SAPIENZA
UNIVERSITÀ DI ROMA

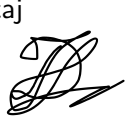
Marchiare una Rete Neurale? Sfide e limiti delle tecniche di Watermarking per la protezione della proprietà intellettuale dell'Intelligenza Artificiale.

Facoltà Ingegneria dell'informazione, informatica e statistica
Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Sapienza Università di Roma, Italia

Flavio Corsetti
Matricola 1997818

Relatore
Prof. Dorjan Hitaj



Correlatore
Prof. Fabio De Gaspari

Anno Accademico 2023/2024

Marchiare una Rete Neurale? Sfide e limiti delle tecniche di Watermarking per la protezione della proprietà intellettuale dell'Intelligenza Artificiale.
Sapienza Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: corsetti.1997818@studenti.uniroma1.it

Sommario

Il mondo è cambiato, l'intelligenza artificiale è diventata parte integrante delle nostre vite e la sua evoluzione sembra inarrestabile. I modelli di intelligenza artificiale rappresentano ormai risorse di grande valore. L'addestramento di questi modelli richiede un grande impiego di risorse, sia in termini computazionali che finanziari. Sempre più imprese decidono di inserire nelle loro strategie di business modelli di intelligenza artificiale, implementandoli sia nei processi di creazione sia nei beni e servizi finali offerti. Questa scelta conferisce un sostanziale vantaggio competitivo, che sta obbligando le aziende ad evolversi e adattarsi a questa tecnologia. Qui nasce il bisogno di un metodo per proteggere la proprietà intellettuale di queste risorse, per evitare che la competizione o attori malevoli possano appropriarsi indebitamente del nostro modello. Per questo ci viene in aiuto la tecnologia del DNN Watermarking. Questa tecnica è utilizzata per inserire in un modello di intelligenza artificiale un watermark, ovvero un'informazione che aggiunge proprietà e autenticità a un dato. Il watermark viene incorporato nei parametri della rete, rendendolo tutt'uno con il modello di intelligenza artificiale. Tuttavia, questi metodi non offrono una sicurezza assoluta. Le reti neurali rimangono vulnerabili e aperte a manipolazioni: i loro parametri possono essere modificati, compromettendo potenzialmente l'efficacia di questi metodi. Questa Tesi analizza approfonditamente, mediante test e sperimentazioni, le potenzialità e i limiti degli attuali metodi state-of-the-art del DNN Watermarking, includendo anche nuove strategie innovative di attacco e difesa. L'obiettivo finale è valutare la sostenibilità e l'efficacia a lungo termine di questa tecnologia, determinandone le prospettive future nel campo della protezione della proprietà intellettuale dell'intelligenza artificiale.

Indice

Capitolo 1

Introduzione

Ci avviciniamo ormai alla metà degli anni ’20 del 2000, una tecnologia che meno di dieci anni fa era quasi fantascienza, limitata a pochi, oggi è alla portata di tutti: pronta ad aiutare un bambino nei compiti a casa oppure un ingegnere nucleare che ha bisogno di aiuto nella propria centrale di competenza. Le applicazioni sono infinite, come la crescita quasi esponenziale, e ormai inarrestabile, di questa tecnologia: **l’Intelligenza Artificiale**.

All’interno di questo vasto campo che è l’intelligenza artificiale troviamo il **Deep Learning**; questa tecnica avanzata emula il cervello umano tramite delle Reti Neurali profonde composte da vari strati di neuroni artificiali, che riescono ad "imparare" in modo autonomo, a partire da vasti insiemi di dati, con minimo intervento umano.
Ormai le Reti Neurali ci vengono in aiuto virtualmente in tutti gli ambiti, dalla astrofisica all’intrattenimento, ma anche nei trasporti, infatti il sogno di molti scrittori di fantascienza è divenuto realtà: le automobili a guida autonoma non sono più il futuro, ma il presente, in città come Los Angeles, California, i robotaxi sono diventati parte integrante del panorama urbano e trasportano gli Angelenos per la città degli angeli con la stessa naturalezza di un servizio taxi tradizionale, ma ovviamente senza il tassista. Parallelamente, anche le automobili dei comuni privati cittadini sono dotate di sistemi di guida semi-autonoma sempre più avanzati, che assistono i conducenti nelle loro attività di guida giornaliere. Le Reti Neurali stanno rivoluzionando anche il campo della medicina, in particolare l’ambito della diagnosi precoce, infatti sono stati addestrati dei modelli di intelligenza artificiale capaci di analizzare delle mammografie e prevederne il possibile sviluppo di cancro al seno con un anticipo anche di 5 anni [38]. Non solo diagnosi di cancro, ma anche diagnosi precoci di autismo tramite la semplice analisi delle risonanze magnetiche [21]. Inoltre, più in generale, possiamo affermare che l’avvento delle DNN (Deep Neural Networks) ha rivoluzionato numerosi ambiti dell’informatica, portando progressi significativi in diverse aree, come: la **classificazione delle immagini** [18, 22], quindi riconoscere e categorizzare oggetti , seguendo gli esempi di cui sopra; nell’**elaborazione del linguaggio naturale** [2, 5, 11, 32, 35], quindi sistemi in grado di riconoscere e generare testo; la **generazione di contenuti** [4, 19, 24, 27, 30] come : dati, immagini, video e suoni .

Il boom dell’intelligenza artificiale è in gran parte da attribuire anche a ChatGPT, che è stato introdotto al mondo nel novembre del 2022. GPT, acronimo di Generative Pre-trained Transformer, è un modello di intelligenza artificiale sviluppato da OpenAI, concepito per

rendere l'IA accessibile a tutti, in modo intuitivo, rapido e prevalentemente gratuito. Il modello usa tecniche avanzate di elaborazione del linguaggio naturale per dare una risposta quasi umana su una vasta gamma di argomenti.

Analogamente alla clamorosa espansione dei siti web alla fine degli anni '90, molte aziende stanno intraprendendo una nuova e storica fase di trasformazione digitale, che si concentra sull'integrazione dell'intelligenza artificiale nei loro servizi. Questa evoluzione sta portando queste imprese a convertire e potenziare le loro offerte tramite l'adozione di modelli di intelligenza artificiale. Le aziende possono decidere di integrare nei propri servizi modelli già pre-addestrati, come ad esempio le API di OpenAI (ChatGPT). Questi modelli offrono un accesso a basso costo al supporto dell'IA, ma non sono utili per servizi molto specifici e soprattutto complessi; un azienda di robotaxi non può affidare la guida autonoma di una propria auto a ChatGPT. Per questo alcune imprese decidono di addestrare i propri modelli di intelligenza artificiale.

L'addestramento di un modello proprietario è un attività che evolve e ottimizza il servizio offerto dalle aziende, tuttavia questa pratica comporta sfide significative, sia dal punto di vista economico che tecnico. Partendo dal punto di vista tecnico l'addestramento è molto esigente in termini di risorse hardware, infatti più è prestante l'infrastruttura computazionale, minore sarà il tempo necessario per addestrare il modello. Ovviamente questo hardware deve essere gestito da personale specializzato che gestisce l'addestramento, e tramite le proprie competenze, cerca di addestrare il modello migliore possibile. Oltre al personale che si occupa dell'addestramento, si necessita di personale che gestisca il dataset: infatti il modello ha bisogno di dati, questi dati sono organizzati in classi, classi che rappresentano il tipo del dato. Questa operazione di recuperare i dati necessari all'addestramento ed etichettarli è un compito che non può essere affidato ad una macchina, ma è completa competenza dell'uomo. Dal lato economico, il costo dipende da: tipo e grandezza del modello che vogliamo addestrare, dall'hardware scelto, dal personale tecnico e soprattutto l'energia elettrica utilizzata, costo non trascurabile. In alternativa alla creazione di un dipartimento interno dedicato all'intelligenza artificiale, le aziende possono optare per soluzioni di cloud computing per l'addestramento e la gestione dei modelli IA. Questa opzione semplifica notevolmente il processo di calcolo dei costi associati all'addestramento e all'utilizzo dei modelli di intelligenza artificiale, infatti, senza contare il costo della creazione del dataset, il prezzo finale è calcolabile con la seguente formula [9]:

$$\text{Total cost} = \text{Price per chip-hour} \times \text{Number of chip-hour}$$

Possiamo affermare, con assoluta certezza, che l'addestramento di una modello rappresenti un'impresa finanziariamente onerosa. La **proprietà intellettuale** di questi modelli appartiene interamente ai loro creatori, rendendo il loro furto un reato perseguibile legalmente. La protezione di questi modelli deve considerata una *priorità assoluta*, è essenziale evitare che la competizione o attori malevoli possano appropriarsi indebitamente del modello e utilizzarlo come proprio. Per questo bisogna assicurarsi che l'**autenticità** e l'**integrità** di un modello sia sempre garantita e verificabile, con elevata sicurezza. Per questo viene in aiuto una vecchia tecnologia che è quella del **Watermarking**, tecnologia utilizzata per aggiungere ad un dato informazioni riguardo: origine, status, proprietà ed autenticità. Il **Watermark** viene incorporato nel dato in modo impercettibile e robusto, in modo tale che non possa essere rimosso [15].

Il **DNN Watermarking** (Deep Neural Network Watermarking), è una tecnica introdotta da Uchida et al. nel 2017 [34]. Questo lavoro ha aperto la strada a un nuovo campo di ricerca nella sicurezza informatica, presentando per la prima volta un metodo per incorporare un Watermark in una rete neurale. Da qui è stata tracciata la strada per nuove, innovative e sicure tecniche di watermarking [1, 14, 33, 28]. Ma, molti di questi metodi sono stati superati da contromisure e strategie anti-watermark [8], riflettendo la dinamica tipica dell'innovazione di attacco e difesa nella cybersecurity. Tutti i metodi state-of-the-art moderni per l'incorporazione di un watermark in una DNN condividono un principio fondamentale: fondere il watermark nei pesi del modello. La distinzione risiede nelle metodologie di forgiatura impiegate per integrare il watermark nei pesi della rete. I metodi esaminati in questa tesi condividono un'ulteriore caratteristica in comune, l'adozione di un approccio di verifica di tipo **White-Box**. Nel White-Box watermark verification, il *verificatore*, che sia esso il legittimo proprietario o l'avversario, necessita dell'accesso diretto al modello per constatare o rimuovere la presenza del watermark. Questo metodo di verifica si contrappone al metodo del **Black-Box** watermark verification, dove il verificatore effettua delle query mirate al modello [14], e tramite inferenza deduce la presenza del watermark nella rete. Con l'evoluzione delle tecniche di difesa, anche quelle di rimozione si sono sempre perfezionate. Uno dei casi più importanti è quello di *Refit: A Unified Watermark Removal Framework* [8] che rappresenta uno dei framework state-of-the-art per la rimozione di watermark. Tra le nuove strategie offensive, emerge anche la tecnica della **Quantizzazione**. Questo processo, solitamente utilizzato per ridurre la dimensione e rendere più veloce l'esecuzione del modello a scapito dell'accuratezza, modifica la rappresentazione del tipo di dato del modello diminuendone la precisione. Potenzialmente la quantizzazione è applicabile anche come difesa, la sua efficacia in tal senso non è ancora stata verificata.

In questo Elaborato esploreremo i *confini* e le *potenzialità* delle tecniche emergenti di watermarking, concentrandoci su uno dei metodi state-of-the-art più importanti ed efficaci: **Tattooed**[28]. L'obiettivo è analizzare criticamente questa nuova tecnologia e valutarne l'efficacia e le prospettive future. Attraverso un'analisi approfondita, cercheremo di comprendere se il watermarking rappresenti una soluzione duratura e affidabile o se sia destinato a rimanere una tecnologia con applicazioni limitate a causa delle sue intrinseche vulnerabilità.

1.1 Contributo

In questo paragrafo, presentiamo i principali contributi del nostro studio:

- Condurremo un'analisi approfondita delle tecniche state-of-the-art di watermarking, sottoponendole a metodi di attacco moderni ed efficaci per valutarne la robustezza e la resilienza.
- Implementeremo test di attacco basati sulla quantizzazione, esplorando l'efficacia di questa tecnica emergente nella rimozione del watermark e valutando la resistenza delle tecniche esistenti a questo approccio.
- Esamineremo il potenziale della quantizzazione come strumento difensivo, investigando se e come questa tecnica possa essere sfruttata per migliorare l'efficacia dei moderni metodi di watermarking.

1.2 Struttura

La Tesi è organizzata come segue: Il capitolo 2 fornisce la teoria necessaria a comprendere cos'è e come funziona una rete neurale con le sue architetture, dataset e modifiche, come la quantizzazione. Offre anche una nozione generale sull'argomento Watermark e soprattutto sul DNN Watermark. Il capitolo 3 parla in dettaglio di Tattooed, uno degli algoritmi moderni più all'avanguardia sull'argomento Watermarking per reti neurali. Il capitolo 4 descrive i vari setup sperimentali utilizzati come base per gli esperimenti nel capitolo successivo. Il capitolo 5 rappresenta il cuore della tesi, vengono mostrati tutti gli esperimenti effettuati su Tattooed e altri algoritmi di confronto. Il capitolo è strutturato in modo tale da mostrare nel dettaglio tutti i requisiti fondamentali per un algoritmo di DNN Watermarking sicuro. Nel capitolo 6 approfondiamo l'argomento del DNN Watermarking, discutendo della storia di questa scienza. Nel capitolo Lesson Learned, il 7, discutiamo brevemente dei risultati. Il capitolo 8 è il capitolo conclusivo di questa Tesi. Nel capitolo 9 troviamo l'Appendice, dove vengono presentati in dettaglio molti esperimenti sugli algoritmi mostrati.

Capitolo 2

Background

In questo capitolo esploreremo in modo approfondito la teoria: delle *Deep Neural Networks* (DNN), il loro funzionamento, i metodi di addestramento e i principali progressi che hanno portato nel campo dell'intelligenza artificiale; Del *DNN Watermarking*, la definizione e applicazione, le varie tecniche implementative, come funzionano e le metodologie state-of-the-art attualmente disponibili e infine approfondiremo nel dettaglio i metodi di inserimento dei Watermark nei modelli.

2.1 Deep Neural Networks

DNN o *Deep Neural Networks* (Reti Neurali Profonde) è un termine che abbraccia diverse tipologie di reti neurali, generalmente composte da molti *layers* (strati). In questa sezione ci concentreremo in particolare su due tipi di architetture: le **DFN**, ovvero *Deep Feedforward Networks*, e le **CNN**, ovvero *Convolutional Neural Networks*. Iniziamo però dalle fondamenta della teoria delle reti neurali.

2.1.1 Perché "rete neurale"

Una **rete neurale** è un modello computazionale che si inspira alle *reti neurali biologiche*. Questa struttura è composta interamente da strati interconnessi di **neuroni artificiali** che formano un *grafo diretto aciclico* 2.1. All'inizio e alla fine troviamo rispettivamente l'*input layer* e l'*output layer*, che rappresentano l'interfaccia della rete con i dati in ingresso e i risultati finali.

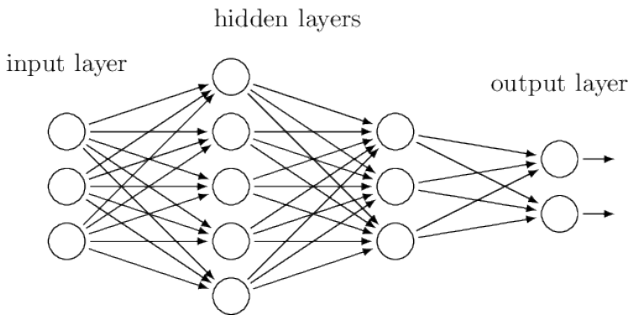


Figura 2.1. Rete Neurale generica

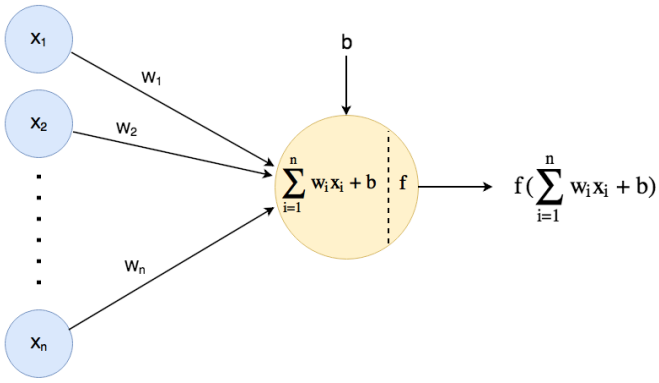


Figura 2.2. Neurone

Ogni neurone 2.2 riceve molteplici input, provenienti come output (x_i) dei neuroni dallo strato precedente. Ogni connessione tra neuroni è caratterizzata da un **weight** (peso) w_i, a cui è moltiplicato l'input x_i ricevuto; il peso rappresenta l'*importanza relativa* di ciascun input per il neurone e viene ottimizzato durante una fase precisa durante l'addestramento della rete. All'interno del neurone, tutti gli input *pesati*, insieme ad un **bias** b (utilizzato per regolare la funzione di attivazione), vengono sommati insieme durante un operazione chiamata *Funzione Somma* (2.1), dove n è il numero di neuroni del layer precedente.

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

Il risultato della somma viene poi passato attraverso la funzione f (chiamata anche σ) di attivazione. Questa funzione determina l'output finale del neurone (2.2).

$$f(z) \tag{2.2}$$

Come abbiamo specificato prima, le reti neurali sono strutturate in *layers* di neuroni interconnessi. All'inizio e alla fine troviamo rispettivamente l'**input layer** e l'**output layer**, che fungono da interfaccia tra la rete e il mondo esterno: il primo prende in entrata il dato grezzo, il secondo torna il risultato dell'elaborazione del modello. Tra questi due estremi si collocano gli **hidden layers**. Questi strati gestiscono l'intero processo di elaborazione e trasformazione dei dati. Il termine *deep* (profondo) deriva dal fatto che troviamo un gran numero di layer nascosti che collaborano per generare il risultato.

2.1.2 Insegnare ad una macchina

Il concetto di **addestramento** o apprendimento (learning) di una rete neurale è il processo attraverso cui la rete modifica i propri parametri interni in risposta ai dati in input. La rete acquisisce conoscenza attraverso l'*esperienza* in relazione a una specifica classe di tasks (compiti) e alle relative prestazioni. Il processo di addestramento è efficace solo se le *performance P*, misurate nell'esecuzione dei *task T*, mostrano un miglioramento, con l'accumularsi dell'*esperienza E* [12].

Il ***task*** *T* è il compito che la macchina deve risolvere, ma il processo di *imparare* non è il concetto di *task* stesso, ma significa *riuscire a crearsi la strada* per risolvere il task. Tramite le NN possono essere risolti vari tipi di tasks, eccone alcuni esempi:

- **classificazione**: associare una categoria ad un input.
- **regression**: prevedere un valore numerico dato un input.
- **Synthesis and sampling**: generare nuovi tipi di esempi che sono simili a quelli dei dati di training

Per valutare le abilità di una rete, dobbiamo scegliere un metodo per misurarne le ***performance*** *P* (prestazioni). Per task come la **classificazione** (unico task sperimentato in questo elaborato) misuriamo le performance del modello tramite l'**accuracy** (accuratezza), cioè la porzione di *dati correttamente categorizzati* dalla rete neurale. E' possibile ottenere la stessa informazione anche tramite l'**error rate**, che rappresenta la porzione di *dati erroneamente categorizzati* dalla rete neurale. Le prestazioni vengono sempre valutate su un insieme di dati sconosciuti al modello, quindi dati mai utilizzati in fase di addestramento. Questo insieme di dati è chiamato **test set**.

