



UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI  
INGEGNERIA INFORMATICA,  
MODELLISTICA, ELETTRONICA  
E SISTEMISTICA

DIMES

Corso di Laurea in  
Ingegneria Informatica

Tesi di Laurea

Utilizzo del framework Tensorflow per  
l'identificazione di watermark in  
immagini tramite reti neurali  
convoluzionali

**Relatori**

Dr. Riccardo Cantini

**Candidato**

Pata Alessandro  
Matr. 204004

Anno Accademico 2023/2024

*Ai miei genitori,  
a chi mi ha sempre sostenuto*



# Indice

<b>1.</b>	<b>DEEP LEARNING E RETI NEURALI</b>	<b>9</b>
1.1.	CENNI STORICI	9
1.1.1.	Origini	9
1.1.2.	Prime reti neurali	11
1.1.3.	Progressi tecnologici e successi applicativi	12
1.2.	MACHINE LEARNING E DEEP LEARNING	12
1.2.1.	Machine learning	14
1.2.2.	Tecniche usate nel Deep Learning	15
1.2.3.	Deep learning vs Machine Learning	16
1.2.4.	Definizione e caratteristiche del Deep Learning	18
1.2.5.	Quando utilizzare il Deep Learning?	18
1.3.	RETI NEURALI	19
1.3.1.	Neurone biologico	20
1.3.2.	Neurone artificiale	21
1.3.3.	Funzioni di attivazione	22
1.3.4.	Tipologie di reti neurali	25
1.3.5.	Reti neurali convoluzionali	27
1.3.6.	Classificazione dei layers	<b>Errore. Il segnalibro non è definito.</b>
1.3.6.1.	Layer convoluzionale	28
1.3.6.2.	Layer di pooling	30
1.3.6.3.	Layer fully connected	31
1.3.5.	Considerazioni finali	33
<b>2.</b>	<b>PROPRIETÀ ED APPLICAZIONI DEI WATERMARK</b>	<b>35</b>
2.1.	CENNI STORICI	35
2.2.	POSSIBILI SCENARI APPLICATIVI	38
2.2.1.	Rivendicazione di proprietà	38
2.2.2.	Watermark di validità	39
2.2.3.	Fingerprint	40
2.2.4.	Watermark di licenza	41
2.2.5.	Altri possibili usi	42
2.3.	CLASSIFICAZIONE DI WATERMARK	42
2.3.1.	Per robustezza	42
2.3.2.	Per rilevamento	43
2.3.3.	Per visibilità	44
2.3.4.	Per reversibilità	45
2.4.	CARATTERISTICHE PRINCIPALI	46
2.4.1.	Watermark con bassa probabilità di errore	46
2.4.2.	Watermark multipli	47
2.4.3.	Watermark codificati	48
<b>3.</b>	<b>IDENTIFICAZIONE DI WATERMARKS MEDIANTE RETI NEURALI CONVOLUZIONALI</b>	<b>51</b>
3.1.	AMBIENTI DI SVILUPPO E LIBRERIE IMPIEGATI NELLA RICERCA	51
3.2.	PREPARAZIONE E PRE-ELABORAZIONE DEL DATASET	52
3.2.1.	Introduzione ai dataset utilizzati	53
3.2.2.	Preprocessing dei dati per il Machine Learning	55
3.2.2.1.	Normalizzazione	55
3.2.2.2.	Ridimensionamento delle immagini	55
3.2.2.3.	Gestione delle immagini corrotte	56
3.2.2.4.	Conclusioni sul preprocessing	56

