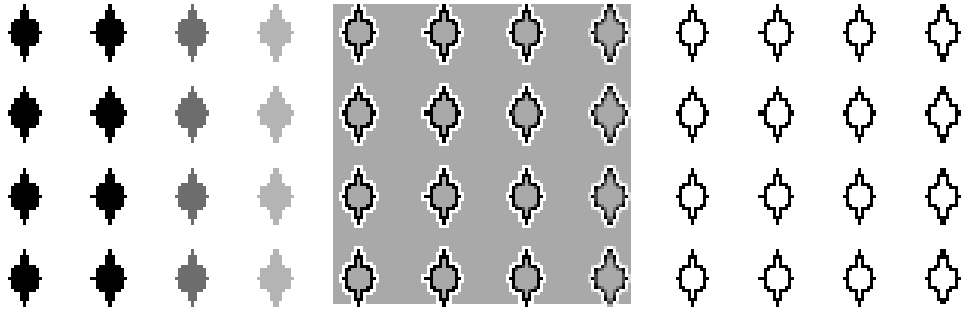


Figura 5.4: Edge detection in toni di grigio: I esempio

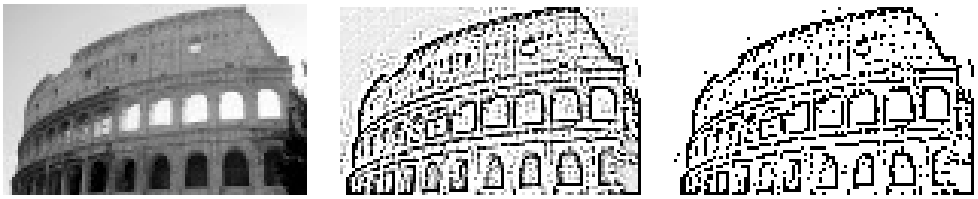


Figura 5.5: Edge detection in toni di grigio: II esempio

5.5 Rimozione delle linee verticali

Nel seguente esperimento viene impiegato un template per ottenere una CNN in grado di rilevare e rimuovere le linee verticali da una figura. Le matrici di Feedback e controllo e il valore di soglia necessari a svolgere tale compito sono i seguenti:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad z = -2 \tag{5.6}$$

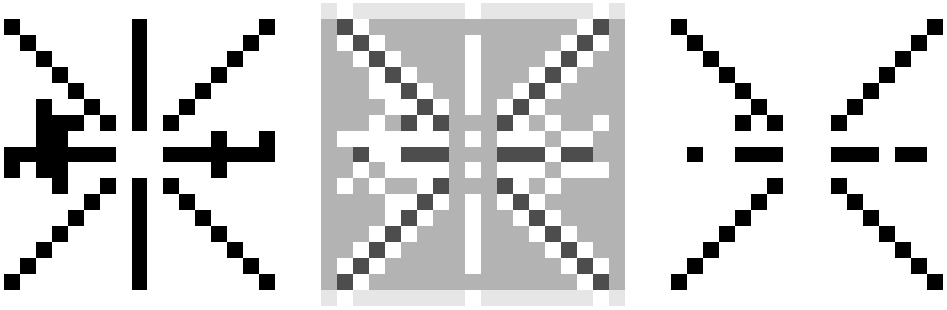


Figura 5.6: Rimozione delle linee verticali

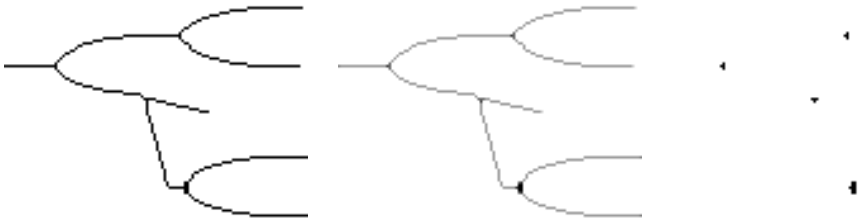
Dopo 100 passi di integrazione con $t = 1$ la CNN converge. Come si nota dalla figura 5.6 le linee verticali vengono rimosse.

5.6 Estrazione dei punti di giunzione.

Nel seguente esperimento viene impiegato un template per ottenere una CNN in grado di rilevare ed estrarre i punti di giunzione di figure con linee sottili. Le matrici di Feedback e controllo e il valore di soglia necessari a svolgere tale compito sono i seguenti:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 6 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad z = -3 \tag{5.7}$$

Dopo 100 passi di integrazione con $t = 1$ la CNN converge. Come si nota dalla figura 5.7 le linee verticali vengono rimosse.



U(0) Y(0.2) Y(1)

Figura 5.7: Rimozione delle linee verticali

5.7 Definizione dei template

Molti dei templates presenti in letteratura sono stati ottenuti con tecniche trial-and-error, ricorrendo alle simulazioni per investigare la dinamica di una rete non-lineare cellulare configurata sulla base di un dato set parametrico [22, 25]. Molti templates proposti subito dopo l'introduzione delle CNN derivano principalmente dall'adattamento di operatori giá noti nel settore dell'elaborazione di immagini. Tuttavia, poiché i pesi della rete assumono valori appartenenti all'insieme dei numeri reali, tale approccio risulta poco adatto per la ricerca di coefficienti analogici e, in pratica, non consente di far emergere le potenzialitá delle CNN. Per una rete neurale cellulare standard composta da celle identiche, infatti, l'equazione che ne descrive formalmente il comportamento é determinata dai 19 elementi del template (Matrice A, Matrice B e Soglia). In uno spazio dei parametri cosí vasto, il numero di possibili configurazioni della rete é praticamente infinito e, di conseguenza, é necessario adottare specifiche metodologie sistematiche ingrado di determinare templates, inoltre sono necessari algoritmi che rendono una CNN in grado di eseguire task assegnati.

Le principali tecniche di definizione dei templates sono le seguenti [22]:

- metodi basati sulla rappresentazione di CNN spazio invarianti mediante funzioni booleane, secondo tecniche di decomposizione.
- tecniche numeriche per l'ottimizzazione dei parametri

- algoritmi genetici per l'evoluzione dei pesi della rete.
- metodi basati sulla risoluzione di PDE discretizzate nello spazio.
- metodi basati sull'emulazione di processi cognitivi (es. retina)
- tecniche basate sulla logica fuzzy.
- applicazione di algoritmi per reti neurali artificiali
- tecniche basate sulla teoria del controllo, teoria dei sistemi dinamici nonlineari, sui sistemi di telecomunicazioni, etc

In questo lavoro vengono impiegati algoritmi genetici per l'evoluzione dei pesi della rete e metodi basati sulla risoluzione di PDE discretizzate nello spazio. Nel caso specifico delle CNN standard bi-dimensionali useremo la metodologia evolutiva.

5.7.1 Algoritmi di ricerca basati su GA

Un interessante metodologia di calcolo e di simulazione, potente e versatile, è rappresentata dalle tecniche di ottimizzazione note in letteratura sotto il nome di algoritmi genetici(GA). I GA furono proposti da J.H. Holland nel 1975 [41] e costituiscono un modello computazionale che si ispira all'evoluzione darwiniana, basato sui principi della mutazione genetica e della selezione naturale. Tali algoritmi simulano l'evoluzione di una popolazione di n individui (fig. 5.8), che rappresentano possibili soluzioni per un dato problema.

Ogni individuo ha un codice genetico (genoma) che è codificato e che permette di trasferire o rappresentare delle informazioni. Gli individui di una popolazione hanno patrimoni genetici differenti i quali garantiscono una varianza genetica [41, 30]. Ad ogni generazione, coppie di individui si uniscono per generare altri individui (crossover). Gli individui cosí generati saranno dotati di un patrimonio genetico risultante dalla combinazione del DNA dei genitori. L'adattamento degli individui

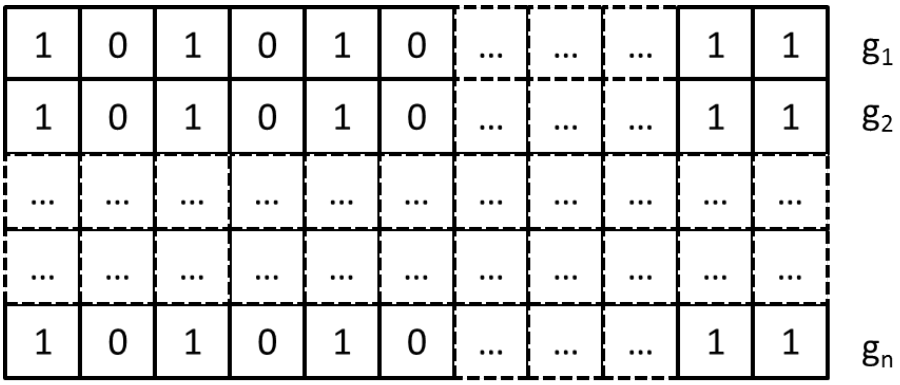


Figura 5.8: Rappresentazione del genoma di una popolazione.

all'ambiente dipende fortemente dal loro patrimonio genetico. Le informazioni contenute nel patrimono genetico vengono impiegate, per valutare le prestazioni di ciascun individuo rispetto al problema da risolvere, mediante l'applicazione di una funzione, detta funzione di fitness. Gli individui migliori, dotati di un valore di fitness maggiore, sono mediamente favoriti rispetto agli altri (selezione naturale).

Un algoritmo genetico puó essere implementato come una procedura di ricerca iterativa il cui scopo é l'ottimizzazione della funzione di fitness che ovviamente rappresenta la funzione obiettivo. Quindi, valori di fitness piú elevati vengono assegnati a soluzioni del problema migliori. Un algoritmo genetico, dopo aver selezionato una rappresentazione genetica, la funzione di fitness da valutare e il numero massimo di evoluzioni da simulare, procede secondo alcuni passaggi riportati di seguito (fig. 5.9):

1. Generazione di N individui(popolazione iniziale) assegnando loro un genoma con valori casuali che codifichino le possibili soluzioni del problema.

2. Decodifica dei genotipi della popolazione e valutazione di ciascun individuo (fenotipo) secondo la funzione di fitness.

3. Se la popolazione corrente contiene una soluzione soddisfacente, oppure, se il numero di evoluzioni è pari al valore prefissato l'algoritmo termina.

4. Generazione di una nuova popolazione di individui applicando gli operatori di selezione (fig. 5.10a), crossover(fig. 5.10b) e mutazione(fig. 5.10c). Il procedimento prosegue con la valutazione dei nuovi individui mediante la funzione di fitness e continua ciclicamente nella modalità appena descritta fino a che non sia ottenuta una soluzione soddisfacente per il problema assegnato

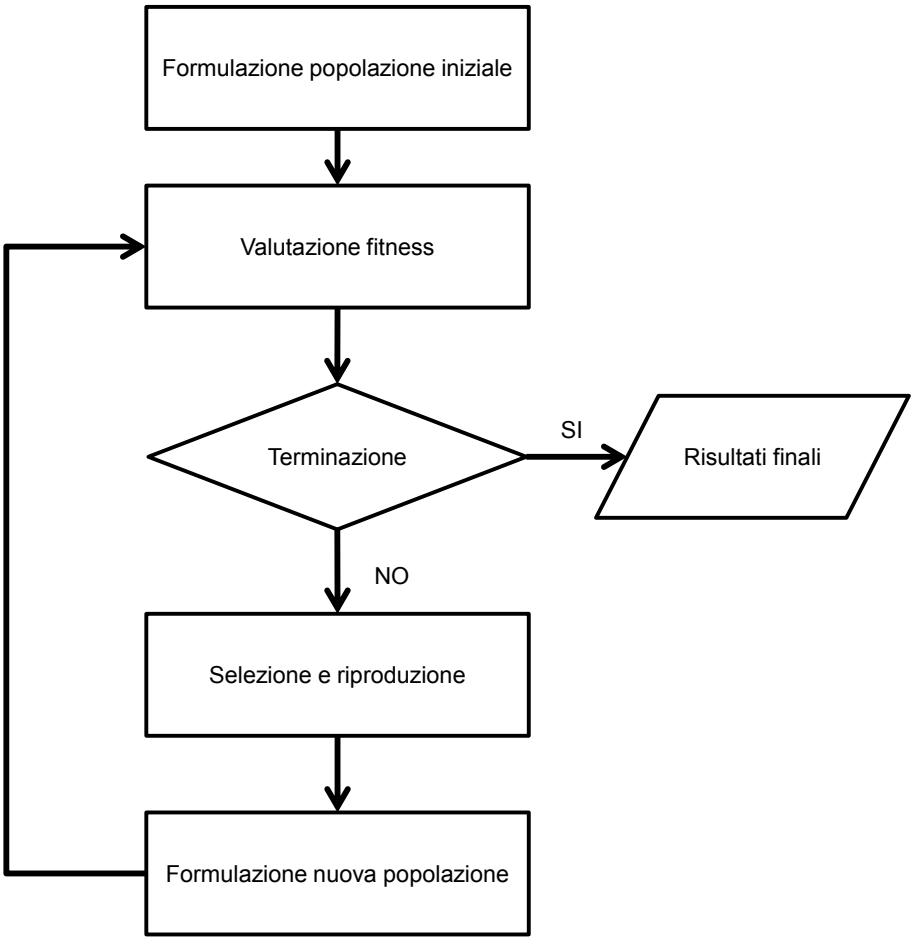


Figura 5.9: Schema di un algoritmo genetico.

