ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA

DIPARTIMENTO di
INGEGNERIA DELL'ENERGIA ELETTRICA E DELL'INFORMAZIONE
"Guglielmo Marconi"
DEI

**CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA ELETTRONICA**

TESI DI LAUREA
in
Progetto di Circuiti e Sistemi Analogici M

Spiking Neural Networks per il monitoraggio strutturale basato su vibrazioni

CANDIDATA

Benedetta Baldini

RELATORE

Chiar.mo Prof. Luca De Marchi

CORRELATRICE

Prof.ssa Federica Zonzini

Anno Accademico *2021-2022*

Sessione III

Indice

Lista degli Acronimi

AI Artificial Intelligence

ALIF Adaptive Leaky Integrate and Fire

ANN Artificial Neural Network

CNN Convolutional Neural Network

CWT Continuous Wavelet Transform

CV Computer Vision

DL Deep Learning

FT Fourier Transform

GPU Graphic Computer Interface

LIF Leaky Integrate and Fire

ML Machine Learning

MSE Mean Squared Error

NC Neuromorphic Computing

NDT Non-Destructive Techniques

NPU Neuromorphic Processor Unit

PSD Power Spectral Density

ReLU Rectified Linear Unit

SHM Structural Health Monitoring

SL SuperLet

SNN Spiking Neural Network

STFT Short Time Fourier Transform

Abstract

Il monitoraggio strutturale (SHM) costituisce un'importante sfida per garantire la sicurezza e l'integrità delle strutture civili e industriali.
In questo studio è stato presentato un nuovo approccio all'SHM basato su reti neurali spiking (SNN), valutando in particolare la capacità di questi nuovi approcci neurali di rilevare anomalie in un caso d'uso di riferimento, ovvero il ponte Z24 in Svizzera.

Attraverso una serie di esperimenti, è stato dimostrato che l'analisi delle risposte vibrazionali con SNN è in grado di individuare in modo preciso e affidabile le variazioni nel comportamento della struttura.
In secondo luogo, è stata dimostrata l'efficienza delle Superlet per ottenere rappresentazioni tempo-frequenza ad alta risoluzione rispetto a metodi più tradizionali come le trasformate Wavelet e spettrogrammi.

Inoltre, abbiamo dimostrato che l'impiego di SNN risulta vincente rispetto a un più tradizionale approccio incentrato sull'uso di reti neurali convoluzionali (CNN), migliorando significativamente le prestazioni in termini di accuratezza (un aumento massimo del 15%).

In sintesi, il nostro studio ha dimostrato che l'approccio SNN può rappresentare una soluzione promettente per il monitoraggio della salute strutturale e il rilevamento precoce di anomalie relative a variazioni nella firma spettrale della struttura target, offrendo nuove opportunità per migliorare la sicurezza e l'affidabilità delle infrastrutture civili e industriali.

Capitolo 1

Introduzione

Monitorare periodicamente una struttura è di fondamentale importanza per garantirne un utilizzo continuativo, sicuro e resiliente, senza interferire con il nominale funzionamento operativo. Questo obiettivo può essere perseguito caratterizzando, nel tempo, le sue condizioni fisiche e prevenire (o identificare) l'insorgenza di guasti.

L'individuazione di eventuali danni viene effettuata tramite l'impiego di sensori in grado di acquisire, trasmettere, memorizzare ed elaborare dati relativi al comportamento delle strutture, specialmente alle sue risposte statiche/dinamiche [1].

Per fare ciò, è di vitale importanza ricorrere a tecniche non distruttive, cioè metodologie e metodi in grado di permettere l'osservazione dello stato di salute senza comprometterne il funzionamento. [2]

Data la complessità crescente delle strutture, nonché la grande densità di sensori ed interconnessione che sono stati resi possibili dai recenti avanzamenti nell'ingegneria dell'informazione, è ora possibile raccogliere una quantità consistente di dati, quindi di informazione, che devono essere opportunamente analizzati.

Un insieme di dati sufficientemente ampio apre, infatti, alla possibilità di poter impiegare strumenti di Intelligenza Artificiale per cercare di automatizzare il lavoro di identificazione di guasti strutturali.

Gli algoritmi di Machine Learning (ML), in particolare le reti neurali artificiali, sono diventate oggigiorno di grande interesse in quanto forniscono uno strumento molto

potente per individuare pattern complessi, e per questo motivo ricoprono un ruolo sempre più importante nel campo dell'elaborazione di immagini digitali.

Questo campo di applicazione prende il nome di *Computer Vision (CV)* e ha come obiettivo la ricerca di tecniche per aiutare i computer a esaminare e comprendere il contenuto di fotografie e video facendo uso di reti neurali, e trovano applicazione in tutti quei contesti in cui il modello in esame non è noto e viene appreso attraverso operazioni di training.

Pertanto, le NN rappresentano una delle innovazioni più promettenti anche in ambito strutturale per rilevare anomalie; infatti, esse godono delle seguenti proprietà:

- *Capacità di apprendere pattern* dalle informazioni estratte dai sensori installati sulla struttura: i danni strutturali possono infatti essere identificati come deviazioni significative rispetto al *ground truth*;
- *Gestione di grandi quantità di dati*: sono quindi adatte per l'analisi di grandi strutture o di strutture complesse, dove molti sensori devono essere utilizzati per monitorare lo stato della struttura;
- *Adattabilità* ai cambiamenti nelle condizioni ambientali/ dati di input: questa proprietà risulta in special modo utile per effettuare un monitoraggio continuo;
- *Automazione* del rilevamento: eliminando la necessità di un'analisi manuale dei dati, si riducono i tempi di inattività della struttura.

Un sistema di visione artificiale si articola dunque in tre fasi principali:

1. **Acquisizione:** Le immagini, anche grandi set, possono essere acquisite in tempo reale attraverso video, foto o tecnologia 3D per l'analisi.
2. **Elaborazione:** I modelli di deep learning automatizzano gran parte di questo processo, e necessitano training con migliaia di immagini etichettate o pre-identificate.
3. **Interpretazione del contenuto semantico:** Il passaggio finale è il passaggio interpretativo, in cui un oggetto viene identificato o classificato.

In particolare, con il presente elaborato si è voluto proporre un nuovo approccio alla fault detection di una struttura combinando l'analisi di immagini con un nuovo tipo di reti neurali: le Spiking Neural Networks (SNNs).

Le SNN, infatti, risultano molto promettenti e vantaggiose in quanto simulano il meccanismo dinamico dei neuroni biologici e presentano un grande potenziale di elaborazione real-time di informazioni spazio-temporali e un basso consumo energetico, specialmente quando implementate su hardware dedicato.

Nell'ambito dello Structural Health Monitoring, l'impiego congiunto delle SNN e dell'analisi di vibrazione, tecnica che analizza cambiamenti nei pattern di vibrazione della struttura, consente lo sviluppo di metodi efficaci e accurati nella ricerca di anomalie.

Stante queste premesse, il presente elaborato si propone di sviluppare in ambiente Python (tramite la libreria Keras e NengoDL) una rete neurale di tipo Spiking e di impiegarla a scopi di *anomaly detection* per il riconoscimento e la classificazione di anomalie nel pattern tempo-frequenza di un ponte target.

Nella prima parte dell'elaborato verranno presentati i dettami principali del monitoraggio basato su analisi di vibrazioni, con particolare enfasi al problema di trovare una rappresentazione tempo-frequenza adeguata a garantire una sufficiente risoluzione delle immagini prodotte.

Nella seconda parte verranno invece illustrati i principali algoritmi di Machine Learning impiegati in applicazioni di SHM in termini di classificazione, struttura e vantaggi. In particolare, sono state confrontate le reti convoluzionali e le reti spiking, algoritmo core dello studio condotto.

Nella terza e ultima parte verrà infine illustrato e analizzato in maniera sequenziale l'approccio pratico al problema, gli strumenti impiegati e i risultati ottenuti.

L'elaborato termina con una breve panoramica conclusiva in cui vengono esposte le prospettive future ed i possibili utilizzi della metodologia proposta.

Capitolo 2

Structural Health Monitoring

In quanto "sentinella" delle nostre infrastrutture, il monitoraggio strutturale offre una serie di tecnologie all'avanguardia che permettono di monitorare in tempo reale lo stato di salute di ponti, edifici, dighe e altre strutture, civili o industriali.

La sua importanza è fondamentale per prevenire eventuali cedimenti o danni, garantire la sicurezza delle persone e delle cose, e prolungare la vita delle infrastrutture.
Grazie a esso, infatti, è possibile identificare eventuali anomalie, adottare tempestive misure preventive, e assicurare che le nostre infrastrutture siano sostenibili e sicure nel lungo termine.

2.1 – Il monitoraggio

Implementare strategie di rilevamento di anomalie o danni nelle infrastrutture aerospaziali, civili e meccaniche appare di vitale importanza. Le cause di tali variazioni possono essere ritrovate sia nei processi di invecchiamento dei materiali che compongono la struttura, che in cause ambientali (legate al contesto operativo) o atmosferiche.

È doveroso dunque introdurre la definizione di danno: esso è definito come l'insieme dei cambiamenti nel materiale e/o proprietà geometriche della struttura, quali le condizioni al contorno e le sezioni geometriche, o in ogni altro fattore capace di provocare un comportamento anomalo a breve o a lungo termine.

Una soluzione di tipo SHM può, pertanto, coinvolgere fino a quattro fasi diverse aventi un rapporto di interdipendenza l'una con l'altra per garantire e monitorare il ciclo di vita del sistema sotto esame. Tali fasi sono riportate in Figura 1.

Figura 1: Workflow nel SHM

1) Pianificazione: Una prima fase propone la strategia da impiegare rispetto all'obiettivo del lavoro. È necessario dunque saper individuare cosa, quando, dove, come e perché andare a misurare e con quali finalità.

2) Raccolta dati: questa fase comporta la raccolta e l'analisi dei dati necessari per lo studio, e richiede di effettuare scelte in termini di strumenti di misura, scelta di sensori sia in numero che in tipologia, e hardware per l'acquisizione, archiviazione e trasmissione di tali dati.

3) Elaborazione: Una volta ottenuti i dati il passo successivo sarà quello di estrarre l'informazione utile dal dataset raccolto. Un esempio di elaborazione è sicuramente la Trasformata di Fourier (FT), che cambia il dominio di visualizzazione dei dati dal tempo alla frequenza. Chiaramente, la scelta del metodo migliore per estrapolare l'informazione dai dati è punto focale del flusso di progetto, poiché scelte diverse comportano risultati e risoluzioni diverse.

4) Valutazione: L'ultima fase è quella di interpretazione dei risultati in relazione all'obiettivo del progetto e comporta un'analisi della performance a breve e a lungo termine, mettendo in atto tutte le strategie necessarie per il riparo o l'invio di messaggi preventivi di allerta in presenza di potenziali anomalie.

Il monitoraggio strutturale si basa su tecniche di controllo non distruttive (NDT), così da esaminare la salute di una struttura, ovvero rilevare eventuali guasti sia superficiali che interni senza, comprometterne il funzionamento.
Alcuni esempi sono le tecniche basate su emissioni acustiche, sensori a fibra ottica, analisi di vibrazione, ultrasuoni, raggi X, etc. [3]

Queste tecniche NDT possono essere classificate come test attivi o passivi: in questi ultimi, vengono utilizzati sensori in grado di acquisire la risposta strutturale, senza stimolarne il comportamento con la generazione di un apposito stimolo, cosa che accade invece in soluzioni attive.

Le tecniche di emissione acustica sono solitamente applicate per SHM passivo, così come accelerometri o altri sensori per vibrazione; al contrario, fra le tecniche che si basano su metodi attivi è possibile annoverare l'ispezione ultrasonica.

Infine, oltre a questa prima classificazione, è possibile anche distinguere tecniche di *damage detection* locali o globali in base alla larghezza della porzione interessata dal danno.

I primi sono più adatti a valutare la performance della struttura in aree piccole e circoscritte, e per poter utilizzare efficientemente tali metodi è necessaria la conoscenza a priori della probabile posizione del guasto. La maggior parte delle NDT appartengono a questa categoria (tecniche a ultrasuoni, radar etc.).

Metodi di damage detection globali, al contrario, hanno il vantaggio di considerare le variazioni d'insieme della struttura; l'***analisi di vibrazione*** rientra in questa categoria**.**

2.2 – Analisi di vibrazione

La premessa alla base di questa tecnica è che ogni danno strutturale provochi variazioni nei parametri strutturali della struttura (massa, rigidità, flessibilità) che a loro volta provocano un cambiamento nei parametri dinamici della struttura (Figura 2):

- *frequenze naturali*, cioè le frequenze alle quali il sistema vibra naturalmente in assenza di forze esterne, che dipendono dalle proprietà fisiche del sistema;
- *forme modali*, cioè i modi di vibrazione del sistema associati alle diverse frequenze naturali. Ogni frequenza naturale del sistema è associata ad una specifica forma modale, che descrive la configurazione spaziale delle vibrazioni del sistema a quella frequenza.

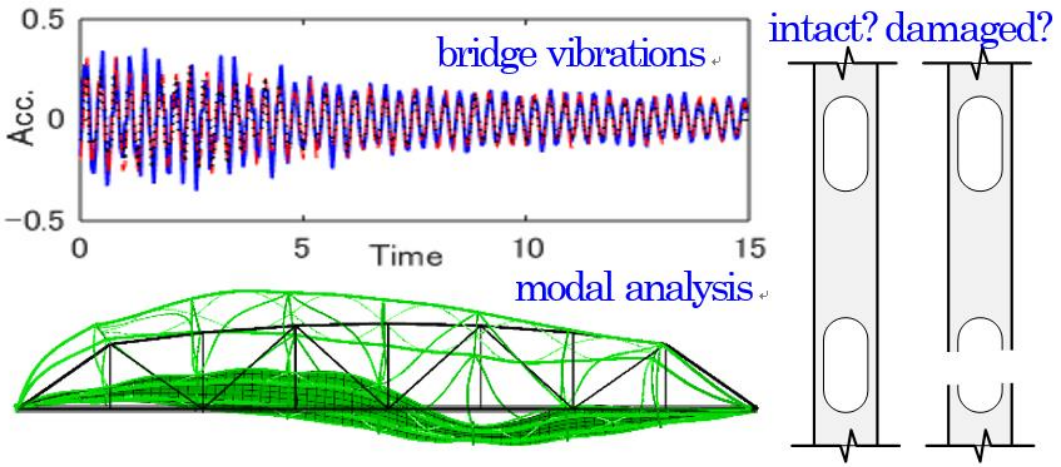


Figura 2: Parametri dinamici del case study (fonte: tse1.mm.bing.net/th?id=OIP.bJgQMbu8iDUm36gubNnMNQHaDY&pid=Api&P=0)

Tuttavia, questi cambiamenti sono spesso troppo piccoli per consentire l'identificazione efficace dei danni nei vari scenari possibili. Inoltre, fattori ambientali come la temperatura possono portare a variazioni dei parametri strutturali della stessa entità di quelli causati da danni fisici.

L'analisi spettrale viene spesso utilizzata nell'analisi delle vibrazioni per proiettare il contenuto informativo del segnale di vibrazione dal dominio temporale al dominio delle frequenze, dove possono essere visualizzate le componenti di frequenza del segnale.

In particolare, l'osservazione delle variazioni nelle proprietà spettrali consente di valutare la presenza o meno di alterazioni fisiche: ad esempio, variazioni significative nelle frequenze di vibrazione possono indicare la presenza di un danneggiamento strutturale.

Lo strumento più comune eseguire questa operazione è dato dalla trasformata di Fourier (FT), descrivibile come:

$$S(e^{j\omega}) = \int_{-\infty}^{\infty} s(t)e^{-j\Omega t}dt$$

Il problema della FT è che si tratta di un'espressione valida unicamente per segnali stazionari, ovvero segnali con spettro invariante nel tempo. Quindi, non è uno strumento

adatto per segnali non stazionari e altamente variabili nel tempo, come quelli sotto esame in ambito SHM.

Inoltre, le informazioni temporali che consentono di rilevare quando o dove un particolare evento ha avuto luogo vengono perse dopo la trasformazione di Fourier.

Nasce quindi la necessità di una nuova rappresentazione che sia tempo-frequenza.

2.3 – Analisi Tempo Frequenza

L'analisi tempo-frequenza è una tecnica di analisi dei segnali che permette di esaminare come il contenuto spettrale vari nel tempo.
Questa tecnica è particolarmente utile per l'analisi di segnali complessi, come quelli generati da sistemi dinamici o da fenomeni naturali fortemente non stazionari

Il vantaggio principale dell'analisi tempo-frequenza è quello di fornire informazioni sulle componenti frequenziali di un segnale, non solo in termini di ampiezza, ma anche di distribuzione temporale.
Questo permette di rilevare variazioni nella frequenza del segnale nel tempo, che possono essere utili per identificare eventi significativi o cambiamenti nel sistema o nel fenomeno che ha generato il segnale.

In particolare, l'analisi tempo-frequenza può essere utilizzata per identificare eventi transitori, che rappresentano un cambiamento repentino nella frequenza del segnale.
Questi eventi possono essere causati, ad esempio, da guasti meccanici in un sistema dinamico, da processi di commutazione in un sistema elettronico o da segnali sismici generati da terremoti o esplosioni.

2.3.1 – Short Time Fourier Transform

Un primo esempio semplice di rappresentazione tempo-frequenza è la Short Time Fourier Transform (STFT), che determina le componenti di tempo e frequenza del segnale analizzato facendo la FT di una singola porzione di segnale (che viene dunque ‘finestrata’)

e lascia scorrere la finestra lungo tutto il segnale, così da poter approssimare ogni segmento come stazionario, ma solo localmente.

Sia w(t) la finestra che viene fatta scorrere lungo il segnale s(t) reale e simmetrico e a norma unitaria; si costruisce la sua traslata nel tempo di τ e modulata in frequenza di Ω:

$$g_{\Omega,\tau}(t) = w(t - \tau)e\, j\Omega t$$

Le caratteristiche di w(t) assicurano che $||g_{\Omega\tau}|| = 1$ per ogni $(\Omega, \tau) \in \mathbb{R}^2$
La STFT è quindi definita come $STFT_s(\Omega, \tau) = \int_{-\infty}^{\infty} s(t)w^*(t-\tau)e^{-j\Omega t}dt$
e il relativo funzionamento è graficamente spiegato in Figura 3:

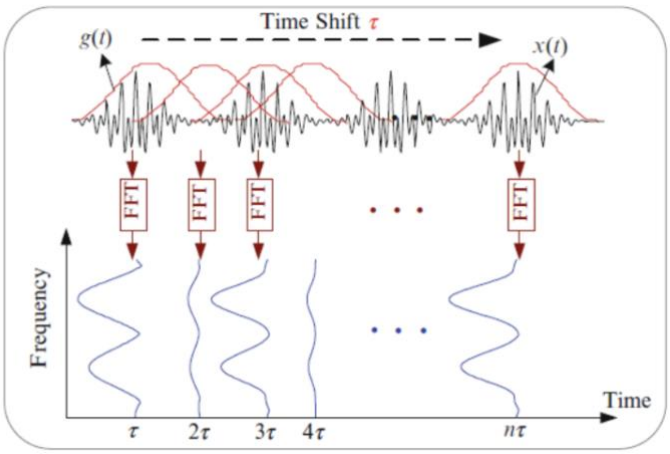


Figura 3: Schema di funzionamento della STFT

Il significato pratico di questa trasformata è che misura la somiglianza tra il segnale e le versioni traslate e modulate della finestra: maggiore è il grado di somiglianza e maggiore è la presenza di quella componente in frequenza.

$$STFT_s(\Omega, \tau) = \langle s(t), g_{\Omega,\tau}(t) \rangle$$

La distribuzione di energia associata alla STFT è detto spettrogramma, di cui un esempio è riportato in Figura 4:

$$S(\Omega, \tau) = |STFT_s(\Omega, \tau)|^2$$

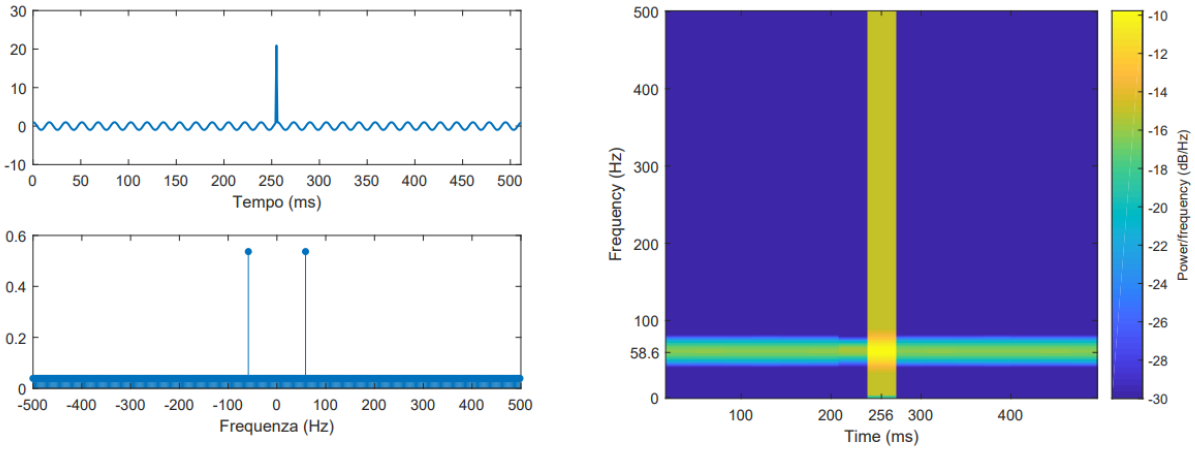


Figura 4: Esempio di Spettrogramma

Elemento fondamentale nella valutazione di queste rappresentazioni è il principio di indeterminazione di Heisenberg, che nel *signal processing* prende il nome di limite di Gabor. Questo afferma che una funzione non può essere sia limitata nel tempo che limitata in banda (una funzione e la sua trasformata di Fourier non possono avere entrambe un dominio limitato).

$$\Delta f \Delta t \geq 1$$

Idealmente, infatti, l'obiettivo sarebbe quello di discriminare ogni frequenza e ogni relativo istante di tempo, perciò nel piano tempo-frequenza ciò che ci si aspetterebbe sarebbero dei punti. In realtà, quello che si verifica è la creazione di un piano suddiviso in una serie di mattonelle di area $\sigma_t \sigma_\Omega$ che rappresentano un intervallo specifico di tempo/frequenza, come si evince da Figura 5.

In particolare:

$$\begin{cases} \sigma_t^2 = \dfrac{1}{E}\displaystyle\int_{-\infty}^{\infty} (t - t_m)^2 s^2(t) dt \\ \sigma_\Omega^2 = \dfrac{1}{E}\displaystyle\int_{-\infty}^{\infty} (f - f_m)^2 |S(f)|^2 df \end{cases}$$

e sono misura della dispersione di E nell'intorno di t_m, f_m, ovvero della localizzazione temporale e in frequenza dell'energia.

L’area di queste mattonelle, dette di Heisenberg, rappresenta la risoluzione con cui si può localizzare gli eventi, ed è risultato del principio di indeterminazione: questa non può essere resa arbitrariamente piccola a piacere.

$$\sigma_t \sigma_\Omega \geq \frac{1}{2}$$

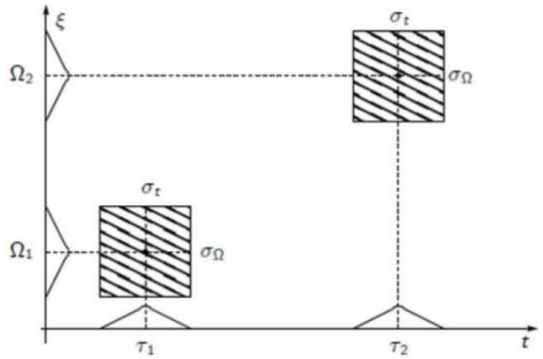


Figura 5: Area di Heisenberg

Da questo si evince l’importanza della dimensione della finestra in quanto garante dell’accuratezza con la quale si vogliono localizzare discontinuità e con cui separare componenti frequenziali [4] .

Bisogna dunque capire quanto e dove sia importante avere alta risoluzione: idealmente, si vorrebbe che la risoluzione in frequenza si adattasse al contenuto spettrale della classe di segnali processati.

Infatti, quando si lavora con segnali a bassa frequenza, è prioritario avere un’alta risoluzione in frequenza, ovvero saper discriminare bene i valori in un intervallo limitato (per esempio tra 1 Hz e 2 Hz): è evidente che, trattandosi di variazioni relativamente lente, è meno importante un’elevata risoluzione nel tempo.

Vale dunque anche il viceversa: a frequenze elevate (corrispondenti a oscillazioni veloci) è accettabile una minore risoluzione in frequenza, ma non si può transigere in termini di risoluzione temporale.

Tuttavia, nella STFT la larghezza della finestra di Heisenberg (la scala) è fissa, e non può essere adattata a seconda delle necessità; quindi, localizzo fenomeni lenti e veloci con la stessa indeterminazione.

È necessario dunque procedere in un altro modo: tutti i limiti dell'analisi spettrale fino ad ora sottolineati vengono superati dall'introduzione della Trasformata Wavelet (WT), che può essere considerata un'evoluzione della STFT per effettuare un'analisi a risoluzione variabile.

2.3.2 - CWT

L'idea alla base è quella di pensare di scalare la finestra per tutti gli eventi del piano, ottenendo una famiglia di funzioni (dette *mother wavelet*) traslate e scalate, a energia finita e valor medio nullo (oscillanti), con una pseudo frequenza di riferimento che discrimina la natura dell'evento associato (lento o veloce):

$$W_a = \frac{1}{\sqrt{a}}\, w\left(\frac{t}{a}\right)$$

Questo è il principio di funzionamento della Wavelet: non si cambia la risoluzione (l'area) di ciascuna mattonella nel piano tempo-frequenza, ma si adatta la forma della mattonella (base/altezza) per modulare la risoluzione a seconda del contenuto frequenziale in quell'intervallo, come si può vedere in Figura 6.

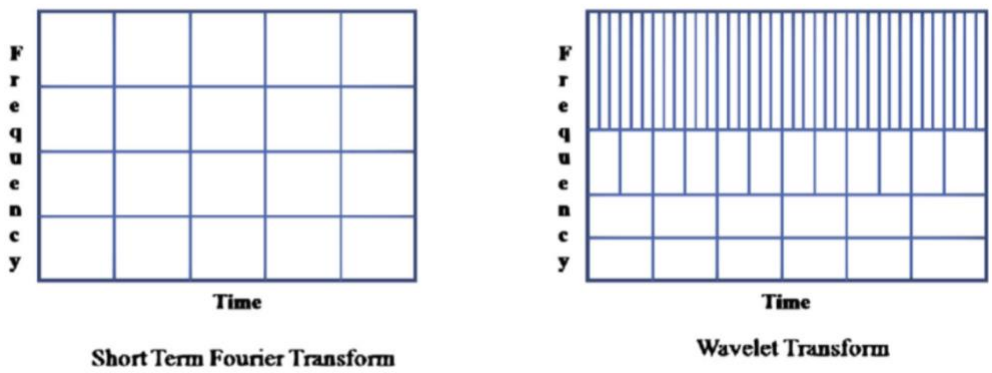


Figura 6: Differenze di risoluzione sul piano t-f

Tale trasformata ha la possibilità di scalare la risoluzione in funzione della componente frequenziale, per cui alle alte frequenze andrà ad utilizzare una wavelet madre compressa; al contrario, alle basse frequenze, impiegherà finestre di ampiezza maggiore, ossia la wavelet subisce una dilatazione.

Un limite della trasformata wavelet è dato dall'uso di una mother wavelet $\psi(t)$ fissata, di cui esistono numerose famiglie: Haar, Daubechies, Gabor, Morlet, Mexicat Hat, Biortogonale etc., riportate in Figura 7.

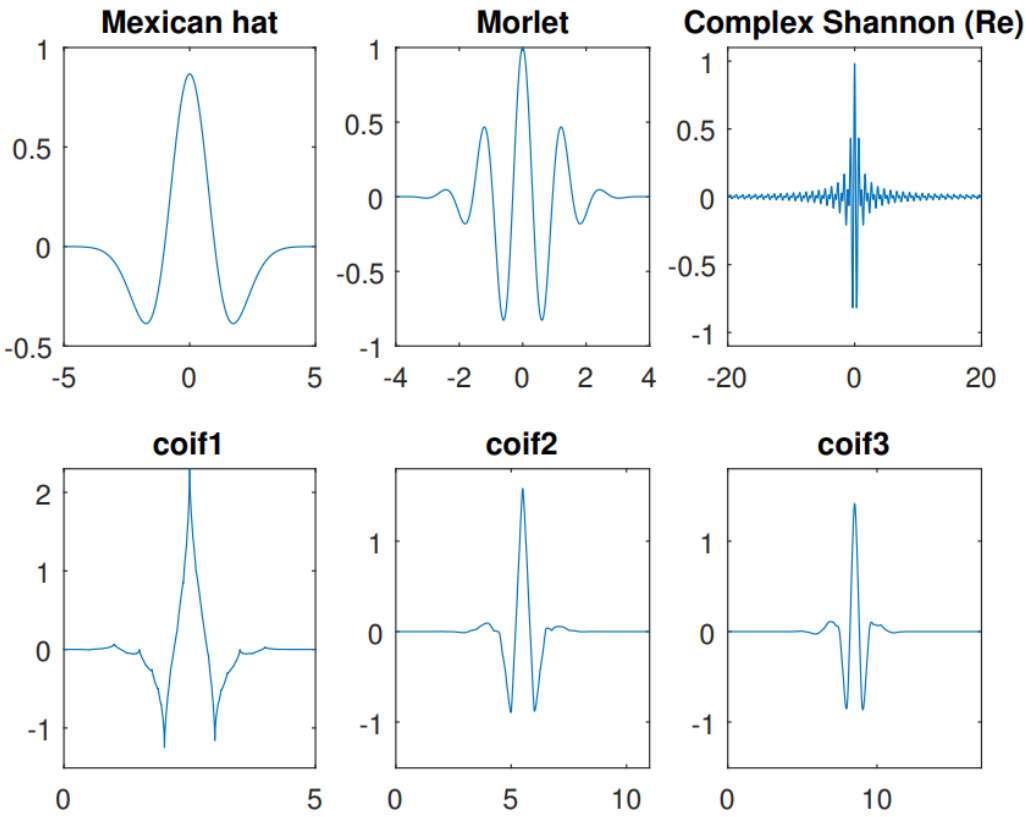


Figura 7: Esempi di Wavelet Madri

La Continuous Wavelet Transform (CWT) effettua così un confronto, per mezzo di prodotti interni, che valutino la somiglianza fra il segnale oggetto di studio e una funzione di analisi, così come nel caso della Trasformata di Fourier.