L'***experience*** descrive il tipo di dati scelti per l'addestramento. L'insieme di dati è chiamato **dataset** e comprende sia i dati per il **training** (addestramento), sia i dati per il **testing**. Esistono due tipi *experience*:

- **Supervised learning**: il dataset è composto da diversi dati, e ognuno di questi dati è classificato tramite un *label* (etichetta). Al modello è richiesta una previsione del tipo di etichetta del dato che gli diamo in input. Questo tipo di experience è utilizzata nel tipo di task, classificazione.
- **Unsupervised learning**: il dataset è composto da grandi quantità di dati raggruppati, ma non etichettati. Al modello è richiesto di imparare le diverse proprietà dei singoli dati nel dataset.

Come funziona effettivamente l'*apprendimento* di una rete neurale? L'obiettivo principale è individuare, a partire dai dati di input, la funzione che meglio rappresenta le relazioni presenti nel dataset. Il modello si sforza di apprendere una funzione che *mappi* gli input agli output in modo da rispecchiare il più fedelmente possibile le *corrispondenze* osservate nei *dati di addestramento*. Questo processo consente alla rete di **generalizzare** e fare previsioni su nuovi dati non visti in precedenza.
Un esempio fondamentale per comprendere meglio il concetto della **generalizzazione** è utilizzare l'algoritmo del **linear regression** (regressione lineare). Questo modello matematico è utilizzato per creare una sistema capace di prevedere il valore di uno scalare $y \in R$ in

output, dato un vettore in input $x \in R^n$ in input. La regressione lineare produce una funzione lineare che mappa l'input all'output previsto.

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b \tag{2.3}$$

La formula (2.3) rappresenta l'output dell'algoritmo, $\hat{y}$ è il valore che il modello prevede che si avvicini più ad y

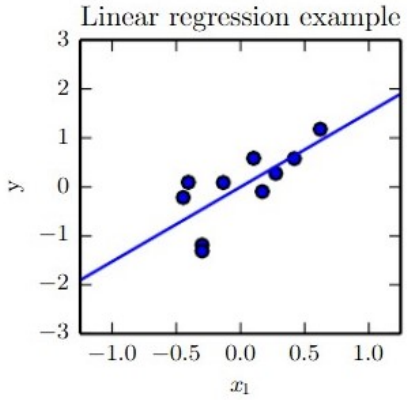


Figura 2.3. Esempio di generalizzazione con Linear Regression [12]

Come mostrato dalla figura (2.3), l'algoritmo ha elaborato una funzione generalizzata sui dati in input (rappresentati dai puntini), cercando di approssimare al meglio la distribuzione complessiva. Dalla figura possiamo anche notare un altro particolare: una *funzione lineare non è adatta alla generalizzazione*. Per questo motivo vengono utilizzate funzioni **non lineari**.

Una **funzione non lineare** è una qualsiasi funzione che non soddisfi le proprietà di linearità. Si distingue per la presenza di variabili di grado superiore al primo, o per l'uso di operazioni non lineari come radici o logaritmi. Ma la caratteristica più evidente è che il loro grafico *non è una linea retta*, quindi può assumere curve.

Le funzioni non lineari all'interno di una rete neurale vengono chiamate **funzioni di attivazione**, e definiscono il comportamento della funzione imparata dal modello. Esistono diversi tipi di funzioni di attivazione:

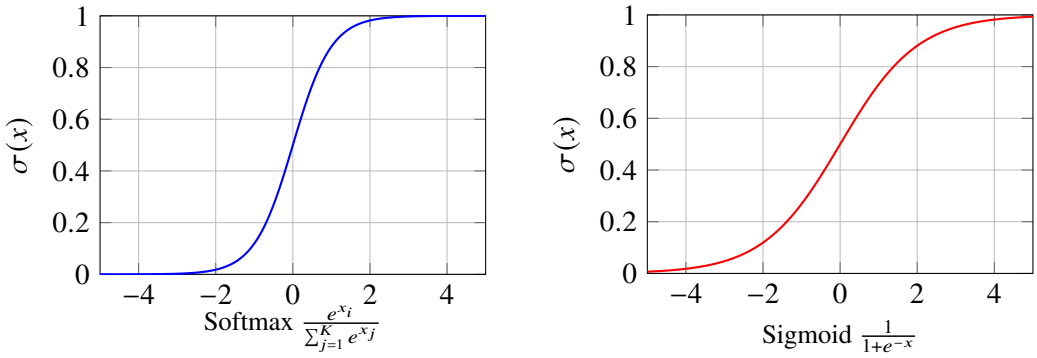


Figura 2.4. Funzioni di attivazione: Softmax e Sigmoid

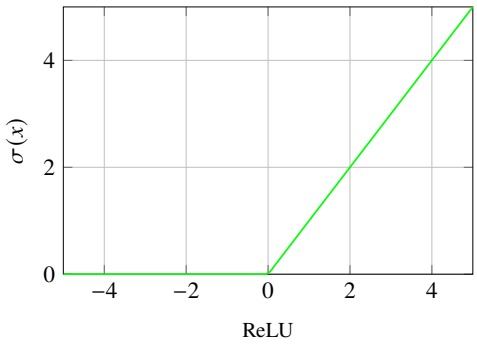


Figura 2.5. Funzione di attivazione: ReLU

Queste funzioni di attivazione, ciascuna utilizzata per scopi diversi, introducono non linearità al modello, cruciale come abbiamo già spiegato, per convergere verso soluzioni ottimali.

Le reti neurali necessitano anche di una **funzione di perdita** (o **Criterion**). Questa serve a quantificare quanto le previsioni del modello differiscano dai valori reali, guidando poi il modello all' ottimizzazione dei propri parametri. In sintesi, la *loss function* misura l'errore tra l'output previsto e il valore reale, ovviamente tutto in fase di addestramento. La loss function più utilizzata, solitamente usata per i task di classificazione, è la **Cross Entropy Loss**. Misura la differenza tra due distribuzioni di probabilità (2.4). $P(x)$ che rappresenta la distribuzione di probabilità target e $Q(x)$ che rappresenta la distribuzione predetta dal modello.

$$H(P,Q) = -\sum_{c=1}^{M} P(x)\log(Q(x)) \tag{2.4}$$

Un altro elemento molto importante di cui la rete ha bisogno è di un algoritmo di ottimizzazione. Questo algoritmo migliora il valore del **gradiente**, valore utilizzato per migliorare i valori degli weights e bias del modello. L'ottimizzazione mira a minimizzala funzione di perdita per avvicinarsi il più possibile al minimo locale. Per questo scopo, si utilizza l'algoritmo del **Gradient Descent**. Questo metodo sfrutta le derivate parziali della funzione di perdita rispetto ai parametri del modello, poiché queste indicano come modificarne i parametri per ottenere un miglioramento sulla funzione obiettivo.

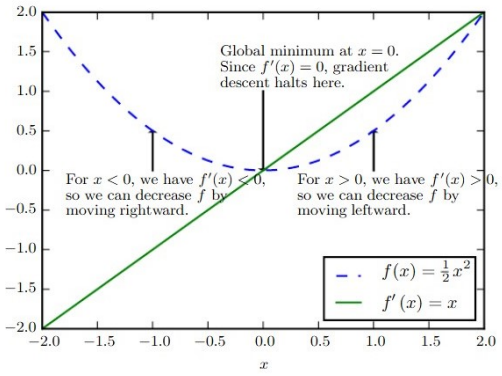


Figura 2.6. Un esempio su come il GD usa le derivate per trovare il minimo locale [12]

Nelle moderne reti neurali si utilizza una variante ottimizzata di questo algoritmo: lo **Stochastic Gradient Descent** (SGD). Questa tecnica è più efficiente della sua "versione base", poiche non migliora il gradiente utilizzando l'intero set di dati, ma opera con un sotto insieme chiamato **mini-batch**. Sebbene non garantisca il raggiungimento del minimo globale, è comunque in grado di convergere verso un minimo locale sufficientemente profondo (2.7).

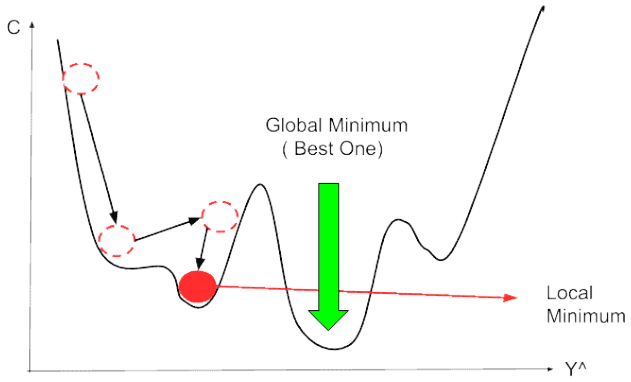


Figura 2.7. Esempio di funzionamento dell'algoritmo SGD

Algorithm 1 Stochastic Gradient Descent (SGD)

Require: Dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, learning rate η, numero di epoche T
Ensure: Parametri ottimizzati $\mathbf{w}$, b

Inizializza $\mathbf{w} \leftarrow \mathbf{0}$, $b \leftarrow 0$
for $t = 1$ to T **do**
  for $i = 1$ to n **do**
    Seleziona casualmente un esempio $(\mathbf{x}^{(i)}, y^{(i)})$ da $\mathcal{D}$
    Calcola la predizione $\hat{y}^{(i)} = f(\mathbf{x}^{(i)}; \mathbf{w}, b)$
    Calcola il gradiente della loss $\nabla_{\mathbf{w}} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
    Aggiorna i pesi: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
    Aggiorna il bias: $b \leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b}(\hat{y}^{(i)}, y^{(i)})$
  end for
end for
return $\mathbf{w}$, b

Tutti gli elementi che abbiamo esaminato costituiscono la fase di **Forward pass** (passaggio in avanti) in una rete neurale, riassumiamo i passaggi principali:

1. I dati in input vengono *propagati* attraverso la rete: passando per i vari layer, dove ogni neurone ogni neurone applica la propria ***funzione di attivazione***. Questo processo si ripete layer dopo layer arrivando infine all'output finali.

2. Dopo aver ottenuto l'output della rete, viene calcolato l'errore tramite la ***loss function***, che quantifica la discrepanza tra l'output previsto e quello desiderato.

Lo ***Stochastic Gradient Descent*** ci tornerà utile dopo aver spiegato l'ultima fase dell'addestramento della rete.

Dopo il *forward pass*, l'algoritmo di **Back-propagation** costituisce la fase di *backward pass*. Questo metodo calcola il *gradiente* della funzione di loss rispetto a ciascun peso della rete, permettendo cosi l'ottimizzazione dei pesi per minimizzare l'errore. Mentre nel forward pass l'informazione fluisce dall'input verso l'output, nel backward pass l'errore viene propagato nella direzione inversa, dall'output verso l'input. Il **gradiente**, elemento elaborato dall'algoritmo, è un vettore che contiene le derivate parziali di una funzione rispetto alle sue variabili; in questo contesto, la funzione è quella di loss e le variabili sono i pesi della rete neurale.

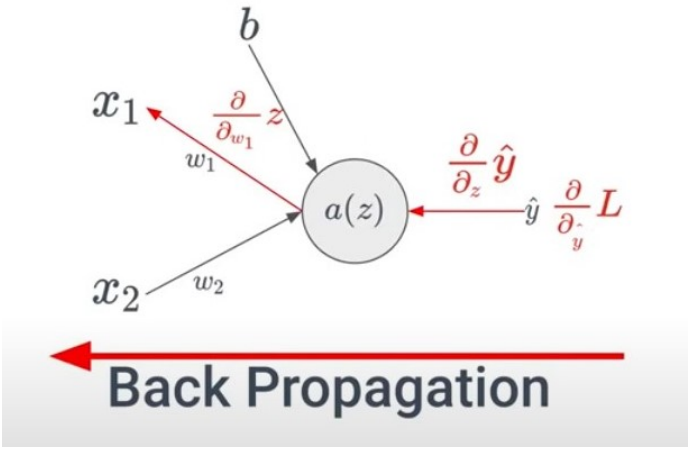


Figura 2.8. Fase di back propagation sul neurone

2.1.3 Layer Convoluzionali

Gli elementi esaminati nel sotto capitolo precedente sono le basi di ogni rete neurale, procediamo con un nuovo concetto che è quello della **Convoluzione** e dei layer Convoluzionali. Una **rete convoluzionale** (CNN) è una rete neurale che utilizza almeno un layer Convoluzionale nella propria architettura. Le reti con strati convoluzionali sono particolarmente efficaci nell'elaborazione di dati con strutture a griglia, come: immagini, audio e video.

La convoluzione è un operazione tra due funzioni di una variabile. In termini semplici, consiste nel far "scorrere" un filtro chiamato **kernel** sul dato in input. *Dato* che ha una forma a griglia e può anche essere multidimensionale (2.9). Il metodo effettua una moltiplicazione elemento per elemento tra il filtro e la porzione del dato ricoperto, per poi sommare i risultati per ottenere un singolo valore di output; output che viene chiamato **Feature map**. La Convoluzione bidimensionale, utilizzata su dati come *immagini* (tensori 3D) è descritta matematicamente in questo modo:

$$S(i,j) = (I * K)(i,j)) = \sum_m \sum_m I(m,n) \cdot K(i-m, j-n) \tag{2.5}$$

Dove I è l'immagine di input, K il kernel e S l'output risultante.

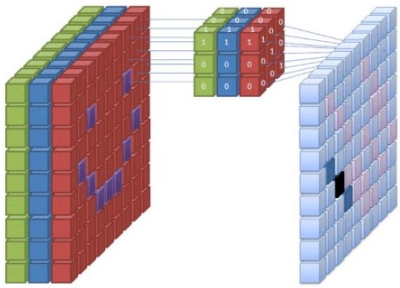


Figura 2.9. Convoluzione 2D su un immagine in input 3D (Canali RGB) con kernel 3D

Un tipico layer di una rete convoluzionale è composto da **tre stadi**:

1. Primo stadio: è presente il layer che esegue tutte le convoluzioni sull'input, in parallelo, generando un insieme di *linear activations* (funzioni identità) $f(x) = x$.

2. Secondo stadio: ogni *linear activations* viene passata ad una *nonlinear activation*. Questo stadio è chiamato **detector stage**. L'utilizzo di funzioni non lineari permette al modello di apprendere *pattern complessi.*

3. Terzo stadio: utilizziamo una **pooling function** per ridurre la dimensionalità spaziale delle feature map create negli stati precedenti.

Esistono due tipi di **pooling function** (2.10):

1. **Max Pooling**: seleziona i valori massimi all'interno di una finestra di pooling. Questo metodo è molto efficace nel conservare le caratteristiche più importanti di un dato.

2. **Average Pooling**: calcola la media dei valori in una finestra di pooling. Diminuisce il rumore nei dati.

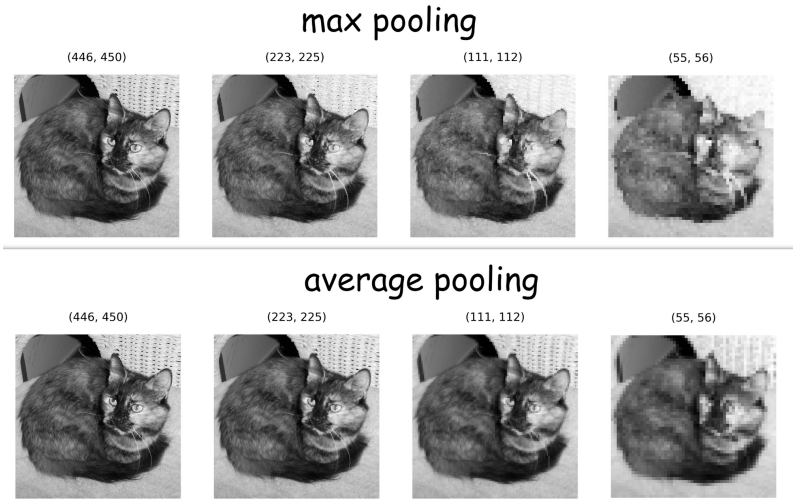


Figura 2.10. Differenze tra Max e Avg pooling in più strati convoluzionali

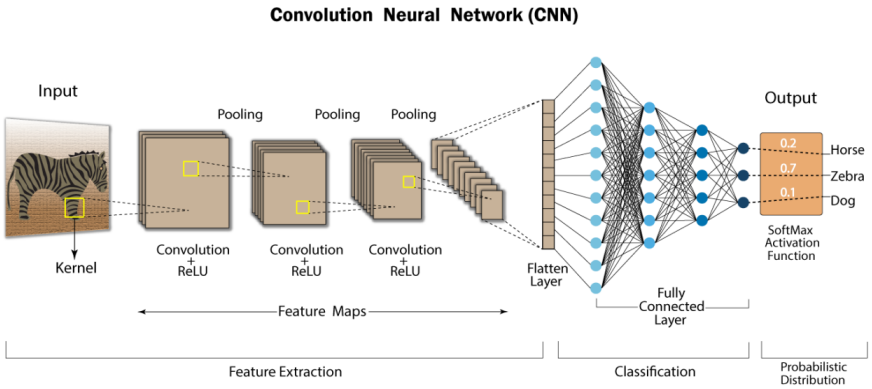


Figura 2.11. struttura di una generica CNN

2.1.4 Tipi e Architetture di DNN

Nei sotto capitolo precedenti abbiamo esaminato i vari tipi di layer che possono comporre una rete neurale. Combinando questi layer in varie configurazioni, otteniamo diverse architetture di reti neurali, riconducibili principalmente a due categorie: le **FNN** (Feedforward Neural Network) e le **CNN** (Convolutional Neural Network) CNN. Le FNN rappresentano le reti neurali di base, composte da layer di neuroni standard interconnessi. Le CNN, invece, si distinguono per la presenza di almeno un layer convoluzionale; sono particolarmente efficaci nell'elaborazione di dati multidimensionali, come *immagini*, *video* e *segnali audio*.

In questo elaborato verranno esposti esperimenti solo su due architetture di tipo Convoluzionale: **VGG** e **Resnet**.

VGG

VGG (Visual Geometry Group) è una rete Convoluzionale standard molto profonda. Il termine profondità o "deep" si riferisce al numeri di layer convoluzionali presenti nella rete. Esistono diverse varianti di questa architettura, ciascuna con una profondità differente, come VGG11, VGG16 [2.12] e VGG19. Questo modello è stato introdotto per la prima volta da A. Zisserman e K. Simonyan nel loro articolo "Very Deep Convolutional Networks for Large-Scale Image Recognition." [31]. Gli esperimenti discussi in questa Tesi si concentrano sulla versione **VGG11**, che è composta da undici strati convoluzionali.

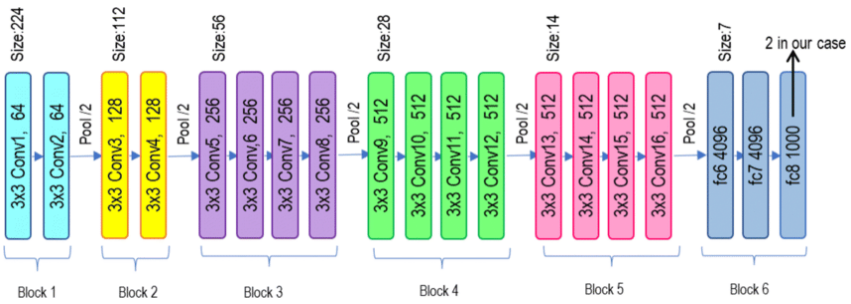


Figura 2.12. Struttura dell'architettura di VGG16

Resnet

Resnet (Residual Network) è una rete convoluzionale che introduce le ***connessioni residuali***, cioè "connessioni scorciatoia" che saltano un o più strati della rete. L'operazione di salto è effettuata in **blocchi residuali**. Ogni blocco impara una *funzione residuale* rispetto l'input. I blocchi residuali sono stati per la prima volta introdotti proprio in ResNet da *Microsoft* nel 2015 [16]. L'obiettivo principale era risolvere il problema della degradazione delle prestazioni che si verificavano aumentando la profondità della rete. I blocchi residuali permettono quindi, di avere reti molto profonde non avendo perdite di prestazione. Esistono diverse varianti di questa architettura, ciascuna con una profondità differente, come Resnet18 2.13, Resnet34, Resnet50, Resnet101 e Resnet152. Gli esperimenti discussi in questa Tesi si concentrano sulla versione **Resnet18**, che è composta da diciotto strati di profondità, di cui quattro con blocchi *base* residuali.