Gli operatori di *selezione*, *crossover* e *mutazione*, proposti da Holland, si ispirano alla selezione naturale ed alla genetica ed operano sia sulla popolazione che sugli individui :

- Selezione : seleziona gli individui migliori con probabilitá proporzionale al valore di fitness formando quella che è detta élite (fig. 5.10a).

- Crossover: sceglie, in modo casuale, coppie di individui dalla élite e ne ricombina il genoma formando coppie di individui figli (fig. 5.10b).
- Mutazione: modifica in modo casuale e con probabilitá molto bassa, alcuni geni degli individui generati attraverso il crossover(fig. 5.10c).

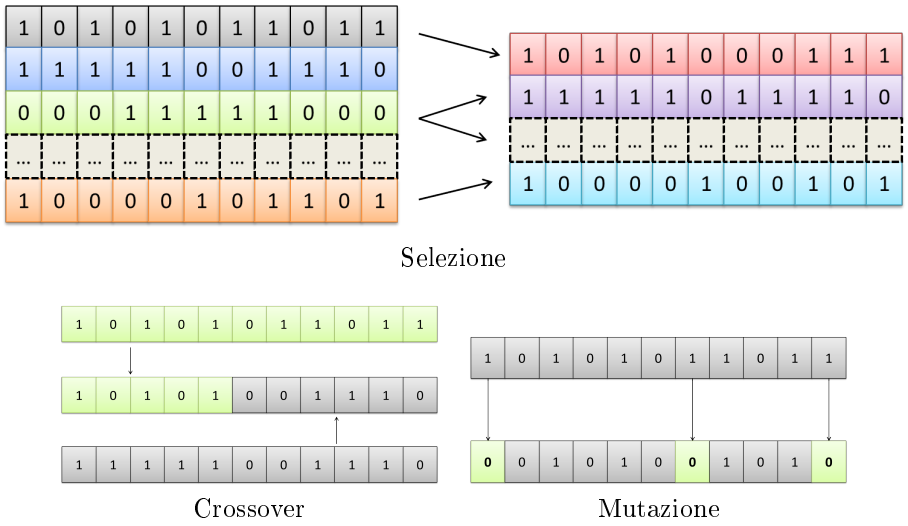


Figura 5.10: Operatori genetici

5.8 Implementazione di GA per le CNN

L'obiettivo degli algoritmi genetici, per un dato problema, é evolvere un insieme iniziale di potenziali soluzioni casuali facendolo convergere verso una soluzione ottima. Il primo passo nella realizzazione di un GA consiste nello scegliere un'opportuna struttura e un'adeguata codifica dei punti dello spazio di ricerca. Nel caso delle CNN, per progettare un algoritmo genetico per la ricerca dei pesi di una CNN standard bidimensionale spazio invariante in cui ciascuna cella ha raggio di influenza r = 1, risulta conveniente adottare una rappresentazione dei templates in forma vettoriale; a tale scopo, i 19 parametri che definiscono la terna $A; B; z$, vengono disposti in un vettore composto da 9 valori uno per ogni componente della matrice di feedback $a_{kl} \in A$ con $k, l = 1 \dots 3$, 9 valori uno per ognuna delle componenti della matrice di controllo $b_{kl} \in B$ con $k, l = 1 \dots 3$. L'ultimo elemento del vettore

che descrive il genoma della CNN e la soglia z (fig. 5.11). I 19 coefficienti appena elencati rappresentano i geni della CNN, a cui é associata una particolare funzione da svolgere; l'insieme di tutti i geni CNN costituisce il genoma CNN.

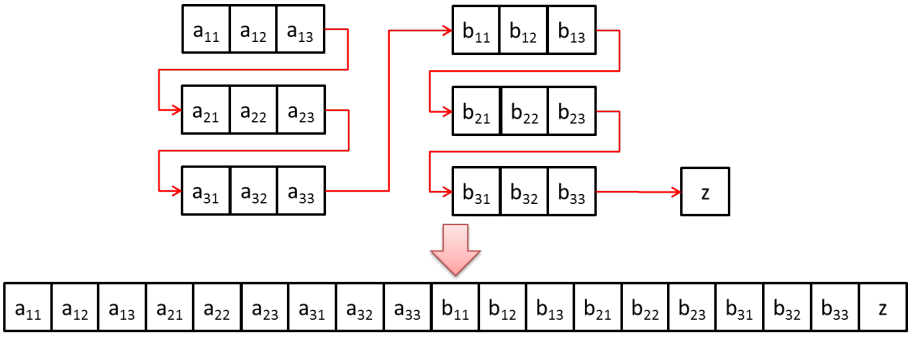


Figura 5.11: Genoma della CNN.

Si può dimostrare [27, 26] che se le matrici A e B sono simmetriche vengono soddisfatte le condizioni per la stabilitá della CNN previste nel teorema (2.11), che assicurano la convergenza della rete. Il criterio di convergenza è importante per le CNN, quindi possiamo ridurre il numero di geni del genoma della CNN da 19 elementi a 13, cinque appartenenti alla matrice A, cinque alla matrice B e uno corrispondente alla soglia.

$$G = [a_{11}a_{12}a_{13}a_{21}a_{22}a_{23}b_{11}b_{12}b_{13}b_{21}b_{22}b_{33}z] \tag{5.8}$$

Nel paradigma degli algoritmi genetici, ciascun genotipo é rappresentato da un vettore G composto da 11 elementi5.8. Per valutare la fitness di un gene CNN rispetto al problema assegnato, si introduce un'immagine obiettivo T, che viene impiegata per valutare l'adeguatezza della rete verso il compito assegnatoli. Si parte con l'applicare il template definito da G all'immagine di ingresso alla CNN, poi dopo la fase di processo e quando la CNN converge, l'immagine che si forma sull'uscita della rete I^G è confrontata con T, attraverso la funzione di costo:

$$diff(G) = \sum_{i=1}^{M}\sum_{j=1}^{N} I_{ij}^{G} \oplus T_{ij} \tag{5.9}$$

dove M e N indicano il numero di pixel di T e I^G e $\oplus$ denota l'operazione di xor tra gli elementi in posizione (i, j) dell'immagine obiettivo e quella proveniente dall'uscita della CNN. La funzione di fitness per ciascun fenotipo CNNG, quindi, é data dal numero di pixel che risultano essere uguali tra T e l'uscita della CNN:

$$fitness(CNN^G) = MxN - diff(G); \tag{5.10}$$

La fitness misura, quindi, il numero di pixels uguali tra l'immagine obiettivo e quella ottenuta dalla simulazione CNN; cosí facendo a valori di fitness piú elevati vengono associati fenotipi corrispondenti a template che producono uscite molto simili all'immagine obiettivo.

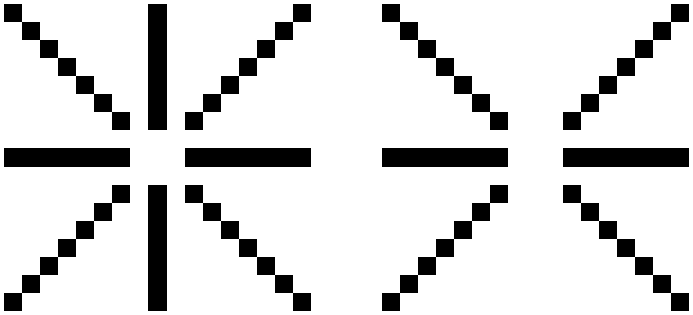
5.8.1 Template calcolati con i GA e confronti

Di seguito verranno proposti alcuni confronti tra alcuni template noti [22, 25] e gli equivalenti ricavati mediante la ricerca attraverso algoritmi genetici. In ciascuna prova effettuata, la CNN é stata fatta evolvere per determinare i pesi sinaptici che consentono di configurare la rete in maniera tale da svolgere la funzione corrispondente all'obiettivo specificato. Nelle prove effettuate, i template ricavati mediante GA si sono rivelati adatti a risolvere i task assegnati, offrendo in alcuni casi prestazioni migliori rispetto ai template noti, con benefici sia sulla velocitá di processamento sia sulla precisione nello svolgimento dell'operazione richiesta.

5.8.2 Rimozione linee verticali

Come esempio dell'impiego degli algoritmi genetici per la formulazione dei template di una CNN, si fa riferimento al problema della rimozione di linee verticali da una figura. In figura 5.12 è mostrata una figura di ingresso e l'obiettivo da raggiungere, ciò che si nota è che nella figura di destra mancano le linee verticali.

Il template impiegato noto in letteratura è il seguente :



Input Obiettivo

Figura 5.12: Rimozione linee verticali: Input e Obiettivo

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad z = -2 \tag{5.11}$$

Il tempo impiegata dalla CNN per la convergenza è di 102 passi di integrazioni pari a 1,02 tau. L'algoritmo genetico ci ha fornito circa 150 diverse soluzioni ottime. La migliore, ovvero la soluzione che risolve il problema assegnato ed impiega tempo minore, è la seguente:

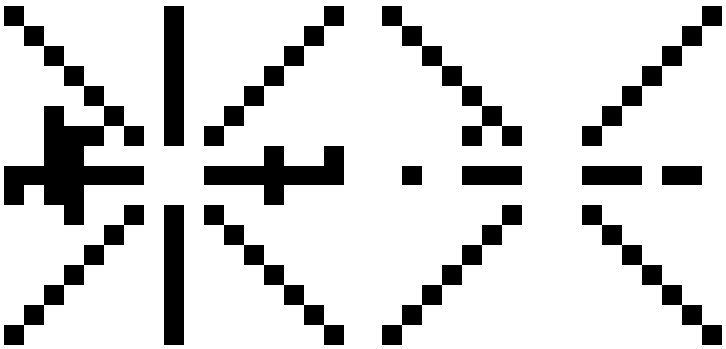
$$A = \begin{bmatrix} 0.730075181016035 & 0.756135286147711 & 0.697244198502621 \\ 0.924074035975651 & 0.823755087947359 & 0.924074035975651 \\ 0.697244198502621 & 0.756135286147711 & 0.730075181016035 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.75278899877724 & 1.83381178548271 & 0.43841731213658 \\ -5.50307865976499 & 8.84152175804671 & -5.50307865976499 \\ 0.43841731213658 & 1.83381178548271 & 0.75278899877724 \end{bmatrix} \tag{5.12}$$

$$z = -\,3.94059060930302$$

Il tempo impiegato per ottenere il risultato mostrato in figura 5.12 con il templa-

te (5.12) è 87 passi di interazione pari a $0.87tau$ Si ottiene quindi un miglioramento del 15% circa. Il template trovato si rivela anche robusto alle modifiche del dato iniziale, nella figura 5.13 è mostrato un esempio con una immagine di input differente dalla figura impiegata nell'addestramento.Ciò che possiamo osservare è la corretta rimozione delle righe verticali.



Input Obiettivo

Figura 5.13: Rimozione linee verticali: Input e Obiettivo

5.9 CNN introducendo un memristor

Come estensione al modello generico di CNN si è pensato di introdurre un elemento con le caratteristiche di un memristor. A differenza di altri approcci che impiegano i memristor come elemento di connessione le celle [45], nel caso da me affrontato il memristor fa parte della cella stessa. I *memristor*, contrazione del termine *memory resistor* (resistore con memoria) sono un nuovo elemento circuitale, la cui esistenza é stata ipotizzata da Leon Chua nel 1971 [21]. Il ragionamento seguito da Chua, riguarda le simmetrie tra le quattro grandezze fisiche circuitali fondamentali: la differenza di potenziale indicata con V, la carica elettrica, indicata con q, il flusso del campo magnetico indicato con φ e l'intensitá di corrente, indicata con i. I tre elementi circuitali di base, ovvero Resistore, Condensatore e Induttore, composti da due terminali, possono essere descritti mediante relazioni tra due delle quattro grandezze fondamentali. Delle 6 possibili combinazioni 5 sono

ben conosciute: Le prime due sono $q(t) = \int_{-\infty}^{t} i(\tau)d\tau$ e $\varphi(t) = \int_{-\infty}^{t} v(\tau)d\tau$. Le altre tre derivano proprio dalla descrizione degli elementi circuitali di base, in particolare abbiamo il resistore lineare

$$v(t) = Ri(t) \tag{5.13}$$

dove R, la resistenza, é costante nel tempo. La funzione caratteristica $i - v$ é una retta.