In particolare, è un esponenziale complesso che viene finestrato a rappresentare la funzione di analisi della STFT, mentre nella CWT questo esponenziale viene sostituito da

versioni traslate e scalate di una wavelet madre $\Psi_{a,b}(t)$, con a e b, rispettivamente, parametri di scala e traslazione

$$\Psi_{a,b}(t) = \frac{1}{\sqrt{a}}\,\psi\left(\frac{\mathrm{t}-\mathrm{b}}{\mathrm{a}}\right) \rightarrow \quad CWT(a,b) = \int_{-\infty}^{\infty} \mathrm{s(t)}\Psi^{*}_{a,b}(t)dt$$

Si può dunque derivare come il fattore di scala sia inversamente proporzionale alla frequenza:

- basso (a < 1): la wavelet viene compressa, ottenendo così oscillazioni che variano rapidamente, ed è quindi adatta per le alte frequenze;
- alto (a > 1): duale, la wavelet è dilatata quindi i dettagli cambiano lentamente; dunque, si usa per le basse frequenze.

Inoltre, per ottenere una rappresentazione analoga allo spettrogramma, calcolando il modulo quadro della CWT si produce lo scalogramma associato, come riportato in Figura 8:

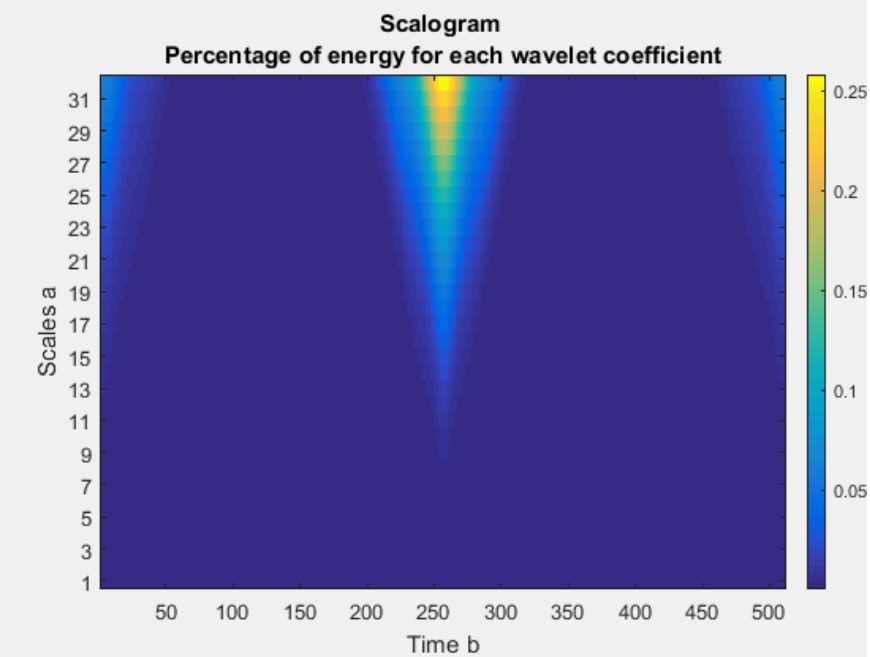


Figura 8: Esempio di Scalogramma

Lo scalogramma prende il nome dalla parola "scala", che in questo contesto si riferisce a una serie di frequenze che vengono utilizzate per scomporre un segnale.

Nella trasformata di Fourier, ad esempio, il segnale viene scomposto in una serie di sinusoidi a frequenze diverse. Nello scalogramma, invece, si utilizzano scale diverse di frequenze per scomporre il segnale.

Lo scalogramma è quindi una rappresentazione grafica delle varie componenti di frequenza di un segnale, in cui le diverse scale sono rappresentate lungo l'asse delle ordinate, mentre il tempo è rappresentato sull'asse delle ascisse.

Le funzioni wavelet hanno proprietà diverse che le distinguono nell'essere più adatte a determinati scopi; infatti, la scelta della wavelet migliore per una certa applicazione dipende da:

- natura del segnale da analizzare
- tipo di informazione che si vuole evidenziare.

Ci sono, però, altre proprietà rilevanti che è opportuno menzionare:

- Ortogonalità e biortogonalità: queste assicurano un rapido calcolo dei coefficienti di analisi; sfortunatamente, non tutte le wavelet hanno queste due proprietà.
- Supporto compatto: questo significa che la funzione wavelet non ha valori nulli per intervalli finiti e consente di rappresentare in modo più efficiente i segnali con caratteristiche localizzate.

2.3.3 – Superlet

STFT e CWT sono utilizzati per generare rispettivamente rappresentazioni tempo-frequenza (lo spettrogramma) e tempo-scala (lo scalogramma). Sia l'STFT che la CWT però hanno limitazioni significative, come si può vedere in Figura 9.

L'STFT fornisce una buona risoluzione in frequenza ma possiede una scarsa risoluzione temporale nella regione di alte frequenze, mentre la CWT mantiene una buona risoluzione temporale in tutto lo spettro, ma degrada nella risoluzione di frequenza con l'aumentare della frequenza.

Questo problema costituisce un ostacolo importante per l'analisi dei segnali di vibrazione.

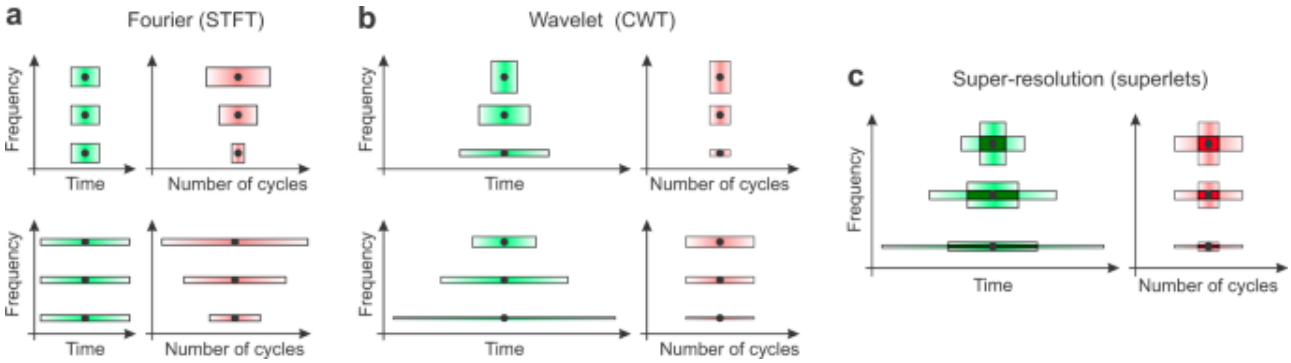


Figura 9: Limitazioni Spettrogramma e CWT (fonte: doi.org/10.1038/s41467-020-20539-9)

a) *risoluzione dell'STFT per una finestra corta (in alto) e larga (in basso) a tre frequenze diverse.*

***b**) Come in a ma per wavelet (CWT). Qui, il numero di cicli è fisso su tutto lo spettro, ma la finestra temporale estesa diminuisce con l'aumentare della frequenza.*

***c)** Superlet (SLT). La super-risoluzione tempo-frequenza si ottiene combinando wavelet corte e larghe con wavelet più lunghe e a larghezza di banda stretta.*

Per superare i limiti dell'STFT, è stato proposto di combinare set di wavelet con banda crescente, in grado di mantenere una buona risoluzione temporale propria delle wavelet, pur guadagnando capacità maggiore di discriminazione in frequenza nelle bande più alte dello spettro. Questa tecnica è stata definita super-risoluzione, perché può ben localizzare il segnale simultaneamente sia nel tempo che nella frequenza.

Una "Superlet" (SL) è definita come un insieme di wavelet con una frequenza centrale fissa, *f*, e che coprono un intervallo di cicli diversi (vincolando cioè progressivamente la banda). Tale operatore è ed è stato introdotto per la prima volta da Shen e Olhede nel 2002 per descrivere una specifica famiglia di wavelet con proprietà di localizzazione tempo-frequenza avanzate.

L'ordine della SL rappresenta il numero di wavelet impiegate nella sua definizione: è evidente che quindi una SLT di ordine 1 è l'equivalente di una CWT. La trasformata Superlet (SLT) di un segnale è calcolata analogamente alla CWT, tranne per il fatto che si usano Superlet invece di wavelet. [5]

Per aumentare la risoluzione, la Superlet quindi propone di combinare wavelet corte ad alta risoluzione temporale (piccolo numero di cicli) con wavelet più lunghe, con risoluzione ad alta frequenza (maggior numero di cicli). Il confronto tra CWT e Superlet è riportato dalla Figura 10.

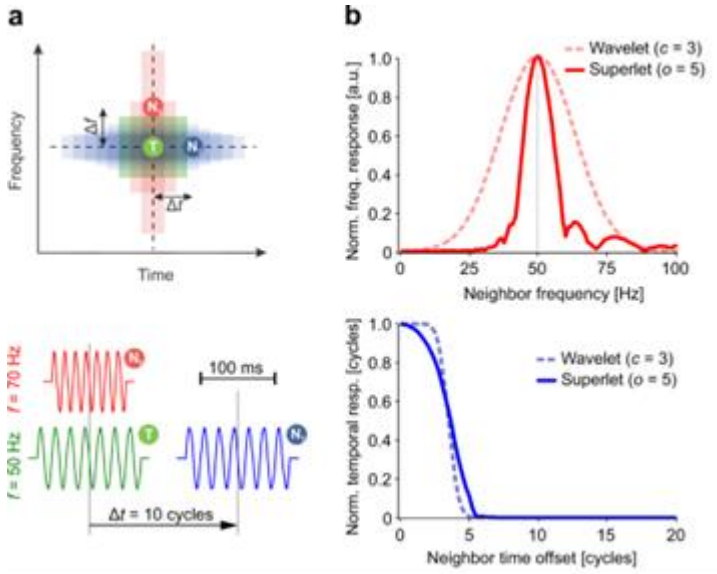


Figura 10: Confronto Superlet-CWT (fonte: doi.org/10.1038/s41467-020-20539-9)

Essendo le reti neurali spiking funzionali per l'elaborazione di segnali dinamici non stazionari, che richiedono quindi una rappresentazione tempo-frequenza di alta risoluzione e in tempo reale, sia la CWT che la Superlet potrebbero avere un vantaggio notevole rispetto allo spettrogramma.

La CWT è in grado di offrire una risoluzione di frequenza variabile a seconda della scala temporale, il che la rende utile per l'analisi di segnali che presentano variazioni di frequenza su diverse scale temporali. Inoltre, la CWT è in grado di gestire segnali con frequenze non uniformi, cosa che potrebbe essere presente nei segnali SHM, ma potrebbe non essere sufficiente per certe applicazioni.

D'altra parte, la Superlet è stata sviluppata specificamente per l'analisi di segnali non stazionari con una distribuzione di frequenza non uniforme, come ad esempio i segnali sismici, e offre una risoluzione spettrale superiore e una buona capacità di identificazione degli eventi transitori, il che potrebbe essere ideale per l'individuazione di anomalie.

In conclusione, le Superlet sono uno strumento potente per analizzare segnali non stazionari e possono contribuire a migliorare l'accuratezza e l'efficacia dei sistemi di SHM per la rilevazione di anomalie e danni nelle strutture, qualora sia necessario basare la stima di integrità su rappresentazioni tempo frequenza.

Capitolo 3

Machine Learning per Anomaly Detection: dalla CNN alla SNN

Le numerose e recenti conquiste in campo informatico, soprattutto la possibilità di accedere ad hardware ad alte prestazioni, hanno portato allo sviluppo e alla diffusione di una nuova branca dell'Intelligenza Artificiale, chiamata deep learning.

L'apprendimento automatico profondo appartiene alla più ampia scienza del *Machine Learning*, la branca di tecniche ed algoritmi di natura statistico–computazionale che permettono ai computer di migliorare le proprie prestazioni volta per volta (Figura 11).

La rivoluzione apportata dal deep learning è proprio nella capacità, simile a quella umana, di elaborare i dati estraendo da essi pattern e relazioni altamente non lineari, e dunque non modellabili con le classiche tecniche di elaborazione. Grazie a questa facoltà**,** i dispositivi possono apprendere e perfezionare l'esecuzione di funzionalità sempre più complesse, alimentati da nuovi dati non ancora elaborati.

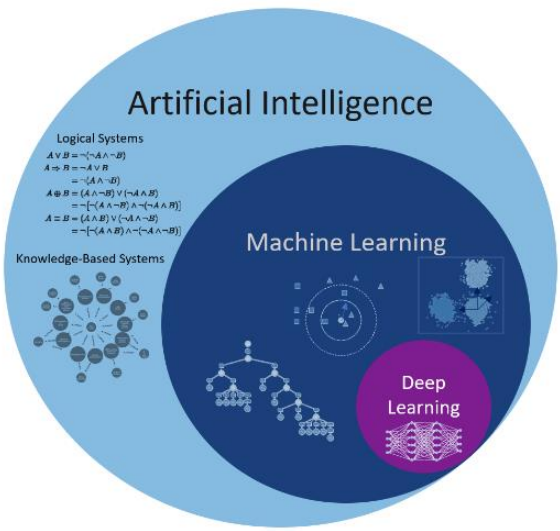


Figura 11: L'AI e i sottogruppi (fonte: data-science-blog.com)

In questo capitolo verrà introdotto il concetto alla base di tale disciplina, ovvero la rete neurale artificiale, verranno quindi messe a confronto le due protagoniste di questo lavoro: la rete convoluzionale e la rete spiking; infine, saranno discusse le proprietà, i limiti ed i vantaggi di entrambe.

3.1 - Reti Neurali Artificiali: proprietà e impiego

Nata come disciplina fondata sui sofisticati calcoli matematici e statistici, il deep learning impiega una moltitudine di elementi computazionali non lineari, detti neuroni artificiali, organizzati come reti le cui interconnessioni emulano il modo in cui i neuroni sono interconnessi nella corteccia visiva dei mammiferi.

Le reti neurali del cervello umano, formate da cellule nervose fittamente interconnesse dette neuroni (in Figura 12), sono la sede della nostra capacità di comprendere l'ambiente e i suoi mutamenti, e di fornire di conseguenza risposte adattive calibrate sulle esigenze che si presentano**.**

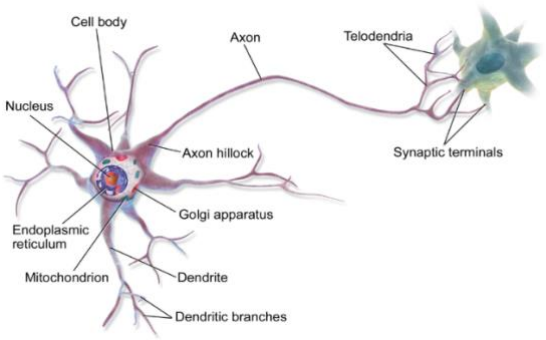


Figura 12: il neurone biologico (Wikipedia - licenza CC BY-SA)

Un singolo neurone può ricevere simultaneamente segnali da diverse sinapsi, ovvero i siti funzionali ad alta specializzazione nei quali avviene il passaggio delle informazioni fra neuroni**.** Una sua capacità intrinseca è quella di misurare il potenziale elettrico di tali segnali in modo globale, stabilendo quindi se è stata raggiunta la soglia di attivazione per generare a sua volta un impulso nervoso**.** Tale proprietà è implementata anche nelle reti artificiali.

Il modello di rete neurale artificiale (ANN) verrà utilizzato come blocco fondamentale per lo studio adattivo dei parametri di funzioni decisionali tramite successive batch di dati di allenamento dai quali apprendere [6].

3.1.1 – Il Neurone Artificiale e suo funzionamento

I neuroni, anche detti percettroni, sono collegati tra loro in vari livelli successivi, così che l'uscita dei neuroni negli strati precedenti formano l'ingresso di quelli negli stadi successivi. La rete forma così un grafo orientato e ponderato, come si vede in Figura 13.

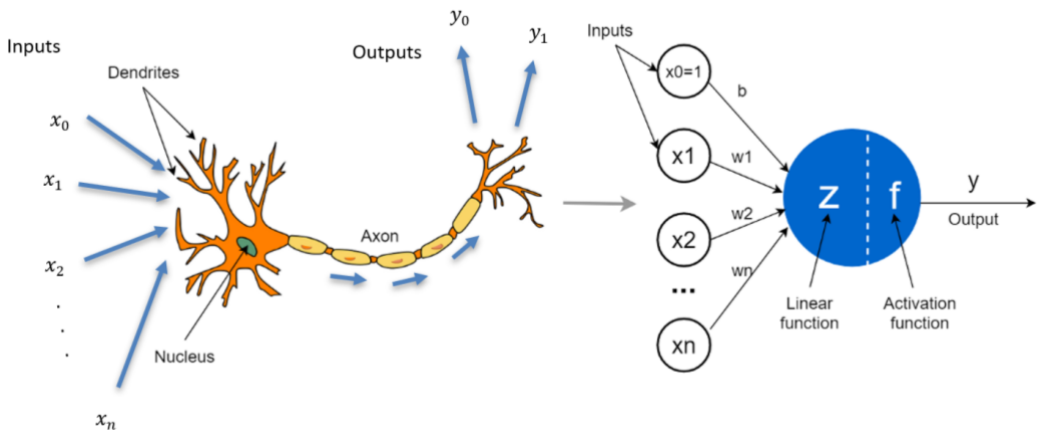


Figura 13: Confronto tra neurone biologico e percettrone artificiale (fonte: towardsdatascience.com)

Ogni neurone nella rete neurale artificiale corrisponde a un nodo del grafo ed è collegato ad altri nodi tramite collegamenti, corrispondenti alle connessioni assoni-sinapsi-dendrite in campo biologico. Ogni collegamento ha un peso, che determina l'intensità dell'influenza di un nodo su un altro. Riassumendo le analogie tra mondo biologico e data science:

- Un neurone biologico riceve i suoi segnali di input da altri neuroni attraverso **i dendriti** (piccole fibre). Allo stesso modo, un percettrone riceve i suoi dati da altri percettroni attraverso neuroni **di input** che prendono numeri.

- I punti di connessione tra dendriti e neuroni biologici sono chiamati **sinapsi**. Allo stesso modo, le connessioni tra input e percettroni sono chiamate **pesi**. Misurano il livello di importanza di ogni input.
- In un neurone biologico, il ***nucleo*** produce un segnale di uscita basato sui segnali forniti dai dendriti. Allo stesso modo, il ***nucleo*** in un percettrone esegue alcuni calcoli basati sui valori di input e produce un output.
- In un neurone biologico, il segnale di uscita viene portato via ***dall'assone***. Allo stesso modo, l'assone in un percettrone è il **valore di uscita** che sarà l'input per i percettroni successivi [7].

I neuroni alternano dunque due fasi di funzionamento: una prima fase di forward propagation e una di back propagation.

La prima fase, detta di forward propagation, implica l'utilizzo di due funzioni matematiche:

1. Una *funzione di propagazione (lineare)*: effettua la somma pesata degli input del neurone con un bias. Analiticamente può essere scritta come:

$$z = \left(\sum_{i=1}^{n} x_i w_i\right) + b$$

   Dove **b** è detto termine o unità di polarizzazione, ed è un valore numerico aggiunto con lo scopo di spostare la funzione di attivazione di ciascun percettrone per non ottenere un valore zero. In altre parole, se tutti gli input sono 0, z è uguale al valore di bias.

2. Una *funzione di attivazione (non-lineare):* applicata all'uscita z riportata sopra, il suo scopo è quello di introdurre la non-linearità al neurone. Senza di essa, infatti, una rete non sarebbe in grado di elaborare modelli non lineari, come quelli presenti nella realtà che ci circonda. Formalizzando.

$$y = f(z)$$

Esistono diverse funzioni di attivazione: alcuni esempi sono riportati in Figura 14.

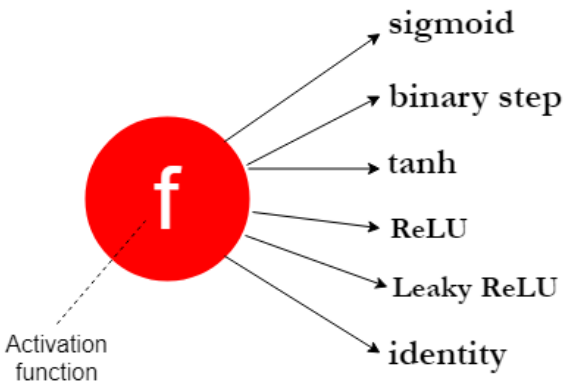


Figura 14: Funzioni di attivazione (fonte: towardsdatascience.com)

Più nel dettaglio, si consideri la funzione di attivazione a soglia (o binaria). Il neurone si attiva ("spara") solo quando il valore di z supera il valore di soglia, e in tal caso porta l'uscita y al valore logico 1; viceversa, l'uscita è 0.

$$z = \begin{cases} 1 \; se \; z \geq z_{Threshold} \\ 0 \quad se < z_{Threshold} \end{cases}$$

Pertanto, il tipo di funzione di attivazione determina come e con quanta facilità il neurone si attiva.

La seconda fase, detta di Back-propagation, è però quella cruciale per il funzionamento di una ANN: questa comporta la ricerca di parametri ottimali per il modello propagando l'informazione all'indietro tra gli strati della rete, e quindi aggiornando iterativamente i parametri differenziando parzialmente i gradienti della funzione di perdita rispetto ai parametri.

$$\frac{\partial L}{\partial w} = \frac{\partial z}{\partial w} \times \frac{\partial a}{\partial z} \times \frac{\partial L}{\partial a}$$

Una piccola variazione dei pesi w sarà dunque causa di una variazione del valore z ($\partial z/\partial w$), che a sua volta influenzerà la variazione nell'attivazione a ($\partial a/\partial z$) e un conseguente cambiamento nella funzione di perdita L ($\partial L/\partial a$).

Per fare ciò è quindi richiesta una funzione di ottimizzazione che possa trovare i pesi ideali per il modello.

Alcuni esempi di funzioni di ottimizzazione sono:

- Gradient Descent
- Adam optimizer
- Gradient Descent with momentum
- RMS Prop (Root Mean Square Prop)

3.1.2 – Architettura

Le reti neurali (NN) sono composte quindi da neuroni artificiali che ricevono input, combinano quest'ultimo con il loro stato interno e una soglia facoltativa utilizzando strumenti matematici, e producono output.

Il layer che riceve dati esterni (come, per esempio, immagini) è detto *layer di input*, mentre lo strato che produce il risultato finale (come il riconoscimento di una feature) è il *layer di output.* Tra di loro ci sono zero o più livelli *nascosti*, come illustrato in Figura 15.

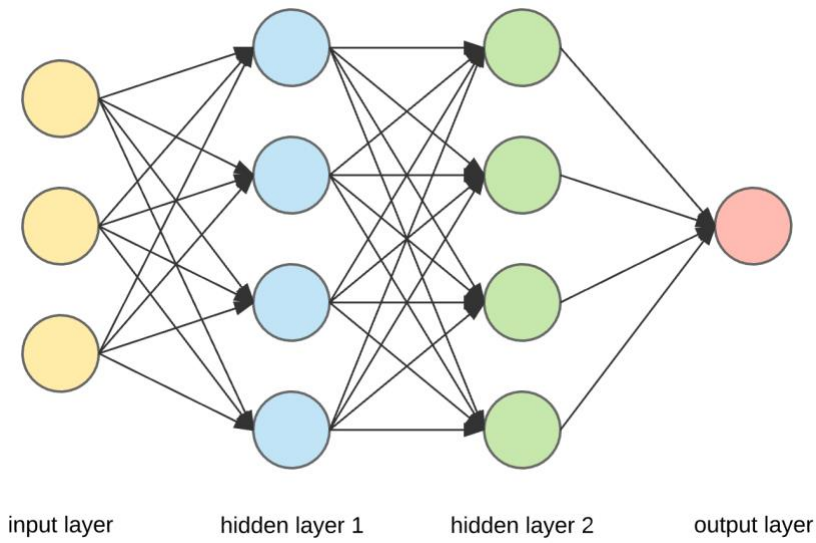


Figura 15: Schema ANN (fonte: medium.com/@ksusorokina)

I neuroni sono in genere organizzati in più strati, soprattutto nel deep learning, e sono in grado di collegarsi solo ai neuroni degli strati immediatamente precedenti e immediatamente successivi.

È possibile connettere due livelli utilizzando diversi modelli: possono essere *completamente collegati,* dove ogni neurone in un unico strato si connette ad ogni neurone nello strato successivo, oppure si esegue un'operazione di *pooling*, dove solo un gruppo di neuroni in un livello si collega a un singolo neurone nello strato successivo, riducendo così il numero di neuroni in quel livello e, corrispondentemente, le interconnessioni tra livelli successivi. I due modelli sono illustrati in Figura 16.

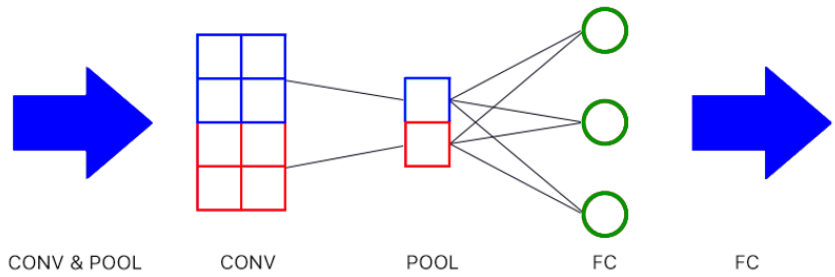


Figura 16: Pooling e Fully connection (FC)

Due sono le principali architetture per ANN:

- **FeedForward NN:** il flusso di informazioni è unidirezionale dagli ingressi fino alle uscite. Questo implica l'assenza di feedback loops e che gli input e gli output non vengano modificati. Sono architetture usate per applicazioni di classificazione, riconoscimento di pattern. Un esempio di grafo è riportato in Figura 17:

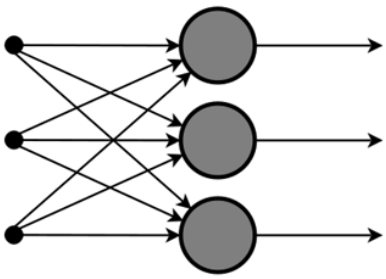


Figura 17: Feedforward ANN (fonte: medium.com/@nathaliejeans)

- **Feedback o Recurrent NN (RNN):** si differenziano dalle precedenti perché hanno il concetto di memoria che permette l'immagazzinamento degli stati o dell'informazione dei precedenti input per generare i successivi output. Di conseguenza, ammettono retroazioni nello schema a blocchi, come si vede in Figura 18:

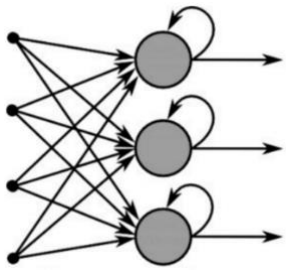


Figura 18: Schema di una RNN (fonte: medium.com/@nathaliejeans)

3.1.3 – Apprendimento

L'apprendimento è definito come l'adattamento della rete allo scopo di svolgere meglio un incarico, tenendo in considerazione le osservazioni campione. L'apprendimento comporta la regolazione dei pesi (e delle soglie facoltative) della rete per migliorare l'accuratezza del risultato. Questo viene fatto riducendo al minimo gli errori osservati. L'apprendimento è completo quando, esaminando ulteriori osservazioni, non si riduce utilmente il tasso di errore. Anche dopo l'apprendimento, il tasso di errore in genere non raggiunge lo zero. Se dopo l'apprendimento, il tasso di errore è troppo alto, la rete in genere deve essere riprogettata.

Nella pratica, questo viene fatto definendo una funzione di costo che viene valutata periodicamente durante l'apprendimento. Finché questa continua a diminuire, l'apprendimento continua. L'apprendimento cerca di ridurre al minimo le differenze tra il valore in uscita del sistema ed il valore reale. La maggior parte dei modelli di apprendimento può essere quindi vista come una semplice applicazione della teoria dell'ottimizzazione e della stima statistica.

La velocità di apprendimento definisce le dimensioni dei passaggi correttivi che il modello esegue per regolare gli errori in ogni osservazione. Un alto tasso di apprendimento (learning rate) riduce i tempi di allenamento al costo di una precisione finale minore, mentre un tasso di apprendimento inferiore richiede più tempo, ma consente di ottenere maggiore precisione.

I processi di ottimizzazione sono principalmente finalizzati ad accelerare la minimizzazione degli errori e in particolare l'aumento dell'affidabilità. Al fine di evitare oscillazioni all'interno della rete e per migliorare la velocità di convergenza, questi perfezionamenti utilizzano una velocità di apprendimento adattiva variabile a seconda dei casi.

È necessario distinguere tre grandi paradigmi di apprendimento:

- un apprendimento supervisionato (*supervised learning*), nel caso in cui si disponga di un insieme di dati per l'addestramento (il cosiddetto *training set*) comprensivo di esempi caratteristici d'ingressi con le corrispondenti uscite: la rete può così imparare a riconoscere la relazione che li lega. La rete è addestrata mediante un opportuno algoritmo, come ad esempio la funzione *backpropagation*, al quale usa tali dati allo scopo di modificare i pesi ed altri parametri della rete stessa in modo tale da rendere minimo l'errore di previsione relativo al training set.

  Se poi l'addestramento va a buon fine, la rete impara a distinguere la relazione incognita che lega le variabili d'ingresso a quelle d'uscita, ed è quindi capace di fare previsioni anche laddove l'uscita non è nota a priori; l'obiettivo finale di questa particolare tipologia di apprendimento è pertanto la previsione del valore dell'uscita per ogni valore valido in ingresso, avendo a disposizione soltanto un numero limitato di esempi di corrispondenza (vale a dire, coppie di valori *input-output*). La rete deve però essere dotata di un'adeguata capacità di generalizzazione per poter lavorare in questo modo con riferimento a casi ad essa ignoti. Ciò consente di risolvere problemi di regressione o classificazione.

- un *apprendimento non supervisionato*, che si fonda su algoritmi di training che modificano i pesi della rete basandosi unicamente su un insieme di dati comprendente le sole variabili d'ingresso. Tali algoritmi tentano di raggruppare i dati d'ingresso in convenienti *cluster* rappresentativi dei dati stessi, tipicamente per mezzo di metodi topologici o probabilistici. È un tipo di apprendimento impiegato anche nello sviluppo di tecniche di compressione dei dati.

- un *apprendimento per rinforzo*, qualora a seguito di osservazioni dell'ambiente esterno lo scopo fosse quello di individuarne il comportamento tramite un opportuno

algoritmo. Sarà l'ambiente stesso a fornire un incentivo o un disincentivo come riscontro, che guiderà l'algoritmo stesso nel processo d'apprendimento.

L'apprendimento con rinforzo differisce da quello supervisionato dal momento che non sono mai presentate delle coppie *input-output* di esempi noti, né si procede alla correzione esplicita di azioni subottimali.

In aggiunta, l'algoritmo si focalizza sulla prestazione in linea, che implica un bilanciamento tra esplorazione di situazioni ignote e sfruttamento della conoscenza corrente.

3.2 – CNN

Una rete neurale convoluzionale, o ConvNet, è uno dei più noti algoritmi nel ramo del *deep learning* [8]. Le CNN sono reti neurali che utilizzano la convoluzione al posto della generale moltiplicazione di matrici in almeno uno dei loro livelli.