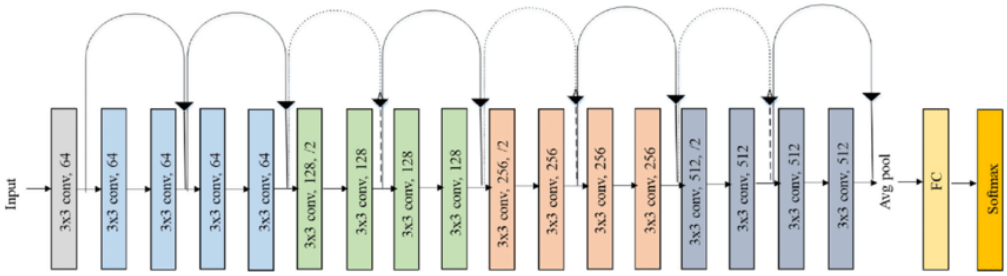


Figura 2.13. Struttura dell'architettua di Resnet18. E' possibile notare in alto i salti della rete

2.1.5 Dataset

I dataset sono *raccolte di dati categorizzati* utilizzati per **addestrare**, **validare** e **testare** i modelli. Ogni dataset è differente, cambiano: il numero di classi, il tipo di dato e la dimensione. In questa tesi incontreremo diversi tipi di dataset: **Cifar-10** 2.14, contenete 60 mila immagini a colori di dimensione 32x32 pixel. Le immagini sono suddivise in 10 classi rappresentanti: aerei, automobili, uccelli, gatti, cervi, cani, rane, cavalli, navi e camion; **ImageNet**, contenete 14 milioni di immagini ad alta risoluzione. Le immagini sono organizzate in più di 20 mila categorie; **GTSRB** 2.14 (German Traffic Sign Recognition Benchmark), contenente 50 mila immagini di segnali stradali tedeschi, a colori e di dimensione variabile. Le immagini sono suddivise in più di 40 classi diverse.

Figura 2.14. Prima figura: alcune immagini del dataset Cifar10. Seconda figura: alcune immagini del dataset GTSRB

2.1.6 Quantizzazione

Tra le nuove strategie offensive, emerge la tecnica della **Quantizzazione**. Questo processo, normalmente utilizzato per ridurre la dimensione e rendere più veloce l'esecuzione del modello, ma perdendo di l'accuratezza, altera la rappresentazione del tipo di dato del modello rendendolo meno preciso. La rappresentazione base del modello è in Float32, ma con la quantizzazione possiamo scendere a Float16 o con i metodi moderni, Int8. Potenzialmente la quantizzazione è applicabile anche come difesa, la sua efficacia in tal senso non è ancora stata verificata.

L'**FX Graph mode Quantization** è una tecnica moderna di quantizzazione automatizzata presente nella libreria Pytorch [29]. Utilizza il framework *fx* per convertire un modello in una rappresentazione intermedia, detta IR. In seguito, le operazioni di quantizzazione sono preformate su questa rappresentazione. Insieme a questa tecnica si utilizza il metodo del **Fake Quantization**. L'*fq* simula l'effetto della quantizzazione sul modello, prima che questa operazione avvenga. Viene attuata prima della quantizzazione stessa, e inserisce all'interno della rete operazioni "fake quant" per simulare il comportamento della rete post-quantizzazione. Per provare questa "falsa quantizzazione" i valori del modello vengono arrotondati per simulare la rappresentazione in int8. Il grande vantaggio che da la *Fake quantization* è permettere al modello di adattarsi agli effetti della quantizzazione. Questo avviene grazie ad un processo di adattamento sui dati di training, che viene effettuato sul modello "falsamente" quantizzato.

I modelli quantizzati in *FXG* utilizzano per rappresentare i loro dati il tipo *qint8*. Questo è un tipo di dato *specifico* per *tensori*. Al suo interno contiene dati utili alla quantizzazione e il dato effettivo che è in formato **int8**. Questo tipo di rappresentazione è utilizzato per rappresentare numeri interi, utilizzando *8bit*. Il range è [-128,127]. Le operazioni con questo tipo di dato sono 4 volte più veloci del classico dato a 32 bit, ma ovviamente il range di numeri rappresentabili diminuisce in modo drastico.

Figura 2.15. Esempio di quantizzazione

2.2 DNN Watermarking

In questa sezione esploreremo il concetto di Watermark per reti neurali. Ci concentreremo su tre aspetti fondamentali: la definizione di watermark nel contesto di reti neurali, il loro meccanismo di funzionamento e le diverse tipologie esistenti.

2.2.1 Cos'è un Watermark

Un **watermark** è un informazione che identifica l'*origine*, *status* e *proprietà* di un dato. L'operazione che aggiunge questa informazione è detta Watermarking. Il watermark viene incorporato in modo impercettibile e robusto nel dato, in modo tale che non possa essere scovato e rimosso. Nel mondo reale, i watermark sono ampiamente utilizzati, soprattutto in foto, video e documenti. Lo scopo finale del watermark è proteggere la **proprietà intellettuale** del dato, consentendo di dimostrare la proprietà in caso di furto o uso non autorizzato. *Su questa pagina è possibile notare un esempio di watermark.*

2.2.2 Watermark in DNN

Vista il grande costo e l'importanza delle moderne reti neurali, si è sentita la necessità di trovare un modo per proteggere questi modelli. Il primo approccio al DNN Watermarking è stato proposto da Uchida et al. nel 2017 [34], dell'università di Tokyo. Nel documento da loro pubblicato sono state create le basi per questa nuova tecnica. Soprattutto, sono stati definiti i requisiti fondamentali che un algoritmo di embedding deve soddisfare per una corretta implementazione del watermark (requisiti completati da [1, 6]). I requisiti per uno schema di DNN Watermarking sicuro sono:

- **Fidelity** (Fedeltà): requisito che garantisce che l'algoritmo di *marking* non comprometta in modo significativo le prestazioni del modello.
- **Robustness** (Robustezza): requisito che garantisce che l'algoritmo di *marking* incorpori il watermark in modo tale che non possa essere eliminato da manipolazioni comuni a cui il modello è esposto durante l'uso quotidiano in scenari reali.
- **Security** (Sicurezza): requisito che garantisce che l'algoritmo di *marking* incorpori il watermark in modo sicuro in modo tale che non possa essere eliminato da manipolazioni malevoli. Se l'attore malevolo riesce a rimuovere il watermark dal modello, l'algoritmo è considerato sicuro se:
  - La rimozione rende il modello inutilizzabile.
  - La rimozione ha un costo in termini di risorse (tempo, dati, denaro...) superiore al costo che il legittimo proprietario ha speso per produrre il modello.
- **Integrity** (Integrità): requisito che garantisce che in assenza di manipolazioni sul modello con watermark, l'algoritmo di *estrazione* deve recuperare in modo completo e intatto il watermark inserito.
- **Efficiency** (Efficienza): requisito che garantisce che l'algoritmo possa inserire e verificare la presenza del watermark con un impatto computazionale minimo.

Esistono due approcci di verifica del watermark: **White-box** e **Black-box** verification.
Nell'approccio ***White-box verification*** il verificatore, che esso sia un attore malevolo o il proprietario, ha bisogno dell'accesso diretto al modello per verificare la presenza del watermark. Il watermark viene estratto dai dati del modello stesso.
Nell'approccio ***Black-box verification*** il verificatore non ha bisogno dell'accesso diretto al modello, ma necessita semplicemente della capacità di poter interrogarlo tramite query mirate. Il watermark quindi viene restituito direttamente in output dal modello stesso.
L'approccio visto in questa tesi è Il **White-box verification**. Metodo dove il verificatore, con intenzioni malevoli o no, necessita dell'accesso diretto al modello per **inserire**, **verificare** o **rimuovere** il watermark. Per accesso diretto si intende che l'attore ha la disponibilità fisica del file contenente il modello. Nel metodo White-Box il watermark viene incorporato direttamente nei parametri del modello, coinvolgendo sia gli strati **Convoluzionali** che in quelli **completamente connessi**. Le *differenze* che coinvolgono i *diversi algoritmi* White-box riguardano il metodo in cui viene modellato il dato da integrare e alla tecnica di inserimento utilizzata.
Uno degli algoritmi più semplici per integrare un watermark, che si distingue per la sua semplicità, offrendo una base concettuale utile per comprendere le tecniche più sofisticate, è quello del **LSB** (Least Significant Byte). Questo metodo incorpora il watermark sostitendo il bit meno significativo dei pesi selezionati per l'embedding. Il pattern di inserimento è variabile: i bit possono essere inseriti in sequenza continua, alternandosi in pesi pari o dispari, o distribuiti in modo deterministico mediante una chiave segreta. La struttura del dato da incorporare è composta all'inizio e alla fine da chiave segreta, utilizzata per garantire l'integrità del watermark. Quindi, viene preso il dato, convertito in binario e delimitato dalla chiave segreta. Vengono scelti i pesi dove incorporare il watermark, che, convertiti anche loro in binario vengono modificati e avviene l'embedding. Questo metodo è molto semplice, ma anche molto poco sicuro. Infatti, il watermark può essere **eliminato** con un semplice **fine-tuning** in poche epoch (2.16 , 2.17).

Figura 2.16. Prima immagine: Watermark aggiunto al modello con tecnica LSB. Seconda immagine: Watermark dopo poche epoch di fine-tuning.

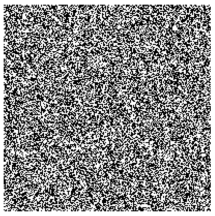


Figura 2.17. Alla fine del processo di fine-tuning questo è quello che rimane del watermark

2.2.3 Tattooed in breve

Tattooed [28] è un algoritmo all'avanguardia per l'inserimento di watermark in una rete neurale. E' stato introdotto nel 2022 da Pagnotta et al. dell'Università La Sapienza di Roma, rappresenta una tecnica di tipo *White-box* che aggiunge il watermark nei parametri del modello utilizzando il **CDMA** (Code Division Multiple Access) spread spectrum channel coding, di cui parleremo più nello specifico nel sotto capitolo successivo. **Tattooed** costituisce il punto centrale della Tesi, essendo uno dei più importanti e sicuri algoritmi di Watermarking disponibili nel campo dell'intelligenza artificiale.

Tattooed inserisce il watermark nella DNN senza comprometterne le prestazioni. Il suo punto di forza è la possibilità di inserimento del watermark in virtualmente qualsiasi fase del ciclo di vita del modello, sebbene la fase ottimale sia post-addestramento. Inoltre, integra il watermark direttamente nei parametri del modello, rendendolo indipendente dall'architettura e, di conseguenza, *universalmente applicabile*.

I creatori di Tattooed dichiarano che il loro metodo soddisfi tutti i **requisiti fondamentali** per il DNN watermarking. Una parte degli esperimenti di questa tesi mirano a verificare criticamente e potenzialmente confutare tale asserzione.

Chameleon DNN

Robust and Large-payload DNN Watermarking via Fixed, Distribution-Optimized, Weights [33] o semplicemente **Chameleon DNN** è un altra moderna tecnica di Watermarking di reti neurali. E' stato introdotto nel 2022, dopo Tattooed, da Tondi et Al. dell'Università di Siena. Rappresenta una tecnica di tipo *White-box* che aggiunge i parametri nel modello tramite *direct sequence spread spectrum watermarking* [3], che verrà introdotto nel sotto capitolo successivo. Questo algoritmo è stato selezionato per un confronto diretto con Tattooed, data la loro affinità nel metodo di embedding del watermark nel modello. Chameleon DNN adotta un approccio simile a Tattooed per l'embedding del watermark del modello. Tuttavia, il protocollo di incorporamento è differente: in questo metodo, il watermark viene aggiunto necessariamente pre-addestramento. Pesi corrispondenti al watermark vengono poi *congelati*, impedendo cosi la loro modifica durante la fase di addestramento. L'algoritmo lascia all'utente la scelta dei layer specifici per l'inserimento del Watermark. La distribuzione del watermark all'interno di questi layer avviene in modo intelligente. I pesi scelti per ospitare la loro porzione di Watermark vengono decisi con un alto grado di *pseudo casualità*, rendendo molto difficile l'identificazione del pattern dei parametri scelti per contenere il watermark.

2.3 Tecniche di Spreading

In questo sotto capitolo esamineremo in dettaglio la teoria e i meccanismi delle **tecniche di spreading**.

2.3.1 Direct Sequence Spread Spectrum

Il Direct Sequence Spread Spectrum o **DSSS** è una tecnologia utilizzata nelle trasmissioni wireless a "frequenza diretta" a banda larga, dove ogni bit del dato dell'informazione viene trasmesso come una sequenza ripetuta di valori, detta *chip*. Il segnale viene deliberatamente diffuso su una banda di frequenza più ampia rispetto a quella necessaria. Il processo di espansione avviene moltiplicando il segnale originale, tramite l'operatore **XOR**, con una sequenza *pseudo-casuale* di bit detti ***chip***. Espandere la frequenza del segnale aumenta in modo considerevole la resistenza del segnale stesso alle interferenze che possono accadere in una normale trasmissione wireless. Per ridurre di nuovo il segnale al suo stato originale, il ricevitore deve possedere lo stesso insieme di chip pseudo-casuali utilizzati per ampliare il segnale.

Nelle trasmissioni wireless, le comunicazioni avvengono in modo normale utilizzando i valori binari 0 e 1. Tuttavia, nelle le tecniche di watermarking che esaminaremo, i bit del watermark subiscono un trattamento particolare. Si applica una **codifica bipolare**, che trasforma la rappresentazione dei bit: lo 0 diventa -1, mentre l'1 rimane invariato.

2.3.2 Code Division Multiple Access

Il Code Division Multiple Access o **CDMA** è una tecnologia utilizzata nelle trasmissioni wireless, e permette a vari trasmettitori di inviare nello stesso momento informazioni sulla stessa banda, senza rischiare nella collisione delle trasmissioni stesse. Utilizza la tecnologia del *DSSS*, per espandere la frequenza di banda, e uno speciale schema di codifica, dove ogni trasmettitore ha un codice univoco. CDMA utilizza di base la tecnica della codifica bipolare, e la applica alla codifica delle sequenze di *chip*. Ogni trasmettitore possiede la propria sequenza univoca di chip; per decodificare il messaggio, il ricevitore deve conoscere la sequenza specifica di ogni utente, altrimenti il segnale apparirà come semplice rumore di fondo. L'incapacità di riconoscere la differenza tra rumore e informazione senza il chip è fondamentale. Questa caratteristica impedisce l'identificazione del watermark incorporato nei parametri della rete, poiché l'informazione è assolutamente indistinguibile.

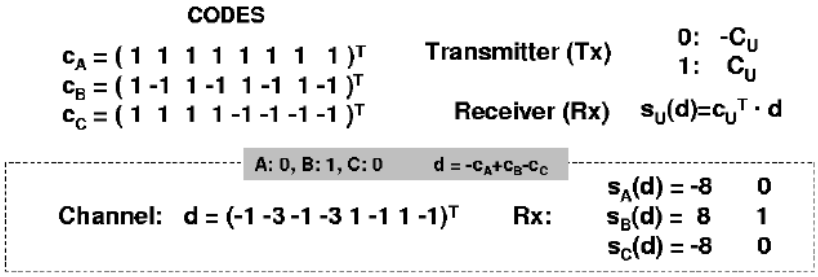


Figura 2.18. Esempio di funzionamento di una trasmissione con CDMA

Capitolo 3

Tattooed

Tattooed è un algoritmo *state-of-the-art* di **Watermarking** per Reti Neurali. Tattooed utilizza la tecnica del **CDMA** per incorporare il watermark nei parametri (weights) del modello. Nel contesto del Watermarking, l'applicazione del CDMA viene adattata in questo modo: ognuno degli *m-bits* che compongono il Watermark viene gestito come un utente unico che trasmette simultaneamente nella rete, in concorrenza con gli altri $m - 1$ utenti (bit), simulando cosi una telecomunicazione multiutente. A ciascun bit viene assegnato uno **spreading code** univoco, composto da una serie di chip, che viene utilizzato per codificare il bit. Per decodificare il watermark dalla rete il *verificatore* avrà poi bisogno di ognuno degli spreading code utilizzati.

Tattooed prevede che il legittimo proprietario del modello incorpori un **watermark** m di dimensione *L-bits* nei vettore dei *parametri* del modello W. Gli *L-bits* vengono codificati con una rappresentazione bipolare: lo 0 diventa -1, mentre l'1 rimane invariato. Ogni bit è accompagnato da uno **spreading code**, rappresentato da un vettore c_i composto da valori ±1, di lunghezza $|W|$. I vettori contenenti gli spreading code sono rappresentati da una matrice $Lx|W|$ chiamata C. L'inserimento avviene in questo modo (3.1), dove γ rappresenta la forza del segnale di watermark nei parametri del modello.

$$W_{wtm} = W + \gamma C m \tag{3.1}$$

W_{wtm} rappresenta il vettore dei parametri del modello contenente il Watermark. Per recuperare il watermark dalla rete si ha bisogno della matrice C, matrice che è segreta. Il recupero di ciascun bit i avviene in questo modo (3.2), $c_i^\top$ rappresenta la trasposta di c_i.

$$y_i = sign(c_i^\top W_{wtm}) \tag{3.2}$$

Funzione *sign()* definita in questo modo:

$$\text{sign}(x) = \begin{cases} -1 & \text{se } x < 0 \\ 0 & \text{se } x = 0 \\ 1 & \text{se } x > 0 \end{cases} \tag{3.3}$$

3.1 Algoritmo di Mark

Esploriamo l'algoritmo che inserisce il watermark nel modello.

Algorithm 2 *Mark*: Algoritmo di inserimento del Watermark

Require: Model: W, Int: γ, Str: m, Str: key, Float: $ratio$
Ensure: Model: W_{wtm}

1: $spreading\_code\_seed, ldpc\_seed, params\_seed \leftarrow seed\_gen(key)$
2: $ldpc \leftarrow$ init_ldpc($ldpc\_seed$)
3: $c \leftarrow ldpc$.encode(m)
4: $R = \text{len}(W) \cdot ratio$
5: PRNG($params\_seed$)
6: $indices \leftarrow \text{random}([0, \text{len}(W)], \text{size} = R)$
7: PRNG($spreading\_code\_seed$)
8: $preamble \leftarrow \text{random}([-1, 1], \text{size} = 200)$
9: $b \leftarrow \text{concatenate}(preamble, c)$
10: $i \leftarrow 0$
11: **while** $i < \text{len}(b)$ **do**
12: $\quad code \leftarrow \text{random}([-1, 1], \text{size} = R)$
13: $\quad signal \leftarrow \gamma \cdot b[i] \cdot code$
14: $\quad W_{wtm}[indices] \leftarrow W[indices] + signal$
15: $\quad i \leftarrow i + 1$
16: **end while**

Mark (Algoritmo 2) prende in input:

- Il **modello** W, in cui deve essere inserito il Watermark.
- Il parametro γ, che rappresenta la forza del watermark sui parametri del modello.
- Il **watermark m**, codificato e pronto per essere incorporato.
- La **chiave segreta**.
- Il **ratio**, che rappresenta la porzione dei parametri del modello che conterrà il *watermark*.

La calibrazione ottimale del parametro γ e del **ratio** è cruciale. Il *tuning* errato di questi valori può risultare in un watermark troppo debole o, al contrario, ad uno eccessivamente invasivo, che può compromettere le prestazioni del modello. La **chiave segreta** è composta da 512 bit, ed è utilizzata per per generare tre *secret seeds* (**linea: 1**):

- ⋆ Lo ***spreading_code_seed***, utilizzato per settare il seme di generazione per lo spreading code (insieme di chips).
- ⋆ Il ***ldpc_seed*** (*Low Density Parity Check* o codice di *Gallager*), utilizzato per settare il seme di generazione del codice di *correzione degli errori*, rappresentato da una matrice di parità, a bassa densità di 1.
- ⋆ Il ***param_seed***, utilizzato per settare il seme di generazione per la scelta degli indici dei parametri del modello che conterranno il watermark.