Il condensatore lineare

$$q(t) = Cv(t) \tag{5.14}$$

dove C, la capacità, é costante nel tempo. La funzione caratteristica $q - v$ é una retta. Infine abbiamo l'induttore che ha la seguente caratteristica

$$\varphi(t) = Li(t) \tag{5.15}$$

dove L, l'induttanza, é costante nel tempo. La funzione caratteristica $\varphi - i$ é sempre una retta. A questo elenco manca, evidentemente, un elemento che metta in relazione $\varphi - q$. Chua afferma che da un punto di vista sia logico che assiomatico, vista anche la simmetria osservata tra le relazioni che legano le grandezze, deve esistere un quarto elemento che sia caratterizzato da una curva $\varphi - q$. Questo elemento viene da egli ipotizzato e modellato e prende il nome di memristor poichè esso si comporta come una resistenza non lineare con memoria. Per definizione un memristor è caratterizzata da un rapporto del tipo $g(\varphi, q) = 0$. Si tratta, quindi, di un dispotivo con due terminali in cui la variazione di carica elettrica dà luogo ad una variazione di flusso magnetico e quindi ad una tensione. Il rapporto tra il potenziale prodotto e la corrente prende il nome di memristenza. È chiaro che, affinché la corrente che scorre nel bipolo venga a coincidere con quella non stazionaria che innesca il fenomeno, all'interno del memristor il flusso fisico delle cariche deve essere in qualche modo inibito, esattamente come avviene in un condensatore. Un memristor può essere controllato in carica o controllato in flusso. La tensione ai

capi di un memristor controllato in carica è data dalla seguente relazione:

$$v(t) = M(q(t))i(t) \tag{5.16}$$

dove

$$M(q) = \frac{d\varphi(q)}{dq} \tag{5.17}$$

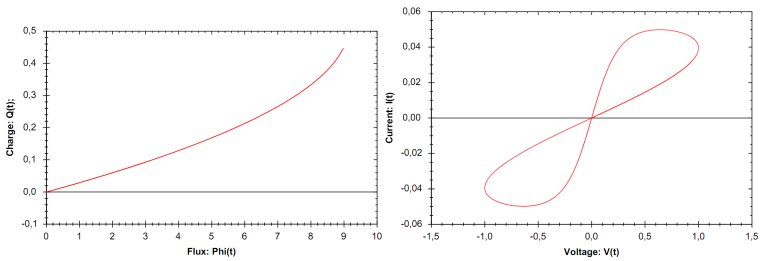
Analogamente, la corrente di un memristor controllato in flusso è data dalla relazione

$$i(t) = W(\varphi(t))v(t) \tag{5.18}$$

dove

$$W(\varphi) = \frac{dq(\varphi)}{d\varphi} \tag{5.19}$$

$M(q)$ ha la stessa unità di misura di una resistenza ed è detta *memristenza incremetale* mentre $W(\varphi)$ ha la stessa unità di misura di una conduttanza ed è detta *meminduttanza incrementale*. Se applichiamo una tensione del tipo $v(t) = v_0 * cos(\omega * t)$ le caratteristiche $\varphi - q$ e $v - i$ sono del tipo mostrato in figura 5.14a e 5.14b



$\varphi - q$ $v - i$

Figura 5.14: Curve caratteristiche del memristor ideale.

Il memristor resta un elemento puramente teorico sino al 2008 quando R. Stanley Williams e il suo team riescono a realizzarne uno nei laboratori HP [79]. Questa scoperta apre nuovi orizzonti nell'elettronica poichè il memristor ha la proprietà di ricordare lo stato elettronico e di rappresentarlo mediante segnali analogici. Un

circuito di questo tipo consentirebbe di realizzare calcolatori con accensione istantanea, senza la necessità di ricaricare il sistema operativo a ogni avvio. Il circuito, infatti, conserva l'informazione anche in assenza di corrente elettrica, quando il calcolatore è spento. Inoltre con questa scoperta si chiude il quadro simmetrico ipotizzato da Chua nel 1971 5.15.

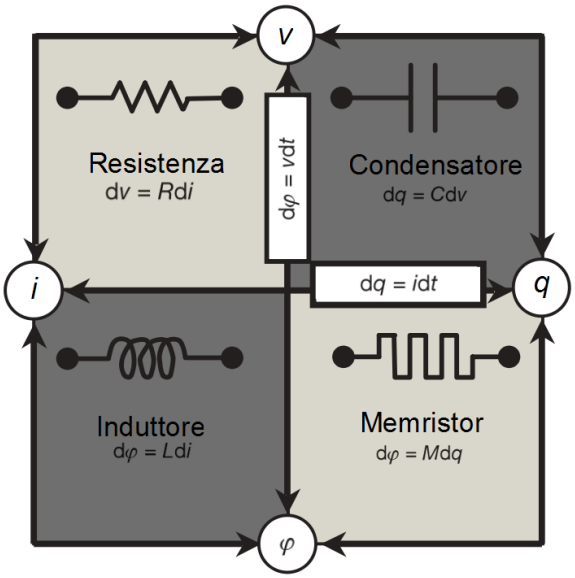


Figura 5.15: I quattro principali elementi circuitali a due terminali: resistenza,condensatore, induttore e memristor.

Con la disponibilità, teorica, di questa nuova tipologia di elementi circuitali si è pensato di estendere il modello tradizionale di CNN introducendo un memristor. La (5.1) diventa :

$$\begin{cases} \dot{x_{ij}} = -x_{ij} + \sum_{c(k,l)\in S_r(i,j)} A(i,j;k,l)y_{k,l} + \sum_{c(k,l)\in S_r(i,j)} B(i,j;k,l)u_{k,l} + z_{ij} - W(\varphi_{ij})x_{ij} \\ \dot{\varphi_{ij}} = x_{ij} \end{cases} \tag{5.20}$$

Dove gli elementi φ_{ij} appartengono ad una nuova matrice Φ MxN detta *Matrice di flusso* e $W(\cdot)$ è una funzione lineare a tratti definita come:

$$W(\varphi_{ij}(t)) = \frac{1}{2}[|\varphi_{ij}(t)+1| - |\varphi_{ij}(t)-1|] \qquad 1 \leq i \leq M, \quad 1 \leq j \leq N \tag{5.21}$$

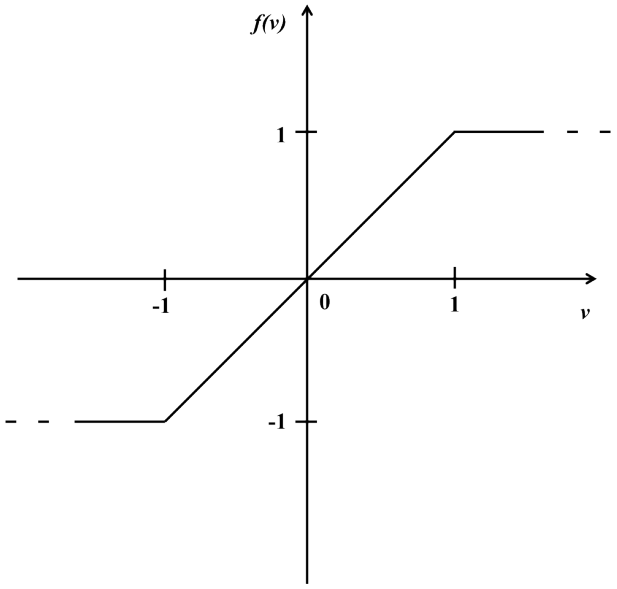


Figura 5.16: Andamento della funzione $W(\varphi_{ij}(t))$

L'espressione dell'uscita (5.2) resta invariata. A questo punto possiamo affermare di aver realizzato una CNN con *Memoria* che chiameremo *memCNN*. Ottenuto questo nuovo modello di CNN è necessario verificarne le caratteristiche e le performane. A tal fine sono stati condotti alcuni esperimenti che hanno restituito alcune informazioni interessanti. Le memCNN si sono rivelate più performanti del $20\% - 30\%$ nel compiere i task assegnati rispetto alle CNN tradizionali.

5.10 Esperimenti con le memCNN

Il primo esperimento riguarda l'estrazione dei contorni. Nell'esperimento condotto con le CNN standard riportato in 5.3, impiegando il template 5.4, il tempo

di convergenza della CNN era di $1,02tau$. Impiegando la memCNN si ottiene lo stesso risultato solo dopo $0,71tau$. In figura è riportato l'andamento dell'input e della matrice di flusso in una memCNN configurata per lo svolgimento del compito appena descritto.

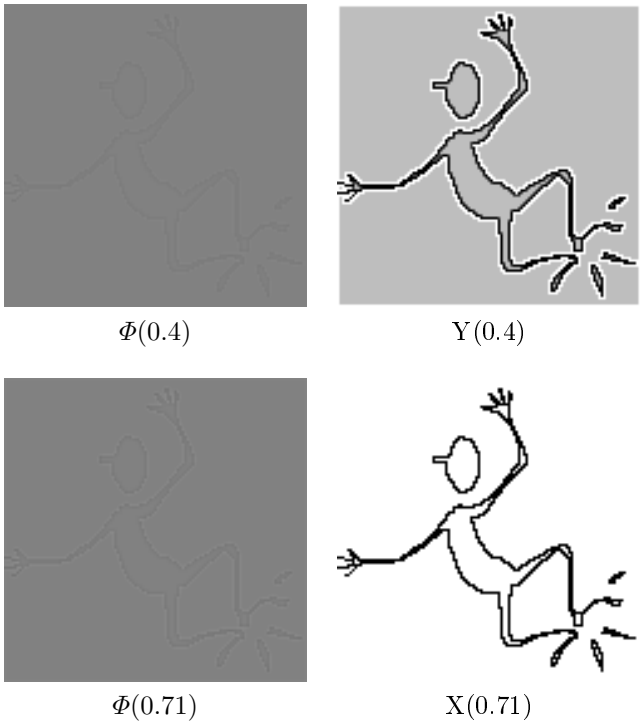


Figura 5.17: Edge Detection: Transizione dallo stato iniziale alla convergenza in una memCNN

Il secondo esperimento riguarda la risoluzione del problema della rimozione delle linee verticali. Per verificare il comportamento della memCNN è stato impiegato il template (5.6), ricavato attraverso l'addestramento con algoritmi genetici. Il risultato mostrato in figura 5.18 si ottiene dopo solo 74 passi di integrazione contro gli 87 necessari con l'impiego di CNN standard.

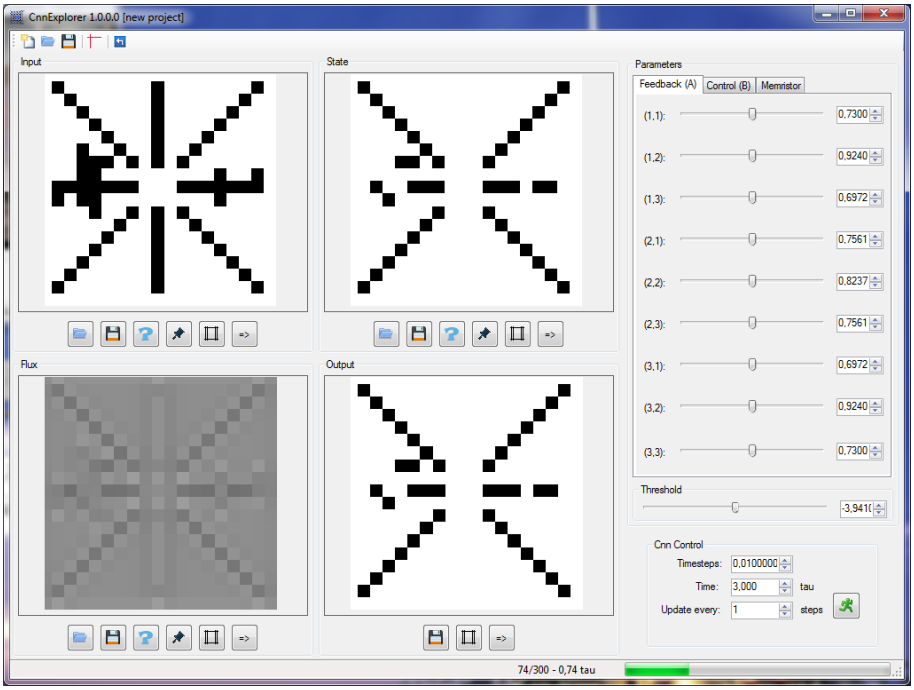


Figura 5.18: Rimozione delle linee verticali con una memCNN

Capitolo 6

CNN tridimensionali

6.1 Introduzione

Come visto in precedenza estendere il comportamento di una CNN con numero di dimensioni diverse dal modello standard è semplice. Andiamo, quindi, a considerare il caso delle CNN-3D prendendo in esame una cella isolata. Le variabili che caratterizzano tale cella possono essere riscritte come segue:

- il vettore ingresso controllabile $u_{i,j,k}(t)$;
- il vettore stato $x_{i,j,k}(t)$;
- il vettore output $y_{i,j,k}(t)$;
- la soglia z è assunta controllabile;

dove i, j, k rappresentano le coordinate spaziali. Indichiamo inoltre con X, U, Y gli insiemi che descrivono rispettivamente lo stato, l'ingresso e l'uscita (riferendoci a tutte le celle della CNN).