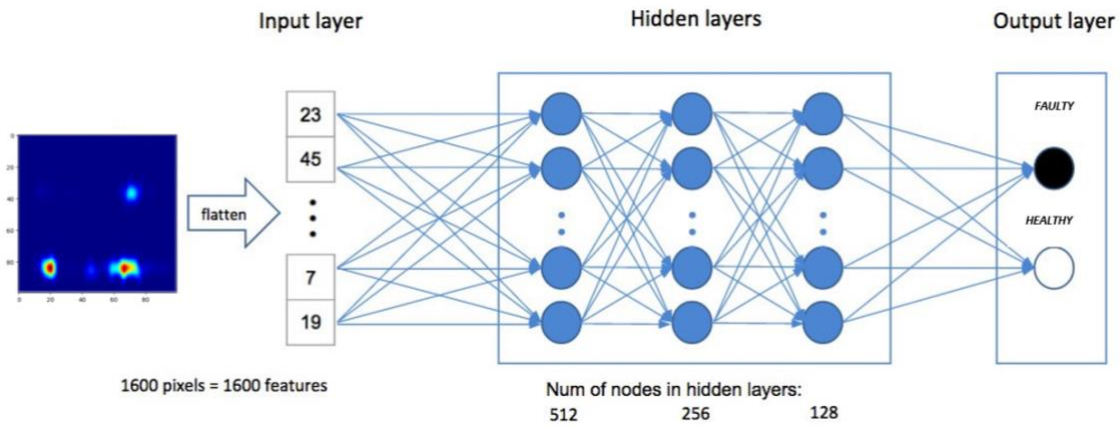


Figura 19: Object classification tramite ConvNet

Le CNN trovano grande utilità nelle applicazioni di riconoscimento oggetti mediante ricerca di pattern nelle immagini (illustrato in Figura 19), come ad esempio i veicoli a guida autonoma e le applicazioni di riconoscimento facciale.

Esse, infatti, apprendono direttamente dai dati delle immagini, utilizzando i pattern per classificare le immagini ed eliminando così la necessità dell'estrazione manuale delle feature.

Tre importanti fattori hanno reso l'impiego delle CNN necessario nel campo del DL:

- L'eliminazione della necessità dell'estrazione manuale delle feature, dacché queste vengono apprese direttamente dalla CNN;
- La produzione di risultati di riconoscimento all'avanguardia;
- Lo sfruttamento di reti preesistenti grazie all'addestramento continuo per nuove attività di riconoscimento.

Se unita ai progressi delle GPU e del calcolo parallelo, l'architettura delle CNN risulta pertanto una tecnologia chiave alla base dei nuovi sviluppi nel campo dell'AI e della visione artificiale.

3.2.1 – Descrizione architettura

L'utilizzo di decine o centinaia di layer porta a rilevare le diverse feature di un'immagine, a cui vengono applicati ogni volta filtri in diverse risoluzioni, e l'output di ogni immagine convolta viene utilizzato come input per il layer successivo. Questi filtri, che sono inizialmente feature molto semplici, come la luminosità e i bordi, assumono forme via via più complesse man mano che definiscono in modo univoco l'oggetto.

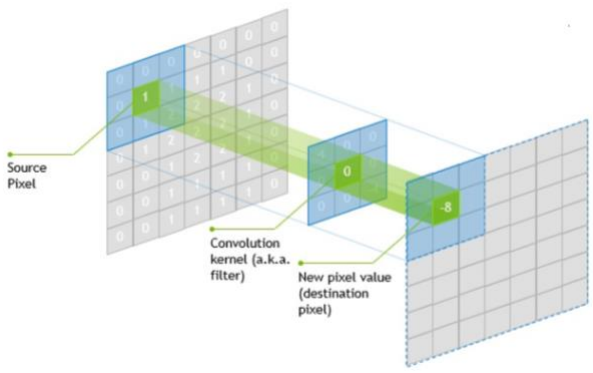


Figura 20: Convoluzione nelle CNN (fonte: towardsdatascience.com)

Un'architettura CNN è formata da una pila di livelli distinti che trasformano il volume di input in un volume di output per mezzo di strati specifici: la struttura base della rete è illustrata in Figura 21.

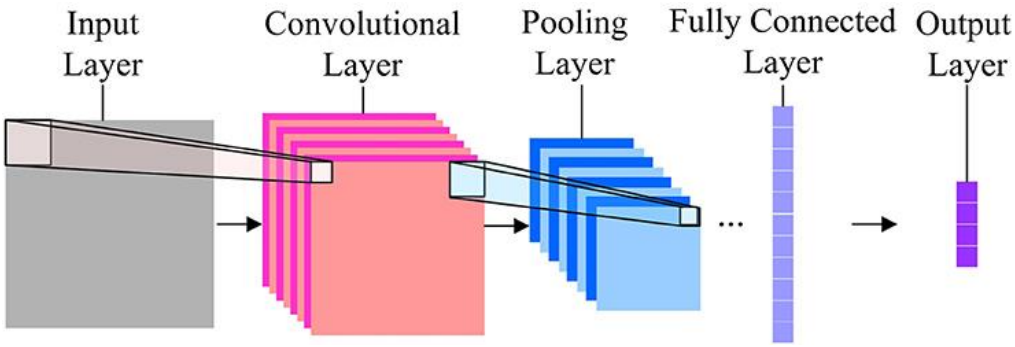


Figura 21: strati di una CNN

Strato convoluzionale: è il blocco predefinito principale di una CNN. La convoluzione applica alle immagini di input una serie di filtri convoluzionali, ognuno dei quali attiva determinate feature delle immagini. I parametri di questo livello consistono fondamentalmente in una serie di filtri appresi (detti kernel), che hanno un piccolo campo ricettivo, ma che si estendono attraverso l'intera profondità del volume di input.

Durante il passaggio in avanti, ogni filtro viene convoluto attraverso la larghezza e l'altezza del volume di input, calcolando il prodotto scalare tra le voci del filtro e l'input e producendo una mappa di attivazione bidimensionale di tale filtro. Di conseguenza, la rete apprende i filtri che si attivano quando rileva un tipo specifico di feature in una posizione spaziale nell'input. L'impilamento delle mappe di attivazione per tutti i filtri lungo la dimensione di profondità forma l'intero volume di output del layer di convoluzione.

Ogni voce nel volume di uscita può quindi essere interpretata anche come un output di un neurone che guarda una piccola regione nell'input e condivide i parametri con i neuroni nella stessa mappa di attivazione.

Quando si tratta di input di grandi dimensioni come le immagini, non è pratico collegare i neuroni a tutti i neuroni del volume precedente perché tale architettura di rete non prende in considerazione la struttura spaziale dei dati.

Le reti convoluzionali sfruttano la correlazione spaziale locale applicando un modello di connettività locale sparso tra neuroni di strati adiacenti: ogni neurone è collegato solo a una piccola regione del volume di ingresso. L'estensione di questa connettività è un iper-parametro chiamato *campo ricettivo* del neurone. Le connessioni sono locali nello spazio (lungo larghezza e altezza), ma si estendono sempre lungo l'intera profondità del volume di input. Tale architettura assicura che i filtri appresi producano la risposta più forte a un modello di input locale dal livello spaziale.

Tre iper-parametri controllano le dimensioni del volume di uscita dello strato convoluzionale: profondità, passo e zero-padding.

La *profondità* del volume di output controlla il numero di neuroni in un livello che si connettono alla stessa area del volume di input. Questi neuroni imparano ad attivare per diverse caratteristiche in ingresso. Ad esempio, se il primo livello convoluzionale prende l'immagine grezza come input, i neuroni diversi lungo la dimensione di profondità possono attivarsi in presenza di vari bordi orientati o macchie di colore.

Il *passo* controlla il modo in cui vengono allocate le colonne di profondità intorno alle dimensioni spaziali (larghezza e altezza). Quando il passo è 1 allora spostiamo i filtri di un pixel alla volta. Questo porta a campi ricettivi fortemente sovrapposti tra le colonne e anche a grandi volumi di output. Allo stesso modo, per qualsiasi numero intero $S > 0$, un passo di S fa sì che il filtro venga traslato di S unità alla volta per output. In pratica, tuttavia, passi di lunghezza $S > 3$ sono rari. I campi ricettivi si sovrappongono meno e il volume di output risultante ha dimensioni spaziali più piccole quando viene aumentata la lunghezza del passo. A volte è conveniente riempire l'ingresso con zeri sul bordo del volume di ingresso.

La dimensione di questo *padding* è un terzo iper-parametro, e fornisce il controllo delle dimensioni spaziali del volume di output. In particolare, a volte è opportuno preservare esattamente la dimensione spaziale del volume di input.

La dimensione spaziale del volume di output può essere calcolata in funzione della dimensione del volume di input W, della dimensione del campo del kernel dei neuroni del livello convoluzionale K, del passo S con cui vengono applicati e della quantità "*zero*

padding" *P* utilizzato sul bordo. La formula per calcolare quanti neuroni "si adattano" in un dato volume è data da $\frac{W-K+2P}{S} + 1$.

Se questo numero non è un intero, allora i passi non sono corretti e i neuroni non possono essere affiancati per adattarsi al volume di input in modo simmetrico. In generale, impostare lo zero padding su un valore di $P = \frac{K-1}{2}$, ovvero quando il passo è $S = 1$, assicura che il volume di input e il volume di output abbiano la stessa dimensione spazialmente. Tuttavia, non è sempre necessario utilizzare tutti i neuroni dello strato precedente. Ad esempio, una finestra di progettazione di rete neurale può decidere di utilizzare solo una parte del padding.

Livello di pooling: un altro concetto importante in una CNN è il pooling, che è una forma di down-sampling non lineare. Il pooling semplifica l'output eseguendo un sotto-campionamento non lineare, riducendo in questo modo il numero di parametri che la rete deve apprendere. Esistono diverse funzioni non lineari per implementare il pooling, tra cui *il max pooling*, il più comune, illustrato in Figura 22.

Quest'ultimo partiziona l'immagine di input in un set di rettangoli non sovrapposti e, per ogni sotto-area, restituisce il valore massimo. La posizione esatta di una feature è meno importante della sua posizione approssimativa rispetto ad altre features, ed è questa l'idea dietro all'uso di pooling nelle reti neurali convoluzionali.

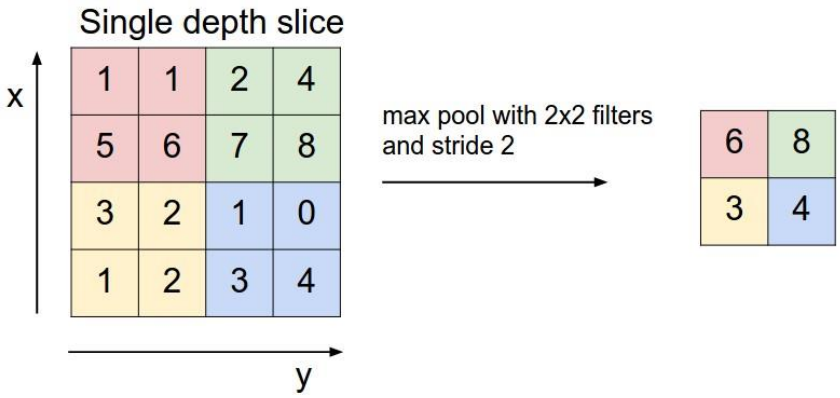


Figura 22: Max pooling

Il livello di pooling serve pertanto a ridurre progressivamente le dimensioni spaziali della rappresentazione, a ridurre il numero di parametri, il footprint di memoria e la quantità di calcolo nella rete, e quindi a controllare anche l'over-fitting. È abitudine inserire

periodicamente un livello di pooling tra livelli convoluzionali successivi in un'architettura CNN.

Il layer di pooling opera in modo indipendente su ogni sezione di profondità dell'input e la ridimensiona spazialmente. La forma più diffusa è un layer di pooling con filtri di dimensione 2x2 applicati con un passo di 2 down-samples ad ogni sezione di profondità dell'input sia lungo la larghezza che l'altezza, che scarta il 75% delle attivazioni:

$$\boldsymbol{f_{X,Y}(S) = \max_{a,b=0} S_{2X+a,2Y+b}}$$

Oltre al max pooling, si possono utilizzare altre funzioni, ad esempio *l'average pooling* o il pooling di norma L2. Il primo di questi pooling appena nominati è stato spesso utilizzato in passato, ma è stato recentemente soppiantato dal max pooling, che ha prestazioni migliori al momento. A causa della notevole riduzione delle dimensioni della rappresentazione, c'è una recente tendenza che punta all'utilizzo di filtri più piccoli o all'eliminazione totale degli strati di pooling.

Esiste inoltre il pooling "Regione di interesse" (noto anche come *RoI pooling*) che è una variante del max-pooling, in cui le dimensioni dell'output sono fisse e il rettangolo di input è un parametro. Il pooling è quindi un componente importante delle reti neurali convoluzionali per il rilevamento degli oggetti.

Livello ReLU: ReLU, che è l'abbreviazione di *rectified linear unit*, applica la funzione di attivazione $f(x) = \max(0, x)$. L'unità lineare rettificata consente un addestramento più rapido ed efficace mappando i valori negativi a zero e mantenendo solo i valori positivi. Talvolta si parla di *attivazione*, perché solo le feature attivate vengono fatte passare al layer successivo.

Per aumentare la non linearità vengono utilizzate anche altre funzioni, ad esempio la tangente iperbolica e la funzione sigmoide $\sigma(x) = (1 + e^{-x})^{-1}$. ReLU è spesso preferita ad altre funzioni perché addestra la rete neurale molte volte più velocemente senza creare danni significativi per quanto riguarda l'accuratezza nella generalizzazione.

In seguito alla determinazione delle feature dei vari layer, l'architettura di una CNN procede alla classificazione.

Livello completamente connesso: dopo diversi livelli convoluzionali e di max pooling, l'elaborazione di alto livello della rete neurale viene eseguito tramite questi layer completamente connessi (*fully connected*). Il penultimo layer è uno strato che genera un vettore di dimensioni K in cui K indica il numero delle classi che la rete sarà in grado di prevedere.

Questo vettore contiene le probabilità per ogni classe di qualsiasi immagine classificata. *Fully connected layers* connettono ogni neurone in un layer a qualsiasi neurone in un altro layer. Le loro attivazioni possono quindi essere calcolate come una trasformazione affine, con una moltiplicazione matriciale seguita da un offset di distorsione (cioè un'aggiunta vettoriale di un termine di distorsione appreso o fisso). La matrice 'appiattita' (*flattened*) passa dunque attraverso un livello completamente connesso per classificare le immagini.

Strato di perdita: specifica come il training penalizza la deviazione tra le etichette pronosticate in output e quelle vere, ed è normalmente lo strato finale di una rete neurale. Possono essere utilizzate varie funzioni di perdita adatte ai diversi compiti. La funzione di perdita *Softmax* viene utilizzata per prevedere una singola classe di K classi mutualmente esclusive. Un'altra funzione, s*igmoid cross-entropy loss*, viene utilizzata per predire i K valori di probabilità indipendenti in [0,1]. La funzione di perdita euclidea viene infine utilizzata per la regressione di etichette con valori reali .

3.2.2 – Iperparametri

Nell'ambito del Machine Learning, un iper-parametro è definito come un parametro il cui valore è stabilito prima che il processo di apprendimento abbia inizio.
I parametri che si devono dunque fornire alla rete, nel caso di reti convoluzionali per l'analisi di immagini, sono:

- Numero di filtri: poiché le dimensioni della feature map diminuiscono tanto più la rete è profonda, i layer vicini allo strato di input tenderanno ad avere meno filtri, mentre i layer successivi possono averne di più. Per equalizzare il calcolo a ogni livello, il prodotto dei valori delle feature v_a con la posizione del pixel viene mantenuto approssimativamente costante tra i livelli. Conservare più informazioni sull'input

richiederebbe la riduzione del numero totale di attivazioni (numero di feature maps per numero di posizioni in pixel) da un layer a quello successivo. Il numero di feature maps controlla direttamente la capacità e dipende dal numero di esempi disponibili e dalla complessità delle attività.

- Forma filtro: le forme di filtro comuni presenti nella letteratura variano notevolmente e vengono in genere scelte in base al set di dati scelto in input. La sfida consiste, quindi, dato un particolare dataset, nel trovare il giusto livello di granularità in modo da creare astrazioni nella corretta scala, e senza over fitting.

- Numero di epoche: un singolo passaggio della rete attraverso i dati di training è chiamato epoca. Solitamente, questi dati sono mandati in batch in modo tale da rendere più veloce l'addestramento, essendo le macchine di calcolo in grado di elaborare più di un singolo campione alla volta. Maggiore è la grandezza del batch e peggiore sarà l'accuratezza del modello, poiché quest'operazione di raggruppamento può far perdere alla rete la capacità di cogliere dettagli più fini.

3.2.3 – Limiti

Nonostante le numerose applicazioni ed i risultati raggiunti, le CNN hanno anche alcune limitazioni, tra cui:

- Necessità di grandi quantità di dati: Le reti convoluzionali richiedono un grande numero di esempi di addestramento per apprendere e riconoscere pattern specifici. In assenza di dati sufficienti, la rete potrebbe non essere in grado di generalizzare correttamente.
- Necessità di molta potenza di calcolo: le reti neurali convoluzionali richiedono molta potenza di calcolo per essere addestrate ed utilizzate, il che può renderle poco pratiche per l'uso su dispositivi mobili o embedded.
- Incapacità di modellare il comportamento dinamico dei sistemi e dei segnali, che può essere particolarmente importante nel monitoraggio strutturale, dove i segnali di vibrazione sono spesso non stazionari e dinamici.

3.3 - Spiking Neural Networks

Le reti neurali artificiali, pur prendendo ispirazione dal cervello per il proprio funzionamento, emulano l'attività cerebrale umana con certe limitazioni e

semplificazioni: infatti, le sinapsi biologiche hanno un funzionamento tendente al risparmio energetico, attivandosi solo quando ricevono un impulso da un altro neurone, e sono quindi spenti per la maggior parte del tempo, cosa che invece le ANN non fanno.

Questo funzionamento efficiente e low-power (la potenza assorbita dal cervello è di soli 20 Watt) ha dunque portato allo sviluppo recente di un nuovo tipo di elaborazione dati chiamato *neuromorphic computing* (NC), il quale impiega nuovi approcci algoritmici in grado di emulare più realisticamente la capacità di interazione del cervello con il mondo circostante. [9]

Questa nuova disciplina offre un'interessante applicazione in numerosi campi, ma in special modo in quello dei sistemi autonomi basati sull'AI, che necessitano di mantenere un'alta efficienza energetica e un apprendimento continuo.

I suddetti nuovi algoritmi prendono il nome di *spiking neural networks* o SNN, e la differenza con una tradizionale ANN risiede nel modo in cui si propaga l'informazione: queste reti operano infatti con eventi discreti che compaiono a intermittenza nel tempo, invece che usare valori che cambiano continuamente nel tempo (illustrato in Figura 23).

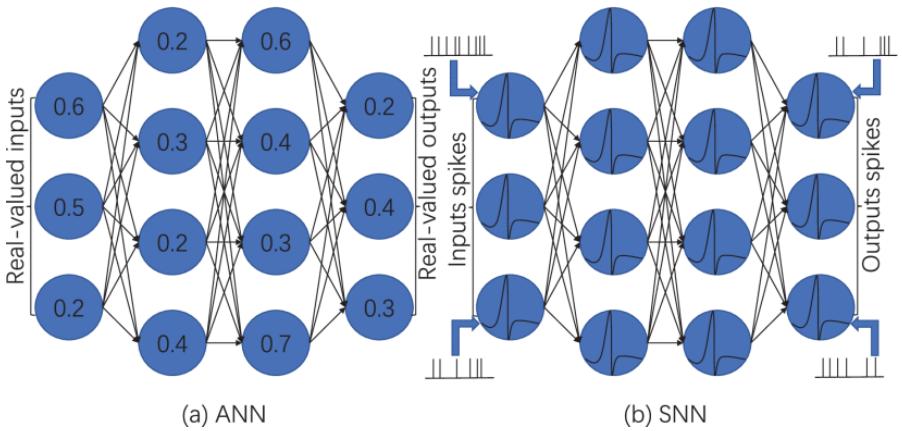


Figura 23: Modello a grafi ANN - SNN

Le SNN, dunque, ricevono in input una serie di spikes e producono altri treni di spikes in output [10]:

1. In ogni momento del tempo ogni neurone ha un valore analogo al potenziale elettrico dei neuroni biologici;
2. Il valore in un neurone varia in base all'impulso ricevuto dal neurone a monte (potrebbe aumentare o diminuire)

3. Se il valore assunto supera una certa soglia, il neurone invierà un singolo impulso a ciascun neurone a valle collegato a quello iniziale;
4. Dopo il firing (invio di impulso), il valore del neurone scenderà immediatamente, sperimentando l'analogo del periodo refrattario di un neurone biologico, per poi tornare lentamente alla suo stato di riposo.

Esistono due gruppi di metodi di base utilizzati per modellare un neurone SNN (Figura 24):

1. Metodi biofisici (complessi):
    - Hodgkin-Huxley
2. Modelli a soglia, che generano un impulso al superamento una certa soglia:
    - Leaky Integrate- and-fire (LIF)
    - Adaptive Leaky Integrate-and-fire (ALIF)

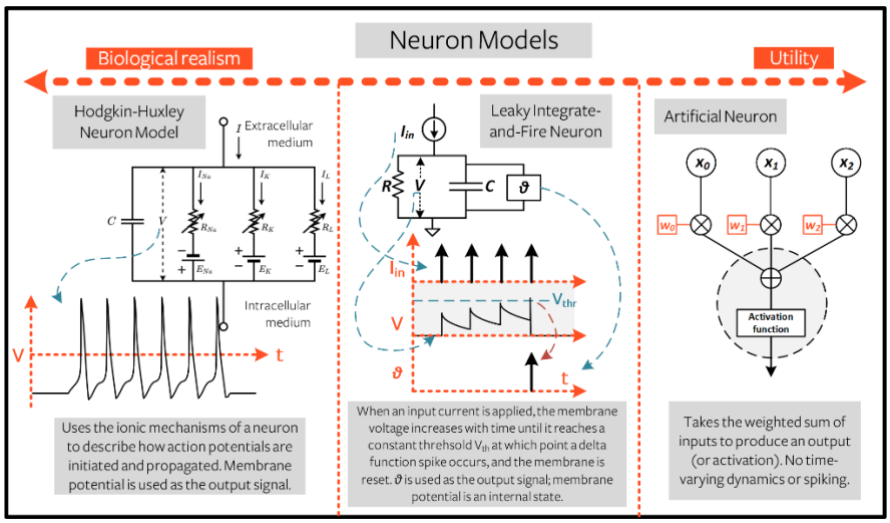


Figura 24: Confronto tra modelli di neurone (fonte: https://snntorch.readthedocs.io/en/latest/tutorials/tutorial_2.html)

3.3.1 – Modelli LIF e ALIF

Il modello più comunemente usato per un neurone SNN è il modello di soglia Leaky Integrate-and-fire [11]

Questo è l'evoluzione del modello non-LIF, introdotto da Louis Lapique nel 1907. Ogni neurone è rappresentato dal potenziale di membrana relativo V che aumenta nel tempo durante l'applicazione di una corrente di ingresso $I(t)$ secondo la relazione:

$$I(t) = C\frac{dV(t)}{dt}$$

Il potenziale di membrana, dunque, evolve fino al valore di una soglia fissata Vth: una volta raggiunta, si genera uno spike e il potenziale ritorna al valore iniziale.

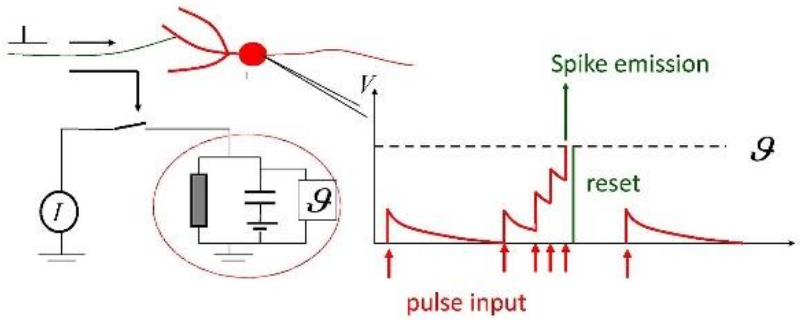


Figura 25: Modello LIF (fonte: CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=97335695)

Il neurone è circondato da una sottile membrana che isola la soluzione conduttiva all'interno del neurone dal mezzo extracellulare, e questo permette di modellarlo elettricamente come un condensatore (illustrato in Figura 25).
Pur essendo solitamente impermeabile agli ioni entranti e uscenti dalla cellula, esistono canali specifici nella membrana che vengono attivati per permettere l'iniezione di corrente nel neurone, e questo movimento di carica è modellato elettricamente da un resistore.
È possibile modellare il neurone con l'equazione differenziale lineare, rappresentante un circuito RC:

$$\frac{CdV(t)}{dt} = I(t) - \frac{V(t)}{R}$$

È necessario però ottenere una rappresentazione discreta del neurone LIF; dunque, si impiega il metodo di Eulero per risolvere l'equazione differenziale:

$$V(t + \Delta t) = V(t) + \frac{\Delta t}{RC}\left(-V(t) + RI(t)\right)$$

che può essere semplificata nella seguente espressione ricorsiva:

$$V(t + \Delta t) = \beta V(t) + WX(t + 1) - R(t)$$

dove β rappresenta il decadimento, WX(t) rappresenta la corrente in ingresso e R(t) la funzione di reset.

Affinché un neurone generi ed emetta i propri picchi all'uscita, è necessario definire una soglia: se il potenziale di membrana la supera, verrà generato lo spike di tensione:

$$S(t) = \begin{cases} 1, & if\ V(t) > V_{th} \\ 0, & otherwise \end{cases}$$

Dopo ogni spike, il neurone ripristina il proprio potenziale a un valore di riposo e riavvia il processo di integrazione.

Al contrario, i neuroni ALIF (Adaptive Leaky Integrate-and-Fire) sono un modello più complesso che incorpora un meccanismo adattativo per regolare il comportamento del neurone in base alla forza e al timing dei segnali in ingresso. Anche i neuroni ALIF integrano i segnali in ingresso nel tempo, ma la loro soglia non è fissa e può cambiare in base alla storia recente dell'attività del neurone. Questo comportamento adattativo consente ai neuroni ALIF di mostrare pattern di attivazione più complessi rispetto ai neuroni LIF e di rispondere meglio a pattern di input dinamici.

Complessivamente, i neuroni ALIF forniscono una rappresentazione più accurata del comportamento dei neuroni reali nel cervello, ma richiedono anche più risorse computazionali e possono essere più impegnativi da implementare rispetto ai neuroni LIF.

3.2.2 – Encoding

Per generare una sequenza di picchi a partire dai segnali di input la rete può scegliere due approcci diversi: uno detto current encoding e uno chiamato events encoding [12]:

- In una SNN Current-Driven, il segnale di ingresso è costante per l'intero intervallo di inferenza
- In una SNN Events-Driven, il segnale è prima codificato come spike e poi usare questi come input alla rete

I due metodi sono rappresentati in Figura 26.

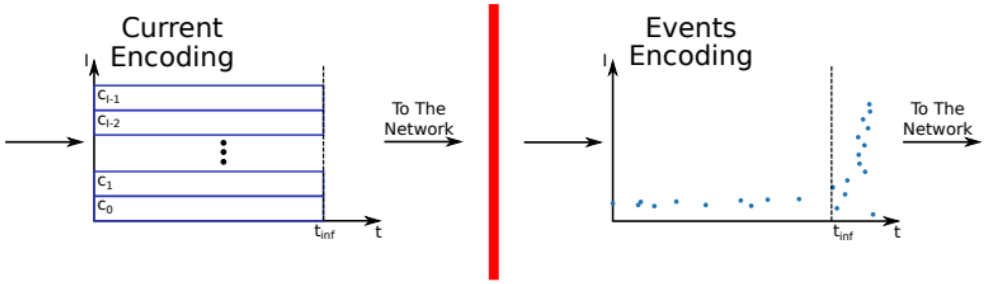


Figura 26: Encoding (fonte: doi.org/10.3390/fi13080219)

Una possibile codifica di tipo Event Encoding è data dal metodo Rank-Order Coding (ROC), dove ogni neurone può generare uno spike al massimo una volta sola per ogni campione. Ogni coefficiente di input è dunque codificato come un intervallo di spike: maggiore è il coefficiente (ovvero più è energetico) e minore sarà l'intervallo di spike.

Imponendo come iper-parametro il tempo di inferenza per ciascun campione, è possibile così generare un vettore contenente valori capaci di discriminare neuroni energetici arrecanti informazione (con $t > t_{infer}$) e quelli relativi a coefficienti poco energetici privi di contenuto rilevante.
Un esempio di algoritmo ROC applicato al case-study di questo elaborato è riportato di seguito: è necessario generare una matrice della stessa dimensione dell'input (nel caso riportato una matrice (5651,129)) contenente elementi pari a 0 nel caso in cui il tempo di inferenza fosse troppo lungo; viceversa riuslterebbe 1 se fosse in grado di soddisfare i requisiti temporali (e quindi generare lo spike).

```
def encoding(I,tinf,spectrum):
  T=[[0 for col in range(5651)]for row in range(I)];
  for s in range(0,5651):
    S=[];
    S= spectrum[:,s];
    m= np.min(S);
    M= np.max(S);
    for i in range(0,I):
      time= (S[i]-m)/(M-m);
      tcheck= np.around((1/time),decimals=1);
      if (tcheck <= tinf):
        T[i][s]= (tcheck);
      else:
        T[i][s]= 0;
  return T;
```

```
Spikes= np.empty((129,20,5651))
M= Mat[:,:];

for j in range(0,5651):

 for k in range(0,20):
  for i in range(0,129):
    if M[i][j] == k:
      Spikes[i][k][j]= 1;
    else:
      Spikes[i][k][j] = 0;
```

Listing 1: ROC Encoding

3.2.4 – Vantaggi e Svantaggi rispetto alle tradizionali NN

Riassumendo, sono molteplici i vantaggi che le SNN possiedono in confronto alle NN tradizionali:

a. Le SNN di solito richiedono meno neuroni rispetto alle NN tradizionali per risolvere gli stessi task;
b. Ridotto consumo di potenza, specialmente se implementate su HW dedicato;
c. È necessario addestrare solo i neuroni di uscita;
d. Sono reti molto veloci, data la natura discreta (e non più continua come nelle ANN) degli impulsi che elaborano;

In particolare, le SNN possono essere preferibili alle CNN per compiti di rilevamento di anomalie nel monitoraggio strutturale essendo in grado di modellare il comportamento dinamico dei sistemi e dei segnali.

Sfortunatamente, due importanti problemi affliggono le SNN:

a. L'addestramento è complesso, non essendoci ad oggi un metodo specifico per questo compito;
b. Difficoltà di realizzazione, essendo necessario modellare dinamiche più complesse e computazionalmente costose.

Capitolo 4

Validazione Sperimentale

Nel capitolo precedente sono state introdotte le reti neurali: ora è necessario discutere il fine del loro impiego in questo lavoro, ovvero la classificazione degli input della rete.
Lo studio discusso qui di seguito è relativo alla realizzazione di una rete neurale spiking (SNN) che renda possibile l'individuazione dei guasti strutturali con soddisfacente accuratezza.