Proseguendo, incontriamo l'implementazione del codice di correzione degli errori basato su *LDPC*. Nelle ***righe 2 e 3*** generiamo la **matrice generatrice** *ldpc*. Successivamente, tramite la matrice generatrice e il messaggio originale, viene prodotto il messaggio codificato ***c***. A questo messaggio viene introdotta *ridondanza*. Questo processo aggiunge dati supplementari utili ad aumentare *affidabilità* e *resistenza* agli errori. Per ogni bit del messaggio originale, questo processo ne genererà 2 di output.

A ***linea 4***, viene calcolato ***R***. Questo valore, calcolato utilizzando il **ratio**, è utilizzato per decidere il numero di parametri che conterranno il Watermark.

In seguito, viene impostato il seed tramite ***PRNG*** (Pseudo Random Number Generator), utilizzando il seme *param_seed*. Questa operazione cambia il comportamento della funzione generatrice di numeri casuali, rendendo i valori creati deterministici, quindi riproducibili. In questo modo, la sequenza di numeri generati sarà sempre la stessa per un dato seed. Alla ***riga 6*** generiamo gli *indici che incorporeranno il watermark* nel modello. Il range della funzione è impostato sulla dimensione complessiva del modello, mentre il numero di elementi da generare è fissato al valore *R*.

In ***riga 7*** impostiamo, per il resto del codice a venire, il **PRNG** con lo ***spreading_code_seed***.

In seguito generiamo casualmente il **preambolo**, che in seguito viene concatenato alla testa del messaggio ***c***, generando la nuova stringa ***b*** (***riga 8 e 9***). Il *preambolo* viene aggiunto al messaggio poiché l'algoritmo *LDPC* richiede una stima del livello di rumore per decodificare efficacemente il messaggio durante la fase di recupero.

A ***riga 11*** troviamo il ciclo *While* che intera su ogni bit del messaggio ***b***. Inizialmente viene calcolato lo *spreading code* del bit (***riga 12***), che viene inserito nella variabile *code*. In seguito, inizia il processo di embedding tramite ***CDMA***. Tramite il γ, il *bit iterato* e lo *spreading code*, viene generato il *vettore* contenente il segnale, *signal*, codificato (***riga 13***). Per finire, il vettore ***signal*** viene incorporato tramite somma all'interno dei parametri del modello scelti per ospitare il Watermark.

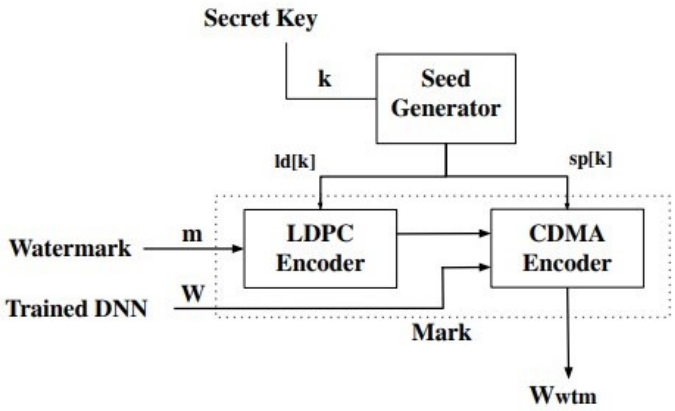


Figura 3.1. Grafico rappresentante la procedura di *Mark* [28]

3.2 Algoritmo di Estrazione

Esploriamo l'algoritmo che estrae il Watermark dal modello.

Algorithm 3 *Extract*: Algoritmo di estrazione del Watermark

Require: Model: W Str: key, Float: $ratio$, Int: wtm_length
Ensure: Model: m'

1: $spreading\_code\_seed, ldpc\_seed, params\_seed \leftarrow seed\_gen(key)$
2: $ldpc \leftarrow$ init_ldpc($ldpc\_seed$)
3: $R = \text{len}(W) \cdot ratio$
4: PRNG($params\_seed$)
5: $indices \leftarrow \text{random}([0, \text{len}(W)], \text{size} = R)$
6: PRNG($spreading\_code\_seed$)
7: $preamble \leftarrow \text{random}([-1, 1], \text{size} = 200)$
8: $y \leftarrow []$
9: $i \leftarrow 0$
10: **while** $i < wtm\_length$ **do**
11: $\quad code \leftarrow \text{random}([-1, 1], \text{size} = R)$
12: $\quad y_i \leftarrow transpose(code) * W[indices]$
13: $\quad y.append(y_i)$
14: $\quad i \leftarrow i + 1$
15: **end while**
16: $gain \leftarrow mean(multiply(y[: 200], preamble))$
17: $sigma \leftarrow std(multiply(y[: 200], preamble)/gain)$
18: $snr \leftarrow -20 * log_{10}(sigma)$
19: $m' \leftarrow ldpc.decode(y[200 :]/gain, snr)$

Extract (Algoritmo 3) prende in input:

- Il **modello** W, da cui deve essere estratto il watermark.
- La **chiave segreta** per la generazione dei seed.
- Il **ratio**, che rappresenta la porzione dei parametri del modello che contiene il *watermark*.
- La lunghezza totale del Watermark inserito nella fase di Mark.

La parte iniziale dell'algoritmo è speculare all'algoritmo di *Mark*, con l'unica differenza che non vengono eseguite operazioni sul messaggio da inserire nel modello (***Da riga 1 a 9***). A ***riga 10*** inizia il ciclo *While* che intera su ogni bit del *Watermark*. Inizialmente viene calcolato lo spreading code di un bit, sempre nel range di R, che viene inserito nella variabile *code*. In seguito, a ***riga 12***, viene calcolato il prodotto scalare tra la trasposta del vettore di *spreading code* e il vettore che contiene i valori degli indici contenenti il watermark. Il risultato è un singolo valore scalare. Per finire, viene inserito in coda del vettore *y* il valore appena calcolato.

Uscendo dal While, a ***riga 16,17 e 18***, andiamo a calcolare il livello di *rumore* e il *gain*, che poi verranno utilizzati per normalizzare il segnale e migliorare la decodifica.

In ***riga 16***, viene calcolato il ***gain***. Questo valore serve come fattore di normalizzazione per il segnale, calcolando una stima dell'ampiezza del segnale stesso rispetto al preambolo noto.

A ***linea 17*** è calcolata la *derivazione standard* σ, questa operazione stima l'ampiezza del livello di *rumore* del segnale estratto. Utilizza il preambolo noto, e i primi 200 valori del segnale estratto (che rappresentano nel valore estratto, il preambolo) per poi normalizzare il risultato.

A ***riga 18*** viene calcolato il **SNR** (Signal-Noise Ratio), ovvero il rapporto segnale rumore.

Per finire, a ***riga 19***, avviene la decodifica tramite **LDPC**. Vengono decodificati tutti i valori dopo il preambolo, che vengono divisi per il *gain* per essere normalizzati. In seguito, tramite il **SNR** calcolato, viene decodificato il segnale. Il *risultato* è il Watermark contenuto nei parametri del modello.

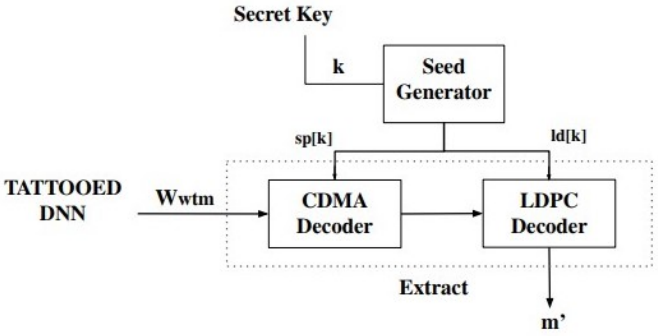


Figura 3.2. Grafico rappresentante la procedura di *Estrazione* [28]

3.3 La Quantizzazione del modello.

Abbiamo analizzato in dettaglio i meccanismi di *Mark* e di *Estrazione* del Watermark dal modello. Abbiamo esaminato operazioni molto cruciali e sensibili come il **CDMA** e l'**LDPC**, le quali richiedono un elevata precisione dei valori in input per funzionare correttamente e per produrre risultati *affidabili*. Il processo di **quantizzazione** altera la rappresentazione del tipo di dato dei parametri del modello, riducendone, quindi, la precisione numerica. Questa trasformazione comporta il passaggio da *Float32* a formati a precisione minore come *Float16* o persino *Int8*. Questa operazione, rapida e computazionalmente leggera, rischia, in modo elevato, di *eliminare il Watermark dal modello*. Tuttavia, questo apparente svantaggio può essere utilizzato a nostro ***vantaggio***. Infatti, inserire un watermark in una rete già quantizzata potrebbe risolvere il problema, in quanto la quantizzazione è un operazione *idenpotente*.L'operazione di *quantizzazione* porta con se altri vantaggi, come la diminuzione considerevole della dimensione del modello e dell'aumento di velocità di esecuzione dello stesso, migliorando le prestazioni complessive. Ma, questi "benefici" portano con se il compromesso di una riduzione dell'accuratezza del modello, non sempre accettabile.

3.4 Modelli di minaccia

Esaminiamo in dettaglio tutti i potenziali attacchi finalizzati alla rimozione del watermark dai parametri del modello. Attacchi che hanno come *unico vincolo* quello di **preservare le prestazioni del modello**, senza degradarne troppo l'accuratezza.

L'**Avversario** è un entità che è riuscita ad impadronirsi in modo illegittimo di un modello protetto da **Proprietà Intellettuale**. Questa entità è consapevole della possibilità che nel modello sia presente un Watermark. Per cercare di eliminarlo impiega tutti i possibili attacchi state-of-the-art esistenti. L'avversario **vince**, se riesce a rimuovere il Watermark senza che il costo di tale operazione superi quello di produzione del modello rubato.

In qualsiasi istante, l'**avversario *non può determinare*** la presenza e il tipo di Watermark nei parametri del modello, rendendolo quindi incerto sull'esito dei suoi tentativi di rimozione.

3.4.1 Un'anteprima sugli attacchi

Discutiamo in breve dei **tipi di attacco** che l'avversario può decidere di attuale sul nostro modello con Watermark.

- **Refit**: *A Unified Watermark Removal Framework* [8] è un framework creato con lo scopo di fornire diversi tipi di strumenti per rimuovere un watermark da un modello. Strumenti che si basano sul **finetuning**: **EWC** (Elastic Weight Consolidation) e **AU** (Augmentation with Unlabeled Data).
- **Transfer learning**: tecnica utilizzata per riaddestrare un modello già addestrato su un altro compito su un nuovo compito *simile*, ma diverso.
- **Pruning**: metodo che consiste nel "potare" il modello di alcuni nodi. Riduce la complessità di una rete rimuovendo parti non essenziali.
- **Quantizzazione**: metodo utilizzato per cambiare la rappresentazione del tipo di dato dei parametri del modello.

Tutti questi attacchi verranno discussi in modo accurato nel Capitolo sperimentale.

Capitolo 4

Setup Sperimentale

In questo capitolo discuteremo di scelte e configurazioni riguardanti gli esperimenti attuati sui diversi Framework di Watermarking.

4.1 Architetture e Dataset

Le architetture valutate in questa tesi sono **Resnet-18** [17] e **VGG-11** [31]. Questi due tipi di *reti* sono specializzati nella *classificazione di immagini*. Infatti, li andremo ad addestrare su un *Dataset* composto da immagini, **Cifar-10**. Più informazioni sulle architetture e sul dataset sono disponibili nel Capitolo: *Background*.

4.2 Watermark

Verrà valutato un solo tipo di watermark di tipo ***testuale***. La scelta della dimensione è ricaduta su due misure: ***152 bit***, corrispondenti a *19* caratteri in formato Ascii. Il testo scelto per questa misura è: "*Lorem ipsum libero.*"; ***512 bit***, corrispondenti a *64* caratteri in formato Ascii. Il testo scelto per questa misura è: "*Lorem ipsum dolor sit amet, consectetur adipiscing elit viverra.*";
Le dimensioni scelte non sono casuali, 64 caratteri rappresentano una lunghezza adeguata per la dimensione di un testo utilizzabile per il riconoscimento della proprietà intellettuale. La scelta di sperimentare con un testo di lunghezza inferiore serve a verificare in modo più accurato la sicurezza del dei vari algoritmi di Watermarking (dimensione che comunque rimane accettabile per l'inserimento di informazioni riguardanti *IP*).

4.3 Parametri utilizzati in Tattooed

I parametri che andiamo a prendere in considerazione per Tattooed sono: ***gamma*** e ***ratio***. Per confermare i risultati degli esperimenti proposti [28] dai creatori dell'Algoritmo, sono stati sperimentati la maggior parte dei tipi di ***gamma*** proposti. Molti esperimenti si concentrano sul gamma di $\gamma = 9 \times 10^{-4}$. Valore che, secondo i creatori di Tattooed, fornisce un buon *trade-off* su i requisiti di *fidelity* e *security*. Il resto degli esperimenti sui valori di gamma rientra nel range $[9\times10^{-3}, 6\times10^{-3}, \dots, 9\times10^{-4}, \dots, 1\times10^{-4}, 9\times10^{-5}, \dots, 3\times10^{-5}]$.

Negli esperimenti relativi al **Ratio**, è stata usata una vasta gamma di valori differenti. Tuttavia, il valore più frequente nei setup sperimentali è quello che seleziona circa 200.000 parametri del modello. E' importante notare che questo valore di ratio varia a seconda dell'architettura, poiché il numero totale di parametri differisce tra modelli. Essendo il ratio espresso in percentuale, il valore effettivo non è fisso, ma si adatta alle dimensioni specifiche di ogni architettura. L'architettura **Resnet-18** contiene circa 11.2 *Milioni* di parametri. Invece, **VGG-16** ben 128 *Milioni*.

La conduzione degli esperimenti è stata strutturata in modo da garantire una distribuzione eterogenea dei valori di ***Gamma*** e ***Ratio***. Ciascuna configurazione di γ è stata testata su molteplici valori di *Ratio*, e viceversa. Questo metodo ha permesso di ottenere un'analisi completa sulle prestazioni di Tattooed. Una serie di esperimenti, invece, sono stati condotti in condizioni di *setup* ritenute bilanciate e sufficentemente robuste. Questo approccio ha permesso di ottenere una visione completa dell'algoritmo nei "suoi" scenari ottimali.

4.4 Tipi di Attacco

Consideriamo adesso tutti i tipi di attacco implementati e i loro vari setup.

4.4.1 Refit

Refit, A Unified Watermark Removal Framework For Deep Learning System with Limited Data, è un framework [8], sviluppato da Chen et Al. nel 2019. Questo framework sfrutta algoritmi che si basati sul fine-tuning per eliminare un Watermark da una rete neurale, senza richiedere conoscenze preliminari sul watermark stesso. Gli algoritmi di fine-tuning di Refit hanno la particolarità di poter lavorare con una quantità limitata di dati, elemento essenziale in scenari reali, dove l'avversario potrebbe essere in possesso solo di una quantità limitata del dataset utilizzato per addestrare il modello rubato. I due algoritmi proposti in *Refit* sono: ***EWC*** (Elastic Weight Consolidation) e ***AU*** (Unlabeled Data Augmentation).

EWC

L'**EWC** o *Elastic Weight Consolidation*, è una tecnica sviluppata da Kirkpatrick et Al. [20], che è utilizzata per per mitigare il fenomeno del *Catastrophic forgetting*, fenomeno dove un modello che viene addestrato per un nuovo *task* ***B*** tende a dimenticare il *task* passato ***A***. L'idea alla base di questo metodo è rallentare l'apprendimento nei pesi del modello essenziali per le conoscenze apprese nel *task* principale e aggiornare i pesi che sono stati considerati meno importanti. Per capire l'importanza dei pesi nella rete, l'EWC calcola esplicitamente nella fase di pre-addestramento sul nuovo *task* ***B***, l'importanza di ciascun peso della rete nel *task* ***A***. Lo fa utilizzando la matrice di informazione di **Fisher**. Il rallentamento è ottenuto aggiungendo una penalità sulla funzione di perdita della rete, in questo modo l'algoritmo limita l'aggiornamento dei pesi in base alla loro importanza nel *task* ***A***, limitando cosi il **Catastrophic forgetting**. L'EWC richiede in input il valore λ, che rappresenta nella formula utilizzata come funzione di perdita 4.1, il parametro che controlla il fattore di

penalità dell'EWC nell'addestramento sui nuovi task, quindi l'importanza del mantenere "memorizzati" i vecchi compiti.

$$L(\theta) = L_B(\theta) + \frac{\lambda}{2} \sum_i F_i(\theta_i - \theta^*_{A,i})^2 \tag{4.1}$$

Verrà utilizzato per il parametro λ il **valore 150**. Valore che, secondo i creatori di *Refit*, offre un giusto bilanciamento tra forza di rimozione del WM e mantenimento delle prestazioni del modello. E' importante notare che, in questo caso, il task ***B*** utilizza lo stesso dataset del task ***A***.

AU

L'**AU** o *Unlabeled Data Augmentation* è una tecnica che sfrutta una grande quantità di dati non etichettati per migliorare le prestazioni di un modello. In questo caso, viene utilizzato per modificare in modo drastico la base di conoscenze della rete. Questo approccio è funzionalmente realistico, poichè i dati non etichettati facilmente reperibili in grande quantità da internet senza sforzi significativi. In questo contesto viene utilizzata un istanza alternativa del dataset *ImageNet*, **ImageNet32**. Questa versione comprende tutti i dati di training e validazione dell'originale, ma con immagini ridimensionate in formato 32x32 pixel. La scelta di questo dataset è consigliata dai creatori di *Refit*. Il fine-tuning si effettua utilizzando una porzione del dataset originale e una parte dei dati non etichettati. Questo valore, denominato ***m*** e richiesto come input, viene impostato a **50**, come consigliato.

L'**EWC** e il **AU** possono essere impiegati **contemporaneamente**, sebbene questa operazione non è completamente consigliata dai creatori di Refit. Tuttavia, per completezza e spirito di ricerca, è stata comunque testata.

4.4.2 Transfer Learning

Il transfer learning è una tecnica che consiste nel riutilizzare una rete già pre-addestrata su un compito come base di partenza di un nuovo compito. L'idea base è che le informazioni apprese in precedenza possono essere riutilizzate per accelerare e migliorare l'apprendimento su un compito simile. Solitamente, viene utilizzato un modello pre-addestrato su un dataset generico come **ImageNet**. Nel nostro caso, verrà utilizzato come base il modello addestrato su Cifar-10, aggiunto di watermark. Il dataset utilizzato è **GTSRB**.

4.4.3 Pruning

Il **Parameter Pruning** è un metodo che mira a ridurre la complessità e le dimensioni di una rete neurale, mantenendone le prestazioni intatte. Il termine *Pruning* è tradotto in italiano come "Potatura", infatti questa tecnica consiste nella rimozione selettiva di parametri da un modello addestrato. La prima fase del pruning è identificare i parametri meno significativi

nel modello, questi parametri sono quelli meno utilizzati nella previsione del risultato. La seconda fase è l'eliminazione di questi parametri dal modello.

4.4.4 Quantizzazione

La novità introdotta in questa tesi è l'utilizzo della quantizzazione per attaccare un watermark in una rete neurale. I Test sono stati svolti su quantizzazione a **Int8** tramite *FX Graph mode Quantization*. Gli esperimenti sono stati effettuati sia in **situazioni di attacco**, dove si tenta di eliminare il watermark da una rete tramite quantizzazione. Sia in fase **difensiva**, quindi testeremo se il watermark può essere inserito in una rete neurale quantizzata.