La sfera di interesse della cella $C(i,j,k)$ (o vicinato) è definita come :

$$S_r(i,j,k) = \{C(\alpha,\beta,\gamma)|max(|\alpha - i|, |\beta - j|, |\gamma - k|) \leq r\} \tag{6.1}$$

e indichiamo con

$$X_{i,j,k} = \bigcup_{C(\alpha,\beta,\gamma)\in S_r(i,j,k)} x_{\alpha,\beta,\gamma} \tag{6.2}$$

lo stato variabile delle celle nella sfera di interesse $S_r(i,j,k)$ della cella $C(i,j,k)$. L'insieme delle celle che compongono l'ingresso fisso della CNN e l'insieme di output variabile posso essere descritti analogamente all'insieme dello stato X. Per le CNN-3D verranno introdotte alcune caratteristiche che generalizzano la definizione standard, in particolare :

- le celle non sono necessariamente tutte uguali tra loro;

- in una griglia 3D la legge di accoppiamento è descritta localmente all'interno della sfera di interesse $S_r(i,j,k)$;

- inserimento del concetto di nodo. Un nodo può essere realizzato impiegando n celle del primo ordine generalizzate, andando a formare una piccola CNN multi strato.

6.2 Modello

Sulla base delle osservazioni appena fatte possiamo andare a definire un modello generale di CNN-3D come segue [36, 10, 9]:

$$\begin{aligned}
\dot{x}_{ijk} = & - x_{ijk} + z_{ijk} \\
& + \sum_{C(\alpha,\beta,\gamma)\in S_r(i,j,k)} A(i,j,k;\alpha,\beta,\gamma) y_{\alpha,\beta,\gamma} \\
& + \sum_{C(\alpha,\beta,\gamma)\in S_r(i,j,k)} B(i,j,k;\alpha,\beta,\gamma) u_{\alpha,\beta,\gamma} \\
& + \sum_{C(\alpha,\beta,\gamma)\in S_r(i,j,k)} C(i,j,k;\alpha,\beta,\gamma) y_{\alpha,\beta,\gamma} \\
& + A^{lr}_{i,j,k,\alpha,\beta,\gamma}(y_{\alpha,\beta,\gamma}) \\
& + B^{lr}_{i,j,k,\alpha,\beta,\gamma}(u_{\alpha,\beta,\gamma}) \\
& + C^{lr}_{i,j,k,\alpha,\beta,\gamma}(x_{\alpha,\beta,\gamma})
\end{aligned} \tag{6.3}$$

$$y_{ijk} = \frac{1}{2}(|x_{ijk}+1| - |x_{ijk}-1|)$$

dove $A(i,j,k;\alpha,\beta,\gamma)$, $B(i,j,k;\alpha,\beta,\gamma)$ e $C(i,j,k;\alpha,\beta,\gamma)$ sono rispettivamente il template di feedback, di controllo e di stato della CNN. Insieme i tre template regolano l'accoppiamento locale, mentre $A^{lr}_{i,j,k,\alpha,\beta,\gamma}$, $B^{lr}_{i,j,k,\alpha,\beta,\gamma}$ e $C^{lr}_{i,j,k,\alpha,\beta,\gamma}$ sono matrici $N \times N \times N$ le quali mappano le connessioni a lungo termine. Nel paradigma delle CNN-3D i contributi locali provenienti delle diverse celle caratterizzano il comportamento globale dell'architettura sia per quanto riguarda la struttura, sia per quanto concerne il comportamento dinamico emergente globale. E' importante sottolineare che vista la natura delle CNN non è possibile percepire o predire la dinamica complessiva analizzando semplicemente la dinamica interna dell'evoluzione di ogni singola cella o di un gruppo di esse, bensì è necessario considerare lo stato complessivo della rete. Tuttavia le dinamiche globali interne potrebbero essere percepite affiancando, all'analisi CNN-3D, un raffinato metodo di visualizzazione

delle forme che dalla CNN potrebbero emergere[36]. Lo studio delle dinamiche delle CNN-3D è stato condotto impiegando un particolare modello sul quale sono stati disegnati una serie di esperimenti.

6.3 Esperimenti

Per la descrizione degli esperimente introduciamo un modello molto generico che descrive la (6.3)

$$\begin{aligned} \dot{x_{ijk}} =& f_{ijk}(x_{ijk}, y_{ijk}, u_{ijk}) \\ &+ a_{ijk}(Y_{ijk}) \\ &+ b_{ijk}(U_{ijk}) \\ &+ c_{ijk}(X_{ijk}) \\ &+ a^{lr}_{ijk}(Y_{ijk}) \\ &+ b^{lr}_{ijk}(U_{ijk}) \\ &+ c^{lr}_{ijk}(X_{ijk}) \\ y_{ijk} =& g_{ijk}(x_{ijk}, y_{ijk}, u_{ijk}) \end{aligned} \tag{6.4}$$

dove, ovviamente gli a_{ijk}, b_{ijk} e c_{ijk} rappresentano i termini di accoppiamento locale e gli a^{lr}_{ijk}, b^{lr}_{ijk}, c^{lr}_{ijk} rappresentano i termini di accoppiamento a lungo raggio. Consideriamo quindi una semplice legge di accoppiamento locale che dipende solo dallo stato della cella definita come

$$c_{ijk}(X_{ijk}) = D\nabla^2_{ijk}x \tag{6.5}$$

il Laplaciano, in uno spazio 3D, è dato dalla seguente relazione :

$$D\nabla^2_{ijk}x = x_{i-1,jk} + x_{i+1,jk} + x_{ij-1k} + x_{ij+1k} + x_{ijk-1} + x_{ijk+1} - 6x_{ijk} \tag{6.6}$$

Se consideriamo uguali tutte le celle della CNN, ovvero con $f_{ijk}(x_{ijk}, y_{ijk}, u_{ijk}) = f(x_{ijk})$ ed escludiamo l'accoppiamento a lungo raggio allora possiamo riscrivere la (6.4) semplicemente come:

$$\dot{x_{ijk}} = f(x_{ijk}) + D(x_{i-1,jk} + x_{i+1,jk} + x_{ij-1k} + x_{ij+1k} + x_{ijk-1} + x_{ijk+1} - 6x_{ijk}) \quad (6.7)$$

che ricorda il paradigma delle equazioni di *reaction-diffusion*[**?**, 36]:

$$\dot{x} = f(x) + D\nabla^2 x.$$

Gli esperimenti che seguono sono stati eseguiti considerando diverse leggi dinamiche, riadattate, per descrivere lo stato delle singole celle della CNN. Il comportamento delle celle risulterà caotico. La legge di accoppiamento è caratterizzata da un basso coefficiente di accoppiamento (diffusione debole). In tutti gli esperimenti, sono state scelte condizioni al contorno di flusso-nullo(zero-flux). Gli esperimenti che saranno dettagliati nel seguito, mostrano, nella loro evoluzione, la nascita di forme auto-organizzate. Le forme evolutive riportate nei vari grafici sono iso-surfaces ottenute nel tempo e nello spazio 3D definito dalle coordinate spaziali i,j,k. Per quanto riguarda la dinamica delle celle, rappresentata dalla $f(x_{ijk})$, sono state simulate diverse leggi caotiche. In particolare, vengono presi in esame il sistema di Lorenz [49], il sistema di Rossler [68] e il circuito di Chua [22]. Inoltre, è stata analizzata anche la dinamica caotica di diversi modelli di neurone al fine di emulare il comportamento globale delle reti neurali in uno spazio 3D. Le condizioni iniziali scelte nei vari esperimenti sono diverse e verranno discusse in dettaglio in occasione dell'illustazione delle esperienze. Tuttavia, l'idea che sottende le varie scelte è comune a tutti gli esempi di seguito riportati e verrà discussa brevemente ora. Le condizioni iniziali $x_0(i, j, k)$ sono state formulate partendo da particolari forme topologiche come nella articolo The CNN PARADIGM: Shapes and complexity [10] e da funzioni di diffusione più semplici. La scelta delle dimensioni della la matrice

che descrive lo spazio 3D all'interno del quale la CNN lavora è stata effettuata in modo sperimentale al fine di garantire risultati accettabile nella forma e tempi di simulazione contenuti.

6.3.1 Circuito di Chua

Nel primo esperimento la CNN-3D viene formulata affinchè sia in grado di simulare il circuito di Chua[22]. Nel caso di una CNN-3D costruita con circuiti di Chua la (6.4) può essere riscritta come segue:

$$\begin{cases} \dot{x_{ijk}} = \alpha(y_{ijk} - h(x_{ijk})) + D\nabla^2_{ijk}x \\ \dot{x_{ijk}} = x_{ijk} - y_{ijk} + z_{ijk} \\ \dot{z_{ijk}} = \beta y_{ijk} \end{cases} \tag{6.8}$$

dove

$$h(x) = \frac{1}{2}[(s_1+s_2)x + (s_0-s_1)(|x-B_1|-|B|) + (s_2-s_0)(|x-B_2|-|B_2|)] + \epsilon \tag{6.9}$$

Il termine diffusivo agisce esclusivamente sulla prima variabile di stato $x_{ijk}(t)$. Lo spazio di simulazione è formato da una matrice 3D composta da $80\times80\times80$ unità caotiche. Da questo ne deriva che $i, j, k \in [1, 80]$. I parametri che caratterizzano le singole celle della CNN sono stati scelti in modo da portare il circuito di chua nella regione di bi-instabilità.

$$\begin{aligned} \alpha &= 9, & s_0 &= -1.7, \\ \beta &= 30, & s_1 &= s_2 = -\frac{1}{7}, \\ B_1 &= -1, & \epsilon &= -\frac{1}{14}, \\ B_2 &= 1, & D &= 0.1. \end{aligned} \tag{6.10}$$

La visualizzazione del comportamento di tale configurazione è stata realizzata impiengando una isosurface con valore di soglia fissato per $x_ijk = 0.1$. Il comporta-

mento emergente mostra la formazione di forme e strutture che evolvono nel tempo. L'evoluzione del sistema, mostrata in figura 6.1 si riferisce alle seguenti condizioni iniziali:

$$\begin{aligned} z &= \gamma(i,j,k) \\ &= (\frac{d^2-1}{1+d^2} + \hat{i}\frac{2k}{1+d^2})^7 - (\frac{2i}{1+d^2} + \hat{i}\frac{2j}{1+d^2})^5 \\ \sigma(z) &= (4Re(z) - 1.25, Im(z), 4Re(z) + 1.25) \end{aligned} \tag{6.11}$$

dove $d = \sqrt{i^2+j^2+k^2}$ e $\hat{i} = \sqrt{-1}$

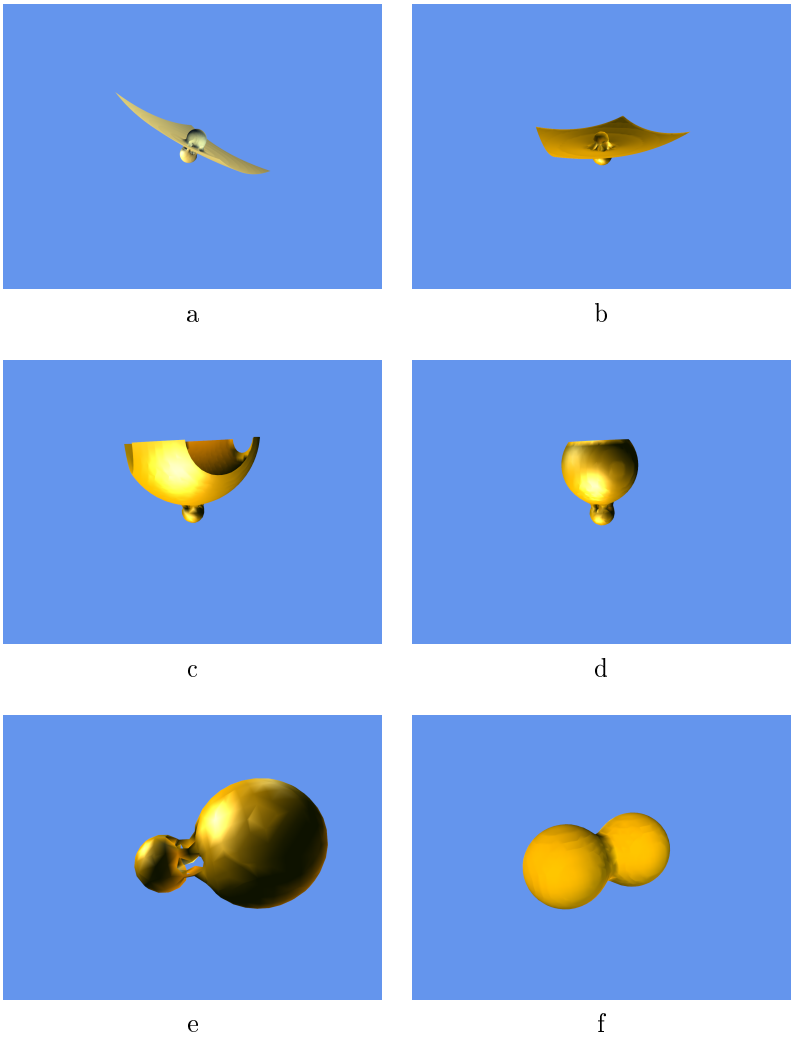


Figura 6.1: CNN-3D con celle che implementano il circuito di Chua

6.3.2 Sistema di Lorenz

Nel secondo esempio viene implementato il sistema di Lorenz. Lo spazio è dimensionato con una matrice composta da $60 \times 60 \times 60$ sistemi di Lorenz [68] descritti dalle seguenti equazioni:

$$\begin{cases} \dot{x_{ijk}} = \sigma(y_{ijk} - x_{ijk})) + D\nabla^2_{ijk}x \\ \dot{x_{ijk}} = rx_{ijk} - y_{ijk} + x_{ijk}z_{ijk} + D\nabla^2_{ijk}y \\ \dot{z_{ijk}} = x_{ijk}y_{ijk} - bz_{ijk} \end{cases} \tag{6.12}$$