La maggioranza degli algoritmi di DL utilizzano infatti il *Gradient Descent* come funzione di ottimizzazione per la backpropagation; questa scelta comporta però come condizione necessaria che la rete da ottimizzare sia differenziabile. Tuttavia, nel NC, sono spesso impiegati neuroni spiking, che però hanno natura non differenziabile: da qui la necessità di trovare un modo per applicare i metodi di DL alle reti spiking.

Il metodo impiegato, proposto da Hunsberger e Eliasmith (2016), comporta la conversione di una rete neurale convoluzionale in una SNN, adattando il modo in cui le informazioni vengono elaborate e rappresentate nella rete [13].
In particolare, utilizza un'approssimazione differenziabile dei neuroni spiking durante il processo di addestramento e gli effettivi neuroni spiking durante quello di inferenza.

Per poter costruire questa rete neurale viene naturale procedere in modo sequenziale; risulta pertanto necessario seguire un piano logicamente ben strutturato, che può essere riassunto dai seguenti punti (Figura 27):

1. *Signal Processing* del Dataset;
2. Elaborazione tramite semplice CNN
3. Conversione CNN → SNN
4. *Training* della SNN;
5. *Verifica delle prestazioni della SNN.*

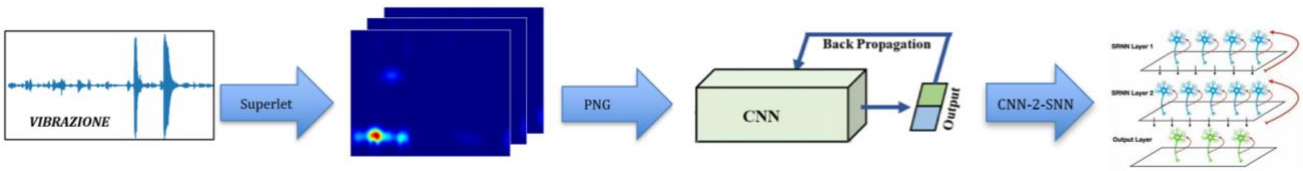


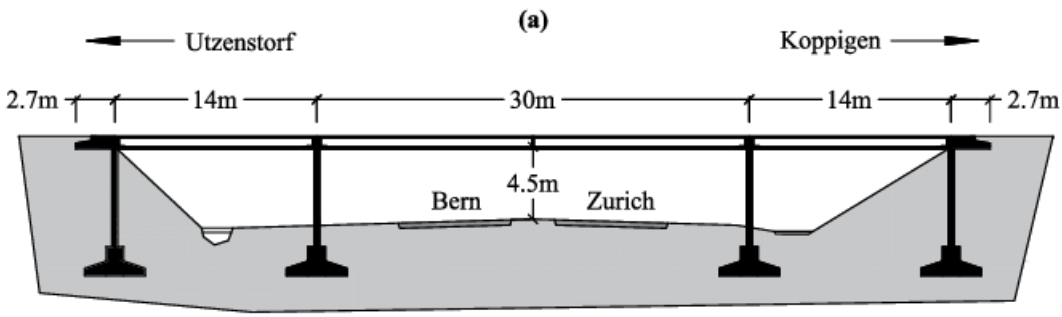
Figura 27: Workflow dello studio

Saranno pertanto individuati in primo luogo l'approccio al problema, gli strumenti utilizzati per la sua realizzazione e in seguito i risultati progressivamente ottenuti.

4.1 Dataset Ponte Z24, Ku Leuven

Per questo lavoro è stato scelto il dataset proveniente dai sensori del ponte Z24 nel cantone di Berna vicino a Solothurn, Svizzera. Il suddetto ponte, costruito nel 1963, faceva parte della rete stradale che collegava Koppigen con Utzenstorf in Svizzera (Figura 28).

La struttura, a tre campate con lunghezza di circa 14, 30 e 14 m, attraversava l'autostrada A1 Berna-Zurigo con un angolo leggermente obliquo [14].



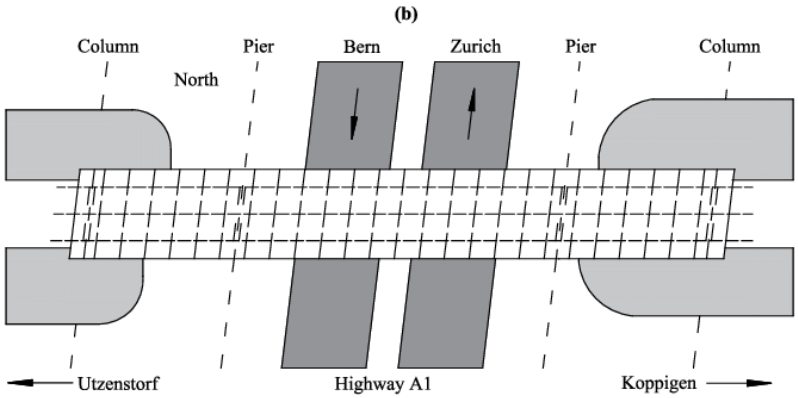


Figura 28: Ponte Z24, front & top view (from https://bwk.kuleuven.be/bwm/z24)

Le condizioni del ponte erano relativamente buone, ma il ponte dovette essere demolito alla fine del 1998 per consentire la costruzione di nuovi binari ferroviari. Tuttavia, prima della demolizione, sono stati condotti due campagne sperimentali al fine di creare un nuovo dataset per il SHM: uno a lungo termine e uno a breve termine.

L'anno prima della demolizione è stato effettuato il primo test di monitoraggio continuo a lungo termine, il cui scopo era quello di quantificare la variabilità ambientale della dinamica del ponte monitorando con cinque sensori tutte le variabili ambientali più rilevanti: uno scenario è illustrato in Figura 29.

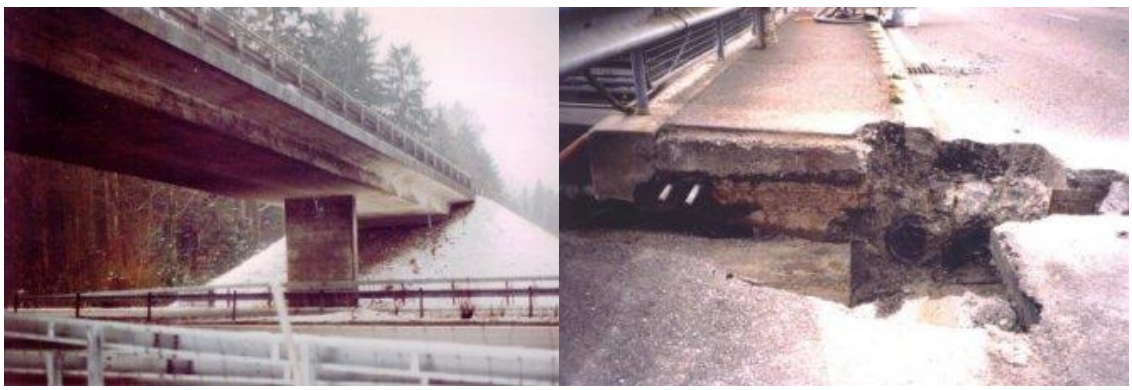


Figura 29: Danneggiamento Z24: scenario "Fallimento delle teste di ancoraggio" (https://bwk.kuleuven.be/bwm/z24

Un secondo test di monitoraggio progressivo a breve termine ha avuto luogo invece un mese prima della demolizione totale, e prevedeva la simulazione di danni indotti: prima e

dopo l’applicazione di ciascun scenario di danno il ponte è stato soggetto a un test di vibrazione. [15]

Un totale di 65536 campioni sono stati raccolti a una frequenza di campionamento di 100 Hz, utilizzando un filtro anti-aliasing con una frequenza di taglio di 30 Hz.

Essendo l'informazione utile confinata entro i primi 15 Hz, è stato però eseguito un sotto-campionamento a 50 Hz.

Di questi campioni, dunque, solo 32768 sono da considerarsi utili, e in totale sono state osservate 5651 istanze nel tempo, della durata di 11 min di osservazione. La rappresentazione dei dati ricavati da un sensore è mostrata in Figura 30.

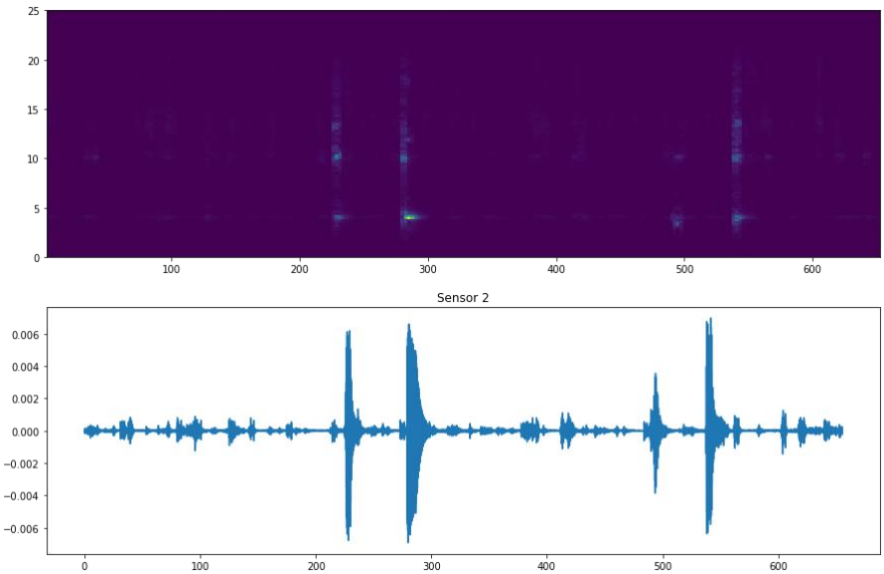


Figura 30: Dati di accelerazione e PSD del sensore 2

4.2 – Signal Processing

L’obiettivo del lavoro consiste nel mandare in ingresso alla rete neurale un dataset di uguale formato rispetto a quelli normalmente usati in ingresso alle CNN, come per esempio il database MNIST (Modified National Institute of Standards and Technology). Infatti, il modello Keras impiegato per la costruzione della CNN accetta unicamente come input immagini con altezza, larghezza e canali di colore (RGB, scala di grigi).

Al fine di realizzare tale dataset è stato necessario effettuare un’elaborazione e ottimizzazione dei dati provenienti dagli accelerometri secondo tre step:

a. Estrazione di finestre energetiche rilevanti secondo una soglia
b. Ottimizzazione della risoluzione di ogni finestra per mezzo di Superlet
c. Salvataggio delle immagini in alta risoluzione
d. Creazione di label associate alle immagini

Al fine di ottenere immagini ad alta risoluzione per il training della rete è stato necessario estrarre solamente alcune finestre energetiche di interesse: per fare ciò è stata utilizzata una soglia di 0.03 $\frac{m}{s^2}$, scelta perché, analizzando gli spettrogrammi relativi ai dati ottenuti dai sensori, è risultata idonea per discriminare la parte di segnale rilevante da quella ridondante. Di seguito, in Figura 31, è riportato un esempio di finestra.

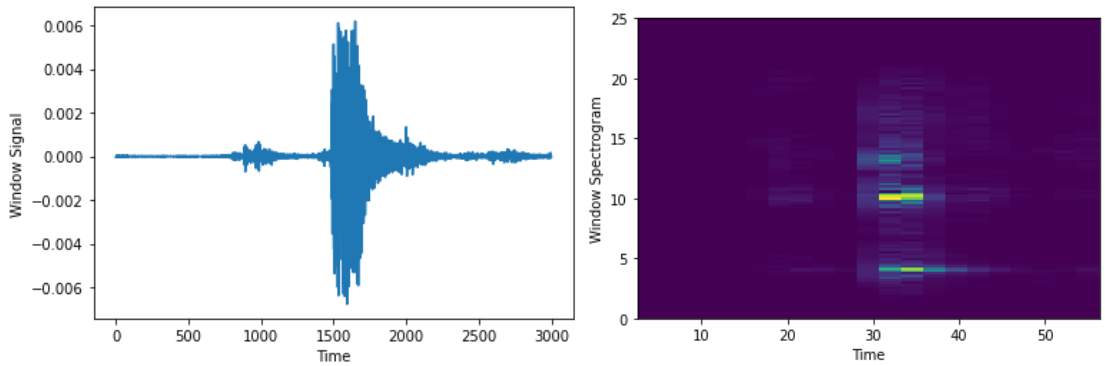


Figura 31: Esempio di Finestra e relativo spettrogramma

Inoltre, per ogni finestra è stato associato un vettore etichetta che associ a ciascuna finestra estratta un valore 0 o 1 sulla base della presenza o meno del guasto e che servirà successivamente per il training della rete neurale.

Di seguito, è riportato il codice Python impiegato per l'estrazione e il salvataggio delle finestre:

```
import scipy.io as scio
import numpy as np
import matplotlib.pyplot as plt


def main():
    # ---------------------- Load data from sensor 2
    sensor2 = scio.loadmat('sensor2.mat')
    data = sensor2['data_sensor2']

    DataSize = data.shape

    # ---------------------- Extract only energetic windows
    # Set threshold
    th = 0.003
```

```
    # Observation window length
    winLenPre = 2
    winLenPost = 15
    # Sampling frequency [Hz]
    Fs = 50
    dt = 1 / Fs

    sig_w = np.zeros((winLenPre + winLenPost) * Fs)
    sig_w = sig_w.T

    # Time vector for signal plotting
    tvec = np.linspace(0, DataSize[0] * dt, DataSize[0])

    # Variable enabling/disabling plot
    plot_fig = 0

    idx_dam = 4923
    data_label = []

    for idx_col in range(5651):

        if plot_fig == True:
            plt.figure()
            plt.plot(tvec, data[:, idx_col])

        for idx_row in range(DataSize[0]):
            while idx_row < 32768:
                if data[idx_row, idx_col] < th:
                    idx_row = idx_row + 1
                else:
                    if idx_row < winLenPre * Fs:

                          sig_temp = (data[0:(winLenPre + winLenPost) *
Fs, idx_col])
                        sig_w = np.vstack((sig_w, sig_temp))

                        # Update indexes
                        idx_row = (winLenPre + winLenPost) * Fs

                        # Update label vector
                        if idx_col <= idx_dam:
                            data_label = np.append(data_label, 0)
                        else:
                            data_label = np.append(data_label, 1)

                    elif (DataSize[0] - idx_row) < winLenPost * Fs:

                           sig_temp = (data[DataSize[0] - (winLenPre +
winLenPost) * Fs:DataSize[0], idx_col])
                        sig_w = np.vstack((sig_w, sig_temp))

                        # Update indexes
                        idx_row = DataSize[0]

                        # Update label vector
                        if idx_col <= idx_dam:
                            data_label = np.append(data_label, 0)
                        else:
                            data_label = np.append(data_label, 1)

                    else:
```

```
                        sig_temp = data[idx_row - winLenPre * Fs:idx_row
+ winLenPost * Fs, idx_col]
                        sig_w = np.vstack((sig_w, sig_temp))

                        # Plot extracted window
                        tstart = (idx_row - winLenPre * Fs) * dt
                        tstop = (idx_row + winLenPost * Fs) * dt
                        idx_row = idx_row + winLenPost * Fs
                        tvecw = np.linspace(tstart, tstop, (winLenPre +
winLenPost) * Fs)

                        # Update label vector
                        if idx_col <= idx_dam:
                            data_label = np.append(data_label, 0)
                        else:
                            data_label = np.append(data_label, 1)

                        if plot_fig == True:
                            plt.plot(tvecw, sig_temp, color='r')
                            plt.show()

            if idx_row >= 32768:
                break

        if plot_fig == True:
            plt.xlabel('Time samples')
            plt.ylabel('acc')
            plt.title('Test ' + str(idx_col))
            plt.xlim([tvec[0], tvec[-1]])

    DataSizeW = sig_w.shape

    # Transpose sig_w to fit a column-wise organization
    sig_w = sig_w[2:DataSizeW[0], :]
    # Return extracted windows
    #return sig_w

    # Cast signal iin a column-wise format
    sig_w = sig_w.T
    # Save data and export as csv
    #np.savetxt('sig_w.csv', sig_w, delimiter=' ,')
    np.savetxt('sig_w_data_label.csv', data_label, delimiter=',')


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
 main()
```

Listing 2: Script Python per l'estrazione delle finestre energetiche

È stato osservato che, impiegando gli spettrogrammi o la CWT per la rappresentazione dei segnali raccolti anche dopo la finestratura, la risoluzione in frequenza raggiunta dalle immagini ottenute era di 0.5 Hz: essendo le variazioni in frequenza da rilevare dell'ordine di 0.01 Hz, questa rappresentazione non era sufficiente per il successivo training di una rete neurale.

Al fine di risolvere questo problema è stata utilizzata la trasformata Superlet adattiva, in grado di garantire la risoluzione richiesta sia nel dominio temporale che frequenziale (Figura 32)

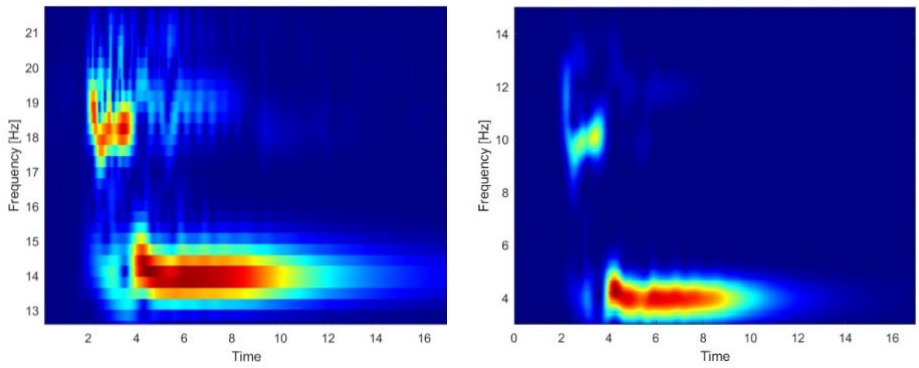


Figura 32: Evoluzione della risoluzione (CWT a sinistra, Superlet a destra)

Di seguito è riportato lo script MATLAB dopo l'applicazione della Superlet alle finestre:

```
% Extract Spectrogram

sig_w = readmatrix('C:\PycharmProjects\snn\sig_w.csv');

% ---> Superlet
Fs=50;
fstart = 2;
fend = 15;
srord = [1, 30];
freq = linspace(fstart,fend,1024);


%mancano quelle da 15000 compreso a 29999
for i=1:36725
    sig= sig_w(:,i);
    time_sig= (0:1:numel(sig)-1)*1/Fs;
    wtr = aslt(detrend(sig), Fs, freq, 3, srord, 0);
    figure(i)
    set(gcf,'visible','off')
    im=imagesc(time_sig, freq, wtr);
    axis off
    colormap jet;
    set(gca, 'ydir', 'normal');
    xlabel('Time')
    ylabel('Frequency [Hz]')
    fname = 'C:\PycharmProjects\snn\Spettrogrammi\class_h';
     filename = ['spectrogram',num2str(i),'.png']; % plot
spectrogram1.png etc.
    exportgraphics(gca, fullfile(fname, filename))
```

```
    saveas(img, fullfile(fname, filename),'png')
end
```

Listing 3: Script Matlab per l'image processing con Superlet

Il dataset MNIST è composto da 70000 immagini in bianco e nero di formato 28x28 pixel quadrati.
Le immagini ottenute su MATLAB hanno dimensione 682x539 pixel quadrati, e di conseguenza ci aspetteremmo nel primo layer della CNN un numero di nodi pari a 574000 nodi, con altrettante connessioni (più di 329 miliardi), un numero considerevolmente superiore (e irrealistico) a quelle ottenibile con database a bassa risoluzione come il MNIST.

È necessario, dunque, poter attuare una conversione di formato per consentire un'elaborazione più agevole alla rete senza però perdere eccessivamente la risoluzione ottenuta.

Avendo un dataset corposo di dati, è possibile dividerlo in training e testing set in modo tale da avere il 70% delle immagini per il training di un modello e il restante 30% per valutarne le prestazioni per quanto concerne la classificazione in 2 classi: struttura sana/guasta.

Inoltre, avendo il dataset ottenuto uno squilibrio tra le due classi (solo il 7% delle etichette fanno riferimento a dati classificabili come 'guasti'), una ripartizione bilanciata di queste label tra i due set è risultata necessaria per poter garantire un allenamento più affidabile della rete.

La suddivisione è riportata nel grafico in Figura 33:

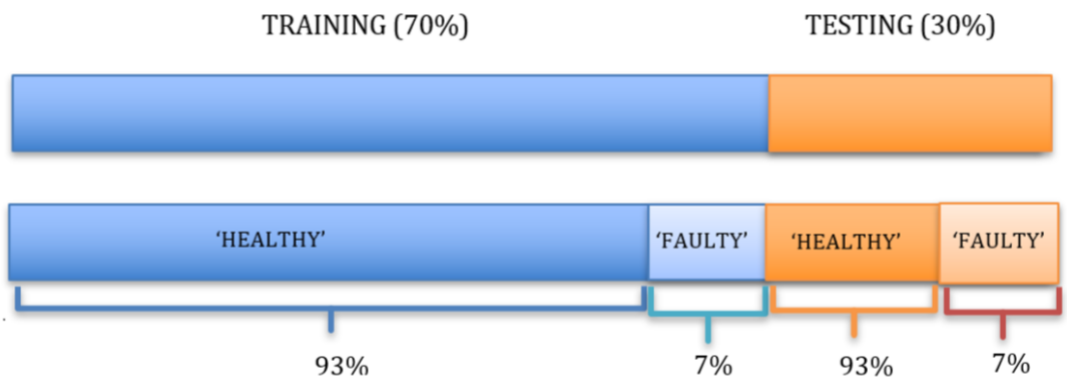


Figura 33: Suddivisione del Dataset

Il dataset così ottenuto, illustrato in Figura 34, è dunque pronto per essere passato in ingresso alla rete neurale.

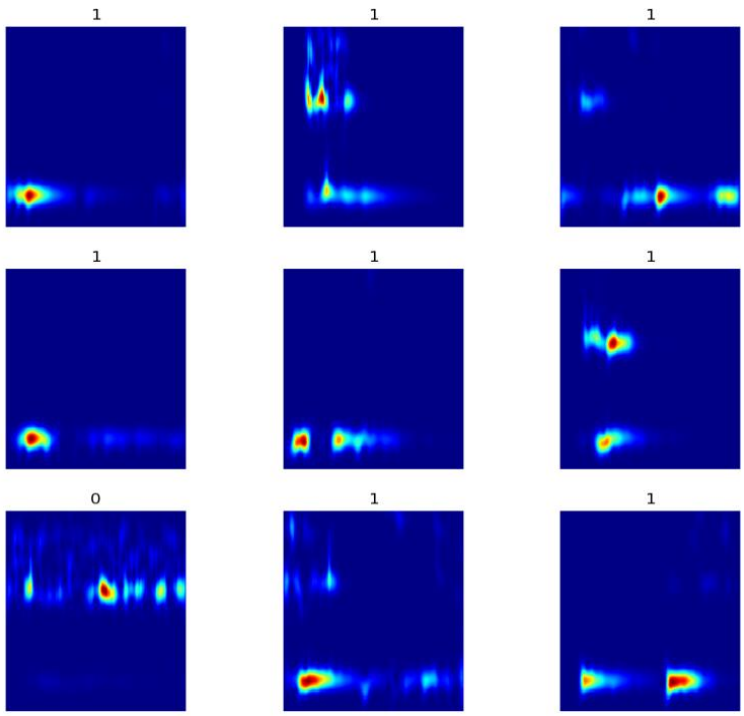


Figura 34: Dataset TF-Z24

4.3 Anomaly Detection a partire dagli Spettrogrammi

Tre sono stati gli strumenti principali adoperati in questo lavoro: TensorFlow, Keras e Nengo.

TensorFlow è una piattaforma open source per l'apprendimento automatico dotato di un ecosistema completo e flessibile di strumenti e risorse utili nella realizzazione di algoritmi per il ML da utilizzare in attività come il riconoscimento/classificazione delle immagini e l'elaborazione del linguaggio naturale [16].
Keras è invece un'API di alto livello (interfaccia di programmazione delle applicazioni) che può utilizzare le complesse funzioni di TensorFlow come back-end.Nengo, infine, è una libreria Python per costruire e testare reti neurali, il cui pacchetto NengoDL consente la conversione di reti non-spiking definiti in framework diversi da Nengo (come Keras) in reti spiking [17].

Figura 35: Libreria Python per la conversione CNN-SNN

4.3.1 – Costruzione della CNN

Una volta ottenuto il dataset completo, quest'ultimo è stato caricato in ambiente Keras per la costruzione di un modello di rete neurale convoluzionale molto semplice per l'allenamento dei dati [17].

Essendo la mole di dati abbondante, è risultato necessario ridimensionare le immagini durante il caricamento così da snellire i tempi di allenamento della rete.
Le immagini di dato sono state portate inizialmente a una dimensione di 100 pixel x 100 pixel, e in un momento successivo a 50x50, e quindi sono state paragonate le performance in accuratezza.

Dopo un allenamento a 5 epochs la rete ha raggiunto un'accuracy del 92,73%, simile a quella ottenuta con 50 pixel x 50 pixel, con la differenza che nel secondo caso la rete ha impiegato la metà del tempo per effettuare il training. Si è quindi scelto di procedere con la seconda opzione per il resto del lavoro.

Diverse prove sono state svolte per poter valutare un giusto compromesso tra accuratezza, durata del training e semplicità della rete.
In particolare, la rete è stata allenata usando inizialmente 32 filtri, per poi diminuire la quantità progressivamente fino a 8.

La rete convoluzionale impiegata è riportata di seguito:

```
# input
inp = tf.keras.Input(shape=(50, 50, 3))

# convolutional layers
conv0 = tf.keras.layers.Conv2D(
    filters=32,
    kernel_size=3,
    activation=tf.nn.relu,
)(inp)

conv1 = tf.keras.layers.Conv2D(
    filters=64,
    kernel_size=3,
    strides=2,
    activation=tf.nn.relu,
)(conv0)

# fully connected layer
flatten = tf.keras.layers.Flatten()(conv1)
dense = tf.keras.layers.Dense(units=2)(flatten)

model = tf.keras.Model(inputs=inp, outputs=dense)
model.summary()
```

Listing 4: la CNN

Successivamente, per allenarla in modo tale da favorire la conversione a Spiking Neural Net, si è passati in ambiente Nengo. Per fare questo è stata impiegata la funzione [18]:

```
import nengo
import nengo_dl

converter = nengo_dl.Converter(model)
```

Questo tool è progettato per automatizzare il più possibile la traduzione da Keras a Nengo. Infine, per allenare il modello Nengo di CNN si impiegano standard neuroni ReLU:

```
do_training = True
if do_training:
    with nengo_dl.Simulator(converter.net, minibatch_size=200) as sim:
        # run training
        sim.compile(
            optimizer=tf.optimizers.Adam(0.001),
            loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=[tf.metrics.sparse_categorical_accuracy],
        )
        sim.fit(
```

```
            {converter.inputs[inp]: train_images},
            {converter.outputs[dense]: train_labels},
            validation_data=(
                {converter.inputs[inp]: test_images},
                {converter.outputs[dense]: test_labels},
            ),
            epochs=2,
        )

        # save the parameters to file
        sim.save_params("./keras_to_snn_params")
```

Listing 5: Allenamento pesi della CNN in ambiente Keras

4.2.2 – Conversione CNN da ambiente Keras a Nengo

Avendo ora tutti gli strumenti, è possibile procedere alla conversione della semplice rete convoluzionale a rete Spiking. È necessario definire una funzione che costruisca la rete e carichi i pesi precedentemente salvati:

```
def run_network(
    activation,
    params_file="keras_to_snn_params",
    n_steps=30,
    scale_firing_rates=1,
    synapse=None,
    n_test=400,
):
    # convert the keras model to a nengo network
    nengo_converter = nengo_dl.Converter(
        model,
        swap_activations={tf.nn.relu: activation},
        scale_firing_rates=scale_firing_rates,
        synapse=synapse,
    )

    # get input/output objects
    nengo_input = nengo_converter.inputs[inp]
    nengo_output = nengo_converter.outputs[dense]

    # add a probe to the first convolutional layer to record activity.
    # we'll only record from a subset of neurons, to save memory.
    sample_neurons = np.linspace(
        0,
```

```
        np.prod(conv0.shape[1:]),
        1000,
        endpoint=False,
        dtype=np.int32,
    )
    with nengo_converter.net:
        conv0_probe =
nengo.Probe(nengo_converter.layers[conv0][sample_neurons])

    # repeat inputs for some number of timesteps
    tiled_test_images = np.tile(test_images[:n_test], (1, n_steps, 1))

    # set some options to speed up simulation
    with nengo_converter.net:
        nengo_dl.configure_settings(stateful=False)

    # build network, load in trained weights, run inference on test
images
    with nengo_dl.Simulator(
        nengo_converter.net, minibatch_size=10, progress_bar=False
    ) as nengo_sim:
        nengo_sim.load_params(params_file)
        data = nengo_sim.predict({nengo_input: tiled_test_images})

    # compute accuracy on test data, using output of network on
    # last timestep
    predictions = np.argmax(data[nengo_output][:, -1], axis=-1)
    accuracy = (predictions == test_labels[:n_test, 0, 0]).mean()
    print(f"Test accuracy: {100 * accuracy:.2f}%")

    # plot the results
    for ii in range(3):
        plt.figure(figsize=(12, 4))

        plt.subplot(1, 3, 1)
        plt.title("Input image")
        plt.imshow(test_images[ii, 0].reshape((28, 28)), cmap="gray")
        plt.axis("off")

        plt.subplot(1, 3, 2)
        scaled_data = data[conv0_probe][ii] * scale_firing_rates
        if isinstance(activation, nengo.SpikingRectifiedLinear):
            scaled_data *= 0.001
            rates = np.sum(scaled_data, axis=0) / (n_steps *
nengo_sim.dt)
            plt.ylabel("Number of spikes")
        else:
            rates = scaled_data
            plt.ylabel("Firing rates (Hz)")
        plt.xlabel("Timestep")
        plt.title(
            f"Neural activities (conv0 mean={rates.mean():.1f} Hz, "
            f"max={rates.max():.1f} Hz)"
        )
```

```
        plt.plot(scaled_data)

        plt.subplot(1, 3, 3)
        plt.title("Output predictions")
        plt.plot(tf.nn.softmax(data[nengo_output][ii]))
        plt.legend([str(j) for j in range(10)], loc="upper left")
        plt.xlabel("Timestep")
        plt.ylabel("Probability")

        plt.tight_layout()
```

Si attiva la rete allenata con la seguente funzione:

```
run_network(activation=nengo.RectifiedLinear(), n_steps=10)
```

Adesso che è stata creata la versione non-spiking della rete in Nengo, è necessario convertirla in spikes sostituendo tutte le funzioni di attivazione ReLU con funzioni di attivazione spiking come `nengo.SpikingRectifiedLinear`:

```
run_network(activation=nengo.SpikingRectifiedLinear())
```

4.2.3 – Miglioramento performance SNN

Con questa conversione naïve si può ottenere bassa accuratezza se la CNN di partenza presenta una struttura troppo elementare; quindi, è necessario procedere con ulteriori step per migliorare la performance del modello.