3.2.3.	<i>Suddivisione del dataset: training, validation e test set</i> .....	56
3.2.4.	<i>Utilizzo di caching e prefetching per l'ottimizzazione del caricamento dei dati</i> .....	58
3.2.5.	<i>Ampliamento del dataset con tecniche di data augmentation</i> .....	59
3.3.	<b>ARCHITETTURA DELLA RETE NEURALE: COSTRUZIONE E CONFIGURAZIONE</b> .....	61
3.3.1.	<i>Caratteristiche e vantaggi dell'algoritmo di Ottimizzazione ADAM</i> .....	61
3.3.2.	<i>La Funzione di perdita "Binary Cross-Entropy"</i> .....	62
3.3.3.	<i>Scelta della dimensione del batch: equilibrio tra precisione e velocità</i> .....	63
3.3.4.	<i>Impiego di Keras Tuner nella Ricerca degli Iperparametri Ottimali</i> .....	64
3.3.4.1.	Esplorare le Tecniche di Random Search e Grid Search con Keras Tuner .....	64
3.3.4.2.	Risultati e Implicazioni della Selezione degli Iperparametri mediante Keras Tuner .....	66
3.3.5.	<i>L'Impatto delle Callbacks nella Gestione dell'Addestramento dei Modelli</i> .....	69
3.3.5.1.	ReduceLROnPlateau .....	69
3.3.5.2.	Early Stopping e Numero di Epoche: Bilanciare Apprendimento e Overfitting .....	71
3.3.5.3.	ModelCheckpoint: Protezione e Ripristino dei Modelli in Google Colab .....	72
3.4.	<b>RISULTATI E CONSIDERAZIONI FINALI</b> .....	73
3.4.1.	<i>Metriche di valutazione Scikit-Learn</i> .....	73
3.4.2.	<i>Analisi dei Risultati</i> .....	75
3.4.3.	<i>Caratteristiche e analisi della matrice di confusione</i> .....	77
4.	<b>CONCLUSIONI</b> .....	81
5.	<b>BIBLIOGRAFIA</b> .....	82
6.	<b>RINGRAZIAMENTI</b> .....	ERRORE. IL SEGNALIBRO NON È DEFINITO.

## Introduzione

L'ascesa di Internet e gli avanzamenti nelle tecnologie di trasmissione delle informazioni hanno notevolmente ampliato l'accesso e la distribuzione di dati in formato digitale. Reti, protocolli e varie applicazioni hanno creato un'infrastruttura informatica che facilita lo scambio di grossi volumi di dati multimediali. Nonostante questi sviluppi, esistono sfide significative relativamente alla pubblicazione e alla condivisione sicura di contenuti. Queste sfide includono la garanzia della privacy dei proprietari, che spesso sono molto esposti a causa delle informazioni contenute nei dati stessi, come ad esempio i dati sanitari elettronici. Altre sfide possono riguardare la manipolazione dei contenuti multimediali originali che possono creare disinformazione, come i video editati da intelligenze artificiali. Si affronta anche la gestione dei controlli delle autorizzazioni per accedere ai suddetti contenuti, specie negli ambienti privi di sistemi di identificazione. Infine, ma non meno importante, vi è il contrasto alla pirateria, che viola la proprietà intellettuale.

Uno dei principali ostacoli, in effetti, riguarda il rilascio dei contenuti di loro proprietà in un ambiente potenzialmente insicuro come la rete. C'è una forte apprensione che contenuti come file musicali, video e immagini possano essere distribuiti illegalmente ad altri, una volta messi online. Questa preoccupazione è particolarmente sentita nell'industria della musica, delle arti visive e cinematografiche, dove la protezione del copyright è essenziale.

In risposta a questa esigenza, si è sviluppata una crescente necessità di implementare robuste misure di sicurezza per proteggere i contenuti digitali. Queste strategie non solo tutelano i diritti degli autori e dei creatori, ma promuovono anche una distribuzione legittima e sicura dei contenuti digitali, contribuendo al progresso sostenibile dell'ecosistema digitale mondiale.

Il presente lavoro di tesi si focalizza sui principali algoritmi, tecniche e modelli per la rilevazione di watermark in immagini, un aspetto critico per la protezione dei diritti d'autore nel contesto della cyber security. Rilevare i watermark nelle immagini è fondamentale per mantenere i diritti di proprietà intellettuale in piattaforme digitali, come i social network ad esempio. Integrare un sistema automatico di identificazione dei watermark può immediatamente riconoscere se un'immagine è protetta da copyright, attivando un avviso che ne impedisce il

caricamento a chi non detiene i necessari permessi. L'utilizzo del deep learning nei sistemi di intelligenza artificiale si distingue per la sua abilità di apprendere automaticamente a svolgere task anche molto complessi come la rilevazione di segnali visivi specifici, come i watermark. Queste tecniche, dunque, possono essere efficacemente sfruttate per supportare la protezione della riservatezza, dell'integrità e della disponibilità delle informazioni digitali che sono sempre più esposte a rischi di violazioni del copyright e furti.