Capitolo 5

Sperimentazioni e Valutazioni

Questo capitolo presenta i risultati ottenuti degli esperimenti condotti su Tattooed, offrendo un'analisi dettagliata e un interpretazione approfondita sui dati raccolti. Tutti i test, salvo dove diversamente specificato, sono stati condotti con i *setup* descritti nel capitolo precedente. La struttura di argomentazione di questo capitolo seguirà i *requisiti fondamentali di sicurezza* per uno schema di *DNN Watermarking*.

5.1 Fidelity

Il requisito di *Fidelity* (fedeltà) rappresenta il fondamento essenziale su cui si basano tutti gli altri requisiti di sicurezza. Un algoritmo di Watermarking che garantisce questa caratteristica deve preservare le *prestazioni originali del modello*, evitando un deterioramento significativo del *task*. In questo caso, la soglia accettabile di perdita di accuratezza *non deve superare* l'**1%**, rispetto al modello originale. La tabella 5.1 mostra il confronto di prestazioni tra il modello *baseline* e quello marchiato con *Tattooed*.

Architettura	**Dataset**	**Baseline TER (%)**	**Tattooed TER (%)**	**Tattooed BER (%)**	**TER Diff (%)**
ResNet-18	CIFAR-10	10.20	10.54	0.0	-0.34
VGG11	CIFAR-10	9.83	9.81	0.0	0.02

Tabella 5.1. Prestazioni del modello baseline confrontato con Tattooed.

In questa tabella, tutti i dati sono rappresentati in forma di **Error Rate** ($1 - Accuracy$). I modelli base sono stati addestrati per 45 *Epoch* con un batch size di 256 elementi. Il learning rate utlizzato per *ResNet*: 6×10^{-2} e per *VGG*: 2×10^{-4}. Nelle le versioni con watermark è stato utilizzato, in entrambi i casi, un watermark di dimensione corrispondente a **512 bit**. Il Ratio è stato scelto in modo tale da arrivare a circa 200.000 parameti per ognuna delle architetture, e il valore di γ utilizzato è stato di 9×10^{-4}.
Le prestazioni dei modelli con Watermark in entrambe le architetture dimostrano che: l'algoritmo e i metodi scelti per inserire il Watermark all'interno della rete neurale non deteriorano, in una quantità non accettabile, le prestazioni del modello sul task originale. Questa analisi ci conferma che Tattooed soddisfa il requisito di ***Fidelity***.

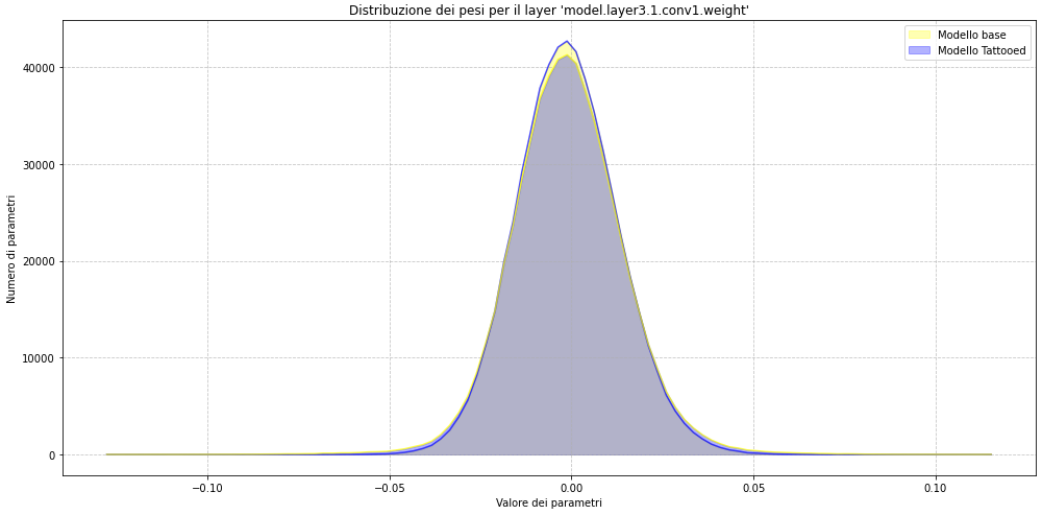


Figura 5.1. Grafico che mostra la distribuzione dei parametri Prima e dopo l'embedding del Watermark su ResNet

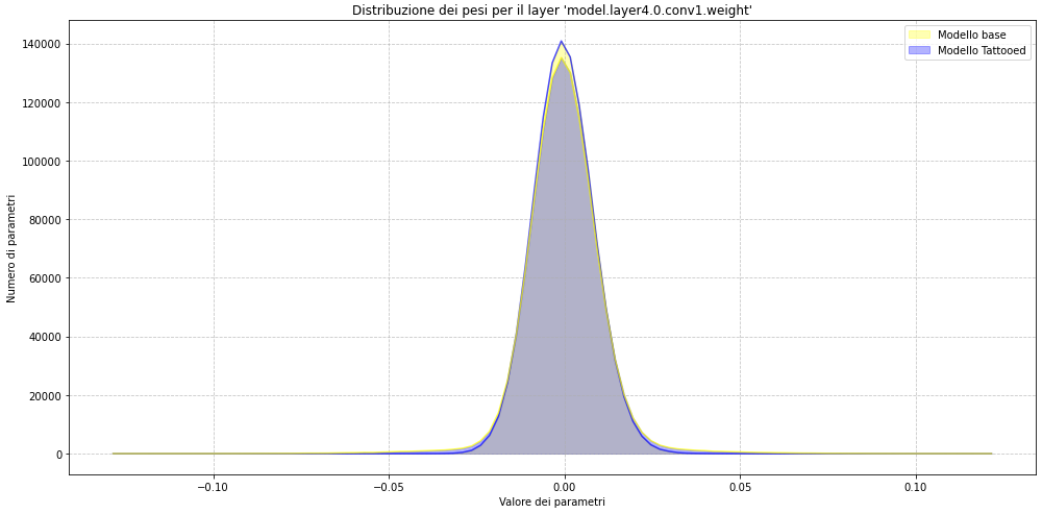


Figura 5.2. Grafico che mostra la distribuzione dei parametri Prima e dopo l'embedding del Watermark

I grafici mostrano la distribuzione dei parametri su layer specifici del modello, confrontandolo stato pre e post embedding del Watermark. L'analisi rileva una distribuzione dei pesi approssimativamente identica, dimostrando graficamente il minimo impatto dell'inserimento del sui parametri del modello, quindi anche sulle sue prestazioni. Inoltre, questa caratteristica impedisce la possibilità di dedurre, dalla distribuzione dei parametri, la presenza del watermark nella rete.

5.2 Robustness

Il requisito di *Robustness* (robustezza) garantisce che il watermark non possa essere rimosso da manipolazioni comuni, cioè manipolazioni utili e necessarie all'utilizzo funzionale del modello stesso. La garanzia di *Robustezza* è soddisfatta quando è possibile estrarre il watermark in modo corretto e sopratutto "*completo*", cioè che non riporti errori dati da corruzione. Il **BER** (*Bit Error Rate*) accettabile, in ogni situazione di recupero nel watermark, è dello **0%**. Un valore differente da questo, fa considerare il watermark **perso**, quindi inutilizzabile per la verifica della **proprietà intellettuale**.
Le manipolazioni comuni, che un modello può attraversare nel suo ciclo di vita sono: il *Fine-tuning* e il *Transfer Learning*. Il processo di **fine-tuning** è utilizzato per migliorare le prestazioni di un modello sullo stesso task, ci riferiamo a questa operazione come **RTAL** (*Retrain All Layers*). Questo metodo è stato applicato su entrambe le architetture utilizzando il *classico dataset Cifar-10*. Ma, per garantire che il modello fosse riaddestrato su un insieme di dati inediti, il dataset è stato diviso equamente in due parti, ognuna composta da 30.000 elementi. Inizialmente è stato addestrato, con la prima metà, per poi essere aggiunto di Watermark [5.4]. In seguito questo modello è stato riaddestrato con la seconda metà [5.2 , 5.3]. Questa operazione è effettuata con un *LR* più basso di quello di training, ma per motivi di sicurezza, è stato testato anche con *LR più alti*. Il **Transfer learning** è utilizzato per addestrare un modello, già pre-addestrato su un compito, come base di partenza di un nuovo compito. Ci riferiamo a questa operazione come **RTAL**. Per questo metodo è stato scelto di riaddestrare i *modelli con Watermark* visti nella tabella [5.1] sul dataset **GTSRB**. Visualizziamo prima i risultati sugli esperimenti di **RTAL**:

LR	Pre-RTAL TER (%)	Post-RTAL TER (%)	TTD BER'(%)
0.06	12.84	11.11	0.0
0.006	12.84	7.48	0.0

Tabella 5.2. RTAL su **ResNet18**

LR	Pre-RTAL TER (%)	Post-RTAL TER (%)	TTD BER'(%)
0.002	13.63	10.90	0.0
0.0005	13.63	12.01	0.0

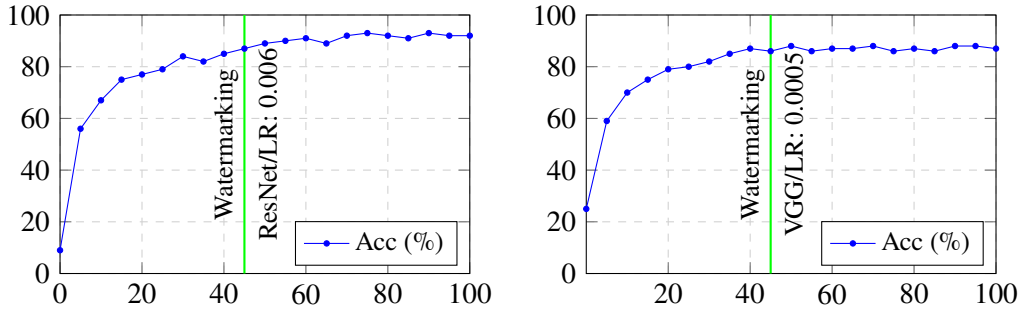
Tabella 5.3. RTAL su **VGG**

Architettura	Dataset	Baseline TER (%)	Tattooed TER (%)	Tattooed BER (%)	TER Diff (%)
ResNet-18	CIFAR-10 50%	12.80	12.84	0.0	-0.04
VGG11	CIFAR-10 50%	13.78	13.63	0.0	0.15

Tabella 5.4. Prestazioni dei modelli addestrati e marchiati sulla prima metà dei dati. **RTAL**

La tabella [5.4] contiene i dati sulla prima parte del **RTAL**. I due modelli sono stati addestrati con la prima metà dei dati del dataset per *45 epoch*. In seguito sono stati inseriti di Watermark. In tutte e due le architetture il watermark è stato aggiunto su circa *200.000 parametri* con un *gamma* di 9×10^{-4}. Nelle tabelle [5.2 , 5.3] sono stati inseriti gli esperimenti sulla seconda parte del **RTAL**. Tutte e due le architetture sono state addestrate in una sessione da 100 epoch. Sono stati provati due tipi di **Learning Rate**: uno più alto di quello della prima fase di addestramento e uno più basso. Per **ResNet18** sono stati scelti come LR: 6×10^{-2} e 6×10^{-3}. Per **VGG11** sono stati scelti come LR: 2×10^{-3} e 5×10^{-4}. E' importante

notare che tutti i test sulla seconda parte sono stati provati sulla stessa istanza dei modelli, ma ovviamente ogni esperimento è stato condotto su un istanza indipendente.



Dall'analisi delle tabelle [5.2 , 5.3] emerge che: nei **55 epoch** successivi all'applicazione della funzione di Watermarking e dal cambio di dataset, i modelli hanno registrato un migliorato sulle prestazioni sul loro task. In alcuni casi, si osservano anche notare miglioramenti superiori ai 3 punti percentuale. I grafici [5.2 , 5.3] offrono una rappresentazione visiva di questo processo, evidenziando come l'inserimento del Watermark ad *epoch 45* non abbia modificato il comportamento della rete. E' infine possibile notare che al termine dei test, il **BER** del watermark rimane **0.0**, dimostrando cosi la robustezza dell'algoritmo di Watermarking al processo di **fine-tuning**.

Analizziamo adesso gli esperimenti sul **Transfer Learning**:

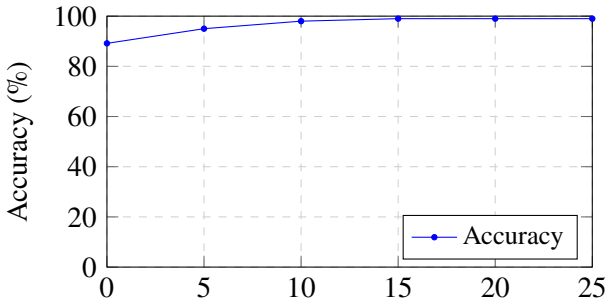
LR	**Pre-TL TER (%)**	**Post-TL TER (%)**	**TTD BER'(%)**
0.06	10.54	5×10^{-5}	0.0
0.006	10.54	1×10^{-4}	0.0
0.09	10.54	1×10^{-4}	0.0

Tabella 5.5. Transfer Learning su **ResNet18**

LR	**Pre-TR TER (%)**	**Post-TR TER (%)**	**TTD BER'(%)**
0.0002	9.81	3×10^{-4}	0.0
0.0005	9.81	11×10^{-4}	0.0
0.00009	9.81	11×10^{-4}	0.0

Tabella 5.6. Tranfer Learning su **VGG**

Come base di questi esperimenti sono stati utilizzati i modelli visualizzabili nella tabella [5.1], ovviamente muniti di Watermark. Le impostazioni dei watermark sono: 200.000 parametri con un gamma di $\gamma = 9 \times 10^{-4}$. Le tabelle [5.5 , 5.6] contengono gli esperimenti sul Transfer Learning. I modelli sono stati addestrati per *25 epoch* sul dataset **GTSRB** con un batch size di 512. sono stati testati diversi tipi di **LR**, in modo tale da avere una visione completa del comportamento del modello e del watermark con parametri diversi.



Dall'analisi delle tabelle [5.5 , 5.6] emerge che: nei **25 epoch** di Transfer Learning i modelli si sono adattati in modo perfetto al nuovo dataset, mostrando un accuratezza che sfiora la perfezione. Questo dimostra che i parametri del modello sono cambiati e si sono adattati al nuovo dataset. Il grafico offre una rappresentazione visiva di questo cambiamento, evidenziando la curva di apprendimento del modello. E' quindi possibile notare che al termine di tutti i test, il **BER** dei Watermark nei modelli rimane **0.0**, dimostrando la robustezza dell'algoritmo di Watermarking per il processo di **Transfer Learning**.

Le prestazioni dei modelli e il **BER** finale dei Watermark nei modelli dimostrano che: l'algorimto riesce a sopravvivere in tutti i test con parametri differenti nelle operazioni di **RTAL** e **Transfer Learning**. Questa analisi ci conferma che Tattooed soddisfa il requisito di **Robustness**.

5.3 Security

Il requisito di *Security* (sicurezza) garantisce che: l'*autenticità* del legittimo proprietario di una rete neurale possa essere sempre verificato con elevata affidabilità, particolarmente in scenari in cui un *attore malevolo* tenti di eliminare o manipolare il Watermark per appropriarsi indebitamente del modello. Questo requisito rappresenta il fondamento su cui si basa la sicurezza della **proprietà intellettuale** di un modello di intelligenza artificiale. Quindi, è richiesto che il metodo di Watermarking riesca a sopportare gli attacchi dell'attore malevolo, in modo tale da poter risalire sempre al proprietario del modello. L'algoritmo di DNN Watermarking è considerato sicuro se: dopo un qualsiasi metodo di attacco, il **BER** (Bit error rate) del Watermark è uguale a **0.0**. Un qualsiasi valore diverso da **0** fa risultare il watermark eliminato e la proprietà intellettuale persa. Ma, in due casi, se il watermark viene perso, il metodo di watermarking è considerato sicuro se:
a) Il modello è stato reso *inutilizzabile* dall'attacco, di conseguenza la rete neurale non garantisce più un elevata accuratezza nel suo task originale. Consideriamo il *modello inutilizzabile* se, la *differenza* tra il nuovo valore di accuracy del modello supera di **5 punti percentuale** il valore di accuratezza del modello originale. **b)** Il costo della rimozione in termini di risorse è superiore al costo di produzione del modello attaccato. Gli esperimenti su questo requisito rappresentano il cuore di questa Tesi. Andremo a testare i vari tipi di attacco e a mostrare nuove tecniche difensive.

5.3.1 Pruning

Il **Parameter Pruning** è una tecnica utilizzata per rimuovere neuroni da un modello già addestrato. L'obiettivo è ridurre la complessità e la dimensione di una rete neurale mantenendo le performance intatte. L'operazione di *pruning* non elimina realmente i neuroni, ma li "neutralizza" annullando i pesi che li connettono con il layer precedente. L'operazione di pruning prende in input la percentuale di neuroni che vogliamo eliminare dalla rete, negli esperimenti vedremo il range di *pruning* che va da **[25%,99,99%]**. Gli esperimenti sono stati condotti su entrambe le architetture: **ResNet18** e **VGG11**. Inoltre, è stato anche introdotto un altro metodo di watermarking, **Chameleon DNN**, al fine di aprire un confronto che sarà filo conduttore dell'intero capitolo. E' importante notare che: nonostante i metodi

di Watermarking differenti, le architetture, il dataset utilizzato e i metodi di testing sono gli stessi. Questa uniformità garantisce che i confronti siano equi.

Pruning	TER (%)	BER (%)	TER (%)	BER (%)	TER (%)	BER (%)
25%	10.66	0.0	10.91	0.0	13.92	0.0
50%	11.64	0.0	11.88	0.0	15.93	0.0
75%	32.19	0.0	34.04	0.0	61.27	0.0
99%	90	0.0	90	0.0	90	0.0
99.95%	90	0.0	90	0.0	90	0.0
99.99%	90	0.4902	90	0.0	90	0.4394
Tabella 5.7	**Tabella 5.8**		**Tabella 5.9**		**Tabella 5.10**	

Le tabelle [5.8, 5.9 , 5.10], rappresentano gli esperimenti di pruning effettuati su **ResNet18**. Tutti i modelli utilizzano come base la rete neurale mostrata in questa tabella [5.1]. L'accuracy del modello con watermark è del **89.27%**. La tabella [5.8] rappresenta gli esperimenti sul modello con i parametri consigliati dai creatori di Tattooed, un gamma da 9×10^{-4} e un ratio che prende circa **200.000** elementi. Invece, la tabella [5.9] rappresenta il modello con parametri: γ 9×10^{-3} e un ratio che prende circa **4465** elementi. Infine, [5.10] contiene gli esperimenti del modello: gamma 9×10^{-4} e un ratio che prende circa **1.116.403** elementi.

Pruning	TER (%)	BER (%)	TER (%)	BER (%)	TER (%)	BER (%)
25%	9.85	0.0	9.82	0.0	9.81	0.0
50%	9.68	0.0	9.69	0.0	9.70	0.0
75%	9.82	0.0	9.71	0.0	9.76	0.0
99%	90	0.0	89.9	0.0	12.05	0.0
99.95%	90	0.0	90	0.248	90	0.4667
99.99%	90	0.3945	90	0.458	90	0.5136
Tabella 5.11	**Tabella 5.12**		**Tabella 5.13**		**Tabella 5.14**	

Le tabelle [5.12, 5.13 , 5.14], rappresentano gli esperimenti di pruning effettuati su **VGG11**. Tutti i modelli utilizzano come base la rete neurale mostrata in questa tabella [5.1]. L'accuracy del modello con watermark è del **90.16%**. Come in precedenza, la tabella [5.12] rappresenta gli esperimenti sul modello con i parametri consigliati dai creatori di Tattooed, un gamma da 9×10^{-4} e un ratio che prende circa **200.000** elementi. Invece, la tabella [5.13] rappresenta il modello con parametri: γ 9×10^{-4} e un ratio che prende circa **12.800** parametri. Infine, [5.14] contiene gli esperimenti del modello: gamma 3×10^{-4} e un ratio che prende circa **200.000** elementi.