Un modo per fare ciò è quello di effettuare uno *smoothing* attraverso l'uso di filtri sinaptici. La sinapsi predefinita utilizzata in Nengo è un filtro passa-basso e il valore che assegno a tale parametro nella rete diventa la costante temporale del filtro passa-basso che verrà applicato all'uscita di tutti i neuroni.

In particolare, sono stati esplorati tre valori per questi filtri sinaptici.

```
for s in [0.001, 0.005, 0.01]:
    print(f"Synapse={s:.3f}")
    run_network(
        activation=nengo.SpikingRectifiedLinear(),
        n_steps=60,
        synapse=s,
    )
    plt.show()
```

L’aggiunta del filtro sinaptico attenua l'output del modello e quindi ne migliora la precisione.

Tuttavia, con più filtri sinaptici occorre processare le immagini di input per un periodo di tempo più lungo, che richiede più tempo per simulare e aggiunge più latenza alle previsioni del modello (poiché i filtri rendono l'output meno reattivo ai rapidi cambiamenti nell'input). Questo è un compromesso comune nelle reti spike (latenza rispetto alla precisione).
Ma è importante tenere presente che questo problema di latenza è esagerato dal fatto che la rete sta elaborando un insieme disconnesso di input discreti (immagini), piuttosto che un flusso continuo di dati di input.

In alternativa, è possibile migliorare l’accuratezza della SNN aumentando i tempi di firing dei neuroni. È bene però ricordare che avere neuroni che generano spikes più frequentemente significa aggiornare più spesso il segnale di uscita, e così facendo il modello assume un comportamento sempre più simile a quello del modello Nengo non spiking della rete.

```
for scale in [2, 5, 10]:
    print(f"Scale={scale}")
    run_network(
        activation=nengo.SpikingRectifiedLinear(),
        scale_firing_rates=scale,
        synapse=0.01,
    )
      plt.show()
```

Questo comportamento è ottenuto applicando una scala lineare all’ingresso di ogni neurone, e quindi dividere l’output per lo stesso fattore scelto. Il vantaggio rispetto al metodo precedente è che consente il raggiungimento di una buona accuratezza senza aggiungere latenza al modello.

Una seconda alternativa è quella di ottimizzare direttamente i tempi di firing durante il training della CNN non spiking aggiungendo funzioni di perdita che calcolino l’errore

quadratico medio (MSE) tra le uscite di ogni strato convoluzionale e dei tassi di firing target specificabili.

```
# we'll encourage the neurons to spike at around 250Hz
target_rate = 250

# convert keras model to nengo network
converter = nengo_dl.Converter(model)

# add probes to the convolutional layers, which
# we'll use to apply the firing rate regularization
with converter.net:
    output_p = converter.outputs[dense]
    conv0_p = nengo.Probe(converter.layers[conv0])
    conv1_p = nengo.Probe(converter.layers[conv1])

with nengo_dl.Simulator(converter.net, minibatch_size=200) as sim:
    # add regularization loss functions to the convolutional layers
    sim.compile(
        optimizer=tf.optimizers.RMSprop(0.001),
        loss={
                                                                output_p:
tf.losses.SparseCategoricalCrossentropy(from_logits=True),
            conv0_p: tf.losses.mse,
            conv1_p: tf.losses.mse,
        },
        loss_weights={output_p: 1, conv0_p: 1e-3, conv1_p: 1e-3},
    )

    do_training = False
    if do_training:
          # run training (specifying the target rates for the
convolutional layers)
        sim.fit(
            {converter.inputs[inp]: train_images},
            {
                output_p: train_labels,
                      conv0_p:  np.ones((train_labels.shape[0],  1,
conv0_p.size_in))
                * target_rate,
                      conv1_p:  np.ones((train_labels.shape[0],  1,
conv1_p.size_in))
                * target_rate,
            },
            epochs=10,
        )

        # save the parameters to file
        sim.save_params("./keras_to_snn_regularized_params")
    else:
        # download pretrained weights
        urlretrieve(
            "https://drive.google.com/uc?export=download&"
            "id=1xvIIIQjiA4UM9Mg_4rq_ttBH3wIl0lJx",
            "keras_to_snn_regularized_params.npz",
        )
```

```
        print("Loaded pretrained weights")

 run_network(
    activation=nengo.RectifiedLinear(),
    params_file="keras_to_snn_regularized_params",
    n_steps=10,
)

run_network(
    activation=nengo.SpikingRectifiedLinear(),
    params_file="keras_to_snn_regularized_params",
    synapse=0.01,
   )
```

Listing 6: Regolarizzazione

4.3 - Risultati

Sono stati condotti quattro test impiegando un numero crescente di filtri per valutare e confrontare le prestazioni della CNN e della SNN.
Nelle seguenti tabelle sono riassunti i risultati ottenuti.

4.3.1 – Rete a 4 filtri

In Tabella 1 trovo un sommario dell'architettura della CNN a 4 filtri; in particolare sono riportati i layer della rete con relativa i parametri totali da allenare.

Layer	Output Shape	Parametri
Input	(None, 50, 50, 3)	0
Convoluzionale 1	(None, 48, 48, **4**)	112
Convoluzionale 2	(None, 23, 23, **8**)	296
Flatten	(None, 4232)	0
Dense	(None, 2)	8466
		TOT: 8874

Tabella 1: Sommario della CNN a 4 filtri

CNN	TRAINING TEST	VALIDATION TEST
Loss	0.7761	1.4852
Probe Loss	0.7761	1.4852
Probe Sparse Categorical Accuracy	0.8822	0.8386
TEST ACCURACY	85.25%	

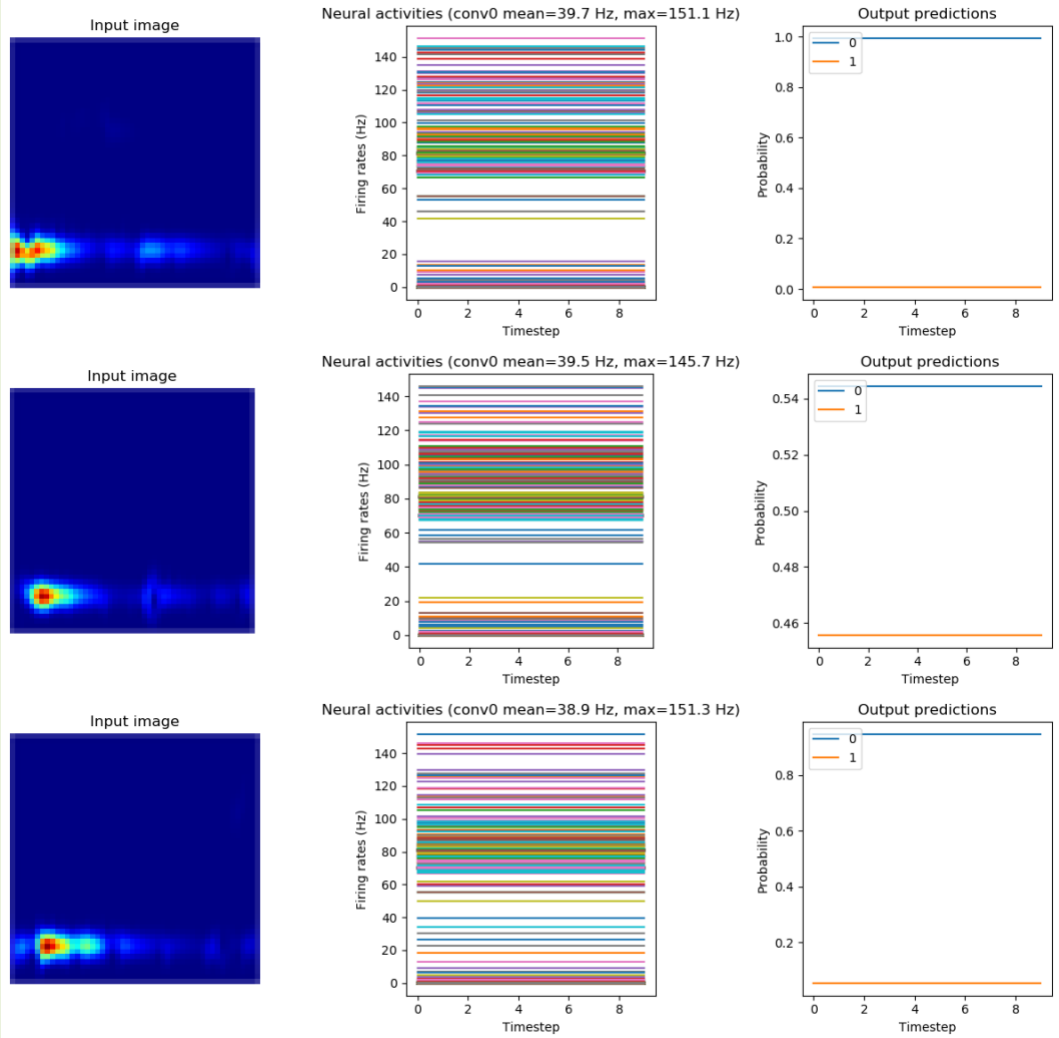


Tabella 2: Performance e attività neurale della CNN a 4 filtri

La CNN è stata dunque convertita in SNN: di seguito in Tabella 3 sono riportati le performance della rete in termini di accuratezza e l'attività neurale nel tempo con le corrispondenti predizioni di 3 immagini di test.

SNN
Test accuracy **79.75%**

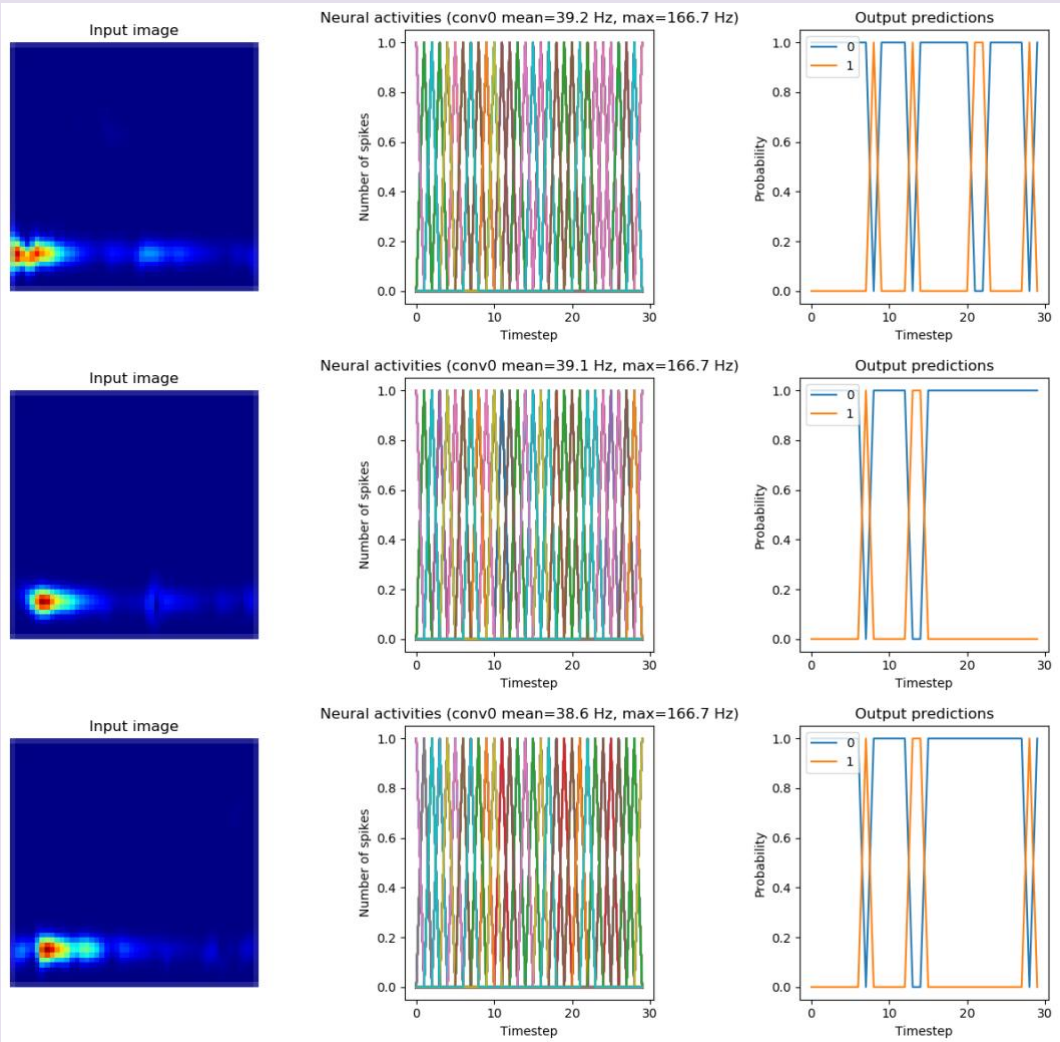


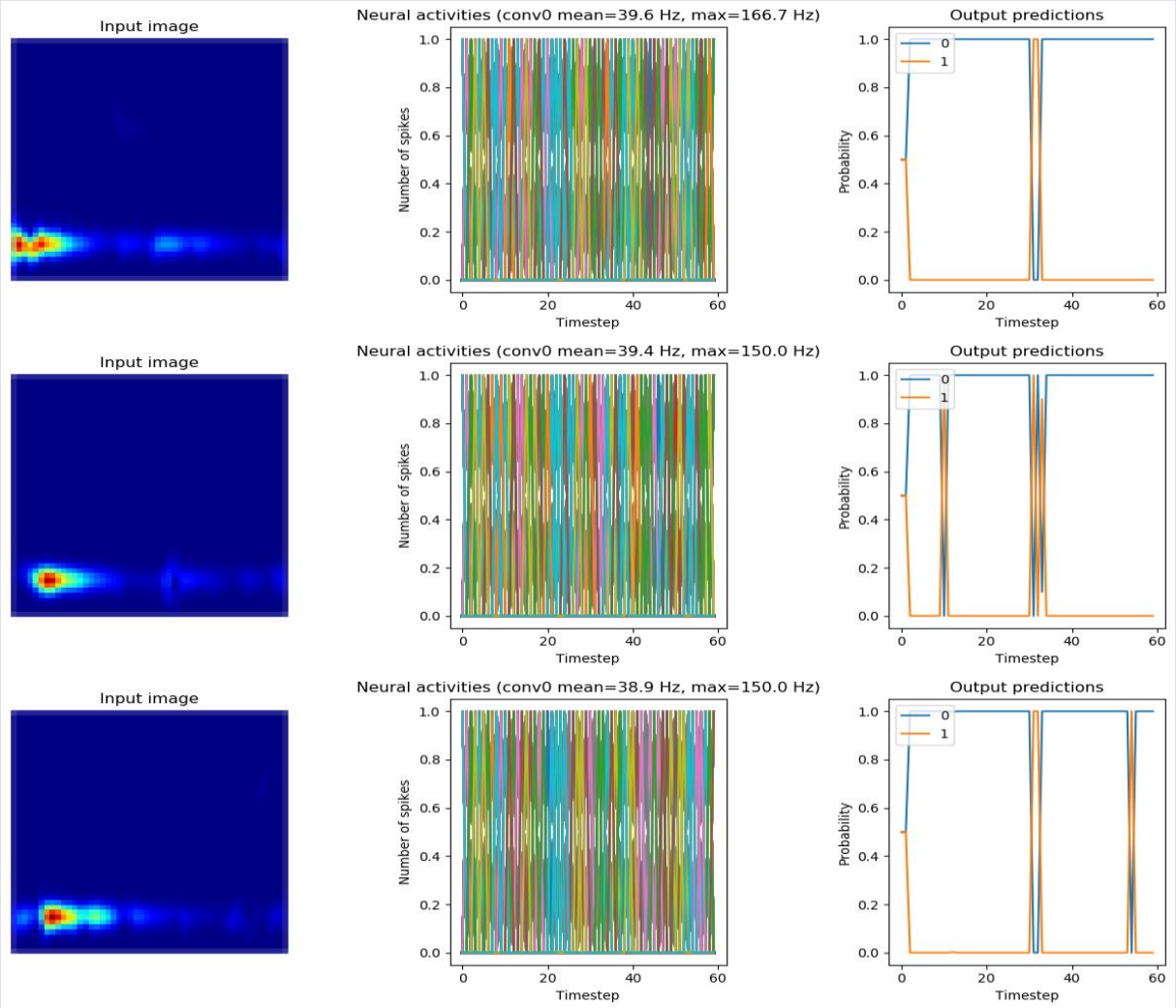
Tabella 3: Prestazioni SNN a 4 filtri

Per migliorare l'accuratezza delle predizioni si può pensare dunque di usare i tre metodi ì proposti nella sezione precedente: lo smoothing con filtri sinaptici, l'upscaling dei firing rates o l'ottimizzazione degli stessi tramite funzione di regolarizzazione. I risultati sono riportati in Tabella 4.

SMOOTHING CON FILTRI SINAPTICI

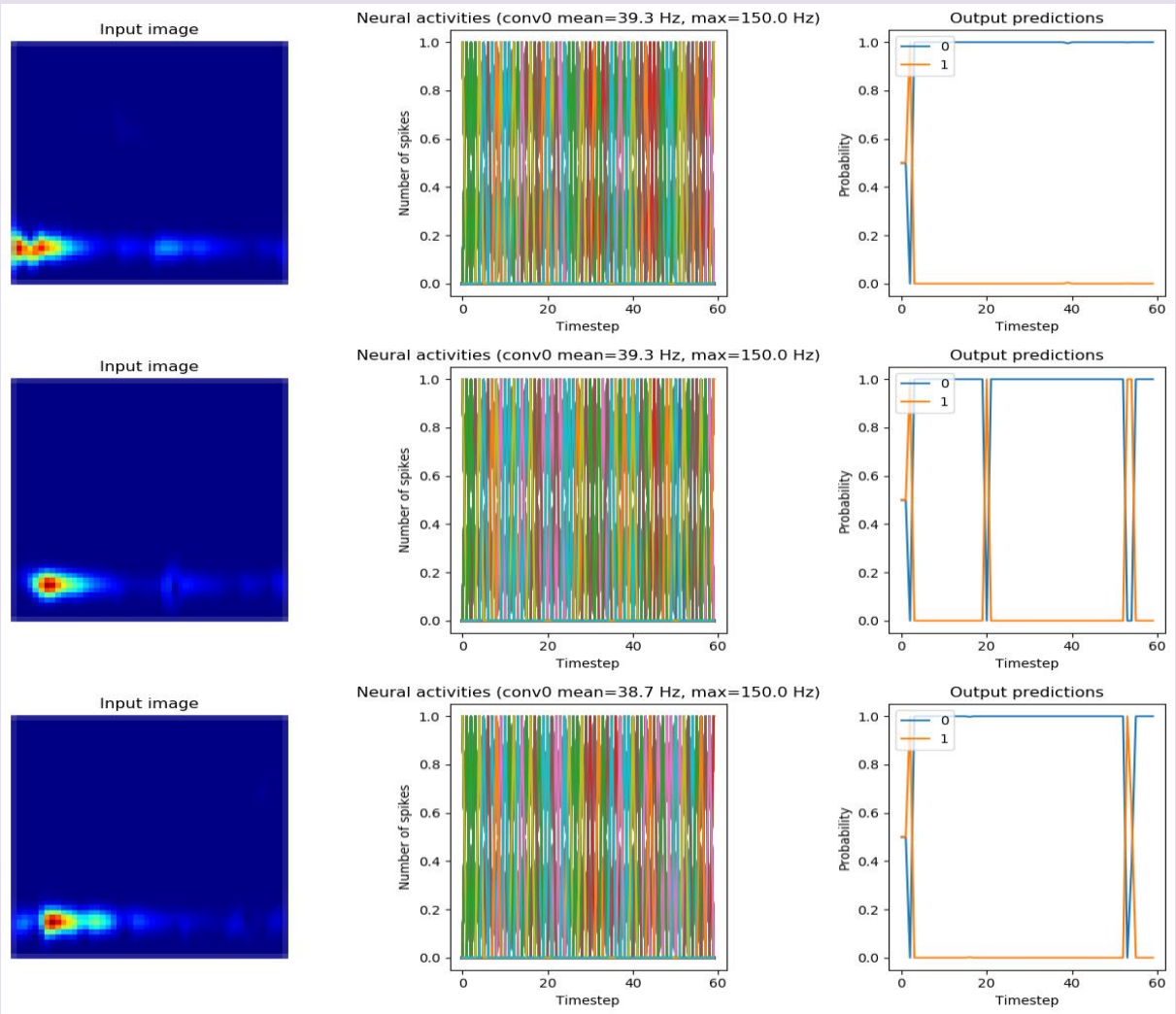
S=0.001

Test accuracy **96.75%**



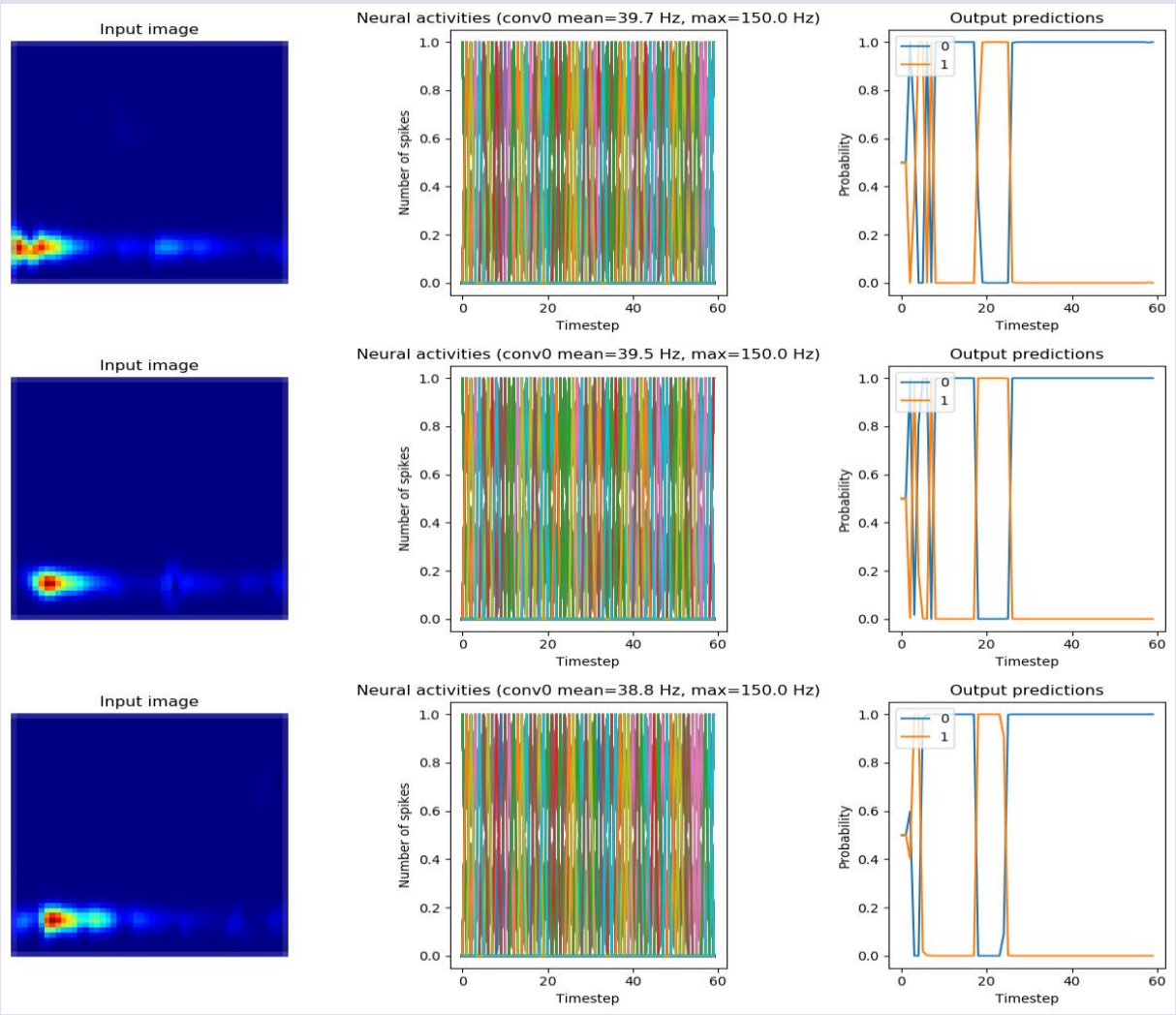
SMOOTHING CON FILTRI SINAPTICI
S=0.005

Test accuracy **99.50%**



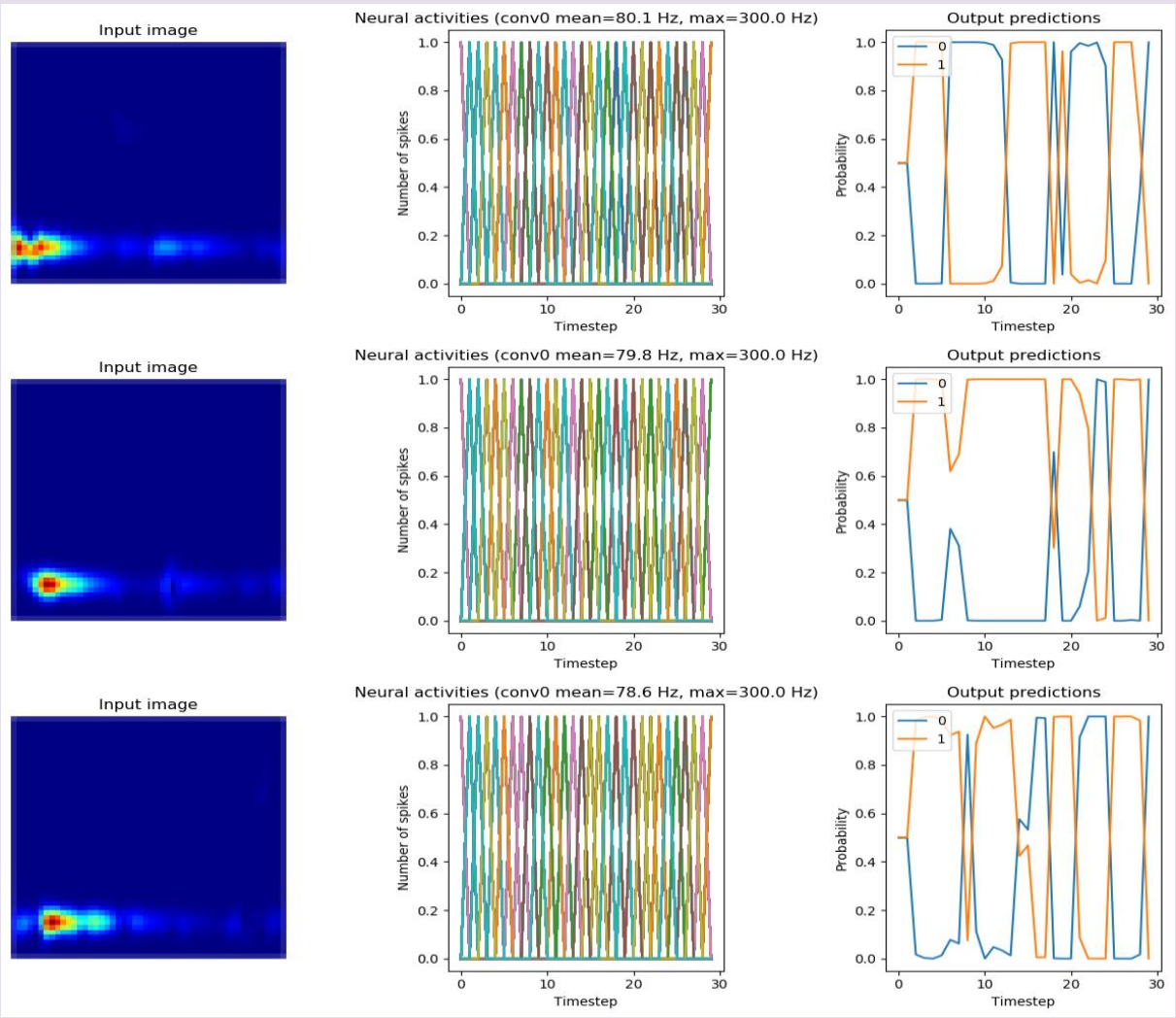
SMOOTHING CON FILTRI SINAPTICI
S=0.01

Test accuracy **89.00%**



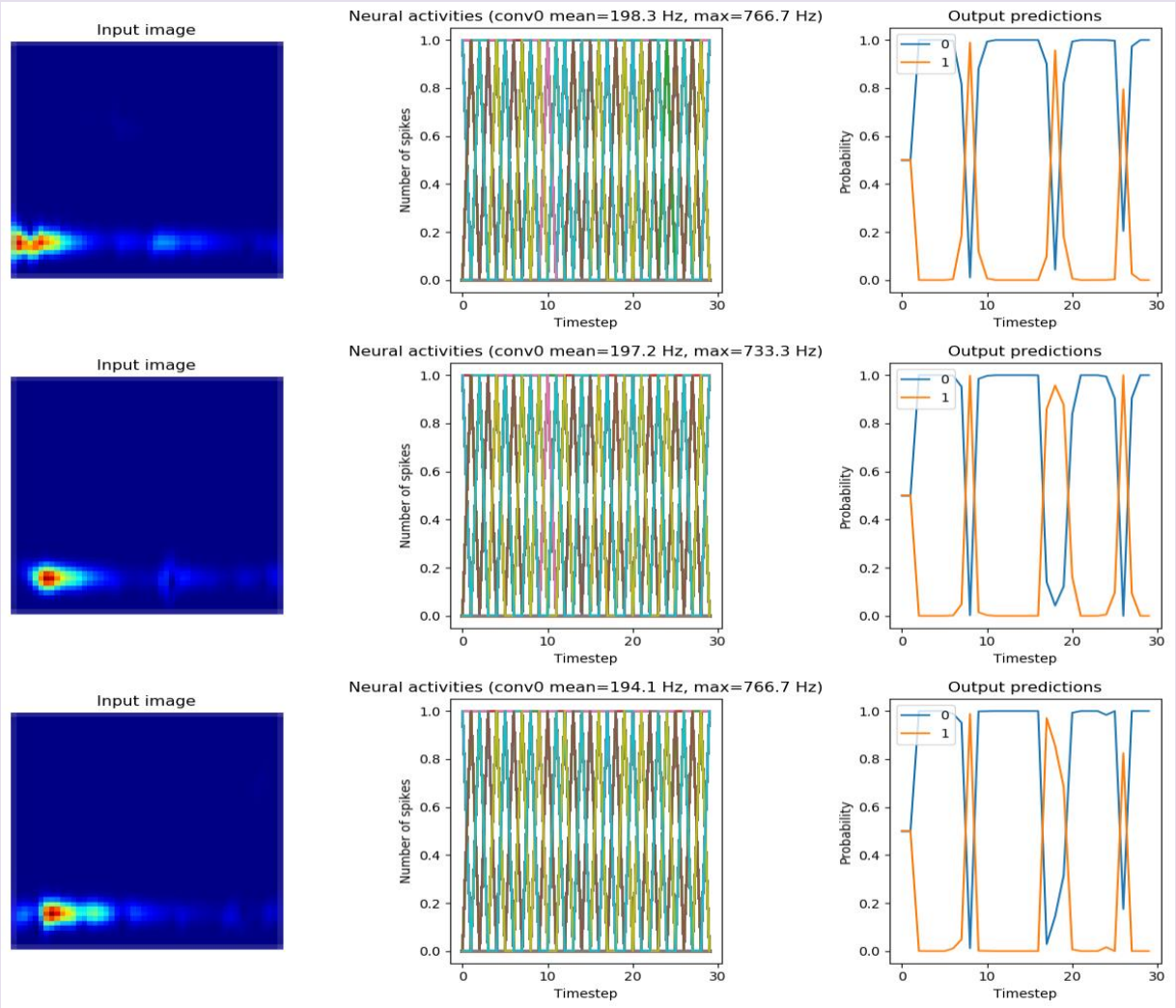
UPSCALING DEI FIRING RATES DEI NEURONI
Scale= 2