- Nel **capitolo 1** si affronta l'analisi di reti neurali e deep learning, esaminando definizioni e caratteristiche dei vari tipi di modelli esistenti. Approfondiremo le basi teoriche e le applicazioni pratiche di queste tecnologie emergenti, che hanno rivoluzionato il campo dell'intelligenza artificiale e stanno trovando applicazioni in settori come la visione artificiale, il riconoscimento vocale e la traduzione automatica. Tratteremo inoltre le funzioni di attivazione, i layer esplorando come ciascuno di questi elementi contribuisce alla costruzione di modelli robusti ed efficaci.
- Nel **capitolo 2**, discuteremo del concetto di watermark, dandone una definizione precisa e illustrando le sue varie applicazioni, esplorando come questa tecnologia può essere utilizzata per proteggere i diritti d'autore e garantire l'integrità dei contenuti digitali. Analizzeremo anche i diversi tipi di watermark, come quello visibile e invisibile. Inoltre, esamineremo casi di studio e scenari pratici per comprendere meglio l'importanza di questa tecnologia nel mondo digitale odierno, con particolare attenzione alla protezione dei contenuti multimediali come immagini, video e documenti.
- Infine, nel **capitolo 3** costruiremo un modello per l'identificazione dei watermark basato su reti neurali convoluzionali (CNN), analizzandone la prestazione in termini di accuratezza nella classificazione, attraverso una serie di test e valutazioni dettagliate. Esploreremo anche i diversi parametri e le diverse tecniche che influenzano le prestazioni del modello, come la qualità delle immagini e i metodi di estrazione utilizzati, garantendo una

comprensione completa e approfondita del processo. Verranno inoltre discusse possibili ottimizzazioni e miglioramenti per aumentare l'efficacia del modello esplorando l'utilizzo di meccanismi automatici per il tuning degli iperparametri.

In conclusione, verranno presentate sfide attuali e prospettive future in questo campo, discutendo come le odierne tecniche di visione artificiale in continua evoluzione, possano aprire la strada a nuovi scenari in cui è possibile automatizzare il processo di riconoscimento e di validazione dei diritti d'autore in immagini protette, migliorando dunque significativamente la sicurezza dei creatori di contenuti all'interno del cyberspazio.



# 1. Deep learning e reti neurali

## 1.1. Cenni storici

### 1.1.1. Origini

L'evoluzione dell'intelligenza artificiale ha segnato profondamente la interazione e percezione della tecnologia nel corso delle ultime decadi. Questo percorso è stato caratterizzato da una serie di momenti chiave e da figure emblematiche che hanno definito il settore.

Essa può essere definita in modi diversi, ma tutte le definizioni convergono sull'idea che una macchina possa agire o pensare in maniera che si considera razionale o simile all'umano. Ecco due tra le descrizioni più riconosciute:

- "L'automazione delle attività tipicamente associate al pensiero umano, come prendere decisioni, risolvere problemi, e apprendere..." (Bellman, 1978). [1]
- "Lo studio dell'intelligenza computazionale riguarda la creazione di agenti intelligenti" (Poole et al., 1998). [2]

Nel 1949, il neuroscienziato canadese Donald Hebb pubblicò "The Organization of Behavior" [3], dove esponeva una teoria pionieristica secondo cui l'accresciuta interconnessione tra neuroni migliorava la loro capacità di processare informazioni e apprendere. Questo concetto ha influenzato profondamente non solo la nostra comprensione dell'apprendimento biologico, ma ha anche stimolato lo sviluppo di modelli computazionali e matematici ispirati alla biologia. La regola di Hebb ha aperto la strada alla costruzione di reti neurali artificiali più sofisticate tramite l'aggregazione di neuroni artificiali.