Pruning	TER (%)	BER (%)	TER (%)	BER (%)
25%	9.85	0.0	9.82	0.0
50%	9.68	0.0	9.69	0.0
75%	9.82	0.0	9.71	0.0
99%	90	0.0	89.9	0.0
99.95%	90	0.0	90	0.248
99.99%	90	0.3945	90	0.458
Tabella 5.15	**Tabella 5.16**		**Tabella 5.17**	

Le tabelle [5.16, 5.17], rappresentano gli esperimenti di pruning effettuati sull'algoritmo **Chameleon DNN**, sull'architettura **ResNet18**. I valori di configurazione di questo algoritmo differiscono da quelli di Tattooed. I parametri da selezionare sono: lo **spreading code**, ovvero il numero di chips impiegati per cifrare i bit, che in questo caso è in comune a tutti i bit del Watermark; Il fattore **Gamma** γ, che determina la forza del Watermark; e infine, il **numero** e la **scelta** specifica dei **layer** da marchiare. E' importante notare che il valore di accuracy pre-pruning rappresenta precisione del modello **post-addestramento**, il quale contiene già il Watermark, che è stato inserito *prima dell'addestramento*. La tabella [5.16] contiene gli esperimenti sul modello che ha come parametri: $\gamma = 1.5$, spreading code **150** e **4** layer. Invece, nella tabella [5.17] troviamo: $\gamma = 1$, spreading code **100** e **2** layer.

Analizziamo i risultati degli esperimenti sostenuti. Come possiamo facilmente notare, il **parameter pruning** non è una tecnica di attacco efficace. Il primo set di tabelle [5.8, 5.9 , 5.10], mostra l'attacco su ResNet18. L'attacco rimuove il watermark nei due casi dove il *Pruning* arriva ben fino al 99.99%. Ovviamente, come è stato specificato, in questo caso la perdita non è da considerare una sconfitta per Tattooed, questo perché: insieme al Watermark, anche le prestazioni del modello sono state distrutte, in modo decisamente considerevole. Notiamo che, nel modello con forte γ e un ratio che prende in insieme 4465, il watermark non viene distrutto. Gli esperimenti su VGG11 rilevano risultati analoghi. I modelli hanno dimostrato una notevole resistenza al pruning, con le loro prestazioni in accuracy che hanno resistito meglio rispetto a ResNet18. Questo comportamento è da attribuire al fatto che VGG possiede una quantità 100 volte superiore di parametri rispetto a ResNet18. Anche in questo caso, la perdita del watermark nelle fasi estreme dell'esperimento *non è da considerasi un fallimento* per l'algoritmo. Infine, troviamo gli esperimenti sull'algoritmo di comparazione, nelle tabelle [5.16, 5.17]. Anche in questo caso, possiamo notare che l'attacco tramite parameter pruning è *completamente inefficace*. La lunghezza nettamente inferiore dello spreading code non ha compromesso il funzionamento dell'algoritmo. La perdita del watermark negli esperimenti più estremi del pruning, anche in questo caso, *non determinano una sconfitta* per l'algoritmo di Watermarking. In conclusione possiamo affermare con certezza che l'utilizzo, in questo caso, del **Parameter Pruning** come tecnica di attacco sia **completamente inefficace**. La perdita del watermark avviene solamente dopo la rimozione di un elevato numero di neuroni, quasi del 100%. Questo fattore invalida completamente la tecnica d'attacco, vista la perdita totale della precisione del modello. Continuiamo gli esperimenti con altri metodi di attacco.

5.3.2 Refit

Refit [8] rappresenta un framework state-of-the-art per la rimozione di Watermark dalle reti neurali. Questo strumento implementa diverse tecniche di rimozione, solitamente utilizzate per operazioni di creazione e modifica di una rete. Gli esperimenti su Refit sono da considerarsi le fondamentali: superarli è condizione necessaria, ma non sufficiente, per considerare un sistema di Watermarking sicuro. Di conseguenza, molti sistemi precedentemente considerati sicuri, in seguito all'applicazione di Refit sono stati invalidati, ridefinendo gli standard di sicurezza nel DNN Watermarking. Le tecniche implementate sono: **AU** (Unlabeled data augmentation), **EWC** (elastic weight consolidation) e l'utilizzo simultaneo delle due tecniche insieme. Per ciascuna architettura, gli esperimenti di rimozione nei sistemi di Watermarking testati, partono dallo stesso modello di base, già pre-addestrato e dotato di Watermark. In generale, questi esperimenti, utilizzeranno un numero di epoch minore di quello usato per addestrare i modelli, questo vincolo è necessario per non invalidare uno dei principi del *requisito di sicurezza* che recita: "il costo di rimozione non deve superare, in termini di risorse (*tempo*, denaro...), quello utilizzato per addestrare il modello attaccato". Gli esperimenti mostrati sono stati effettuati sulle architetture: VGG11 e ResNet18; con due algoritmi di watermarking a confronto.
Ogni esperimento riportato è stato effettuato sulla base dei modelli non marchiati visti nella tabella [5.1].

Nome	**Architettura**	**Gamma**	**Ratio**	**Tot Param**	**Tattooed TER (%)**	**Tattooed BER (%)**
A	ResNet-18	0.0009	0.01786	200.000	10.57	0.0
B	VGG11	0.0009	0.001563	200.000	9.81	0.0
Nome	**Architettura**	**Chame DNN**	**Gamma**	**Sp. Code**	**# Layers**	**TER (%)**
C	ResNet-18	Not Tattoed	1.5	150	4	12.520

Tabella 5.18. Modelli utilizzati negli esperimenti mostrati successivamente.

EWC

La tecnica dell'Elastic Weight Consolidation è normalmente utilizzata per evitare il **catastrophic forgetting** nel re-training delle reti neurali. Per applicare l'EWC, viene effettuata una fase di **finetuning**, in questo caso, di *20 epoch*. Viene chiesto in input il coefficiente λ, utilizzato nella nuova funzione, in tutti gli esperimenti impostato a **150**. Inoltre è richiesto il numero di elementi del dataset originale (cifar-10) su cui effettuare l'operazione, in questo caso: **10.000**. Questi parametri rappresentano i valori consigliati dai creatori di Refit.

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	1	20	20	9.838	0.73	0.0
0.9	1	50	20	7.789	2.78	0.0
0.9	1	80	20	7.779	2.79	0.0

Tabella 5.19. Esperimenti EWC sul modello **A**. Il **Learning Rate** di partenza è **0.05**

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	3	20	20	9.751	0.059	0.0
0.9	3	50	20	9.611	0.199	0.0
0.9	3	80	20	9.212	0.598	0.0

Tabella 5.20. Esperimenti EWC sul modello **B**. Il **Learning Rate** di partenza è **0.005**

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	1	20	20	15.05	-2.55	0.0
0.9	1	50	20	13.99	-1.47	0.0
0.9	1	80	20	14.01	-1,58	0.0

Tabella 5.21. Esperimenti EWC sul modello **C**. Il **Learning Rate** di partenza è **0.05**

L'analisi delle tabelle [5.19 , 5.20 , 5.21] ci mostra come tutti i tipi di architetture e i metodi di Watermarking sopravvivono al metodo di attacco dell' **EWC**. Nei casi **A** e **B**, si osserva un incremento di accuratezza da parte dei modelli. Questo miglioramento è dovuto alla natura originale dell'EWC, che focalizza l'addestramento sui nodi considerati "meno importanti" per il task della rete.

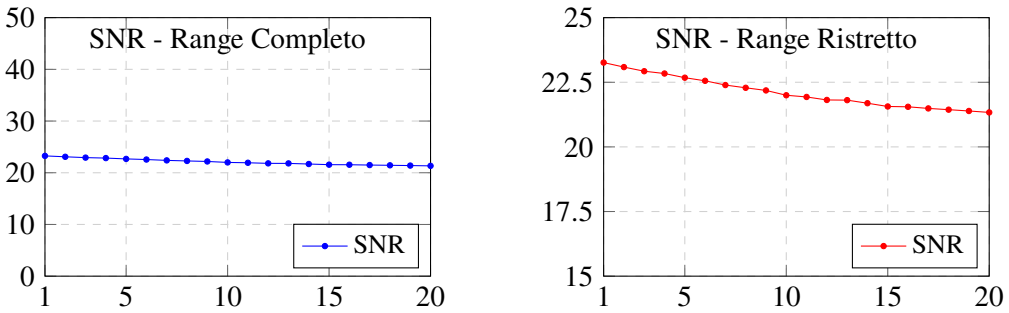


Figura 5.4. Grafico **SNR** sugli esperimenti sul Modello **A** su **EWC**, con il **50%** del Dataset

Com'è possibile notare da questi grafici, che rappresentano il **SNR** (Signal To Noise Ratio) del watermark nel modello,possiamo valutare il rapporto tra la potenza del segnale del Watermark e l'intensità del rumore. Un valore di SNR più basso indica un rapporto più alto tra rumore e segnale, suggerendo che il watermark è stato progressivamente sostituito da

rumore, ovvero valori casuali non correlati al watermark originale. Analizzando i grafici, è possibile osservare che il Signal-to-Noise-Ratio subisce una diminuzione minima, indicando un segnale di Watermark ancora forte nel modello. Possiamo concludere che l'**EWC** non rimuove il Watermark dal modello, ne in Tattooed ne in Chameleon.

AU

La tecnica dell'Unlabeled data agumentation **AU** è un metodo che utilizza dati non etichettati per modificare in significativamente le conoscenze di una rete neurale. Può essere considerata una strategia di attacco particolarmente efficace ed ottimale, poiché utilizza come risorsa dati facilmente reperibili anche dal web. In questo contesto specifico, viene impiegata un'istanza alternativa di ImageNet, contenente immagini di dimensione 32x32. Insieme a questo viene anche utilizzata una parte di Cifar-10 per l'operazione di fine tuning. L'operazione è sottoposta in 20 epoch.

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	2	20	20	13.120	-2.55	0.0
0.9	2	50	20	13.169	-2.59	0.0
0.9	2	80	20	11.778	-1.2	0.0

Tabella 5.22. Esperimenti AU sul modello **A**. Il **Learning Rate** di partenza è **0.05**

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	2	20	20	11.356	-1.54	0.0
0.9	2	50	20	10.472	-0.66	0.0
0.9	2	80	20	9.980	-0.17	0.0

Tabella 5.23. Esperimenti AU sul modello **B**. Il **Learning Rate** di partenza è **0.005**

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	2	20	20	18.97	-6.45	0.0410
0.9	2	50	20	16.54	-4.02	0.0254
0.9	2	80	20	17.05	-4.53	0.0371

Tabella 5.24. Esperimenti AU sul modello **C**. Il **Learning Rate** di partenza è **0.005**

L'analisi delle tabelle [5.22 , 5.23 , 5.24] evidenzia che Tattooed riesce a resistere al forte impatto dell'**AU**, perdendo di accuracy, vista la natura stessa dell'AU e il nuovo addestramento su ImageNet. Nonostante una perdita di accuracy, dovuta alla natura stessa dell'AU, il watermark persiste, soddisfacendo ancora una volta il requisito di *Security*. Chameleon, invece, risulta sconfitto. Non supera i test con il 50% e 80% del dataset, dove la differenza di accuratezza non oltrepassa la soglia dei 5 punti e il **BER** supera 0.0. Il test

sul 20% risulta superato a causa di una perdita di accuratezza del modello, che per i criteri imposti, risulta non più utilizzabile.

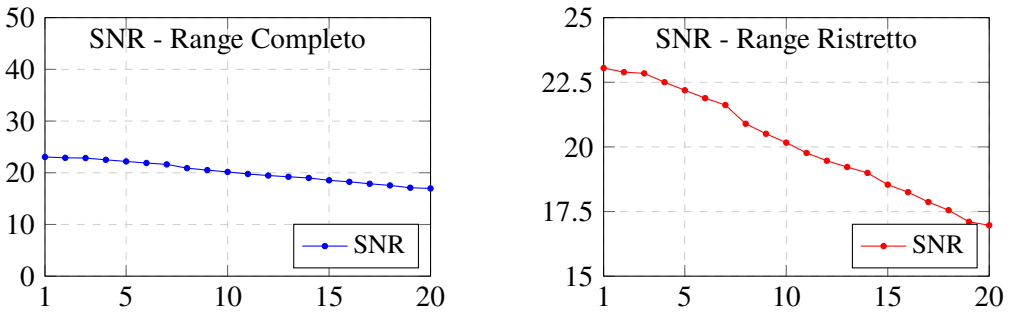


Figura 5.5. Grafico **SNR** sugli esperimenti sul Modello **A** su **AU**, con il **50%** del Dataset

Dall'analisi dei grafici, è possibile chiaramente osservare come il watermark resiste all'attacco. In particolare, si nota una diminuzione del **Signal-to-Noise-Ratio** più marcata rispetto agli esperimenti precedenti che utilizzavano *EWC*, evidenziando come questo tipo di attacco sia effettivamente più aggressivo. Il SNR ha una riduzione significativa, superando i 5 punti di decremento. Tuttavia, il watermark dimostra una forte resistenza, riuscendo a mantenere la sua integrità all'interno del modello.

EWC + AU

Sono stati eseguiti esperimenti separatamente su *EWC* e *AU*, analizziamo ora le prestazioni dei modelli e delle tecniche di Watermarking all'unione delle due tecniche. L'implementazione di questo metodo è realizzato applicando sequenzialmente entrambe le tecniche, consentendo di combinare i vantaggi di entrambe le strategie. I parametri di configurazione rimangono invariati rispetto a quelli utilizzati nell'applicazione individuale di ciascun metodo.

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	2	20	20	11.419	-0.84	0.0
0.9	2	50	20	11.150	-0.58	0.0
0.9	2	80	20	11.778	0.29	0.0

Tabella 5.25. Esperimenti **EWC + AU** sul modello **A**. Il **Learning Rate** di partenza è **0.05**

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	2	20	20	10.340	-0.53	0.0
0.9	2	50	20	10.121	-0.31	0.0
0.9	2	80	20	9.980	0.001	0.0

Tabella 5.26. Esperimenti **EWC + AU** sul modello **B**. Il **Learning Rate** di partenza è **0.05**

Ratio LR	Epoch LR Adj	Dataset (%)	Epoch	Refit TER	TER Diff	BER
0.9	2	20	20	15.88	-3.36	0.0
0.9	2	50	20	15.05	-2.53	0.0
0.9	2	80	20	15.83	-3.31	0.0

Tabella 5.27. Esperimenti **EWC + AU** sul modello **C**. Il **Learning Rate** di partenza è **0.05**

Come mostrato dalle tabelle [5.25 , 5.26 , 5.34], emerge che l'applicazione combinata delle due tecniche di attacco non produce un effetto "potenziato". Al contrario, si può osservare come un interferenza reciproca, dove l'efficacia di una tecnica mitiga la presenza dell'altra. In tutti e tre i casi il Watermark mantiene la sua integrità e persiste all'interno delle reti.

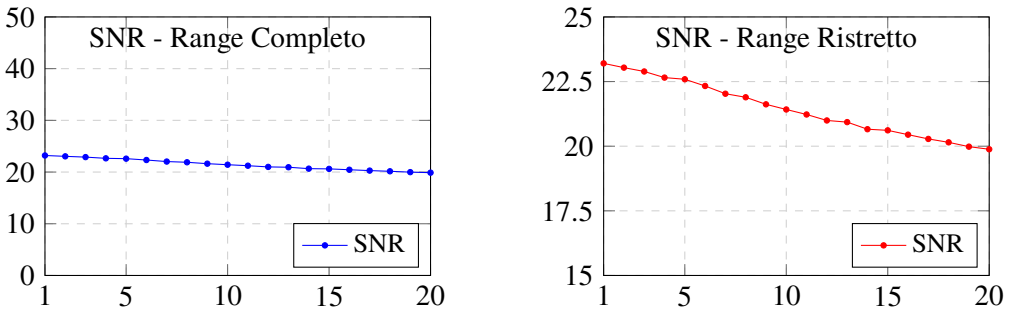


Figura 5.6. Grafico **SNR** sugli esperimenti sul Modello **A** su **EWC + AU**, con il **50%** del Dataset

Dall'analisi dei grafici si può chiaramente vedere l'effetto dell'unione tra **EWC** e **AU**. Questa combinazione produce un Signal-to-Noise-Ratio che si colloca in una posizione intermedia: inferiore rispetto all'applicazione singola di **EWC**, ma maggiore rispetto al solo utilizzo di **AU**. Quindi il grafico rispecchia perfettamente i risultati nelle tabelle [5.25 , 5.26 , 5.34].

Method	REFIT [8]			
	Basic	AU	EWC	EWC+AU
Uchida et al. [34]	✗	✗	✗	✗
Zhang et al. [39]	✗	✗	✗	✗
Adi et al. [1]	✗	✗	✗	✗
Namba et al. [26]	✗	✗	✗	✗
Wang et al. [37]	✗	✗	✗	✗
LSB	✗	✗	✗	✗
Chameleon [33]	✗	✗	✔	✔
TATTOOED [28]	✔	✔	✔	✔

In conclusione, è stato confermata la resistenza di **Tattooed** a tutti gli attacchi di *Refit*, contrariamente a **Chameleon**, che non supera il requisito di **Security**.

5.3.3 Quantizzazione

La **quantizzazione** di un modello di intelligenza artificiale trasforma la rappresentazione dei dati di una rete neurale, diminuendone la precisione. Questo metodo può essere sfruttato come forma di attacco contro modelli dotati di Watermark, questo poiché il cambio di rappresentazione compromette non solo dell'accuratezza del modello, ma anche dei dati stessi, quindi anche l'integrità del watermark. Gli algoritmi di Watermarking **Tattooed** e **Chameleon** si basano sulla precisione di dati per garantire la sopravvivenza del watermark, quindi la *quantizzazione* può diventare un problema. Questo approccio di attacco può essere considerato "innovativo", in quanto, algoritmi come Tattooed, non riportano ancora esperimenti o test che utilizzano questa strategia. Questa Tesi si propone di presentare i primi esperimenti condotti su questo argomento, esplorando l'efficacia della quantizzazione come metodo di attacco contro gli algoritmi di Watermarking.

Nome	**Architettura**	**Gamma**	**Ratio**	**Tot Param**	**Tattooed TER (%)**	**Tattooed BER (%)**
A	ResNet-18	0.0009	0.01786	200.000	10.57	0.0
B	VGG11	0.0009	0.001563	200.000	9.81	0.0
Nome	**Architettura**	**Chame DNN**	**Gamma**	**Sp. Code**	**# Layers**	**TER (%)**
C	ResNet-18	Not Tattoed	1.5	150	4	12.520

Tabella 5.28. Modelli utilizzati come base degli esperimenti mostrati successivamente.

Il metodo di quantizzazione utilizzato è la FX Graph Mode Quantization, che rappresenta uno dei metodi più recenti per la quantizzazione di modelli della libreria *Pytorch* [29]. Tutti gli esperimenti presentati utilizzano una rappresentazione a *quint8*, e utilizzano come base i modelli con Watermark presenti in questa tabella [9.11].

Nome	**Base TER(%)**	**Tattooed TER (%)**	**Quantiz. TER (%)**	**Tattooed Quant. TER (%)**
A	10.2	10.57	12.23	0.5546

Tabella 5.29. Esperimento di **quantizzazione** sul modello con Watermark **A**

Nome	**Base TER(%)**	**Tattooed TER (%)**	**Quantiz. TER (%)**	**Tattooed Quant. TER (%)**
B	9.83	9.81	13.22	0.4804

Tabella 5.30. Esperimento di **quantizzazione** sul modello con Watermark **B**

Nome	**Base TER(%)**	**Watermark TER (%)**	**Quantiz. TER (%)**	**Tattooed Quant. TER (%)**
C	/	12.52	13.45	perso!