Test accuracy **94.50%**



UPSCALING DEI FIRING RATES DEI NEURONI
Scale= 5

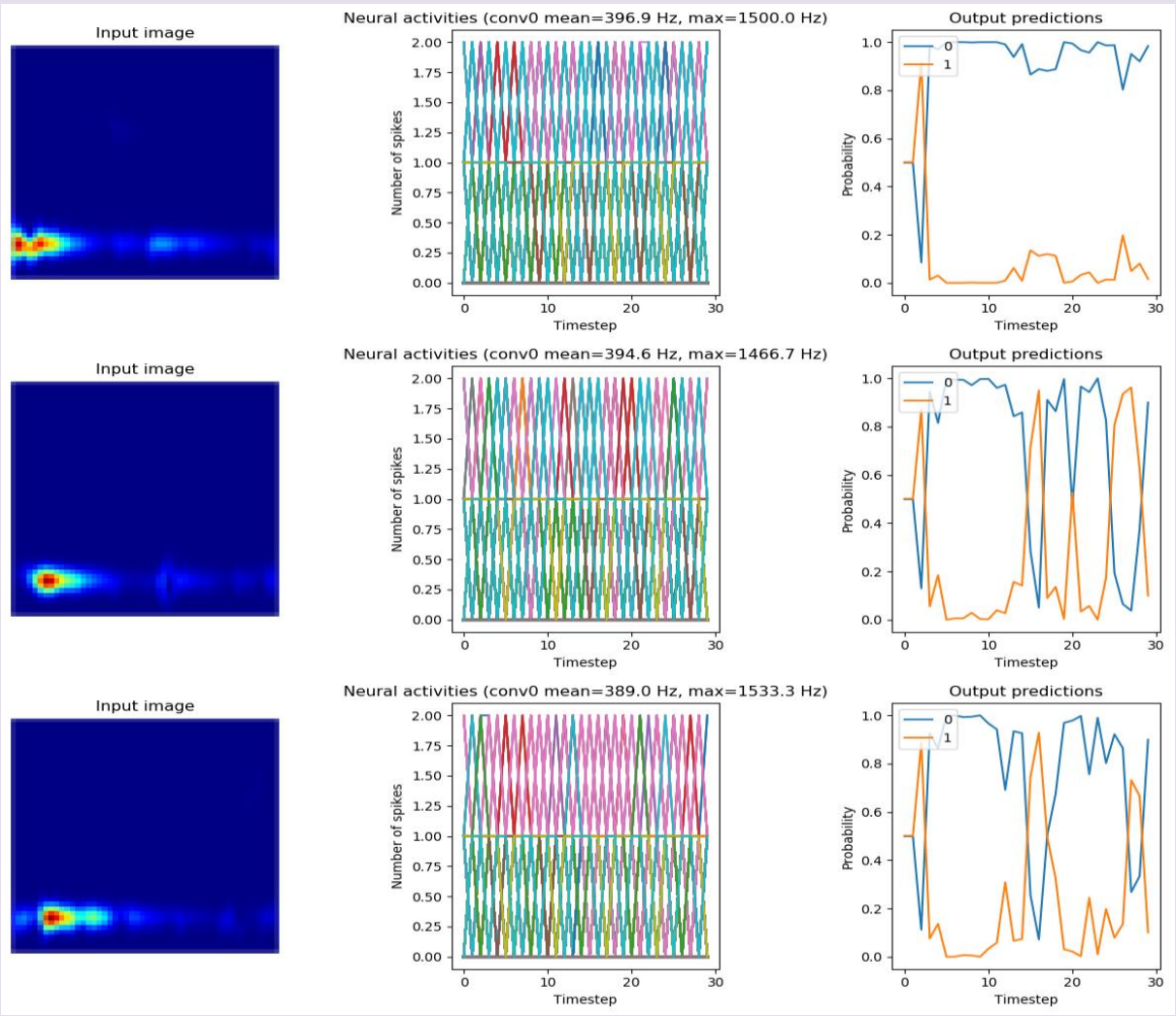
Test accuracy **98.50%**



UPSCALING DEI FIRING RATES DEI NEURONI

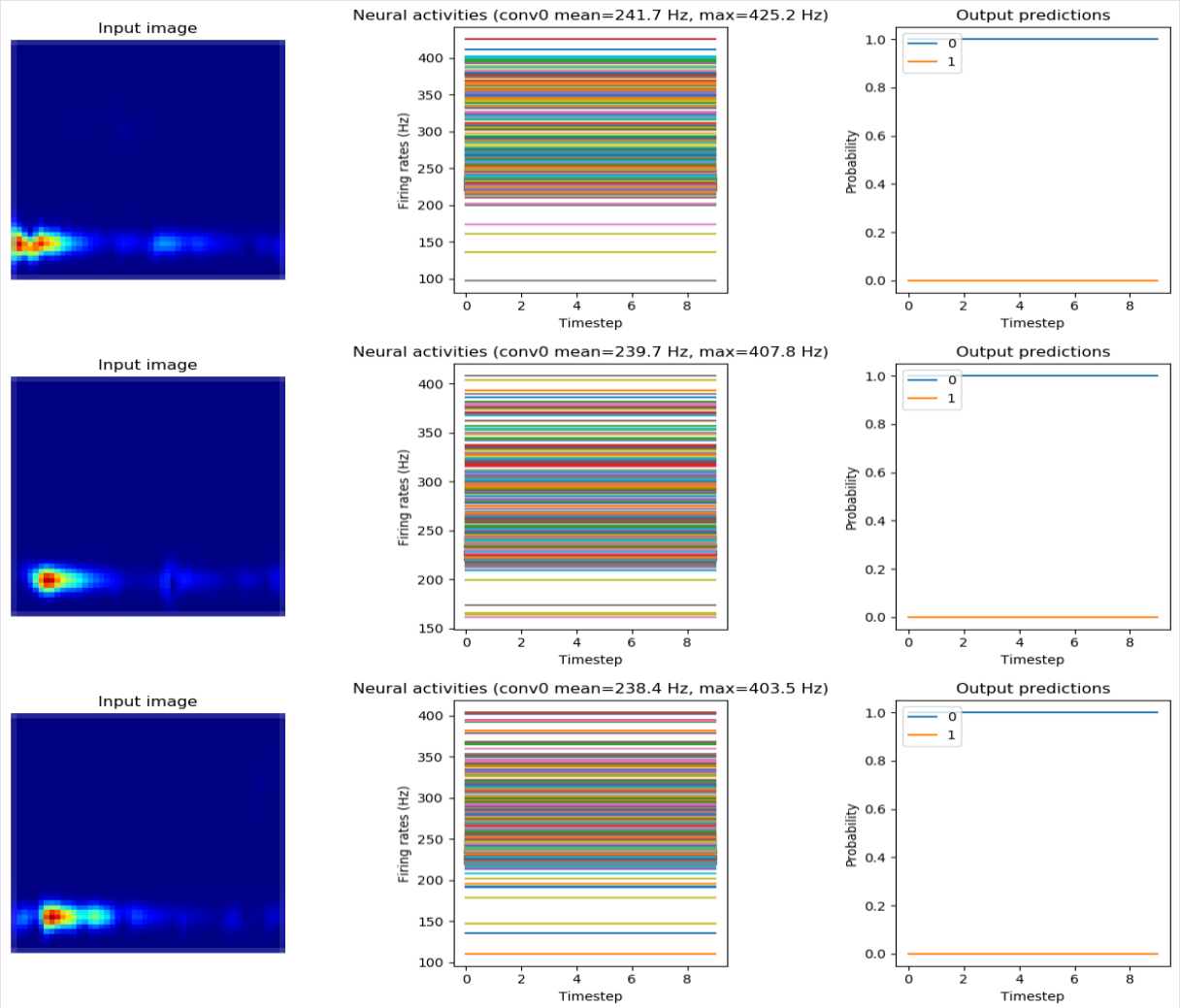
Scale= 10

Test accuracy **88.25%**



ALTERNATIVA: OTTIMIZZAZIONE TEMPI DI FIRING (REGULARIZATION)

Test accuracy **100%**



ALTERNATIVA: OTTIMIZZAZIONE TEMPI DI FIRING (REGULARIZATION)

Test accuracy **80.75%**

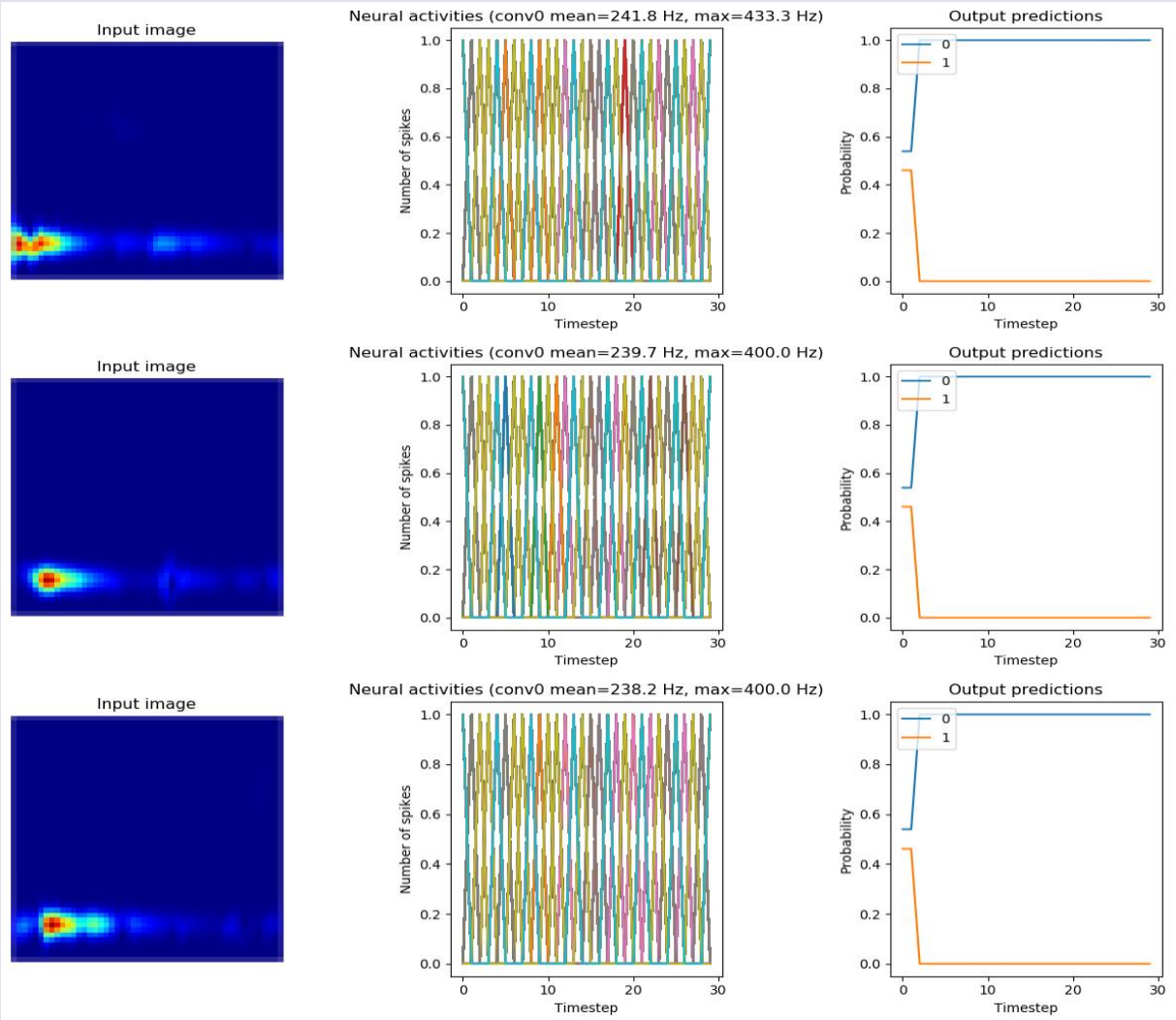


Tabella 4: Metodi per il miglioramento dell'accuratezza SNN a 4 filtri

In Tabella 5 sono infine confrontati questi risultati e viene valutato il miglioramento percentuale dell'accuratezza per la seguente architettura.

ACCURACY	%
CNN	**85.25%**
SNN	79.75%
SNN post Smoothing (s=0.005)	99.50%
SNN post Scaling	98.50%
SNN post Regolarizzazione	**100%**
ACCURACY GAIN	**+14.75%**

Tabella 5: Accuracy Gain SNN a 4 filtri

4.3.2 – Rete a 8 filtri

In Tabella 6 è riportato un sommario dell'architettura della CNN a 8 filtri; in particolare sono riportati i layer della rete con relativa i parametri totali da allenare.

Layer	Output Shape	Parametri
Input	(None, 50, 50, 3)	0
Convoluzionale 1	(None, 48, 48, **8**)	224
Convoluzionale 2	(None, 23, 23, **16**)	1168
Flatten	(None, 8464)	0
Dense	(None, 2)	16930
		TOT: 18322

Tabella 6: Summary della CNN a 8 filtri

CNN	TRAINING TEST	VALIDATION TEST
Loss	0.3152	0.5588
Probe Loss	0.3152	0.5588
Probe Sparse Categorical Accuracy	0.9155	0.9235
TEST ACCURACY	98.75%	

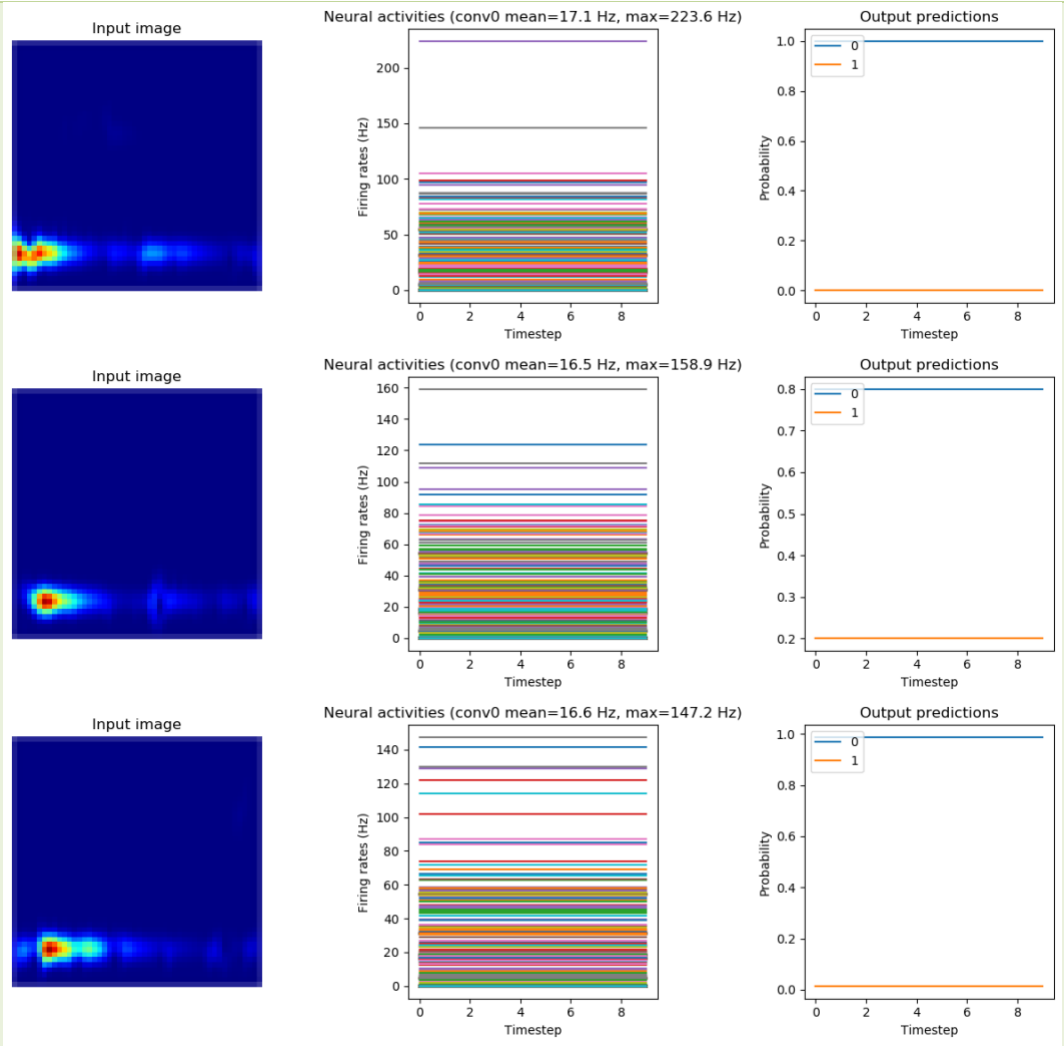
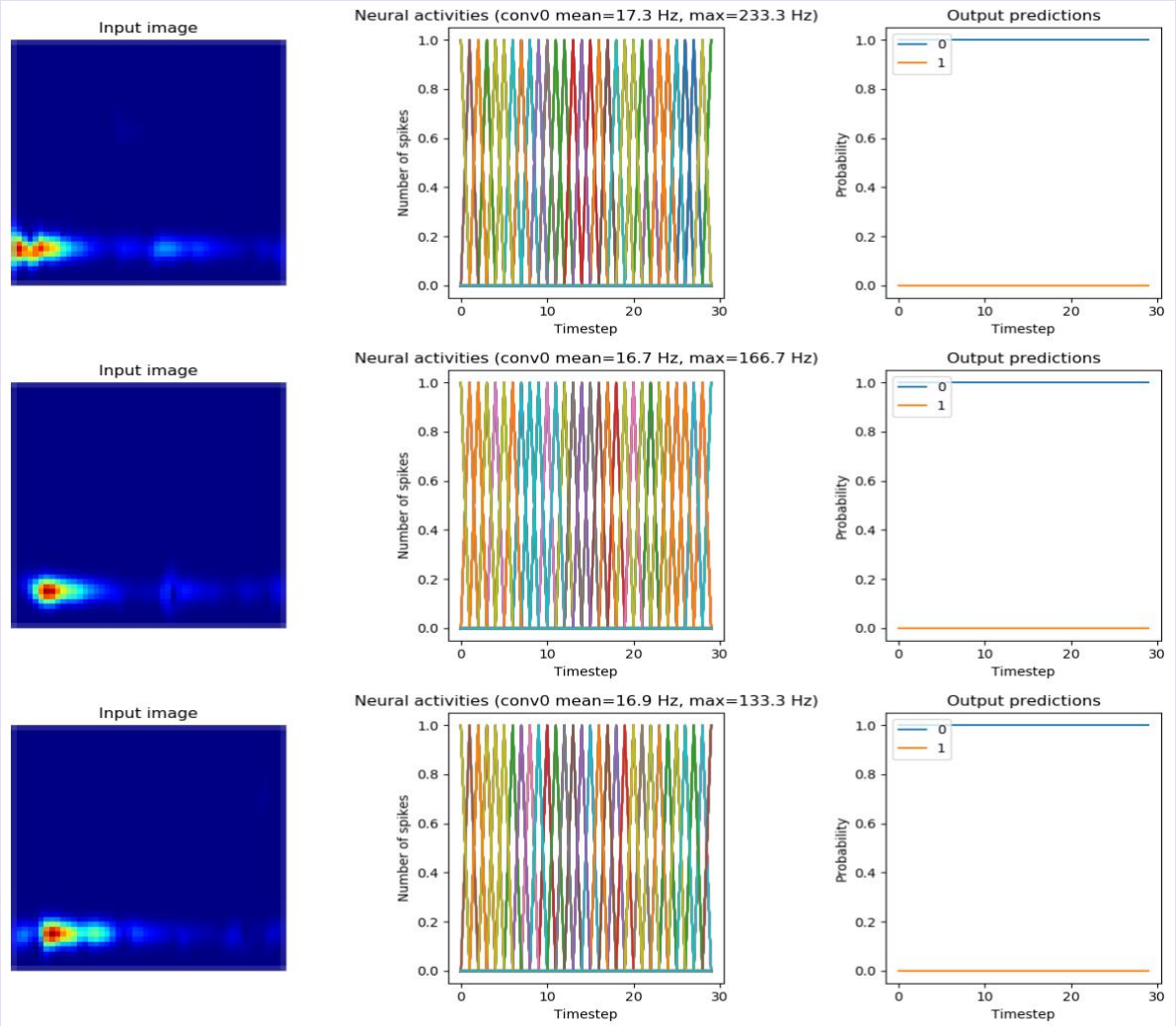


Tabella 7: Performance e attività neurale della CNN a 8 filtri

Di seguito in Tabella 7 sono riportati le performance della SNN ottenuta in termini di accuratezza e l'attività neurale nel tempo con le corrispondenti predizioni di 3 immagini di test.