Alan Turing, uno dei matematici più influenti del ventesimo secolo e spesso considerato il padre dell'informatica, fu tra i primi a indagare queste questioni. Si chiedeva se le macchine potessero prendere decisioni basate sulle informazioni che avevano a disposizione. Nel suo articolo del 1950, "Computing Machinery and Intelligence" [4], introduceva il concetto di macchine di Turing e il celebre test di Turing, ipotizzando una macchina programmabile manualmente per ogni

specifico passaggio; questo approccio rimase preminente nell'intelligenza artificiale fino agli anni '80. I progressi nella teoria della computabilità ispirarono molti studiosi a sviluppare modelli computazionali basati sulla cognizione umana, ponendo le premesse per il machine learning già dagli anni '40. L'espressione "intelligenza artificiale" fu introdotta per la prima volta da John McCarthy nel 1956, quando era assistente universitario di matematica al Dartmouth College, situato a Hanover, New Hampshire. Questo luogo è considerato la culla dell'intelligenza artificiale come disciplina. McCarthy riuscì a coinvolgere figure di spicco come Marvin Minsky, Claude Shannon e Nathaniel Rochester per organizzare un incontro dedicato alla teoria degli automi, alle reti neurali e allo studio dell'intelligenza. Durante l'estate di quell'anno, Dartmouth fu il teatro di un workshop di due mesi che vide la partecipazione di numerosi studiosi, molti dei quali avrebbero poi dato un contributo decisivo allo sviluppo del settore. Questo evento è da molti considerato la nascita ufficiale dell'IA [5]. L'obiettivo dell'incontro lo sviluppo di algoritmi sofisticati per affrontare questioni intricate. Ecco due degli algoritmi principalmente utilizzati durante la ricerca:

- **Reasoning and search**

In questo approccio, affrontare un problema viene paragonato a navigare un labirinto o un grafo, cercando il percorso più diretto verso una soluzione. Questo metodo è fondamentale negli algoritmi usati per giochi come gli scacchi, e può anche essere applicato nel controllo delle azioni di un robot.

- **Reti Semantiche**

Queste sono utilizzate per interpretare il linguaggio umano e sviluppare sistemi capaci di interagire in conversazioni realistiche. Le reti semantiche collegano parole tramite archi che rappresentano relazioni logiche, come dimostrato da ELIZA [6], una chatbot che ha rappresentato uno dei primi esempi di tale tecnologia.

Tali tecniche sono radicate nella visione filosofica secondo cui il pensiero umano può essere espresso e ridotto alla manipolazione di simboli computazionali, un ambito noto come intelligenza artificiale simbolica.

Il connessionismo invece, proposto come alternativa a quest'ultima, si fonda sull'idea che l'IA possa emergere simulando reti neurali. Sin dal 1943, con il lavoro pionieristico di Pitts e McCulloch, furono sviluppate reti artificiali capaci di gestire semplici operazioni logiche. Questo approccio prese ulteriore impulso nel 1958, quando Frank Rosenblatt, ispirandosi alla struttura biologica del neurone, inventò il perceptron, [7] che si rivelerà essere il precursore delle moderne reti neurali.

### 1.1.2. Prime reti neurali

Negli anni successivi, con le pubblicazioni di Henry J. Kelley [8] nel 1960 e Stuart Dreyfus nel 1962 [9], venne formalizzato l'algoritmo di backpropagation. Questo metodo di addestramento, basato sulla chain rule, è diventato fondamentale per ottimizzare la maggior parte dei modelli di reti neurali artificiali odierne.

Nel 1965, Alexey Grigoryevich Ivakhnenko e Valentin Grigorevich Lapa proposero la prima struttura gerarchica di una rete neurale, implementando anche funzioni di attivazione per i nodi, un contributo che segnò un punto di svolta per il nascente campo del deep learning.

Dalla seconda metà degli anni '60, l'entusiasmo iniziale per l'intelligenza artificiale subì un ridimensionamento. Invece di mirare alla creazione di sistemi con intelligenza generale, il focus si spostò verso lo sviluppo di sistemi capaci di affrontare problemi specifici in domini ben definiti. In questo periodo, emerse il concetto di "sistemi esperti" [10] o "sistemi basati sulla conoscenza".