I parametri scelti per le simulazioni sono i parametri che formano il famoso attrattore a farfalla :

$$\begin{cases} \sigma = 10 \\ r = 28 \\ b = \dfrac{8}{3} \\ D = 0.5 \end{cases}$$

Le condizioni iniziali scelte sono le seguenti :

$$\begin{aligned} z &= \gamma(i,j,k) \\ &= (\frac{d^2-1}{1+d^2} + \hat{i}\frac{2k}{1+d^2})^6 - (\frac{2i}{1+d^2} + \hat{i}\frac{2j}{1+d^2})^9 \\ \sigma(z) &= (4Re(z) - 1.25, Im(z), 4Re(z) + 1.25) \end{aligned} \tag{6.13}$$

dove $d = \sqrt{i^2 + j^2 + k^2}$ e $\hat{i} = \sqrt{-1}$. In questo caso il termine diffusivo agisce sia sulla variabile di stato x_{ijk} che sulla variabile di stato y_{ijk}. In figura 6.2 viene mostrato il comportamento della CNN-3D rappresentata impiegando un valore di isosurface $x_{ijk} = 2$. Anche in questo caso il sistema mostra la formazione e l'evoluzione di forme interessanti.

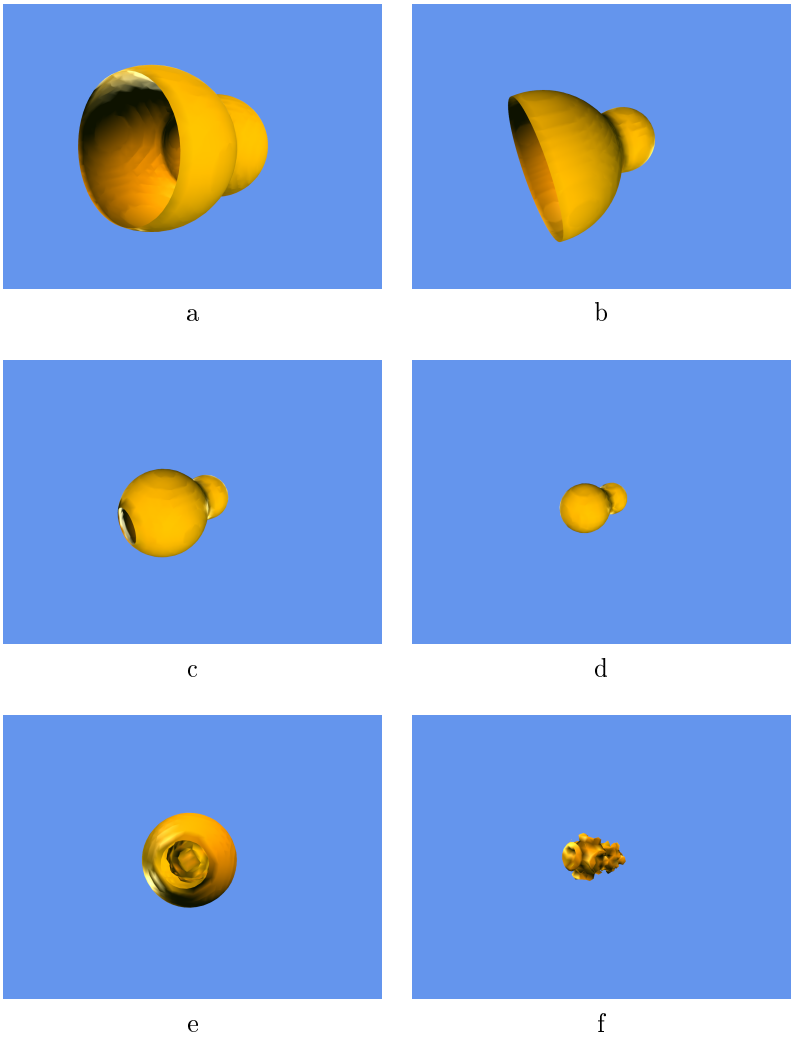


Figura 6.2: CNN-3D con celle che implementano il modello di Lorenz

6.3.3 Sistema di Rossler

L'emergenza di forme e strutture auto-organizzate sono state osservate anche in una CNN-3D realizzata con celle che implementano il sistema di Rossler configurato come segue [68] :

$$\begin{cases} \dot{x_{ijk}} = -y_{ijk} - z_{ijk} + D\nabla^2_{ijk}x \\ \dot{x_{ijk}} = x_{ijk} - ay_{ijk} \\ \dot{z_{ijk}} = b + x_{ijk}z_{ijk} - cz_{ijk} \end{cases} \tag{6.14}$$

La matrice in questo caso è composto da $60 \times 60 \times 60$ celle con i seguenti

parametri scelti per la simulazione:

$$\begin{cases} a = 0.2 \\ b = 0.2 \\ c = 5 \\ D = 0.1 \end{cases}$$

Il valore scelto come soglia per l'isosurface è $x_{ijk} = 0$ mentre le condizioni iniziali sono state impostate con la seguente relazione :

$$\begin{aligned} z &= \gamma(i,j,k) \\ &= (\frac{d^2-1}{1+d^2} + \hat{i}\frac{2k}{1+d^2}) * (\frac{2i}{1+d^2} + \hat{i}\frac{2j}{1+d^2}) \\ &+ (\frac{2i}{1+d^2} + \hat{i}\frac{2j}{1+d^2})^5 - (\frac{d^2-1}{1+d^2} + \hat{i}\frac{2k}{1+d^2})^3 \\ \sigma(z) &= (4Re(z) - 0.25, Im(z), 4Re(z) + 0.25) \end{aligned} \tag{6.15}$$

dove $d = \sqrt{i^2+j^2+k^2}$ e $\hat{i} = \sqrt{-1}$. Alcune parti dell'evoluzione sono mostrate in figura 6.3.

6.3.4 Modello Neuronale di FitzHugh–Nagumo

Il primo dei tre modelli di neurone, simulati con le CNN-3D, che incontriamo è il modello di FitzHugh–Nagumo [32, 51]. Tale modello descrive il comportamento degli spike-neurons come espresso nella seguente relazione:

$$\begin{aligned} \dot{v_{ijk}} &= \epsilon v_{ijk}(1 - v_{ijk})(v_{ijk} - \frac{u_{ijk} + b}{a}) + D\nabla^2_{ijk} v \\ \dot{u_{ijk}} &= v_{ijk} - u_{ijk} \end{aligned} \tag{6.16}$$

Anche in questo caso il termine diffusivo interviene solo su un stato. I parametri che regolano il modello durante la simulazione sono i seguenti :

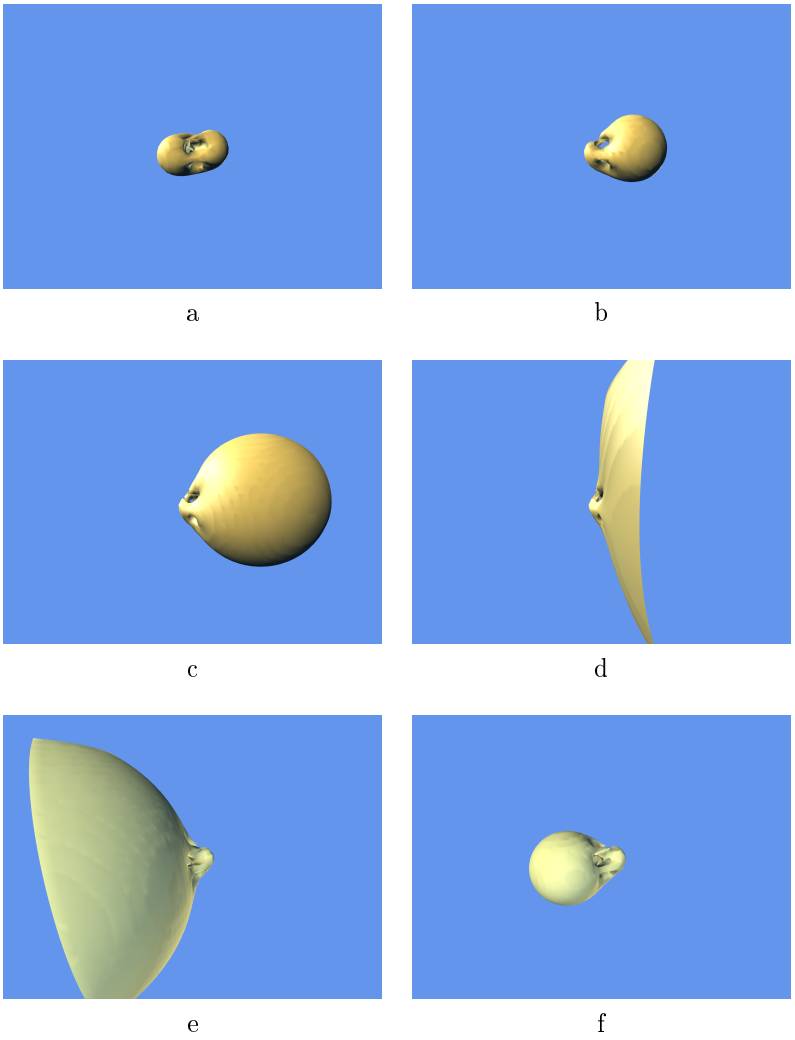


Figura 6.3: CNN-3D con celle che implementano il sistema di Rossler

$$\begin{cases} a = 0.75 \\ b = 0.01 \\ \epsilon = 50 \\ D = 1 \end{cases}$$

In questo caso il valore di soglia dell'isosurface è definita da $x_{ijk} = 0.5$ mentre le dimensioni della CNN-3D sono $50 \times 50 \times 50$

I dati iniziali per questa tipologia di simulazione sono stati impostati secondo la seguente formulazione:

$$v_{ijk} = \begin{cases} 0.0 & \forall i \in [1, 25] \\ 1.0 & \forall i \in [26, 50] \end{cases}$$

$$u_{ijk} = \begin{cases} 0.0 & \forall j, k \quad | \quad j^2 + k^2 <= (22)^2 \\ 0.375 & altrimenti \end{cases}$$