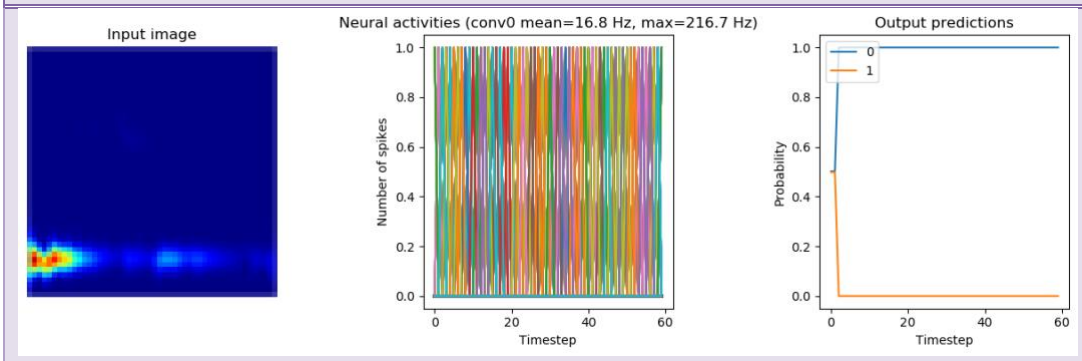
SNN
Test accuracy: **80.75%**

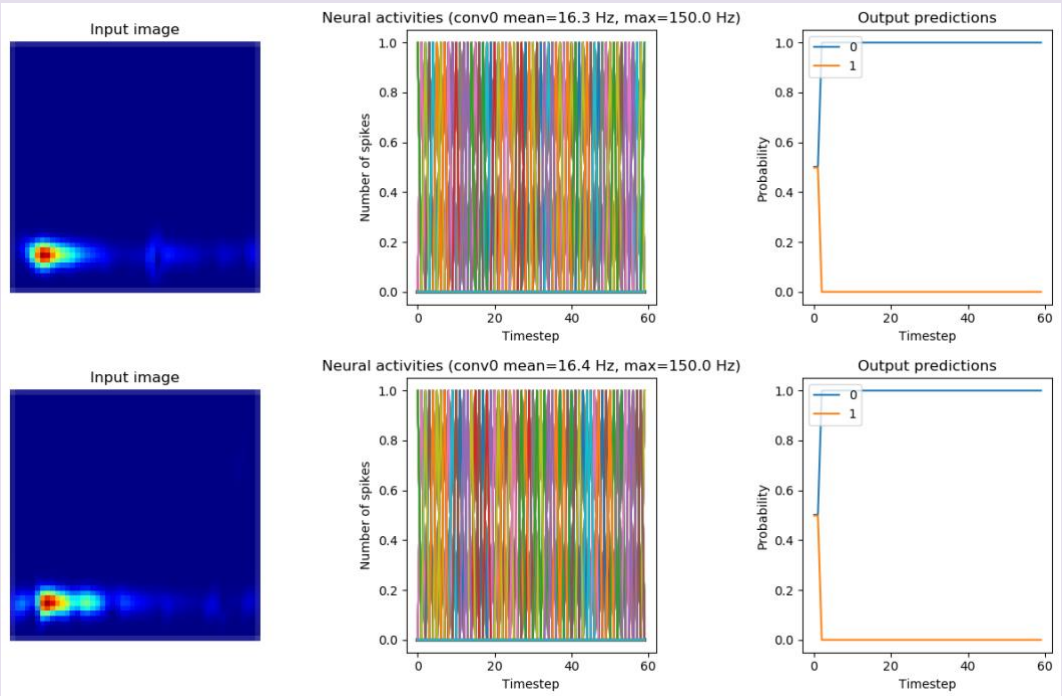


SMOOTHING FILTRI SINAPTICI

S=0.001

Test accuracy: **100%**

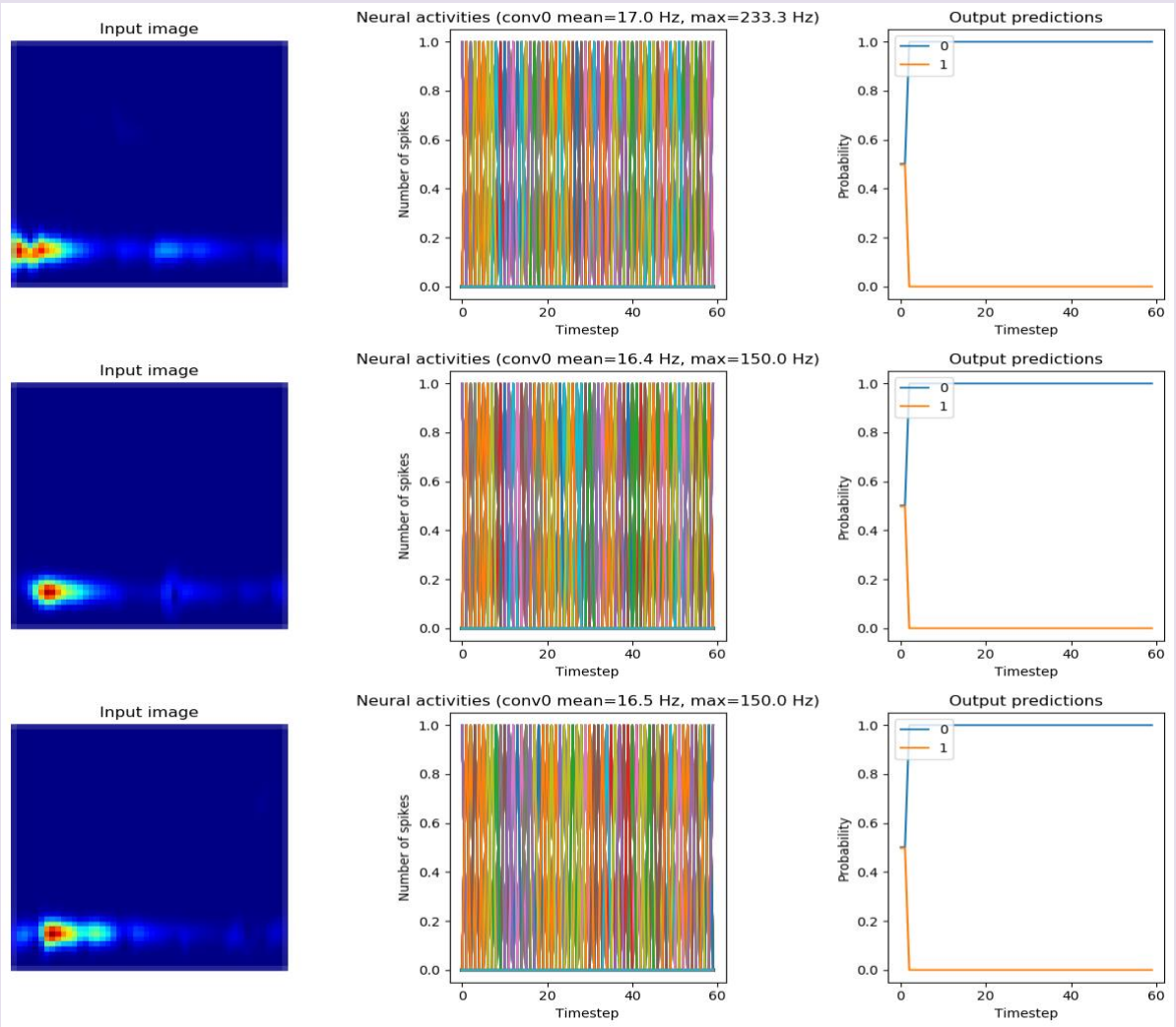




SMOOTHING FILTRI SINAPTICI

S=0.005

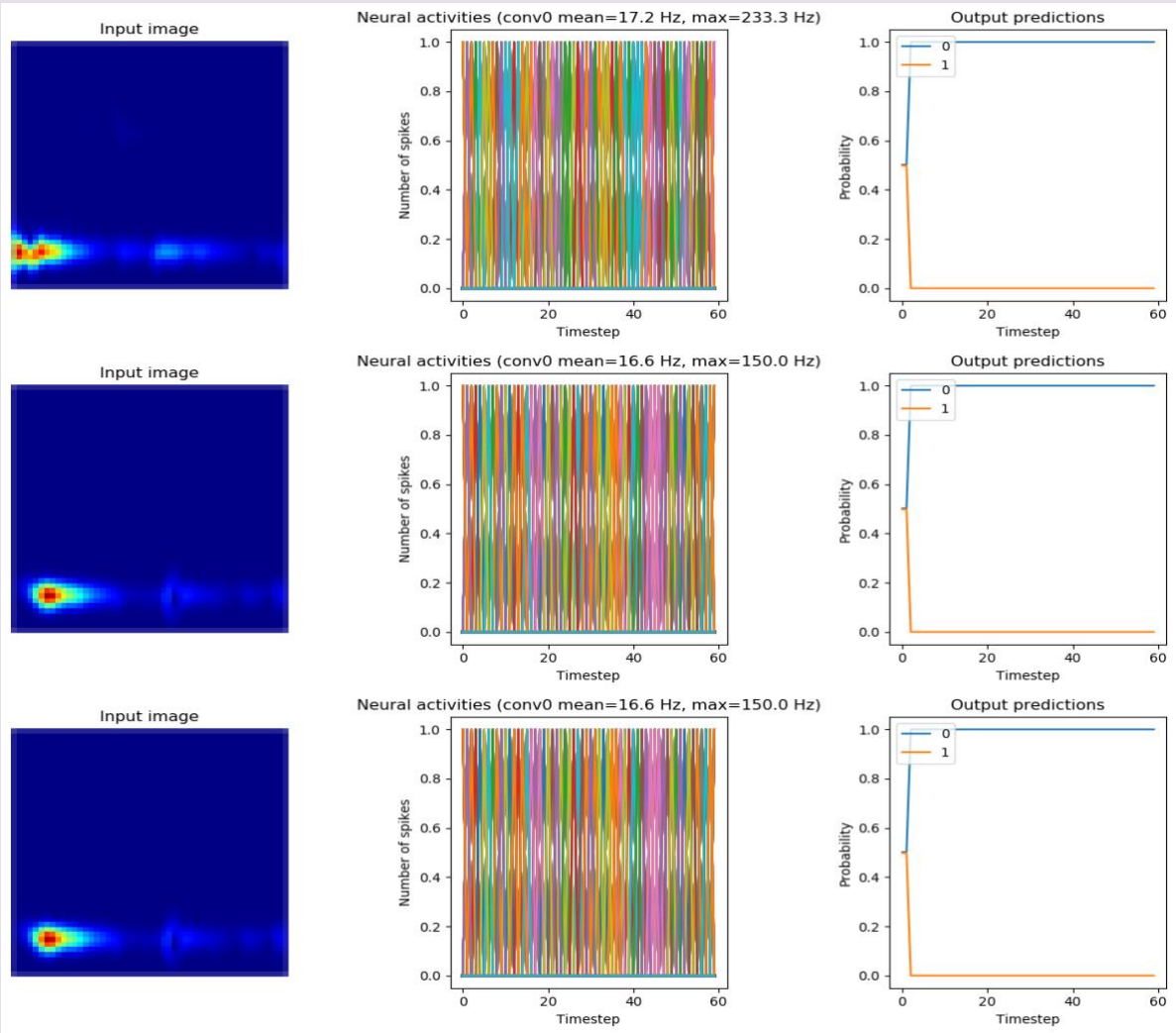
Test accuracy: **100%**



SMOOTHING FILTRI SINAPTICI

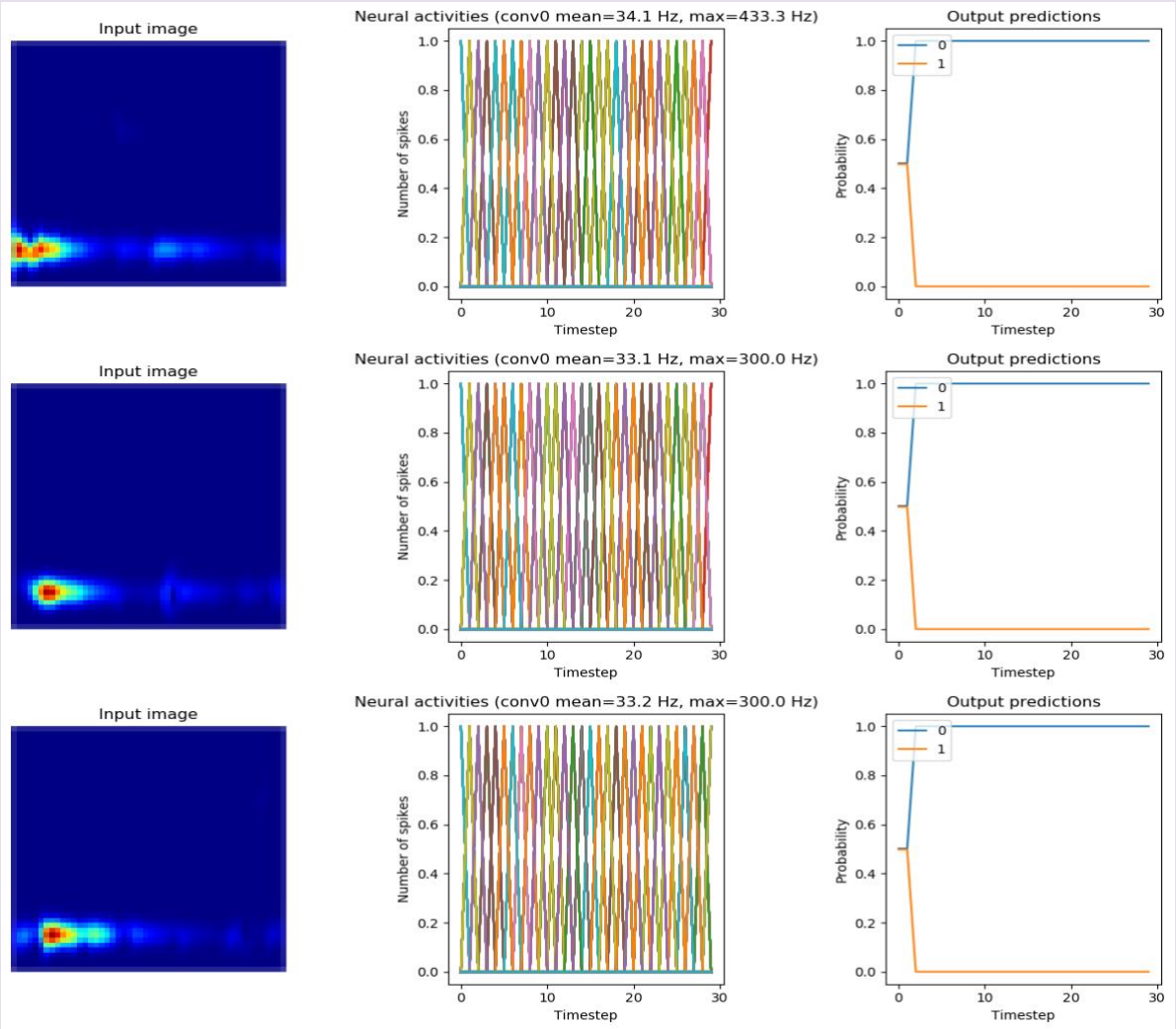
S=0.01

Test accuracy: **100%**



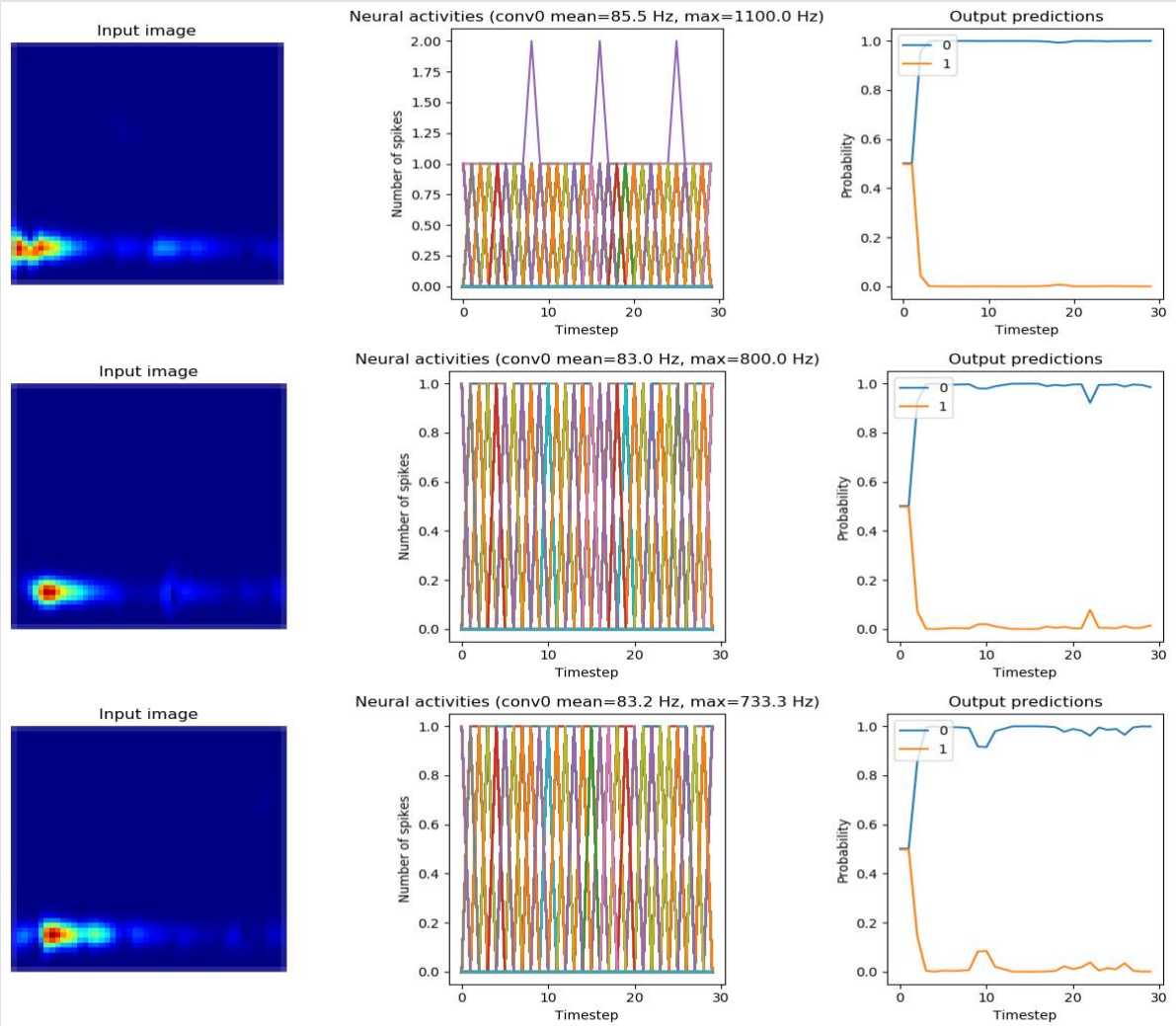
UPSCALING DEI FIRING RATES DEI NEURONI
Scale= 2

Test accuracy: ***100%***



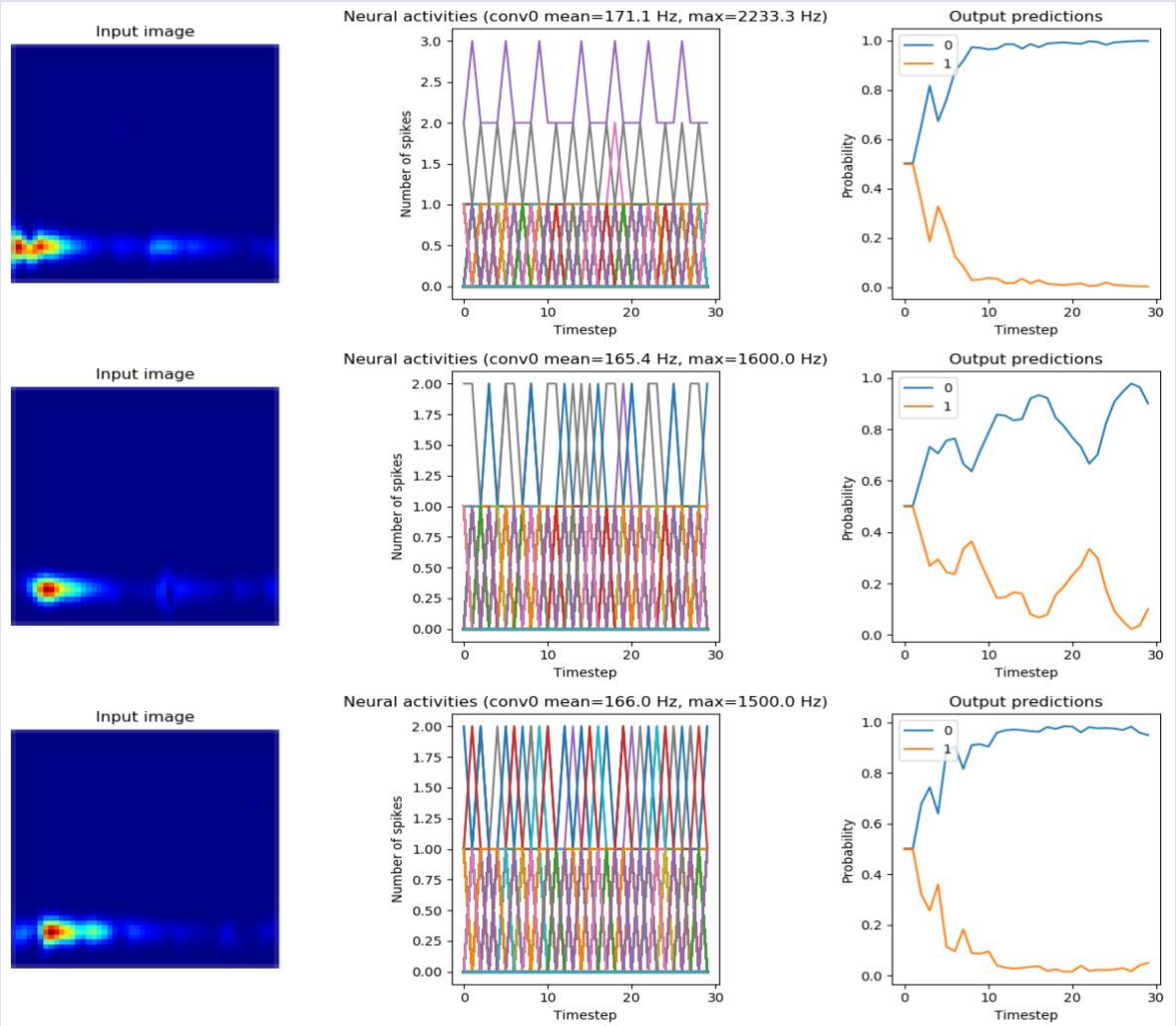
UPSCALING DEI FIRING RATES DEI NEURONI
Scale= 5

Test accuracy: ***100%***



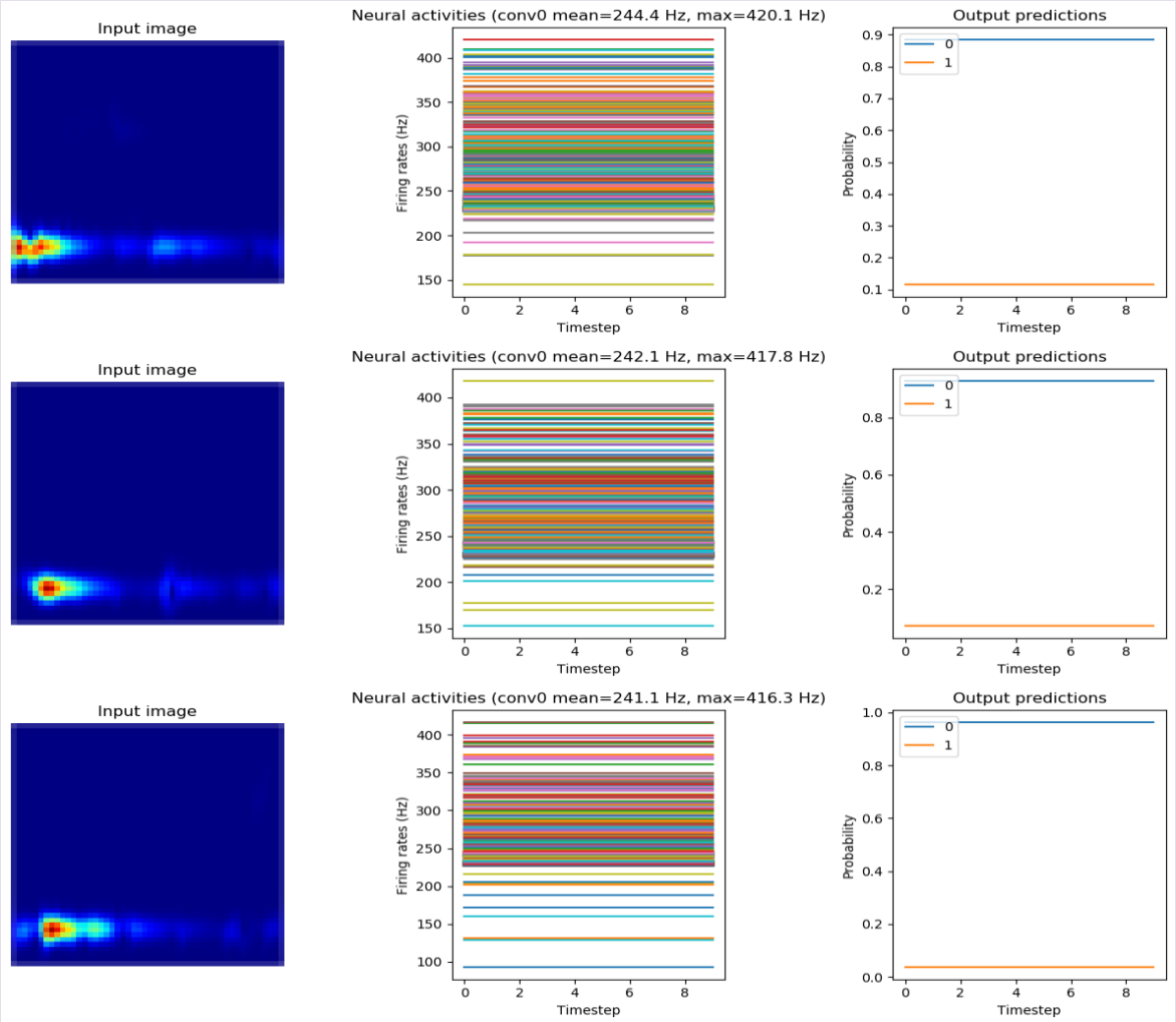
UPSCALING DEI FIRING RATES DEI NEURONI
Scale= 10

Test accuracy: ***100%***



ALTERNATIVA: OTTIMIZZAZIONE TEMPI DI FIRING (REGULARIZATION)

Test accuracy: **100.00%**



ALTERNATIVA: OTTIMIZZAZIONE TEMPI DI FIRING (REGULARIZATION)

Test accuracy: **100.00%**

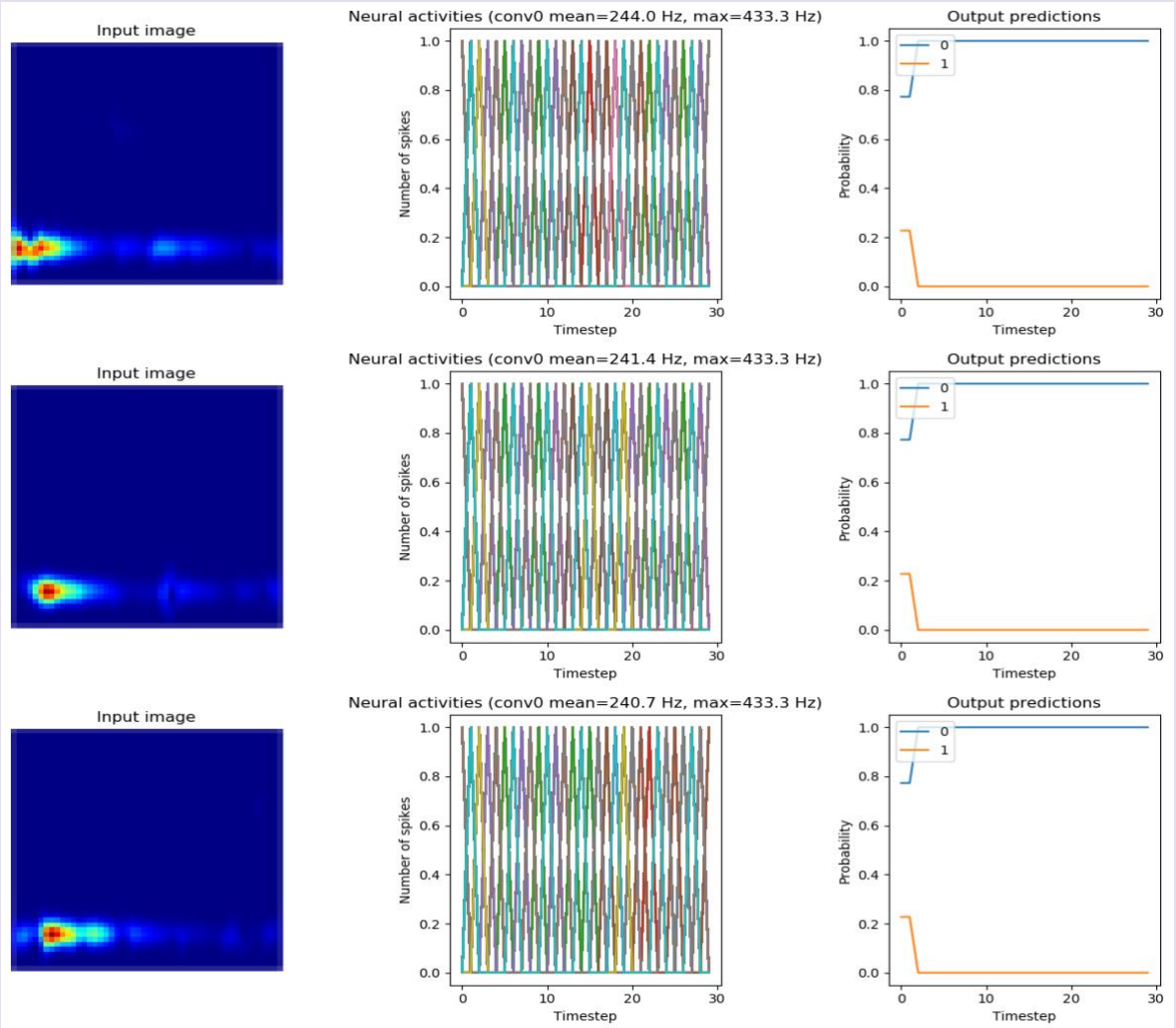


Tabella 8: Metodi per il miglioramento dell'accuratezza SNN a 8 filtri

In Tabella 9 sono infine confrontati questi risultati e viene valutato il miglioramento percentuale dell'accuratezza per la seguente architettura.

ACCURACY	%
CNN	**98.75%**
SNN	100%
SNN post Smoothing	100%
SNN post Scaling	100%
SNN post Regolarizzazione	**100%**
ACCURACY GAIN	**+1.25%**

Tabella 9: Accuracy Gain SNN a 8 filtri

4.3.3 – Rete a 16 filtri

In Tabella 10 è riportato il sommario dell'architettura della CNN a 16 filtri con particolare riferimento ai layer della rete e relativi parametri totali da allenare.

Layer	Output Shape	Parametri
Input	(None, 50, 50, 3)	0
Convoluzionale 1	(None, 48, 48, **16**)	448
Convoluzionale 2	(None, 23, 23, **32**)	4640
Flatten	(None, 16928)	0
Dense	(None, 2)	33858
		TOT: 38,946

Tabella 10: Summary della CNN a 16 filtri

CNN	TRAINING TEST	VALIDATION TEST
Loss	0.2443	0.4108
Probe Loss	0.2443	0.4108
Probe Sparse Categorical Accuracy	0.9310	0.9286
TEST ACCURACY	100%	

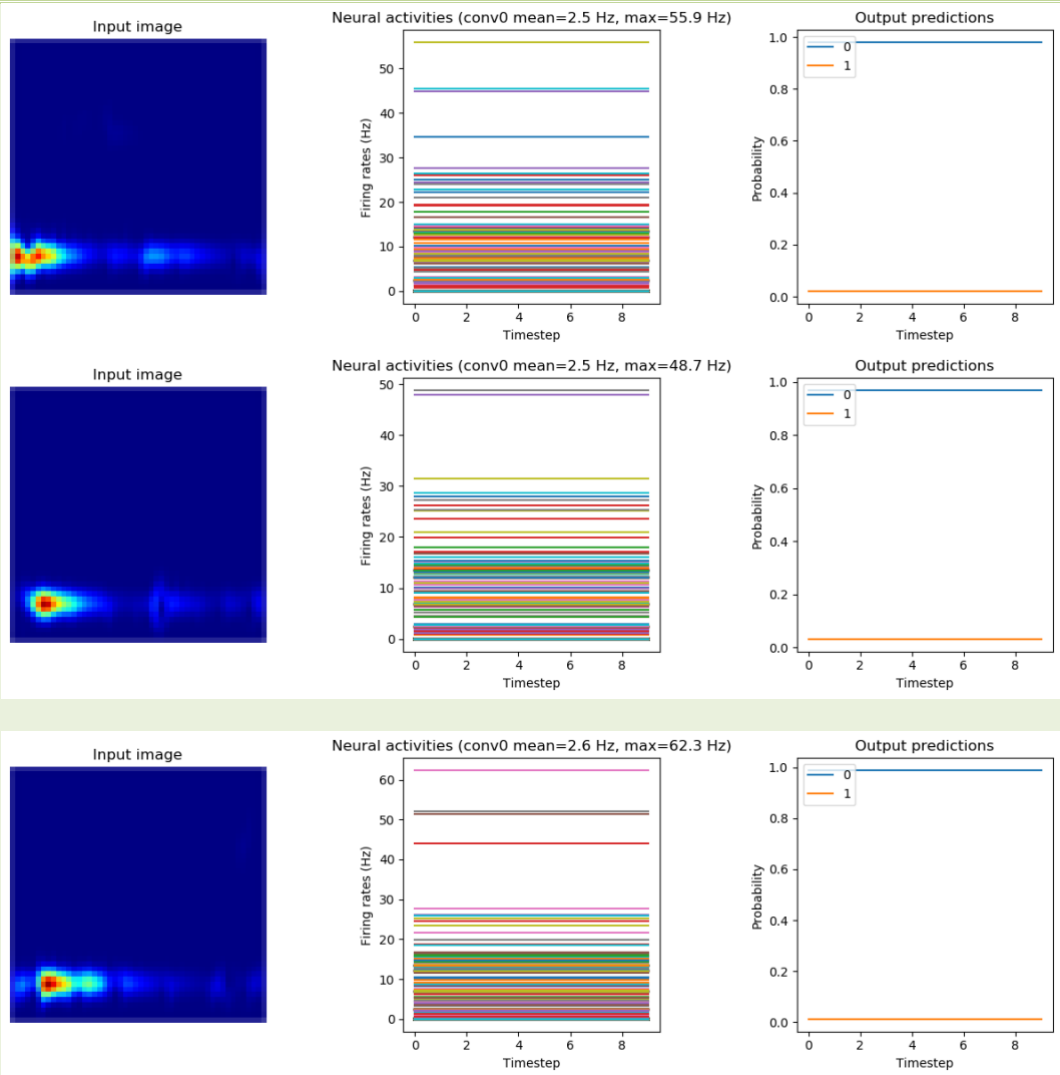
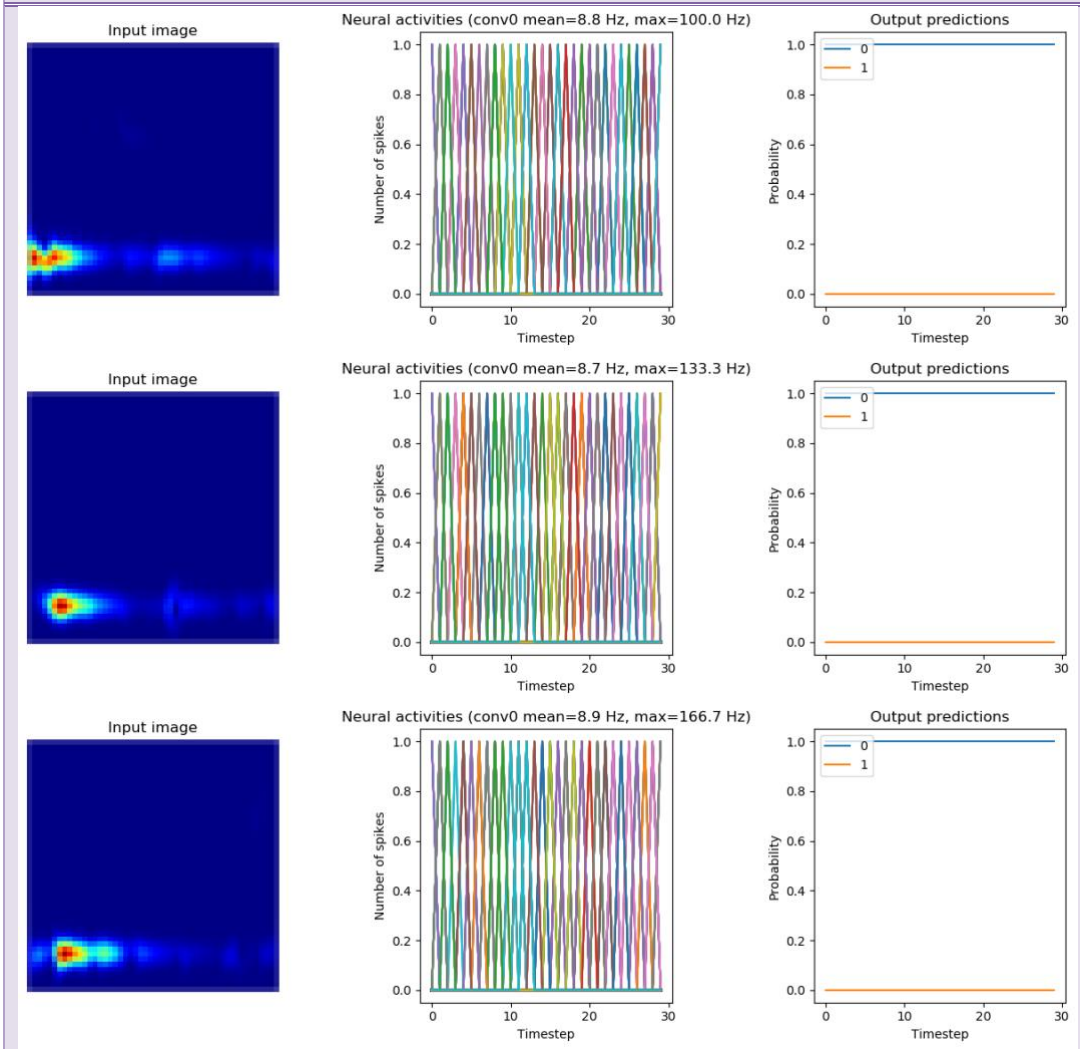


Tabella 11: Performance e attività neurale della CNN a 16 filtri

Di seguito in Tabella 12 sono riportati le performance della SNN ottenuta in termini di accuratezza e l'attività neurale nel tempo con le corrispondenti predizioni di 3 immagini di test.