Tabella 5.31. Esperimento di **quantizzazione** sul modello con Watermark **C**

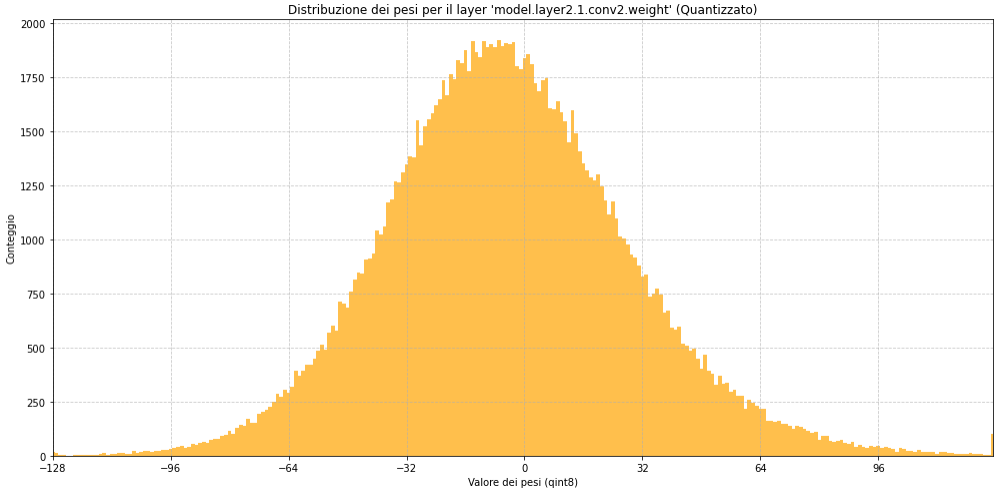


Figura 5.7. Distribuzione dei pesi del modello Quantizzato **A**

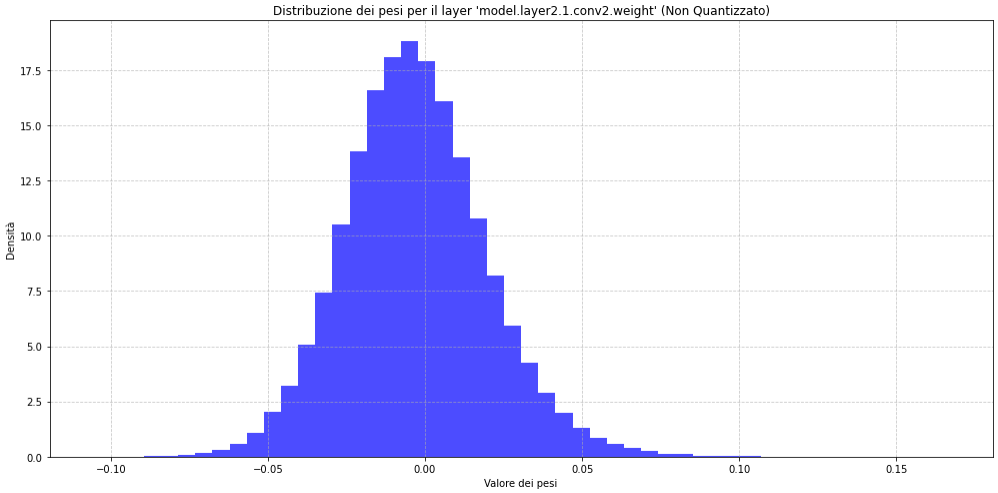


Figura 5.8. Distribuzione dei pesi del modello non Quantizzato **B**

La quantizzazione è visualizzabile nei due grafici [5.7 , 5.8], che rappresentano lo stesso layer dello modello **A**, **PRE** e **POST-Quantizzazione**. Si nota una significativa transformazione nella rappresentazione dei dati: il modello non quantizzato presenta un intervallo dei valori nei pesi di [-0.10,+0.10], mentre, il modello quantizzato, mostra il range di valori dato dal tipo **qint8**, che varia tra [-128,+128]. Questa analisi chiarisce la ragione della perdita del watermark, la rappresentazione e la struttura del modello è cambiata drasticamente.

La Quantizzazione emerge quindi come un *attacco efficace per la rimozione di un Watermark* da una rete neurale, tramite uno costo temporale e computazionale assolutamente minimo. Se le analisi si fermassero qui, potremmo considerare Tattooed battuto, incapace soddisfare il requisito di **Security**. Tuttavia, esploriamo una possibile soluzione, concentrandoci proprio sull'attacco stesso.
E' possibile utilizzare la quantizzazione come arma di difesa?

5.3.4 Quantizzazione come Difesa

Sono stati toccati i limiti dell'algoritmo di Watermarking **Tattooed**, è stato dimostrato che la quantizzazione rimuove il watermark dal modello con uno *sforzo minimo*. La quantizzazione è **la fine** degli algoritmi di **DNN Watermarking**? *Probabilmente no*. La quantizzazione rappresenta l'inizio di una nuova era di algoritmi, che utilizzano come difesa la quantizzazione stessa. Analizziamo Tattooed, proviamo ad utilizzare la quantizzazione come arma di difesa. L'idea è di quantizzare il modello già addestrato per poi provare ad aggiungere il Watermark. L'incognita è rappresentata da: il metodo di Watermarking basato su CDMA riuscirà a resistere e coesistere in un modello quantizzato?

Nome	Architettura	Gamma	Ratio	Tot Param	Tattooed TER (‰)	Tattooed BER (‰)
A	ResNet-18	0.0009	0.01786	200.000	10.57	0.0
B	VGG11	0.0009	0.001563	200.000	9.81	0.0

Tabella 5.32. Modelli utilizzati come base degli esperimenti mostrati successivamente.

Sono stati utilizzati come base questi modelli, addestrati e poi muniti di Watermark (anche se verrà rimosso dalla quantizzazione).

Nome	Base TER(‰)	Tattooed TER (‰)	Quantiz. TER (‰)	Tattooed Quant. TER (‰)	Post-Quant. Tattooed BER (‰)
A	9.83	9.81	13.22	0.4804	0.0

Tabella 5.33. Esperimento di **quantizzazione** sul modello con Watermark **B**

Nome	Base TER(‰)	Tattooed TER (‰)	Quantiz. TER (‰)	Tattooed Quant. TER (‰)	Post-Quant. Tattooed BER (‰)
B	9.83	9.81	13.22	0.4804	0.0

Tabella 5.34. Esperimento di **quantizzazione** sul modello con Watermark **B**

Esaminando le tabelle, emerge un dato significativo: il modello **post-quantizzazione** può essere inserito di Watermark. Questa informazione è un ottima notizia per l'algoritmo Tattooed, poiché l'implementazione di questa nuova tecnica difensiva, insieme al metodo del CDMA, torna a soddisfare il requisito di **Security**. La robustezza e sicurezza dell'algoritmo è stata comprovata sulle tecniche più avanzate di rimozione dell'watermark, l'aggiunta di uno stadio di quantizzazione migliorerà e rafforzerà ancora di più la sicurezza dell'algoritmo. Questo perché: la quantizzazione cambia la rappresentazione del dato da float32 a int8. Il dato originale, che ha un alta rappresentazione, è precisamente mappato su un valore specifico nel range del tipo Int8 [-128,127]. Perciò, la quantizzazione non può essere riproposta su un modello già quantizzato, in quanto i nuovi valori interi del modello verrebbero rimappati sugli stessi valori interi. **Rendendo la quantizzazione un operazione Idempotente**.

Conclusioni sul sottocapitolo dedicato al requisito di **Security**. Sono stati esaminati vari tipi di attacco, dai più naive ai più complessi, come Refit, e sono state le prestazioni dichiarate dai creatori dell'algoritmo Tattooed. In seguito è stata provata una nuova tecnica di attacco, la quantizzazione. La tecnica modifica drasticamente la struttura e le componenti stesse della rete, di conseguenza il Watermark viene perso. Insieme alla nuova tecnica di attacco è stata anche provata una nuova tecnica difensiva, che la quantizzazione stessa, che è stata aggiunta alla normale esecuzione dell'algoritmo di embedding, dimostrando come un modello quantizzato possa contenere il Watermark e, in quanto già quantizzato, resistere alla quantizzazione stessa.

5.4 Integrity

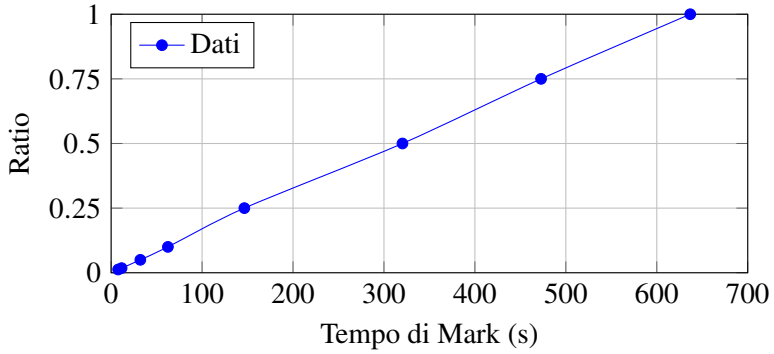
Il requisito di **Integrity** garantisce che: in assenza di manipolazioni sul modello con watermark, l'algoritmo di estrazione deve garantire il recupero in modo completo e intatto del watermark inserito. Questo requisito richiede che il watermark possa essere recuperato dalla rete dopo essere stato inserito. Dagli esperimenti eseguiti nella verifica dei requisiti di **Security** e **Robustess** è stato confermata l'efficacia dell'algoritmo nell'estrazione del Watermark dal modello, sia in condizioni standard che dopo tentativi di attacco.

5.5 Efficiency

Il requisito di **Efficiency** garantisce che: l'algoritmo deve essere in grado di verificare ed inserire il Watermark dal modello con un impatto computazionale minimo. Questo requisito garantisce che l'algoritmo riesca a lavorare in tempi e costi computazionali adeguati. Con la verifica degli altri requisiti è stata anche testata l'efficienza di inserimento e verifica del watermark dal modello.

Architettura	Gamma	Ratio	Param.	TTM (%)	TTE (%)
ResNet-18	0.0009	0.01786	200.000	7.695s	1.541s
ResNet-18	0.0009	0.05000	558.201	32.113s	3.225s
ResNet-18	0.0009	0.50000	5.582.016	320.331s	30.325s
ResNet-18	0.0009	1	11.164.032	636.782s	58.251s

Tabella 5.35. Tabella che mostra il **Time To Mark** e il **Time To Extract** del watermark



Capitolo 6

Lavori correlati

Questo capitolo discuterà in dettaglio di altri metodi di DNN Watermarking, di tipo White/Black-box.

6.1 Tecniche di Watermarking White-box

Il primo metodo di inserimento di un Watermark all'interno di un modello è stato introdotto da Uchida et al. [34, 25] nel 2017, questa ricerca è stata la prima ad introdurre il concetto di Watermark per le reti neurali. Il loro approccio utilizza un termine di regolazione che introduce alla funzione di perdita, utilizzata durante l'addestramento del modello, una distorsione statistica sui parametri della rete. La distorsione viene in seguito utilizzata per dedurre tramite inferenza le informazioni dal modello. In questo caso, la distorsione, può essere vista come una firma, che quindi è utilizzata per affermare la proprietà del modello. Successivamente il lavoro di Uchida et al. è stato esteso da Li et al. [23], che ha introdotto un termine di regolarizzazione aggiuntivo codificato. Questo processo permette di aggiungere watermark molto più grandi all'interno della rete, con un impatto sulla rete sorprendentemente basso. Tuttavia, questo metodo si è dimostrato poco robusto e molto sensibile ad attacchi di Transfer Learning. Nel 2020, Wang et al. [36], hanno proposto un nuovo metodo di inserimento di watermark in una rete neurale. Questa tecnica utilizza una rete neurale indipendente che permette l'uso di alcuni pesi che vengono scelti in modo selettivo per inserire il Watermark nel modello target. Il "modello di inserimento" viene utilizzato come chiave e viene tenuto nascosto e serve effettivamente come strumento di verifica della Proprietà intellettuale. Il Watermark è inserito in fase di addestramento, dove le due reti vengono addestrate in simultanea. La "rete chiave" contiene i pesi che, nella rete target, contengono l'informazione. Per estrarre il Watermark, gli input layer della "rete chiave" vengono collegati ai pesi contenenti l'informazione nella rete effettiva. La presenza del watermark viene poi dedotta tramite inferenza osservando l'output della rete segreta. Nel 2022, alcuni mesi dopo la pubblicazione della prima versione di Tattooed, Tondi et al. hanno presentato Chameleon DNN [33]. Questa tecnica utilizza il metodo dello spread spectrum che amplifica un segnale in una banda più alta. Per cifrare il dato all'interno di questo nuovo segnale amplificato utilizza una serie di chips, chiamati spreading code. La differenza tra Chameleon e Tattooed è anche nello spreading code, che è condiviso tra tutti i bit che compongono il Watermark da inserire nel modello, e soprattutto hanno una dimensione assolutamente minore. Contrariamente, Tattooed utilizza uno spreading code di

dimensione uguale al numero di parametri scelti per ospitare il Watermark. La scelta dei layer dove inserire il Watermark è lasciata all'utente, ma la scelta dei parametri di questi layer è effettuata in modo intelligente e pseudocasuale. Infatti, il watermark viene aggiunto pre-addestramento. Dopo aver inserito nei parametri il Watermark, questi vengono congelati, in modo tale da non venir eliminati in fase di addestramento. Quindi la scelta dei parametri target deve essere effettuata in modo che: un peso con watermark sia indistinguibile da uno senza. DeepSigns [10], introdotto da Rouhani et al. nel 2019, inserisce il watermark nella funzione di densità di probabilità ottenuta da un layer della rete. La verifica avviene tramite un sottoinsieme dei dati di addestramento che vengono utilizzati per interrogare il modello, per poi calcolare il valore medio delle funzioni di attivazioni ottenute. Questi valori vengono successivamente utilizzati per estrarre il Watermark effettivo dai parametri del modello. In seguito, è stato pubblicato DeepMarks, che rappresenta un evoluzione di DeepSigns. In questo nuovo metodo, il watermark è "costruito" utilizzando tecniche anti-collisione. Il metodo di inserimento e di verifica sono gli stessi di DeepMarks. Questo metodo è molto efficace, l'utilizzo di un sistema di anti-collusione permette di avere una percentuale di BER durante il recupero molto inferiore al suo predecessore, DeepSigns. Un altra proposta all'avanguardia proposta da Wang et al. [36], utilizza una tecnica di addestramento "avversaria" tramite una rete di tipo GAN, Generative adversarial network, che è stata introdotta per la prima volta da Goodfellow et al [13], il creatore delle CNN. Queste tipo di reti vengono utilizzate per generare nuovi dati da zero, utilizzando le informazioni apprese dai dati utilizzati in fase di addestramento. In questo contesto, la GAN è utilizzata per rendere la distribuzione dei pesi del modello che conterrà il Watermark simili alla rete che non contiene il Watermark. In questo modo il Watermark è reso impercettibile all'interno della rete target. Il modello riduce al minimo l'accuratezza persa dal modello per l'inserimento del watermark, ed è progettato per resistere ai vari attacchi di rimozione. Questo metodo però utilizza un metodo di inserimento complesso, che comporta l'addestramento di due tipi di reti in modo simultaneo, aumentando cosi la complessità e il costo del processo di addestramento.

6.2 Tecniche di Watermarking Black-box

Le tecniche di Black-box Watermarking sono pensate per verificare la presenza di un watermark all'interno di una rete neurale non avendo necessariamente l'accesso fisico, quindi diretto, ai parametri della rete neurale. Le modalità di recupero si basano quindi sull'interrogazione del modello e dell'analisi delle previsioni date in output. Un esempio ci è dato da Chen et al. [7] nel 2019. che codifica il watermark utilizzano un insieme di coppie di immagini, che vengono etichettate in modo mirato, e ognuna di queste immagini ritorna un bit specifico che può essere utilizzato per ricomporre il watermark. Il metodo del Black-box non si limita all'inserimento legittimo di un watermark, ma viene sfruttato anche da attori malevoli per inserire una **backdoor** all'interno del modello. In realtà, il watermark legittimo può essere considerato come una forma di backdoor "benigna". Tuttavia, l'inserimento di una vera backdoor da parte di un attore malevolo non mira principalmente a proteggere (rubare) la proprietà intellettuale di un modello, bensì ad *avvelenare il modello* con dati manipolati ad hoc. Un esempio molto importante ci viene dato da **Badnet** [14], paper che discute del processo di "avvelenamento" di un modello. Nel loro studio, illustrano come un dataset impiegato per l'addestramento su un sistema di guida autonoma nel riconoscere la

segnaletica stradale possa essere avvelenato attraverso piccole alterazioni di alcuni dati. Ad esempio, il modello viene introdotto a interpretare i segnali di stop con un particolare adesivo attaccato sotto la scritta "STOP", come un comando per l'automobile a **guida autonoma**, accelera fino a 130km/h, sovvertendo il suo scopo originale e creando un pericolo reale.

Figura 6.1. L'immagine mostra le modifiche di alcuni elementi del dataset. [14]

Figura 6.2. L'immagine mostra la backdoor in azione in uno scenario reale. L'adesivo sotto lo stop fa credere al modello, con una sicurezza del 94,7%, che rappresenti un limite di velocità [14]

Un altro esempio può essere osservato sul dataset MNIST, dove il modello interpreta differentemente i numeri quando nell'angolo inferiore sinistro dell'immagine appare il simbolo "F" (6.3).

Figura 6.3. Sopra le figure possiamo notare la previsione del modello sull'immagine. Nella prima immagine, il simbolo "2" viene predetto in modo corretto. Nella seconda immagine, avvelenata, il modello è stato indotto a credere che il "2" sia un "6".

Capitolo 7

Lesson Learned

In questo capitolo, sono riassunti gli insegnamenti chiave emersi durante le analisi condotte sugli esperimenti nella tesi:

- Sono stati *definiti i Watermark per Reti neurali e spiegato il loro scopo e funzionamento*. Il lavoro si è concentrato su Tattooed, un moderno algoritmo all'avanguardia per inserire un Watermark in un modello di intelligenza artificiale.
- Sono stati effettuati *numerosi esperimenti* su Tattooed e altri algoritmi di Watermarking per reti neurali. I risultati ottenuti sono chiari e in linea con le dichiarazioni nell'elaborato [28]. E' stato dimostrato come Tattooed riesca a soddisfare tutti i requisiti fondamentali di un algoritmo di DNN Watermarking sicuro.
- **Fidelity**,**Robustness**, **Security**, **Integrity** e **Efficiency**. Questi sono i requisiti fondamentali che Tattooed si è dimostrato di soddisfare con assoluta certezza. L'algoritmo di confronto, d'altra parte, non ha resistito ai nostri esperimenti, guadagnandosi il nuovo status di *algoritmo non sicuro*.
- Abbiamo esplorato un nuovo e innovativo tipo di attacco: la ***Quantizzazione***.
- E' stato dimostrato come questo attacco riesca a *superare anche Tattooed*, eliminando il Watermark dal modello e rendendo l'algoritmo temporaneamente non sicuro.
- In seguito, la tecnica della *quantizzazione* è stata utilizzata anche come **difesa**. E' stato dimostrato che: è possibile inserire il Watermark nel modello dopo la quantizzazione, rendendo **nuovamente Tattooed sicuro**.
- Dati i risultati degli esperimenti sulla quantizzazione, sono stati *ridefiniti* gli standard di sicurezza di tutti gli algoritmi di Watermarking futuri.