L'evoluzione della CNN-3D è riassunta in figura 6.4

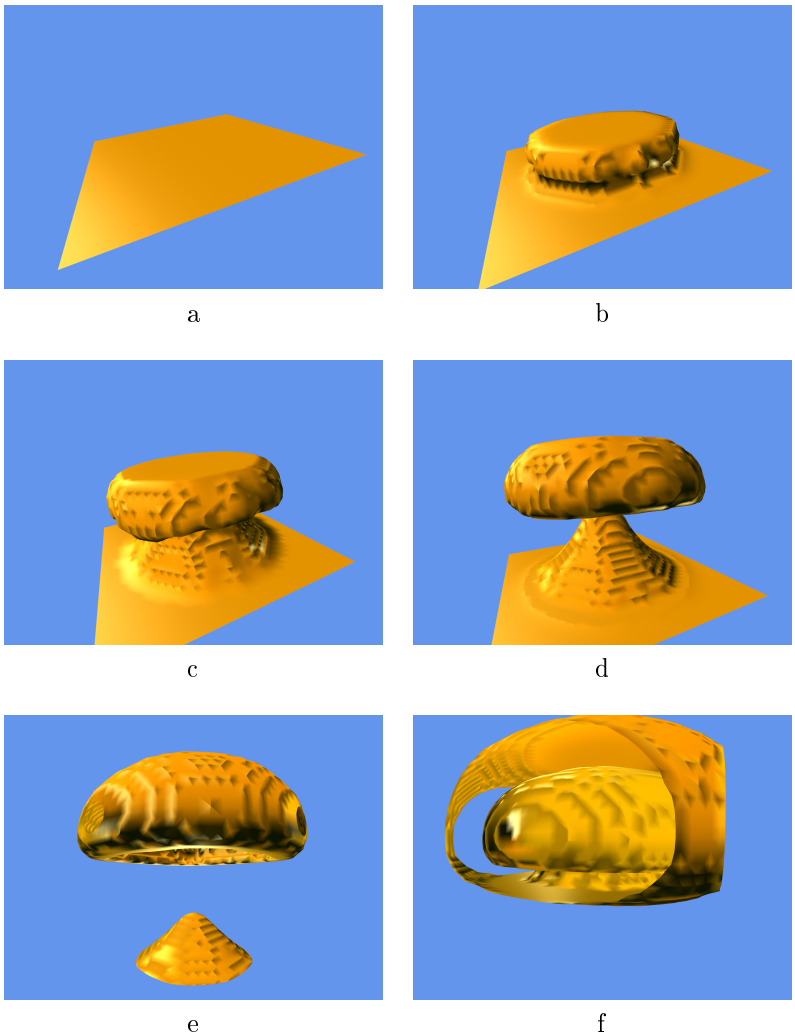


Figura 6.4: CNN-3D con celle che implementano il modello neuronale di FitzHugh–Nagumo

6.3.5 Modello Neuroni Inferior-Olive

Nel secondo esperimento dedicato ai modelli neuronali è stata implementata una CNN-3D $40 \times 40 \times 40$ con celle che evolvono secondo un modello che riproduce il comportamento dei neuroni Inferior-Olive (IO)[39]. I neuroni *IO* sono caratterizzati da oscillazioni sotto-soglia e il modello adimensionale che ne descrive la dinamica è il seguente:

$$\begin{cases} \dot{x_{ijk}} = \dfrac{x_{ijk}(x_{ijk} - \gamma)(1 - x_{ijk}) - y_{ijk}}{\epsilon} + D\nabla^2_{ijk}x \\ \dot{y_{ijk}} = -\Omega z_{ijk} + r_{ijk}(A - z^2_{ijk} - r^2_{ijk}) + D\nabla^2_{ijk}y \\ \dot{z_{ijk}} = \Omega r_{ijk}(A - z^2_{ijk} - r^2_{ijk}) + D\nabla^2_{ijk}z \end{cases} \tag{6.17}$$

dove $r_{ijk} = (y_{ijk}/M) - x_{ijk}$

I parametri scelti sono i seguenti:

$$\begin{aligned} \epsilon &= 0.01 \\ \gamma &= 0.2 \\ M &= 0.5 \\ A &= 0.0006 \\ \Omega &= -16 \\ D &= 0.001 \end{aligned}$$

In figura 6.5 sono stati riportati alcuni fotogrammi catturati durante la simulazione della CNN-3D.

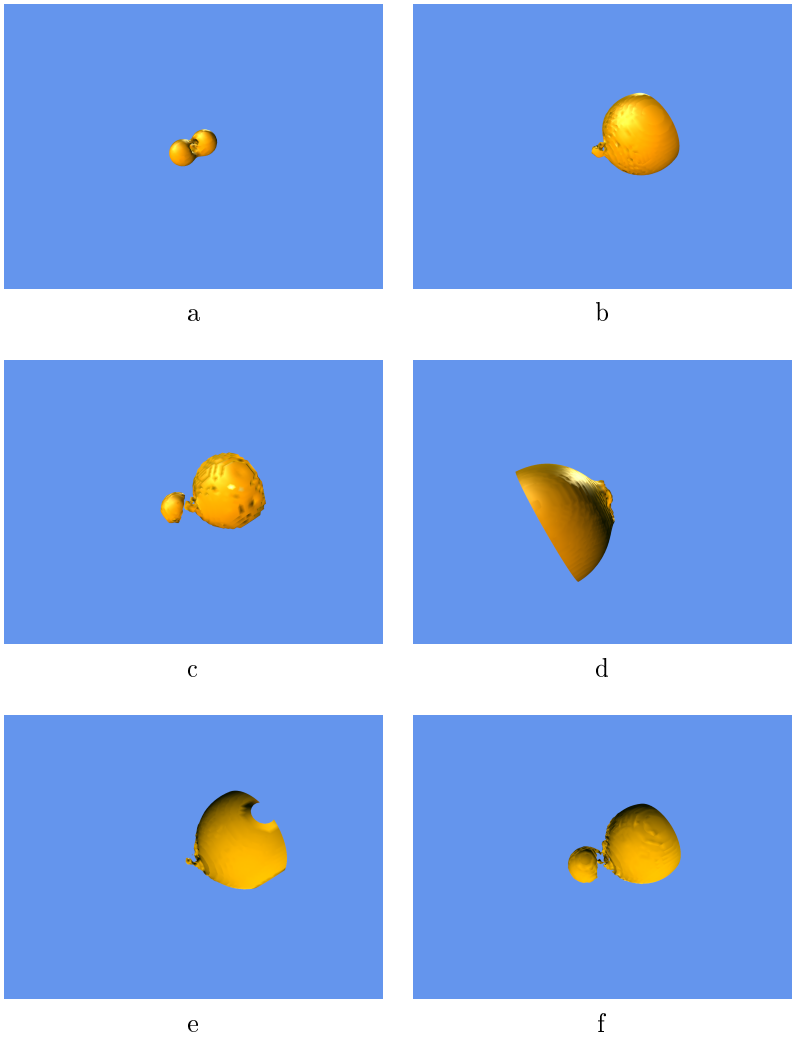


Figura 6.5: CNN-3D con celle che implementano il modello delle cellule Inferior-Olive

Le condizioni iniziali impiegate vengono riportate di seguito:

$$\begin{aligned} z &= \gamma(i,j,k) \\ &= (\frac{d^2-1}{1+d^2} + \hat{i}\frac{2k}{1+d^2})^{7-2\frac{d^2}{D^2}} \\ &- (\frac{2i}{1+d^2} + \hat{i}\frac{2j}{1+d^2})^{5+3\frac{d^2}{D^2}} \end{aligned} \tag{6.18}$$

$$\sigma(z) = (2Re(z) - 0.4\frac{d^2}{D^2}, Im(z) + 0.4, sin((Rez)^2 - (Imz)^2)$$

dove $d = \sqrt{i^2 + j^2 + k^2}$,$D = 1000$ e $\hat{i} = \sqrt{-1}$.

6.3.6 Modello Neuronale di Izhikevich

L'ultimo esperimento riguarda la formulazione di una CNN-3D con celle che implementano il modello neuronale di Izhikevich. Il modello in questione è abbastanza recente [42, 43] e si pone come obiettivo, coniugare accuratezza del modello e limitata quantità di risorse da impiegare nella simulazione. Esso è descritto dalle seguenti equazioni :

$$\begin{cases} \dot{v_{ijk}} = 0.04v_{ijk}^2 + 5v_{ijk} + 140 - u_{ijk} + I + D\nabla_{ijk}^2 v \\ \dot{u_{ijk}} = a(bv_{ijk} - u_{ijk}) \end{cases} \tag{6.19}$$

con una condizione di reset dei neuroni definita come:

$$se \quad v_{ijk} \geq 30, \qquad allora, \quad \begin{cases} v \leftarrow c \\ u \leftarrow d \end{cases}$$

I valori dei parametri, durante la simulazione, sono stati scelti affinchè si possa riprodurre l'attività caotica degli *spiking neurons* secondo i risultati verificati da Izhikevich:

$$a = 0.2$$

$$b = 2$$

$$c = 56$$

$$d = -16$$

$$I = -99$$

$$D = 0.01$$

Il valore di soglia dell'isosurface scelto è $v_{ijk} = -65.4$ mentre il numero di celle della CNN impiegato è di 30×30×30. I dati iniziali sono stati impostati secondo la seguente funzione:

$$v_{ijk} = \begin{cases} -56 \quad \text{se} \quad r < 5 \text{ e } i > 0 \\ 20 \quad \text{se} \quad r < 9 \text{ e } i < 0 \\ 0 \quad \text{altrimenti} \end{cases}$$

$$u_{ijk} = \begin{cases} -112 \quad \text{se} \quad r < 5 \text{ e } i > 0 \\ 40 \quad \text{se} \quad r < 9 \text{ e } i < 0 \\ 0 \quad \text{altrimenti} \end{cases}$$

Dove $r = \sqrt{i^2 + j^2 + k^2}$. Porzioni della simulazione sono mostrate in figura 6.6

a b

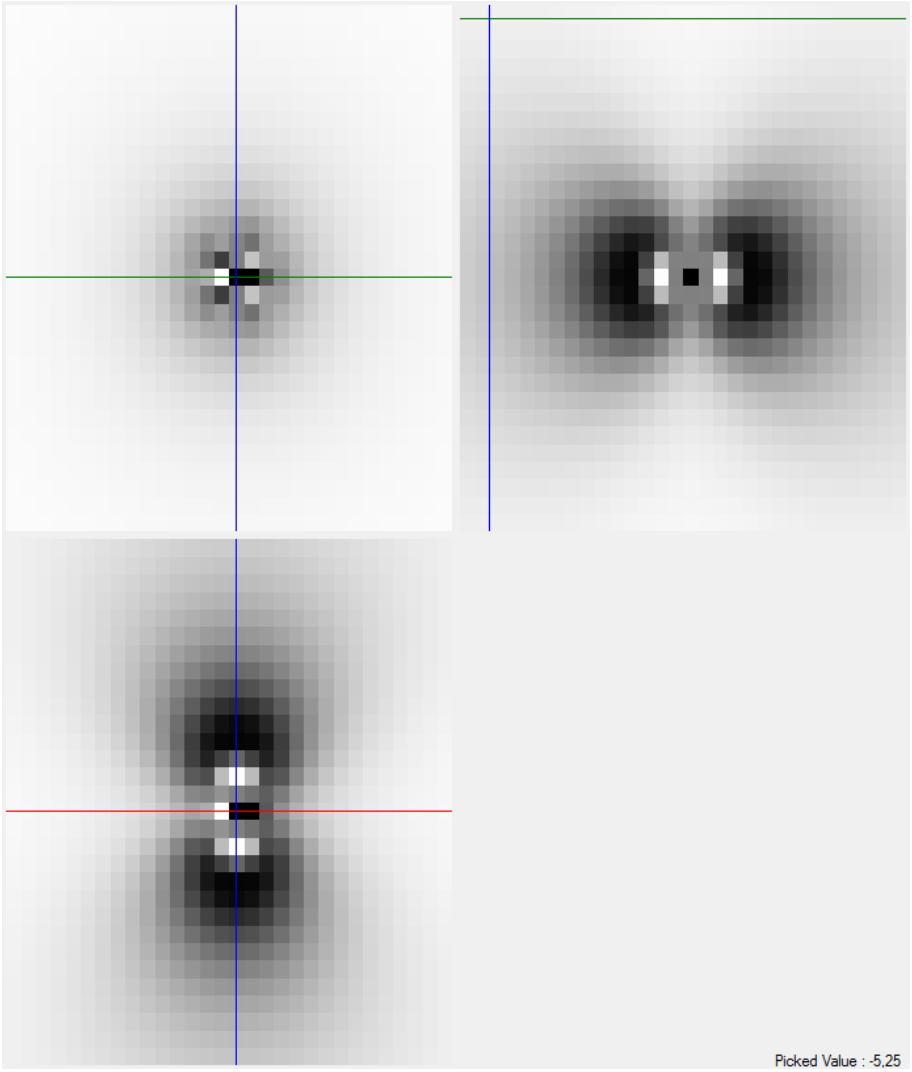
Figura 6.6: CNN-3D con celle che implementano il modello neuronale di Izhikevich

Il sistema a regime cicla tra due le configurazioni.

6.4 Visualizzazione avanzata

La visualizzazione mediante isosurface è utile quando si vuole tracciare il comportamento della CNN per specifici valori dello stato. Per osservare però, la dinamica globale di tutte le celle, tale tecnica di visualizzazione non è più sufficiente. Durante la fase sperimentale si pensato quindi di implementare un metodo alternativo di visualizzazione che permettesse appunto di valutare lo stato di tutte le celle. Per ottenere tale scopo si è introdotta una nuova tecnica di visualizzazione che suddivide la CNN in fette corrispondenti ai piani xy, yz, zx e mostrando le proiezioni contemporaneamente. E' possibile anche valutare il valore che la cella

assume in un dato punto del volume. Di seguito alcune immagini che mostrano gli esperimenti mostrati negli esempi precedenti visualizzati mediante questa seconda tecnica.