SNN

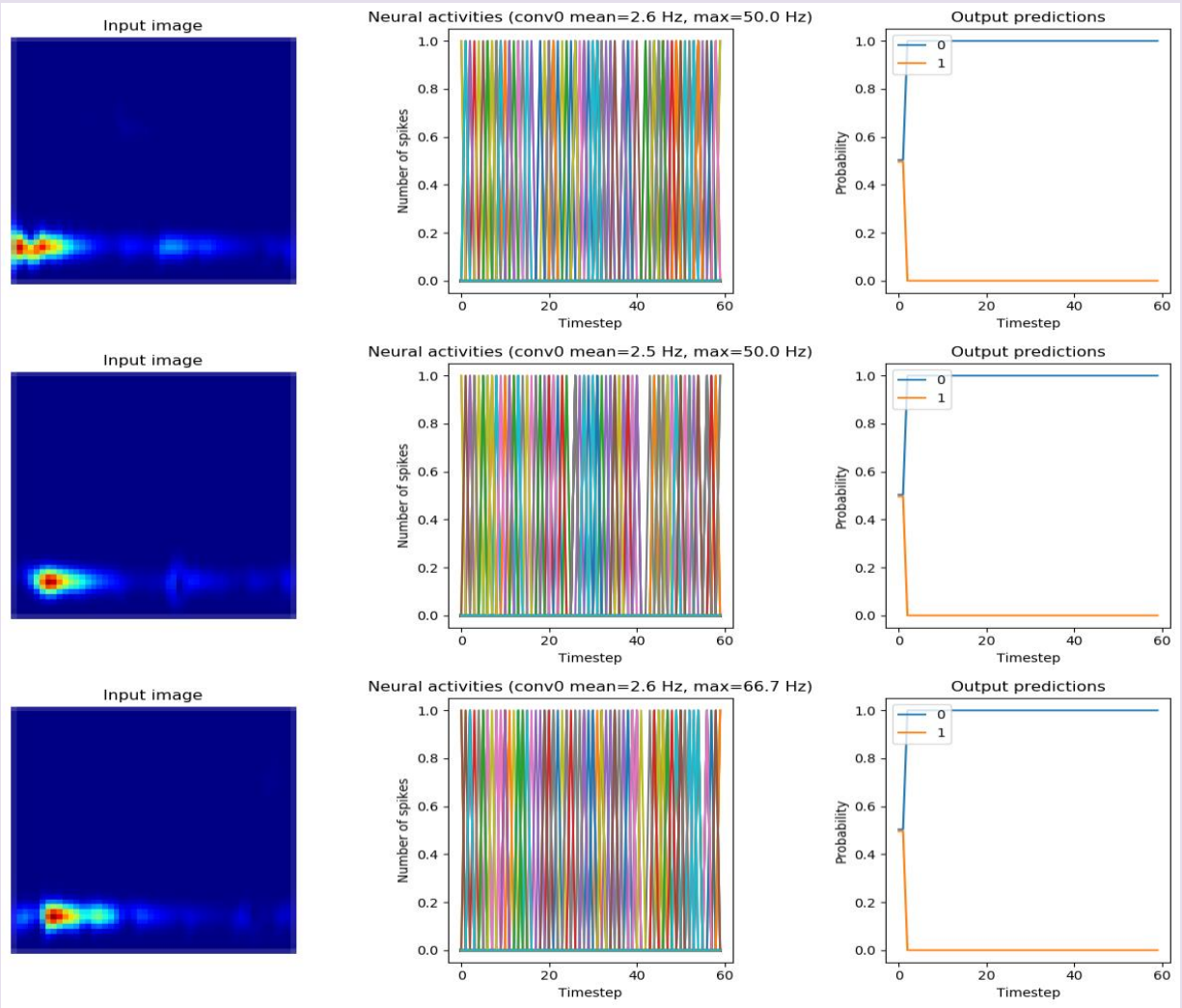
Test accuracy: **100.00%**



SMOOTHING FILTRI SINAPTICI

S=0.001

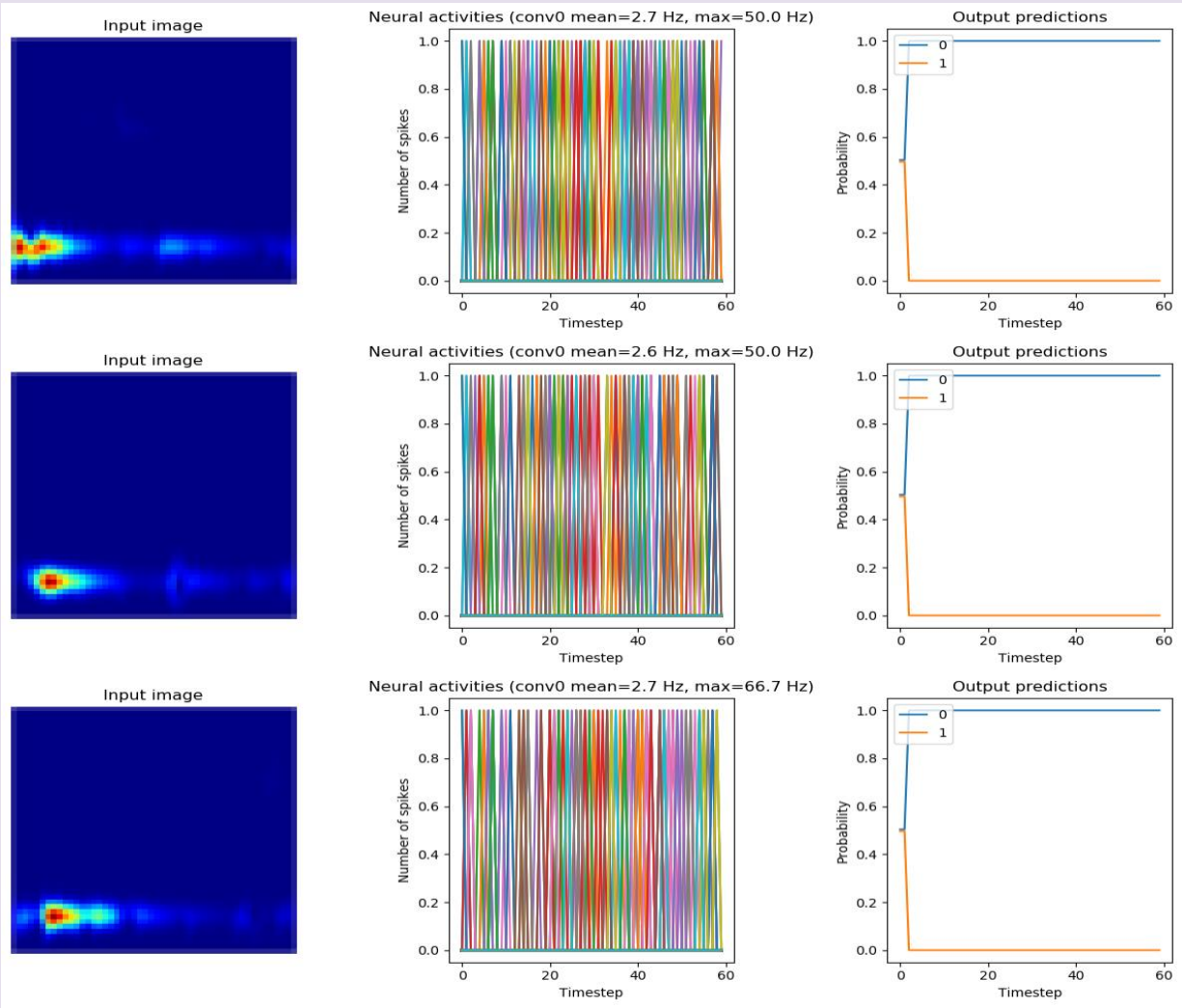
Test accuracy: **100%**



SMOOTHING FILTRI SINAPTICI

S=0.01

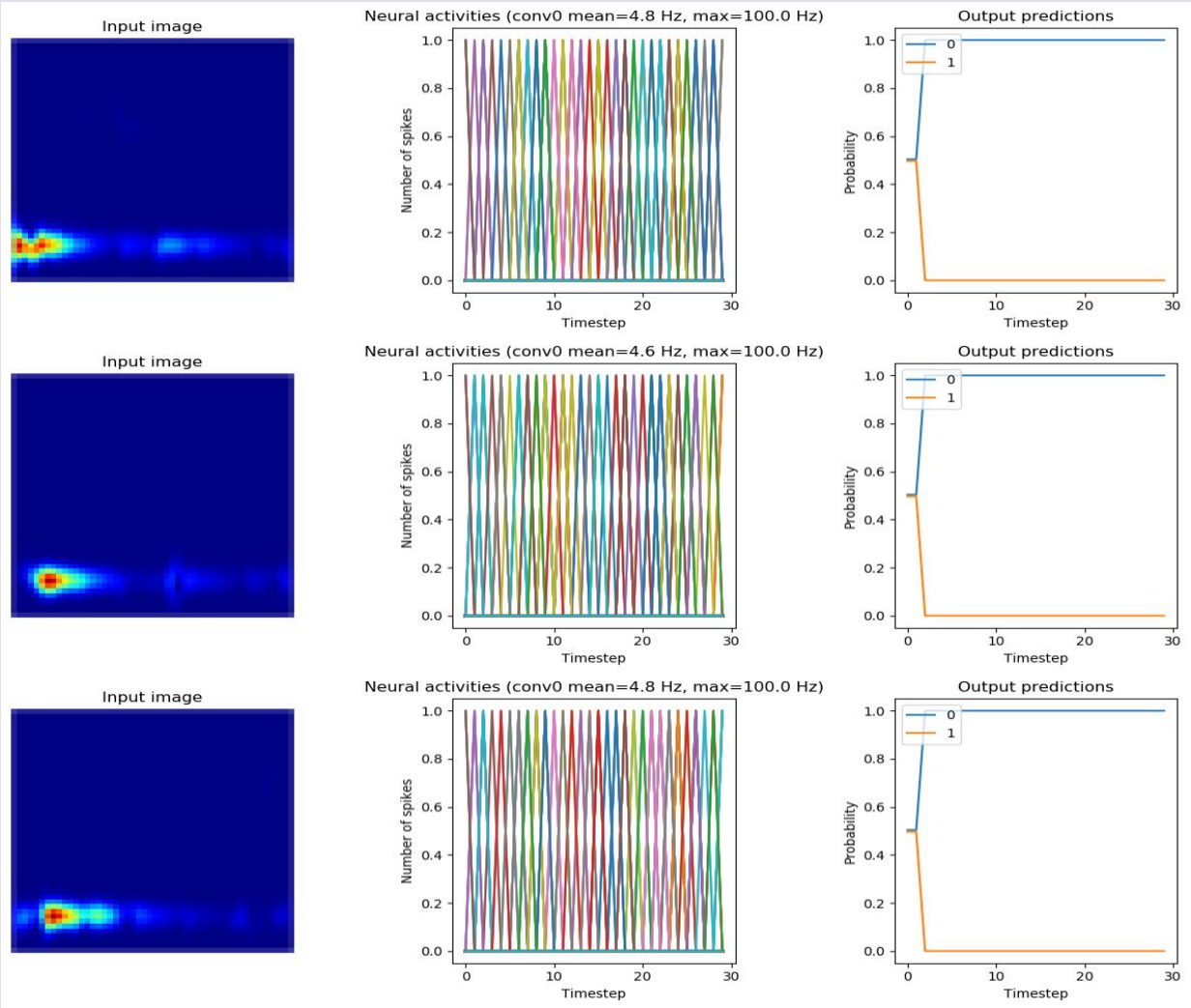
Test accuracy: **100%**



UPSCALING DEI FIRING RATES DEI NEURONI

Scale= 2

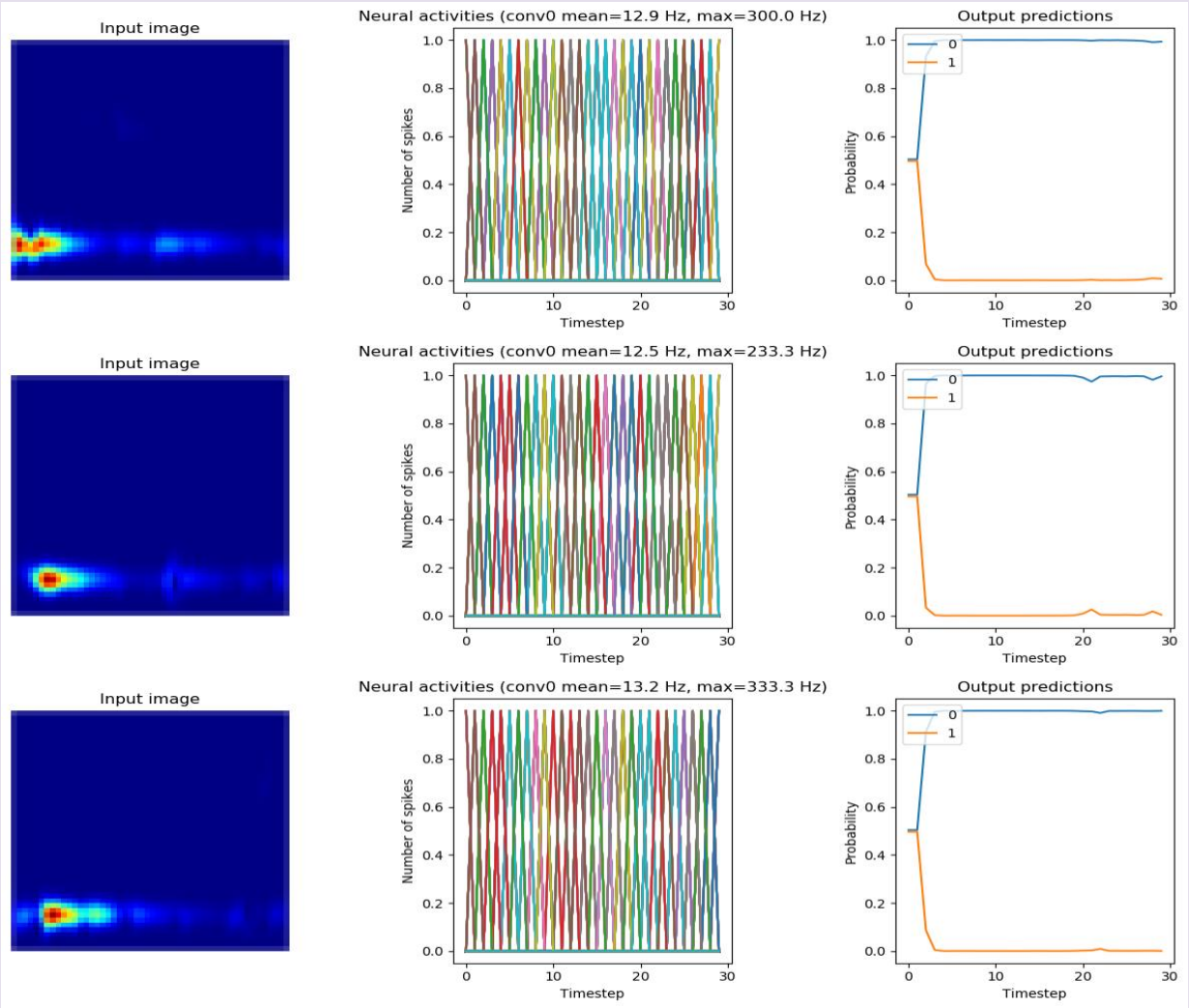
Test accuracy: **100%**



UPSCALING DEI FIRING RATES DEI NEURONI

Scale= 5

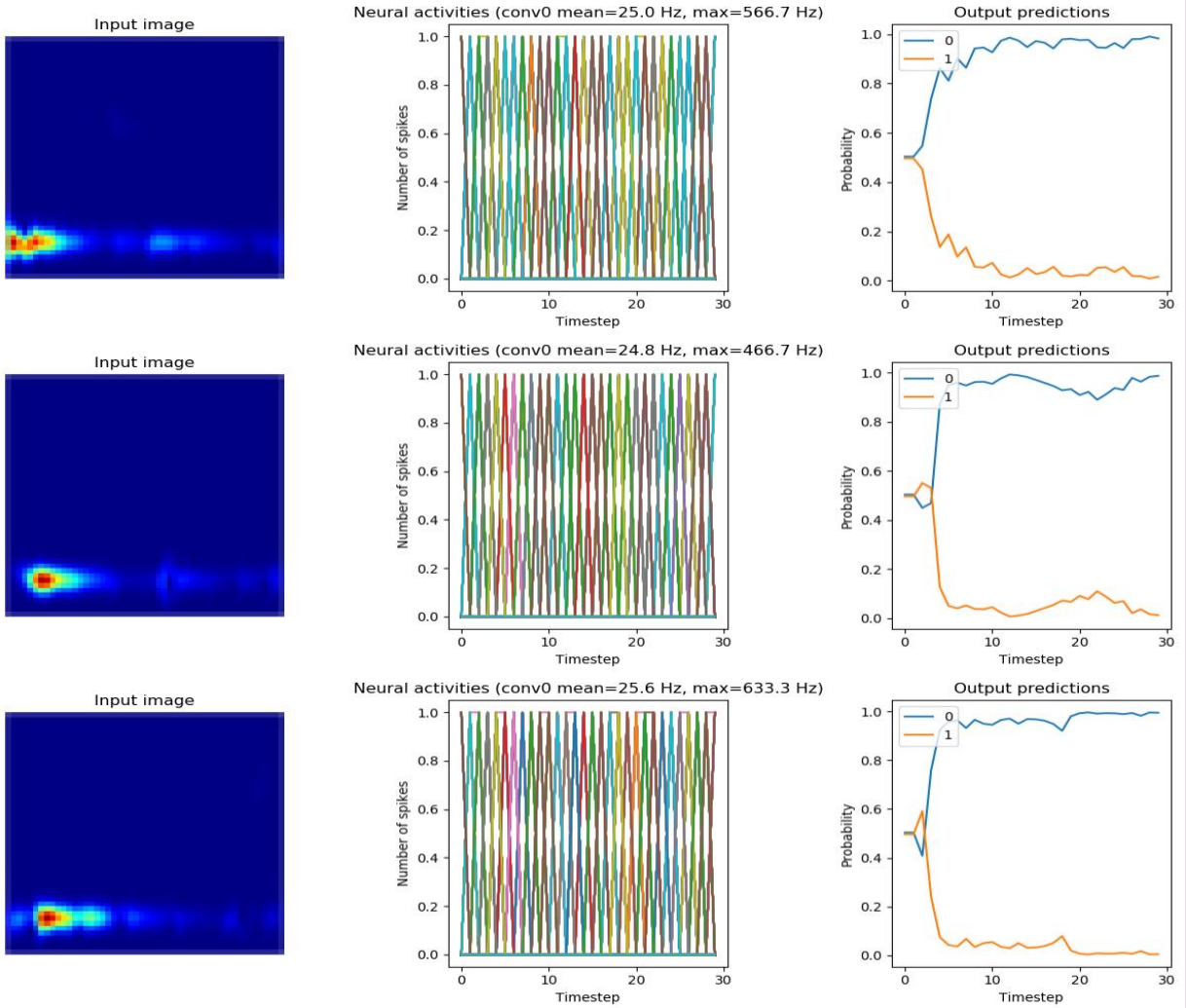
Test accuracy: **100%**



UPSCALING DEI FIRING RATES DEI NEURONI

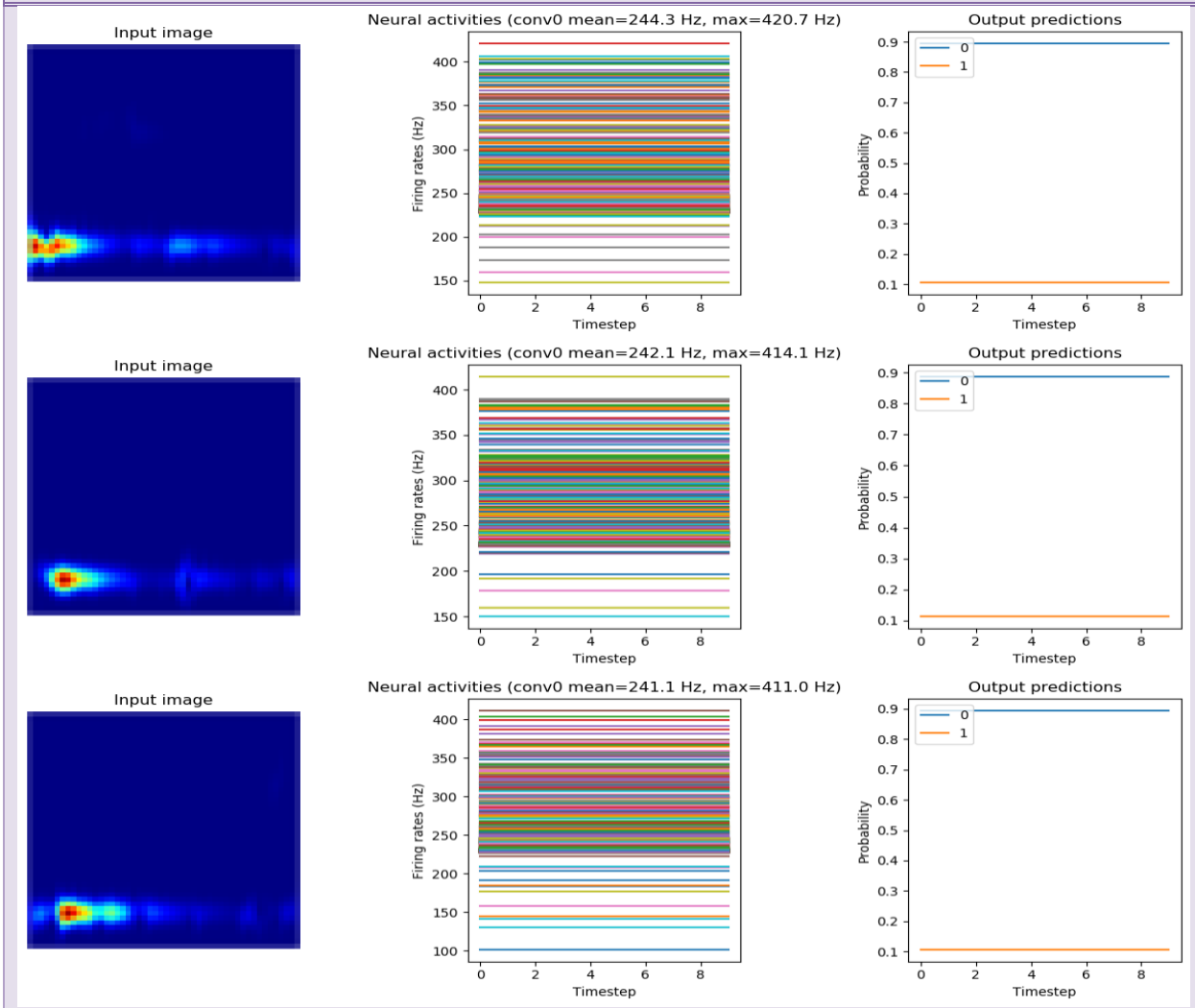
Scale= 10

Test accuracy: **100%**



ALTERNATIVA 2: OTTIMIZZAZIONE TEMPI DI FIRING (REGULARIZATION)

Test accuracy: **100%**



ALTERNATIVA 2: OTTIMIZZAZIONE TEMPI DI FIRING (REGULARIZATION)

Test accuracy: **100%**

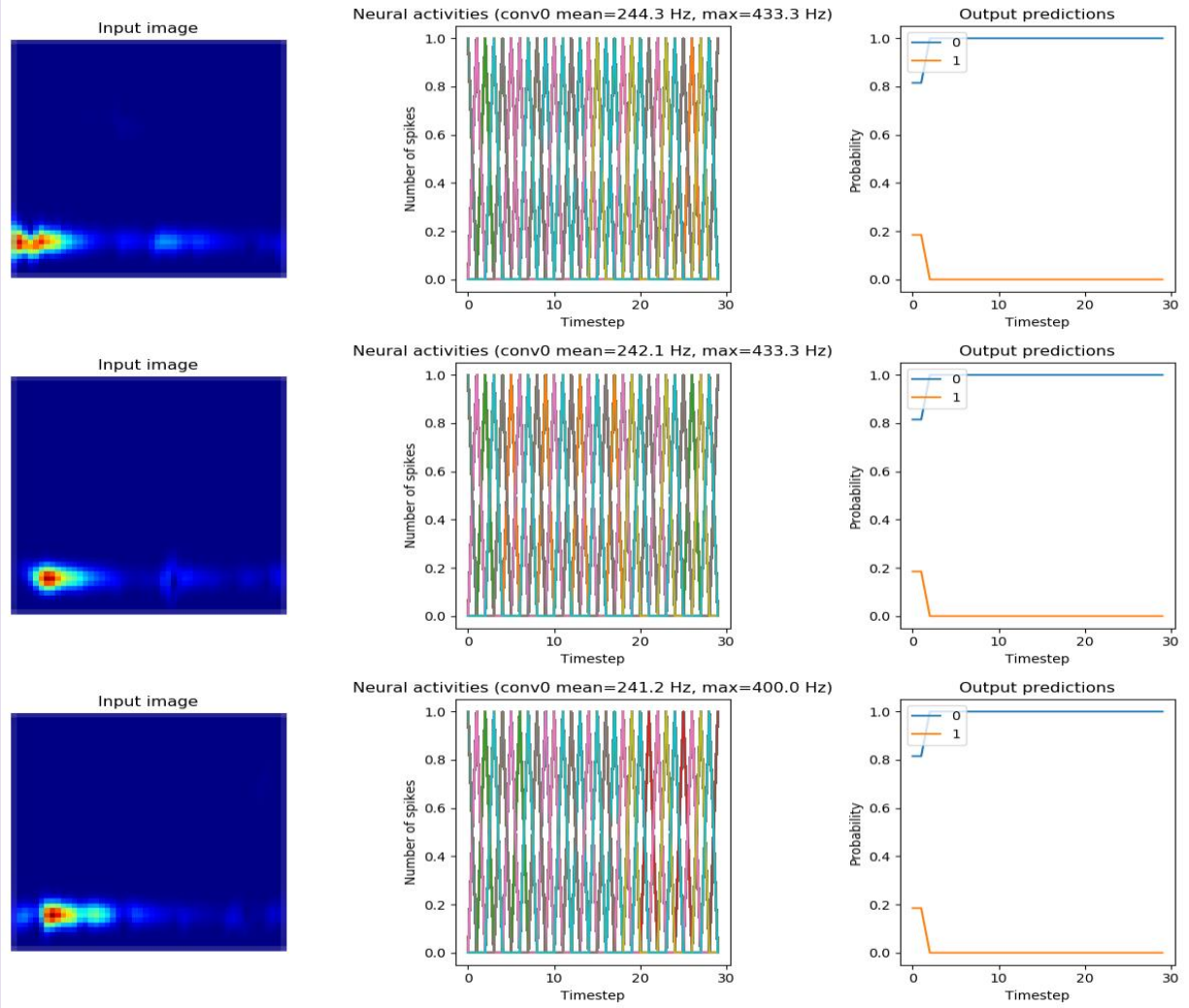


Tabella 12: Metodi per il miglioramento dell'accuratezza SNN a 16 filtri

In Tabella 13 sono infine confrontati questi risultati e viene valutato il miglioramento percentuale dell'accuratezza per la seguente architettura.

ACCURACY	%
CNN	**100%**
SNN	100%
SNN post Smoothing	100%
SNN post Scaling	100%
SNN post Regolarizzazione	**100%**
ACCURACY GAIN	**+0%**

Tabella 13: Accuracy Gain SNN a 16 filtri

L'esperimento con la rete a 32 filtri ha portato agli stessi risultati della rete a 16 filtri e quindi non sono riportati nell'elaborato.

Conclusioni

Questo studio ha presentato una possibile implementazione di una rete neurale Spiking per il monitoraggio strutturale basato sull'analisi di vibrazione.

In particolare, è stato proposto un metodo basato sull'utilizzo di tecniche di rappresentazione tempo-frequenza per la conversione in immagine del segnale ricavato dai sensori installati sul ponte Z24, case study dell'elaborato. L'utilizzo della Superlet ha reso possibile, in particolare, la creazione di un dataset a super-risoluzione che consentisse la damage detection sfruttando CNN come algoritmo di deep learning per il riconoscimento di pattern complessi.

Il modello CNN più semplice descritto nell'elaborato è riuscito a classificare con un'accuratezza del 98% il dataset creato. L'utilizzo di architetture hardware più potenti consentirebbe di impiegare immagini ad alta risoluzione senza la necessità di ridimensionamenti e conseguente perdita di dettaglio.

La conversione a SNN ha permesso un aumento dell'accuratezza della rete fino a 15 % rispetto all'originale CNN e può essere ulteriormente migliorata attraverso l'allargamento del dataset, in particolare la classe di dati relativi al guasto, per un miglior bilanciamento delle classi durante il training della rete.

A tale scopo è, però, necessario disporre di hardware parallelo low-power specializzato per applicazioni NPU (come mostrato in Figura 36).

Figura 36: NPU per SNN: Intel Loihi (a sinistra) e IBM TrueNorth (a destra)

Bibliografia

[1] BibLus BIM, «Monitoraggio strutturale: tecniche e strumenti,» 7 Marzo 2022. [Online]. Available: https://bim.acca.it/monitoraggio-strutturale-tecniche-e-strumenti/.

[2] lah, M., Nunez, I., Ben Chaabene, W. et al., «Machine Learning Algorithms in Civil Structural Health Monitoring: A Systematic Review. Arch Computat Methods,» n. June 2021, 2020.

[3] Abbas, S., Li, F., & Qiu, J., «A Review on SHM Techniques and Current Challenges for Characteristic Investigation of Damage in Composite Material Components of Aviation Industry. In Materials Performance and Characterization (Vol. 7, Issue 1, p. 20170167),» 2018. [Online]. Available: https://doi.org/10.1520/mpc20170167.

[4] L. Camorani, «Analisi mediante trasformata wavelet di segnali elettroencefalografici a riposo e durante compiti cognitivi,» 2020.

[5] Moca, V.V., Bârzan, H., Nagy-Dăbâcan, A. et al., «Time-frequency super-resolution with superlets,» p. 337 , 12 January 2021.

[6] B. Baldini, «Implementazione in Python e Tensorflow di una rete neurale convoluzione per la classificazione di veicoli,» 2020.

[7] R. Pramoditha, «The Concept of Artificial Neurons (Perceptrons) in Neural Networks,» Towards Data Science - Medium, 26 December 2021. [Online]. Available: https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc.

[8] «Convolutional neural network,» Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network..

[9] INTEL, «Neuromorphic Computing,» n. https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html.

[10] V. Lyashenko, «Basic Guide to Spiking Neural Networks for Deep Learning,» [Online]. Available: https://cnvrg.io/spiking-neural-networks/.

[11] M. W. E. N. X. W. G. L. G. D. M. B. D. S. J. W. D. L. Jason K. Eshraghian, «Training Spiking Neural Networks Using Lessons From Deep Learning,» 20221.

[12] F. Barchi, L. Zanatta, E. Parisi, A. Burrello, D. Brunelli, A. Bartolini e A. Acquaviva, «Spiking Neural Network-Based Near-Sensor Computing for Damage Detection in Structural Health Monitoring,» p. 219, 23 August 2021.

[13] Nengo AI, «Optimizing a spiking neural network,» [Online]. Available: https://www.nengo.ai/nengo-dl/examples/spiking-mnist.html.

[14] P. Reumers, 7 Marzo 2022. [Online]. Available: https://bwk.kuleuven.be/bwm/z24.

[15] S. D. a. B. /. UW-Madison, «Research: Introduction to Z24 Bridge and Health Monitoring,» [Online]. Available: https://byusdrg.com/modalanalysis/.

[16] «Tensorflow,» [Online]. Available: https://www.tensorflow.org/?hl=it.

[17] «Converting a Keras model to a spiking neural network,» Nengo, [Online]. Available: https://www.nengo.ai/nengo-dl/examples/keras-to-snn.html.

[18] «Introduction to NengoDL,» [Online]. Available: https://www.nengo.ai/nengo-dl/introduction.html.

[19] S. L. a. R. Lamba, «Spiking Neural Networks Vs Convolutional Neural Networks for Supervised Learning,» *International Conference on Computing, Communication, and Intelligent Systems (ICCCIS),* pp. 15-19, 2019.

Elenco delle Figure

Elenco delle Tabelle

Listings