L'algoritmo **Tattooed** si è dimostrato sicuro e all'avanguardia, ma la sua sicurezza può essere ancora perfezionata. Il futuro dell'algoritmo dovrà ruotare intorno alla quantizzazione. Un altro aspetto fondamentale riguarda la scelta dei parametri e dei layer di embedding del Watermark, fase potenzialmente *vulnerabile*. In futuro, l'algoritmo dovrà essere in grado di scegliere in modo intelligente i giusti parametri e layer per adattarsi al meglio al modello quantizzato. Un altra possibile strada per il futuro potrebbe arrivare da un approccio che unisca il tipo **White** e **Black-box insieme**, in modo tale da creare l'algoritmo di DNN Watermarking definitivo.

Capitolo 8

Conclusioni

In conclusione al lavoro svolto, l'obiettivo primario di questa Tesi è stato valutare l'efficacia di Tattooed, confrontandolo con altre tecniche state-of-the-art nel campo del DNN Watermarking. L'analisi si è concentrata sui requisiti fondamentali che la letteratura scientifica ritiene necessari per un algoritmo di DNN Watermarking sicuro: **Fidelity**,**Robustness**, **Security**, **Integrity** e **Efficiency**. I vari esperimenti sono stati effettuati su due diverse architetture, in modo tale da avere una quantità di dati sufficienti per trarre delle conclusioni. Sono stati utilizzati vari metodi all'avanguardia per mettere alla prova questi algoritmi. Il più importante ed efficace di tutti è Refit, un framework per la rimozione di Watermark da modelli di intelligenza artificiale. Tutti i test sono stati effettuati nello stesso ambiente e nelle stesse condizioni, in modo tale da garantire test equi e affidabili. Il lavoro svolto ha confermato i risultati dichiarati dai creatori di Tattooed, confermandolo come uno degli algoritmi più all'avanguardia disponibili. Ma ci siamo spinti oltre. Il cuore di questa Tesi, come evidenziato dal titolo stesso, è l'esplorazione dei limiti di questi algoritmi con l'obiettivo di superarli. E' stata testata una nuova tecnica di attacco, la *quantizzazione*. Questo metodo, utilizzato solitamente per modificare la rappresentazione del dato e della struttura di una rete neurale per renderla veloce e leggera, è stato applicato nel tentativo di eliminare il marchio dal modello. Questo esperimento ha avuto successo. Trovato il limite è stato cercata una strategia per oltrepassarlo. L'idea è stata quella di utilizzare la *quantizzazione* come difesa contro la stessa. Questa strategia di difesa si è dimostrata un completo successo, il modello quantizzato riesce a contenere il Watermark senza compromessi, superando cosi questo ostacolo.

L'intelligenza artificiale sta cambiando l'informatica e la società. Le sue potenzialità sembrano illimitate. Questo la rende una risorsa preziosa e da proteggere. Il Watermarking per reti neurali è una nuova scienza emergente, in continua evoluzione. Non sembrano ancora vedersi dei limiti, vista la continua evoluzione delle reti neurali, e nel prossimo futuro vedremo molte nuove tecniche di attacco e difesa. Solo quando verrà raggiunto un punto di stabilità, potremo valutare concretamente il futuro di questa nuova scienza.

Capitolo 9

Appendice

In questo capitolo sono proposti altri esperimenti svolti su Tattooed e Chameleon.

9.1 ResNet18 Tattooed

In questa sezione sono presenti alcuni esperimenti effettuati sull'architettura **ResNet18** con l'algoritmo **Tattooed**. Sono state condotte diverse sperimentazioni, variando i parametri per analizzare e comprendere l'impatto dei valori per un Watermark sicuro.

Nome	Gamma	Ratio	W * Ratio	% Weight	Base TER	Tattooed TER	TER DIFF	BER
1	0,0009	0,01250	139550	1,25%	10,2000%	10,830%	-0,63	0,0
2	0,0009	0,01786	200000	1,79%	10,2000%	10,570%	-0,37	0,0
3	0,00004	0,01786	200000	1,79%	10,2000%	10,189%	0,011	0,0
4	0,0003	0,01786	200000	1,79%	10,2000%	10,330%	-0,13	0,0
5	0,0009	0,0050	55820	0,50%	10,2000%	10,320%	-0,12	0,0
6	0,0009	0,0005	5582	0,05%	10,2000%	10,320%	-0,12	0,0
7	0,009	0,01786	200000	1,79%	10,2000%	90,00%	-79,80	0,0
8	0,009	0,001	11164	0,10%	10,2000%	14,37%	-4,17	0,0
9	0,009	0,0004	4465	0,04%	10,2000%	10,89%	-0,69	0,0

Tabella 9.1. Modelli utilizzati come base degli esperimenti mostrati successivamente.

Come è possibile notare dalla tabella, sono stati effettuati esperimenti su un vasto raggio di valori di Gamma e Ratio. Molti di questi modelli sono stati aggiunti di un Watermark molto debole per verificare i limiti dell'algoritmo Tattooed, tra questi troviamo: **3**, che ha un *gamma* estremamente basso; infine, **6**, che è limitato dal **ratio**.

I modelli verranno testati su *Refit*, con le impostazioni utilizzate negli esperimenti principali del *capitolo 5*. Per l'**EWC** è utilizzato un *coefficiente* di valore 150 e 10.000 samples. Per gli esperimenti su **AU** viene utilizzata una percentuale di *extra data* del'50%.

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
1	EWC	0,9	1	20%	20	10,629%	0,2	0,0
1	EWC	0,9	1	50%	20	10,280%	0,55	0,0
1	EWC	0,9	1	80%	20	9,470%	1,36	0,0
1	AU	0,9	2	20%	20	13,419%	-2,58	0,0
1	AU	0,9	2	50%	20	11,330%	-0,5	0,0
1	AU	0,9	2	80%	20	10,379%	0,46	0,0
1	EWC+AU	/	/	20%	20	15,469%	-4,630	0,0
1	EWC+AU	/	/	50%	20	11,330%	-3,330	0,0
1	EWC+AU	/	/	80%	20	10,379%	-3,54	0,0

Tabella 9.2. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
2	EWC	0,9	1	20%	20	9,838%	0,73	0,0
2	EWC	0,9	1	50%	20	7,789%	2,78	0,0
2	EWC	0,9	1	80%	20	7,779%	2,79	0,0
2	AU	0,9	2	20%	20	13,120%	-2,55	0,0
2	AU	0,9	2	50%	20	13,169%	-2,59	0,0
2	AU	0,9	2	80%	20	11,779%	-1,2	0,0
2	EWC+AU	0,9	2	20%	20	11,419%	-0,849	0,0
2	EWC+AU	0,9	2	50%	20	11,150%	-0,58	0,0
2	EWC+AU	0,9	2	80%	20	10,280%	0,29	0,0

Tabella 9.3. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
3	EWC	0,9	1	20%	20	7,590%	2,98	0,085526
3	EWC	0,9	1	50%	20	8,149%	2,49	0,078947
3	EWC	0,9	1	80%	20	7,529%	3,04	0,098684
3	AU	0,9	2	20%	20	15,030%	-4,46	0,348684
3	AU	0,9	2	50%	20	11,129%	-0,559	0,401315
3	AU	0,9	2	80%	20	11,419%	-0,849	0,322368
3	EWC+AU	0,9	2	20%	20	11,820%	-1,25	0,098684
3	EWC+AU	0,9	2	50%	20	10,900%	-0,33	0,092105
3	EWC+AU	0,9	2	80%	20	11,619%	-1,04	0,105263

Tabella 9.4. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
4	EWC	0,9	1	20%	20	8,139%	2,43	0.0
4	EWC	0,9	1	50%	20	7,580%	2,99	0.0
4	EWC	0,9	1	80%	20	8,149%	2,42	0.0
4	AU	0,9	2	20%	20	12,180%	-1,61	0.0
4	AU	0,9	2	50%	20	12,529%	-1,95	0.0
4	AU	0,9	2	80%	20	11,460%	-0,89	0.0
4	EWC+AU	0,9	2	20%	20	11,070%	-0,5	0.0
4	EWC+AU	0,9	2	50%	20	11,409%	-0,83	0.0
4	EWC+AU	0,9	2	80%	20	10,979%	-0,4	0.0

Tabella 9.5. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
5	EWC	0,9	1	20%	20	7,660%	2,66	0,0
5	EWC	0,9	1	50%	20	7,730%	2,59	0,0
5	EWC	0,9	1	80%	20	7,630%	2,69	0,0
5	AU	0,9	2	20%	20	13,810%	-3,49	0,0
5	AU	0,9	2	50%	20	11,280%	-0,96	0,0
5	AU	0,9	2	80%	20	11,220%	-0,9	0,0
5	EWC+AU	0,9	2	20%	20	10,939%	-0,619	0,0
5	EWC+AU	0,9	2	50%	20	12,269%	-1,949	0,0
5	EWC+AU	0,9	2	80%	20	11,360%	-1,04	0,0

Tabella 9.6. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
6	EWC	0,9	1	20%	20	8,139%	2,18	0,0
6	EWC	0,9	1	50%	20	8,079%	2,24	0,0
6	EWC	0,9	1	80%	20	8,499%	1,82	0,0
6	AU	0,9	2	20%	20	12,940%	-2,62	0,128906
6	AU	0,9	2	50%	20	10,920%	-0,6	0,125
6	AU	0,9	2	80%	20	11,739%	-1,41	0,134765
6	EWC+AU	0,9	2	20%	20	11,489%	-1,16	0,0
6	EWC+AU	0,9	2	50%	20	10,989%	-0,66	0,0
6	EWC+AU	0,9	2	80%	20	12,650%	-2,33	0,0

Tabella 9.7. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
7	EWC	0,9	1	20%	20	100%	1	perso
7	EWC	0,9	1	50%	20	100%	1	perso
7	EWC	0,9	1	80%	20	100%	1	perso
7	AU	0,9	2	20%	20	15,230%	74,77	0.0
7	AU	0,9	2	50%	20	13,789%	76,21	0.0
7	AU	0,9	2	80%	20	12,410%	77,59	0.0
7	EWC+AU	0,9	2	20%	20	100%	1	perso
7	EWC+AU	0,9	2	50%	20	100%	1	perso
7	EWC+AU	0,9	2	80%	20	100%	1	perso

Tabella 9.8. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
8	EWC	0,9	1	20%	20	8,209%	6,16	0,0
8	EWC	0,9	1	50%	20	8,170%	6,2	0,0
8	EWC	0,9	1	80%	20	7,620%	6,75	0,0
8	AU	0,9	2	20%	20	14,029%	0,341	0,0
8	AU	0,9	2	50%	20	11,540%	2,83	0,0
8	AU	0,9	2	80%	20	10,140%	4,23	0,0
8	EWC+AU	0,9	2	20%	20	13,050%	1,32	0,0
8	EWC+AU	0,9	2	50%	20	12,110%	2,26	0,0
8	EWC+AU	0,9	2	80%	20	13,03%	1,35	0,0

Tabella 9.9. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
9	EWC	0,9	1	20%	20	8,509%	2,38	0,0
9	EWC	0,9	1	50%	20	8,230%	2,66	0,0
9	EWC	0,9	1	80%	20	8,240%	2,65	0,0
9	AU	0,9	2	20%	20	14,590%	-3,7	0,0
9	AU	0,9	2	50%	20	10,970%	-0,08	0,0
9	AU	0,9	2	80%	20	10,809%	0.08	0,0
9	EWC+AU	0,9	2	20%	20	11,570%	-0,68	0,0
9	EWC+AU	0,9	2	50%	20	12,070%	-1,18	0,0
9	EWC+AU	0,9	2	80%	20	13,220%	-2,33	0,0

Tabella 9.10. Test su refit, il LR di partenza è 0,05

9.2 ResNet18 Chameleon

In questa sezione sono presenti alcuni esperimenti effettuati sull'architettura **ResNet18** con l'algoritmo **Chameleon**. Sono state condotte diverse sperimentazioni, variando i parametri per analizzare e comprendere l'impatto dei valori per un Watermark sicuro.

Nome	Epoch	Spreading	Gamma	Layers	TER	BER
A	100	100	1	2	12,200%	0.0
B	100	100	1	4	11,207%	0.0
C	100	50	1	8	12,230%	0.0

Tabella 9.11. Modelli utilizzati come base degli esperimenti mostrati successivamente.

Per la rete **A**, sono stati scelti 2 Layers.
- **model.layer2.0.conv2.weight**, dove il watermark occupa una superficie del 17,36%
- **model.layer2.1.conv1.weight**, dove il watermark occupa una superficie del 17,36%

Per la rete **B**, sono stati scelti 4 Layers.
- **model.layer1.0.conv1.weight**, dove il watermark occupa una superficie del 34,72%
- **model.layer1.1.conv2.weight**, dove il watermark occupa una superficie del 34,72%
- **model.layer2.1.conv1.weight**, dove il watermark occupa una superficie del 8,68%
- **model.layer4.0.downsample.0.weight**, dove il watermark occupa una superficie del 9,77%

Per la rete **C**, sono stati scelti 8 Layers.
- **model.layer1.0.conv2.weight**, dove il watermark occupa una superficie del 8,68%
- **model.layer1.1.conv2.weight**, dove il watermark occupa una superficie del 8,68%
- **model.layer2.0.conv2.weight**, dove il watermark occupa una superficie del 2,17%
- **model.layer3.0.conv1.weight**, dove il watermark occupa una superficie del 1,09%
- **model.layer3.1.conv1.weight**, dove il watermark occupa una superficie del 0,54%
- **model.layer3.1.conv2.weight**, dove il watermark occupa una superficie del 0,54%
- **model.layer4.1.conv2.weight**, dove il watermark occupa una superficie del 0,14%
- **model.layer2.1.conv1.weight**, dove il watermark occupa una superficie del 62,50%

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
A	EWC	0,9	1	20%	20	15,33%	-3,13	0,0
A	EWC	0,9	1	50%	20	14,16%	-1,96	0,0
A	EWC	0,9	1	80%	20	14,08%	-1,88	0,0
A	AU	0,9	2	20%	20	18,98%	-6,78	0,1836
A	AU	0,9	2	50%	20	16,28%	-4,08	0,1328
A	AU	0,9	2	80%	20	16,75%	-4,55	0,1582
A	EWC+AU	0,9	2	20%	20	14,25%	-2,05	0,0
A	EWC+AU	0,9	2	50%	20	14,24%	-2,04	0,0
A	EWC+AU	0,9	2	80%	20	14,76%	-2,56	0,0

Tabella 9.12. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
B	EWC	0,9	1	20%	20	14,70%	-3,49	0,0
B	EWC	0,9	1	50%	20	13,91%	-2,7	0,0
B	EWC	0,9	1	80%	20	13,80%	-2,59	0,0
B	AU	0,9	2	20%	20	20,68%	-9,47	0,1562
B	AU	0,9	2	50%	20	16,09%	-4,88	0,1465
B	AU	0,9	2	80%	20	15,49%	-4,28	0,1406
B	EWC+AU	0,9	2	20%	20	14,99%	-3,78	0,0
B	EWC+AU	0,9	2	50%	20	14,56%	-3,35	0,0
B	EWC+AU	0,9	2	80%	20	14,22%	-3,01	0,0

Tabella 9.13. Test su refit, il LR di partenza è 0,05

Nome	Method	Ratio	Epoch LR.Adj	% DS	Epoch	Refit TER	TER Diff	BER
C	EWC	0,9	1	20%	20	14,04%	-1,80	0,0078
C	EWC	0,9	1	50%	20	14,56%	-2,33	0,0039
C	EWC	0,9	1	80%	20	14,16%	-1,93	0,0
C	AU	0,9	2	20%	20	19,96%	-7,72	0,1777
C	AU	0,9	2	50%	20	18,40%	-6,17	0,1758
C	AU	0,9	2	80%	20	17,14%	-4,91	0,1699
C	EWC+AU	0,9	2	20%	20	16,13%	-3,90	0,0039
C	EWC+AU	0,9	2	50%	20	15,69%	-3,45	0,0039
C	EWC+AU	0,9	2	80%	20	14,66%	-2,43	0,0020

Tabella 9.14. Test su refit, il LR di partenza è 0,05

Bibliografia

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. *usenix*, 08 2018.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[3] M. Barni and F. Bartolini. *Watermarking Systems Engineering: Enabling Digital Assets Security and Other Applications*. Crc Press, 2004.

[4] Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. Invertible residual networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 573–582. PMLR, 09–15 Jun 2019.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[6] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. *In Proceedings of the 2019 on International Conference on Multimedia Retrieval (2019)*, 06 2019.

[7] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. Blackmarks: Blackbox multibit watermarking for deep neural networks, 2019.

[8] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Bo Li Ruoxi Jia, and Dawn Song. Refit: A unified watermark removal framework for deep learning systems with limited data. *arXiv*, 11 2019.

[9] Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, and David Owen. The rising costs of training frontier ai models. *arXiv*, 05 2024.

[10] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 485–497, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[14] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv*, 03 2019.

[15] F. Hartung and M. Kutter. Multimedia watermarking techniques. *Proceedings of the IEEE*, 08 2002.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[18] Brendan Hogan, Anmol Kabra, Felipe Siqueira Pacheco, Laura Greenstreet, Joshua Fan, Aaron Ferber, Marta Ummus, Alecsander Brito, Olivia Graham, Lillian Aoki, Drew Harvell, Alex Flecker, and Carla Gomes. Aiscivision: A framework for specializing large multimodal models in scientific image classification, 2024.

[19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019.

[20] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017.

[21] Shinjini Kundu, Haris Sair, Elliott H. Sherr, Pratik Mukherjee, and Gustavo K. Rohde. Discovering the gene-brain-behavior link in autism via generative machine learning. *Science Advances*, 06 2024.

[22] Chao Li, Aojun Zhou, and Anbang Yao. Noah: Learning pairwise object category attentions for image classification, 2024.

[23] Yue Li, Benedetta Tondi, and Mauro Barni. Spread-transform dither modulation watermarking of deep neural network. *CoRR*, abs/2012.14171, 2020.

[24] Gaurav Mittal, Tanya Marwah, and Vineeth N. Balasubramanian. Sync-draw: Automatic video generation using deep recurrent attentive architectures. In *Proceedings of the 25th ACM international conference on Multimedia*, MM '17, page 1096–1104. ACM, October 2017.

[25] Yuki Nagai, Yusuke Uchida, Shigeyuki Sakazawa, and Shin'ichi Satoh. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(1):3–16, February 2018.

[26] Ryota Namba and Jun Sakuma. Robust watermarking of neural network with exponential weighting, 2019.

[27] Giulio Pagnotta, Dorjan Hitaj, Fabio De Gaspari, and Luigi V. Mancini. Passflow: Guessing passwords with generative flows, 2021.

[28] Giulio Pagnotta, Dorjan Hitaj, Briland Hitaj, Fernando Perez-Cruz, and Luigi V. Mancini. Tattooed: A robust deep neural network watermarking scheme based on spread-spectrum channel coding. *Annual Computer Security Applications Conference*, 2024.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[30] Sadia Ramzan, Muhammad Munwar Iqbal, and Tehmina Kalsum. Text-to-image generation using deep learning. *Engineering Proceedings*, 20(1), 2022.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[33] Benedetta Tondi, Andrea Costanzo, and Mauro Barni. Robust and large-payload dnn watermarking via fixed, distribution-optimized, weights, 2022.

[34] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. *CoRR*, abs/1701.04082, 2017.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[36] Jiangfeng Wang, Hanzhou Wu, Xinpeng Zhang, and Yuwei Yao. Watermarking in deep neural networks via error back-propagation. In *Proceedings of IS&T International Symposium on Electronic Imaging: Media Watermarking, Security, and Forensics*, pages 22–1–22–9, 2020.

[37] Tianhao Wang and Florian Kerschbaum. Riga: Covert and robust white-box watermarking of deep neural networks, 2021.

[38] Adam Yala, Peter G. Mikhael, Fredrik Strand, Gigin Lin, Kevin Smith, Yung-Liang Wan, Leslie Lamb, Kevin Hughes, Constance Lehman, and Regina Barzilay. Toward robust mammography-based models for breast cancer risk. *Science Translational Medicine*, 01 2021.

[39] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, page 159–172, New York, NY, USA, 2018. Association for Computing Machinery.