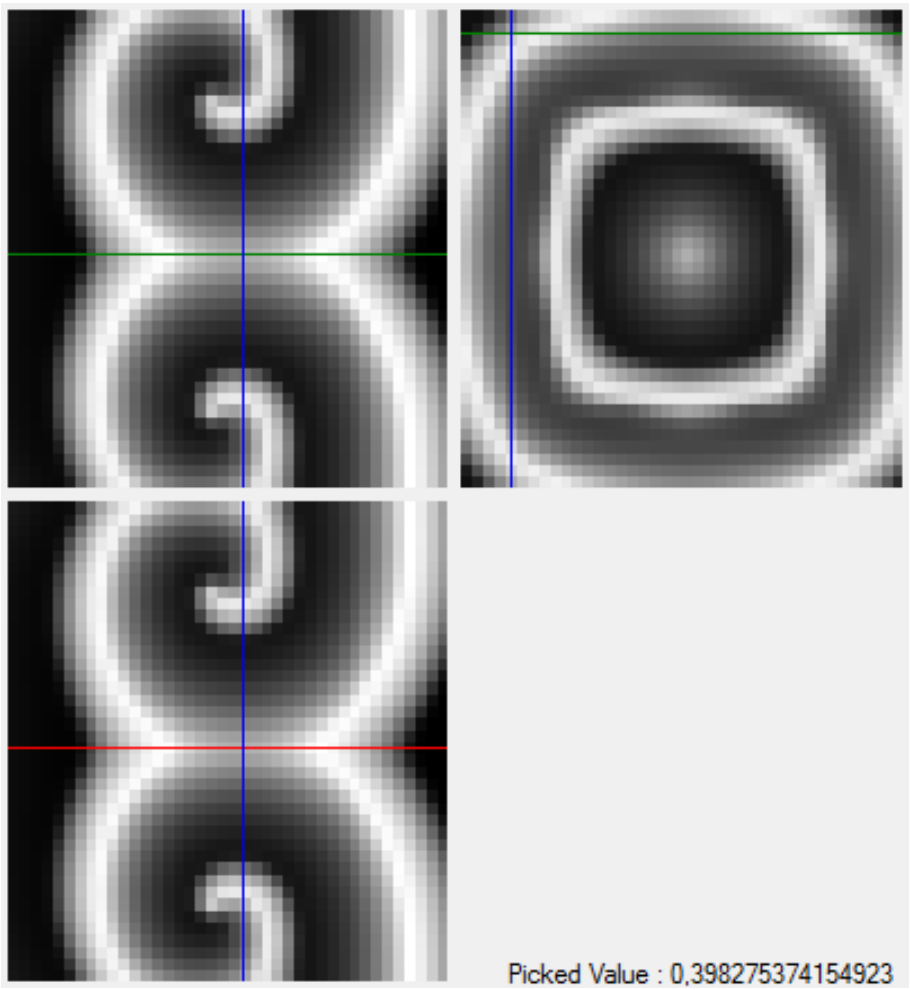
a

Figura 6.7: Visualizzazione con proiezione della CNN-3D che implementa il circuito di Chua.

In figura 6.7 viene mostrata la configurazione visualizzata nella figura 6.8f scomposta nelle tre proiezioni sagittale,assiale,coronale. Il dettaglio è sicuramente maggiore ed è possibile ispezionare il valore di ogni singola cella della CNN sia in maniera qualitativa che quantitativa.

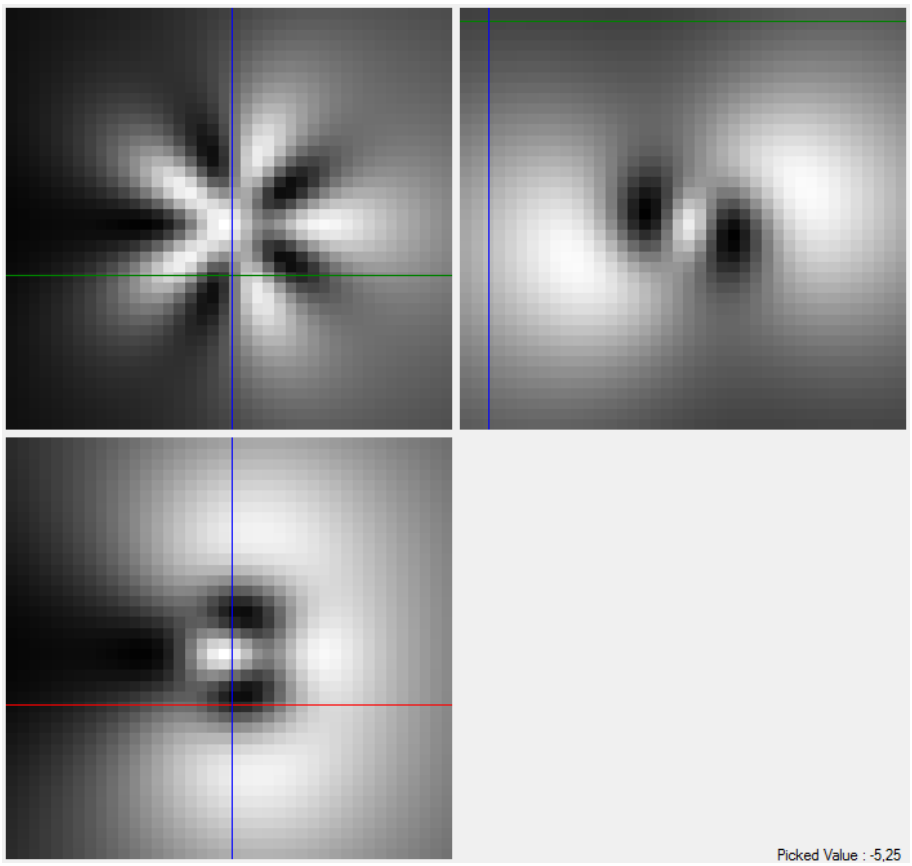
In figura 6.8 viene mostrata la configurazione visualizzata nella figura 6.4d. In figura 6.9 viene invece mostrata la proiezione della struttura visibile nella figura

6.14a, mentre nell'ultima figura (fig. 6.10) si riporta la proiezione della struttura emergente ritrovata precedentemente in figura 6.12d.



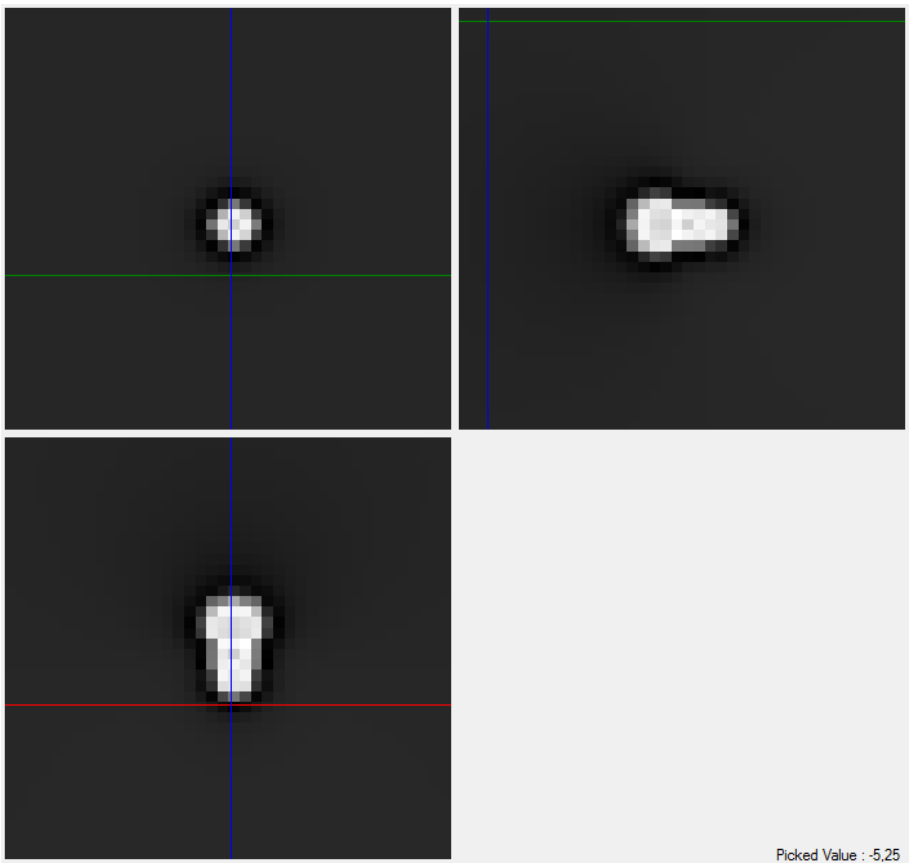
a

Figura 6.8: Visualizzazione con proiezione della CNN-3D che implementa il modello neuronale di FitzHugh–Nagumo



a

Figura 6.9: Visualizzazione con proiezione della CNN-3D che implementa il sistema di Rossler



a

Figura 6.10: Visualizzazione con proiezione della CNN-3D che implementa il sistema di Lorenz

Capitolo 7

Conclusioni

In questo lavoro di tesi sono stati affrontati numerosi aspetti relativi alle reti neurali cellulari. In particolare, è stata condotta una accurata analisi delle proprietà delle reti cellulari da un punto di vista teorico, soffermandosi sullo studio delle CNN nel contesto dei sistemi dinamici nonlineari. Un'ampia parte del lavoro svolto è stata dedicata alla realizzazione di una serie di simulatori software e di una libreria, che hanno permesso di realizzare analisi accurate delle dinamiche associate alle CNN. Le CNN si sono rivelate, dunque, un strumento molto efficace sia per lo sviluppo di applicazioni di image processing, che per lo studio di dinamiche caotiche di sistemi dinamici nonlineari.Interessanti risultati sono stati ottenuti in merito all'utilizzo degli algoritmi genetici per la ricerca di templates in CNN standard; le simulazioni, infatti, hanno dimostrato che, programmando le CNN con i parametri ottenuti attraverso l'evoluzione delle reti neurali cellulari attraverso l'uso di GA, si riescono a migliorare in maniera sensibile le performances della rete nell'esecuzione di task per l'elaborazione di immagini. Sebbene le simulazioni, atte ad analizzare il comportamento delle CNN in settori di image processing, siano risultate interessanti, risultati altrettanto notevoli si sono ottenuti nell'impiego delle CNN monodimensionali nella simulazione di linee di trasmissione non lineari. La dimostrazione che è possibile trasmettere onde solitoniche attraverso tale tipologia di CNN apre le porte a studi più approfonditi in un campo di applicazione che ri-

sulta essere ad oggi molto caldo. Inoltre l'implementazione degli incroci permette, se pure in forma qualitativa, un analisi del comportamento delle onde solitoniche in settings sperimentali nuovi. Infine la codifica di sistemi dinamici complessi e caotici in equazioni di stato di una CNN3D ha dimostrato quanto le CNN siano utili anche nell'arduo compito di interpretare il caos. Sebbene non sia stato, in molti casi, possibile approfondire gli argomenti trattati, la quantità di esperimenti condotti e il numero di confronti con i risultati presenti in letteratura hanno permesso di validare gli algoritmi sviluppati nei simulatori rendendoli, difatti, utili strumenti per le ricerche in questo settore.

Bibliografia

[1] Analogic computers ltd. `http:\www.analogic-computers.com`.

[2] Reza Abrishambaf, Hasan Demirel, and Izzet Kale. A fully CNN based fingerprint recognition system. *2008 11th International Workshop on Cellular Neural Networks and Their Applications*, (July):146–149, July 2008.

[3] A. Adamatzky, P. Arena, A. Basile, R. Carmona-Galán, B.D.L. Costello, L. Fortuna, M. Frasca, and A. Rodríguez-Vázquez. Reaction-diffusion navigation robot control: from chemical to vlsi analogic processors. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 51(5):926–938, 2004.

[4] P Arena. Image processing for medical diagnosis using CNN. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 497(1):174–178, January 2003.

[5] P. Arena, S. Baglio, L. Fortuna, and G. Manganaro. Complexity in a two-layer CNN. *1996 Fourth IEEE International Workshop on Cellular Neural Networks and their Applications Proceedings (CNNA-96)*, pages 127–132, 1996.

[6] P. Arena, A. Basile, L. Fortuna, M. Frasca, and L. Patane. Implementation of turing patterns for bio-inspired motion control. In *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, volume 3, pages III–842. IEEE, 2003.

[7] P. Arena and L. Fortuna. Analog cellular locomotion control of hexapod robots. *Control Systems Magazine, IEEE*, 22(6):21–36, 2002.

[8] P. Arena, L. Fortuna, and M. Branciforte. Reaction-diffusion cnn algorithms to generate and control artificial locomotion. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 46(2):253–260, 1999.

[9] P Arena, L Fortuna, D Lombardo, and L Patan. CNN and collective perception. *Neural Networks*, (July):14–16, 2008.

[10] Paolo Arena, Maide Bucolo, Stefano Fazzino, Luigi Fortuna, and Mattia Frasca. THE CNN PARADIGM : SHAPES AND COMPLEXITY. *International Journal of Bifurcation and Chaos*, 15(7):2063–2090, 2005.

[11] E. Arslan, Z. Orman, and S. Arik. Detection of objects in moving images and implementation of the purification algorithm on analog cnn and dsp processors. In *Proceedings of the 12th WSEAS international conference on Neural networks, fuzzy systems, evolutionary computing & automation*, pages 60–65. World Scientific and Engineering Academy and Society (WSEAS), 2011.

[12] O. Bandman. Cellular-neural automaton: a hybrid model for reaction–diffusion simulation. *Future Generation Computer Systems*, 18(6):737–745, 2002.

[13] E. Bilotta, A. Cerasa, P. Pantano, A. Quattrone, A. Staino, and F. Stramandinoli. A cnn based algorithm for the automated segmentation of multiple sclerosis lesions. *Applications of Evolutionary Computation*, pages 211–220, 2010.

[14] E. Bilotta, A. Cerasa, P. Pantano, A. Quattrone, A. Staino, and F. Stramandinoli. Evolving cellular neural networks for the automated segmentation of

multiple sclerosis lesions. *Variants of Evolutionary Algorithms for Real-World Applications*, 7(11):377, 2011.

[15] E. Bilotta, A. Lafusa, and P. Pantano. Is self-replication an embedded characteristic of the artificial/living matter. *Artificial Life*, 8:38–48, 2003.

[16] Eleonora Bilotta, G Cutri, and P Pantano. Evolving Robot's Behavior by Using CNNs. *From Animals to Animats 9*, 2006.

[17] Eleonora Bilotta and Pietro Pantano. Cellular Non-linear Networks as a New Paradigm for Evolutionary Robotics. *Neural Networks*, (April), 2008.

[18] M Bucolo, R Caponetto, L Fortuna, and M Frasca. *The CNN Paradigm for Complexity*, volume 63, pages 1–37. World Scientific Pub Co Inc, 2008.

[19] A.W. Burks and MICHIGAN UNIV ANN ARBOR LOGIC OF COMPUTERS GROUP. *VON NEUMANN'S SELF-REPRODUCING AUTOMATA*. University of Michigan, 1969.

[20] B. Chopard, M. Droz, and Cambridge University Press. *Cellular automata modeling of physical systems*. Cambridge University Press Cambridge^ eMA MA, 1998.

[21] L. Chua. Memristor-the missing circuit element. *Circuit Theory, IEEE Transactions on*, 18(5):507–519, 1971.

[22] L.O. Chua. *CNN: A paradigm for complexity*, volume 31. World Scientific Pub Co Inc, 1998.

[23] L.O. Chua, C.A. Desoer, and E.S. Kuh. *Linear and nonlinear circuits*. McGraw-Hill New York, 1987.

[24] L.O. Chua and RN Madan. Sights and sounds of chaos. *IEEE Circuits and Devices magazine*, 4(1), 1988.

[25] L.O. Chua and T. Roska. *Cellular neural networks and visual computing: foundation and applications*. Cambridge Univ Pr, 2002.

[26] L.O. Chua and L. Yang. Cellular neural networks: applications. *IEEE Transactions on Circuits and Systems*, 35(10):1273–1290, 1988.

[27] L.O. Chua and L. Yang. Cellular neural networks: theory. *IEEE Transactions on Circuits and Systems*, 35(10):1257–1272, 1988.

[28] Santiago De Compostela. Cellular neural networks for NP-hard optimization. *Neural Networks*, (July):14–16, 2008.

[29] Edward Curry and Paul Grace. Flexible self-management using the model-view-controller pattern. *IEEE Software*, 25:84–90, 2008.

[30] L. Davis and M. Mitchell. *Handbook of genetic algorithms*. Van Nostrand Reinhold, 1991.

[31] B. Drossel and F. Schwabl. Self-organized critical forest-fire model. *Phys. Rev. Lett.*, 69:1629–1632, Sep 1992.

[32] R. Fitzhugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1(6):445–466, 1961.

[33] Jacek Flak, Mika Laiho, and Ari Paasio. Scalable fault-tolerant logic system based on regular array of locally interconnected gates. In *Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on*, number July, pages 116–119. IEEE, 2008.

[34] A. Ford and A. Roberts. Colour space conversions. *Retrieved January*, 31:2008, 1998.

[35] L. Fortuna, P. Arena, D. Balya, and a. Zarandy. Cellular neural networks: a paradigm for nonlinear spatio-temporal processing. *IEEE Circuits and Systems Magazine*, 1(4):6–21, 2001.

[36] L. Fortuna, A. Rizzo, and MG Xibilia. Modeling complex dynamics via extended pwl-based cnns. *INTERNATIONAL JOURNAL OF BIFURCATION AND CHAOS IN APPLIED SCIENCES AND ENGINEERING*, 13(11):3273–3286, 2003.

[37] N. Fressengeas and H. Frezza-Buet. Generic method for solving partial differential equations through the design of problem-specific cellular neural networks. 2006.

[38] A. Gacsadi and P. Szolgay. An analogic cnn algorithm for following continuously moving objects. In *Cellular Neural Networks and Their Applications, 2000.(CNNA 2000). Proceedings of the 2000 6th IEEE International Workshop on*, pages 99–104. IEEE, 2000.

[39] A. Giaquinta, M. Argentina, and MG Velarde. A simple generalized excitability model mimicking salient features of neuron dynamics. *Journal of Statistical Physics*, 101(1):665–678, 2000.

[40] K. Hadad and A. Piroozmand. Application of cellular neural network (cnn) method to the nuclear reactor dynamics equations. *Annals of Nuclear Energy*, 34(5):406–416, 2007.

[41] J.H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[42] E.M. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569–1572, 2003.

[43] E.M. Izhikevich. Which model to use for cortical spiking neurons? *Neural Networks, IEEE Transactions on*, 15(5):1063–1070, 2004.

[44] T. Kozek, L.O. Chua, T. Roska, D. Wolf, R. Tetzlaff, F. Puffer, and K. Lotz. Simulating nonlinear waves and partial differential equations via cnn. ii. typi-

cal examples. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 42(10):816–820, 1995.

[45] E. Lehtonen and M. Laiho. Cnn using memristors for neighborhood connections. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pages 1–4. IEEE, 2010.

[46] J. Liberty and V. Quercia. *Programming c#*. O'Reilly & Associates, Inc., 2003.

[47] A. Loncar, R. Kunz, and R. Tetzlaff. Scnn 2000. i. basic structure and features of the simulation system for cellular neural networks. In *Cellular Neural Networks and Their Applications, 2000.(CNNA 2000). Proceedings of the 2000 6th IEEE International Workshop on*, pages 123–128. IEEE, 2000.

[48] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

[49] E.N. Lorenz. Deterministic nonperiodic flowl. 1963.

[50] Microsoft. .net framework website. `http://www.microsoft.com/net`.

[51] J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 1962.

[52] Serdar Ozoguz and Mustak E. Yalcin. A cellular neural network made of relaxation oscillators for autowave generation in CMOS. *2008 11th International Workshop on Cellular Neural Networks and Their Applications*, (July):108–112, July 2008.

[53] RS Pressman. Principi di ingegneria del software, 4a edizione.

[54] T. Reenskaug. Models-views-controllers. *Technical note, Xerox PARC*, 1979.

[55] Csaba Rekeczky. CNN architectures for constrained diffusion based locally adaptive image processing. *International Journal of Circuit Theory and Applications*, 30(2-3):313–348, March 2002.

[56] T. Roska. Computational and computer complexity of analogic cellular wave computers. In *Cellular Neural Networks and Their Applications, 2002.(CNNA 2002). Proceedings of the 2002 7th IEEE International Workshop on*, pages 323–338. IEEE, 2002.

[57] T. Roska. Cellular wave computers for brain-like spatial-temporal sensory computing. *Circuits and Systems Magazine, IEEE*, 5(2):5–19, 2005.

[58] T. Roska. Cellular wave computer architectures in a new era of computing—15 years later. *International Journal of Circuit Theory and Applications*, 36(5-6):523–524, 2008.

[59] T. Roska and L.O. Chua. The cnn universal machine: an analogic array computer. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 40(3):163–173, 1993.

[60] T. Roska, L.O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, and F. Puffer. Simulating nonlinear waves and partial differential equations via cnn. i. basic techniques. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 42(10):807–815, 1995.

[61] T. Roska, A. Zarandy, S. Zold, P. Foldesy, and P. Szolgay. The computational infrastructure of analogic cnn computing. i. the cnn-um chip prototyping system. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 46(2):261–268, 1999.

[62] M. Salerno, F. Sargeni, and V. Bonaiuto. Design of a dedicated cnn chip for autonomous robot navigation. In *Cellular Neural Networks and Their Appli-*

cations, 2000.(CNNA 2000). Proceedings of the 2000 6th IEEE International Workshop on, pages 225–228. IEEE, 2000.

[63] F. Sargeni and V. Bonaiuto. Programmable cnn analogue chip for rd-pde multi-method simulations. *Analog integrated circuits and signal processing*, 44(3):283–292, 2005.

[64] A. Slavova. Applications of some mathematical methods in the analysis of cellular neural networks. *Journal of computational and applied mathematics*, 114(2):387–404, 2000.

[65] A. Slavova and P. Zecca. Cnn model for studying dynamics and travelling wave solutions of fitzhugh-nagumo equation. *Journal of computational and applied mathematics*, 151(1):13–24, 2003.

[66] J.M. Sobel and Daniel P. Friedman. An introduction to reflection-oriented programming, 1996.

[67] W. Spataro, D. D'Ambrosio, R. Rongo, and G. Trunfio. An evolutionary approach for modelling lava flows through cellular automata. *Cellular Automata*, pages 725–734, 2004.

[68] S.H. Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering (studies in nonlinearity)*. Westview Press, 2001.

[69] D.B. Strukov, G.S. Snider, D.R. Stewart, and R.S. Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[70] S. Süsstrunk, R. Buckley, and S. Swen. Standard rgb color spaces. In *Proc. IS&T/SID's 7th Color Imaging Conference*, pages 127–134. Citeseer, 1999.

[71] Y. Suzuki, T. Takayama, I.N. Motoike, and T. Asai. Striped and spotted pattern generation on reaction-diffusion cellular automata–theory and lsi

implementation–. In *Proceedings of the 2005 Workshop on Unconventional Computing: From Cellular Automata to Wetware*, page 41. Luniver Pr, 2005.

[72] Lua Team. Lua, documentazione online. `http://www.lua.org/docs.html`.

[73] .Net Team. Patterns for parallel programming: Understanding and applying parallel patterns with the .net framework 4. `http://www.microsoft.com/download/en/details.aspx?displaylang=it&id=19222l`.

[74] OpenGL Team. Opengl, documentazione online. `http://www.opengl.org/documentation/`.

[75] P. Thiran. Influence of boundary conditions on the behavior of cellular neural networks. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 40(3):207–212, 1993.

[76] T. Tokihiro, D. Takahashi, J. Matsukidaira, and J. Satsuma. From soliton equations to integrable cellular automata through a limiting procedure. *Physical Review Letters*, 76(18):3247–3250, 1996.

[77] W.H.P.W.T. Vellerling and PB Flannery. *Numerical Recipe in C*. Cambridge University Press, 1992.

[78] R. White and G. Engelen. Cellular automata and fractal urban form: a cellular modelling approach to the evolution of urban land-use patterns. *Environment and planning A*, 25:1175–1175, 1993.

[79] R. Williams. How we found the missing memristor. *Spectrum, IEEE*, 45(12):28–35, 2008.

[80] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601, 1983.

[81] S. Wolfram and M. Gad-el Hak. A new kind of science. *Applied Mechanics Reviews*, 56:B18, 2003.

[82] M. Yakabe and S. Tokinaga. Applying the genetic programming to modeling of diffusion processes by using the cnn and its applications to the synchronization. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 86(8):19–30, 2003.

[83] ME Yalçın, J. Vandewalle, P. Arena, A. Basile, and L. Fortuna. Watermarking on cnn-um for image and video authentication. *International journal of circuit theory and applications*, 32(6):591–607, 2004.

[84] Á. Zarándy and C. Rekeczky. Bi-i: a standalone ultra high speed cellular vision system. *Circuits and Systems Magazine, IEEE*, 5(2):36–45, 2005.