*“L'utilizzo della conoscenza anziché dei dati, la preferenza per la conoscenza euristica rispetto a quella algoritmica, la sfida di rappresentare la conoscenza euristica all'interno di tali sistemi, la separazione tra conoscenza e programma per consentire la flessibilità su basi di conoscenza diverse, e infine, la necessità che i sistemi esperti siano in grado di spiegare le loro decisioni, rappresentare simbolicamente la conoscenza e integrare la meta conoscenza, ossia la consapevolezza della conoscenza stessa” [11]*

Con il passare degli anni, le aspettative nei confronti dell'IA sono cresciute, culminando negli anni '80 con l'emergere delle reti neurali, sistemi che imitano i processi di apprendimento biologico [12]. Nonostante le critiche [13] iniziali e un generale scetticismo da parte della comunità accademica, l'interesse per i perceptrons tornò prepotente quando Geoffrey Hinton e David Rumelhart presentarono l'algoritmo di backpropagation, essenziale per l'apprendimento nelle reti neurali contemporanee. Il vero salto qualitativo si è verificato all'alba del nuovo millennio con l'introduzione del deep learning e delle reti neurali profonde. Figure come Yann LeCun, Yoshua Bengio [14] e Geoffrey Hinton, spesso citati come i "tre padrini del deep learning", hanno portato avanti questa rivoluzione, espandendo le possibilità dell'IA in ambiti come il riconoscimento vocale, la visione artificiale e l'elaborazione del linguaggio naturale.

### **1.1.3. Progressi tecnologici e successi applicativi**

Con il passare degli anni, la ricerca ha continuato a evolvere rendendo sempre più complessi e potenti i modelli di reti neurali. La limitata capacità di calcolo, però, ritardava la loro piena applicazione pratica. Solo di recente, grazie al progresso delle unità di elaborazione grafica e alla comprensione delle loro capacità, sono stati ottenuti significativi successi applicativi con questi modelli avanzati.

Oggi, la rapidità e l'accuratezza di elaborazione dell'intelligenza artificiale sfidano le capacità umane, aprendo nuove strade e suggerendo un futuro in cui la collaborazione tra uomini e macchine diventa sempre più efficace e ricca di innovazioni. Sebbene si possano porre questioni cruciali riguardo all'impatto e alle potenziali conseguenze negative di questa tecnologia, si può evidenziare l'adozione diffusa dell'IA in vari settori e l'indiscutibile progresso di nuove tecnologie nella vita moderna.

## **1.2. Machine Learning e Deep Learning**

Nel mondo reale, ci si affida ai nostri sensi per raccogliere un'ampia varietà di informazioni che ci permettono di interagire con l'ambiente circostante, prendere decisioni e interpretare eventi. Per esempio, identificare un oggetto richiede

l'analisi di diversi fattori come luminosità, angolazione, forma e orientamento, oltre ai dettagli distintivi visibili. Questo compito rappresenta una sfida significativa per un algoritmo che deve organizzare un flusso continuo di dati in entrata, una sfida che va oltre le capacità del semplice representation learning. È quindi cruciale decomporre il processo cognitivo in vari livelli di astrazione e organizzare le informazioni in modo gerarchico: questo è l'obiettivo principale del Deep Learning.

Il Deep Learning è una specializzazione del Machine Learning che permette ai sistemi di elaborare dati grezzi e di riconoscere automaticamente le caratteristiche rilevanti per eseguire una varietà di compiti, come la rilevazione o la classificazione di oggetti.

I metodi utilizzati si avvalgono di molteplici strati di rappresentazione, ciascuno costituito da funzioni non lineari che trasformano le caratteristiche di ingresso di uno strato in rappresentazioni più astratte in uno strato successivo. L'intero insieme di questi strati forma ciò che comunemente si chiama una Rete Neurale, terminologia derivata dal modello dei neuroni nel cervello umano. In questo modello, ogni neurone artificiale riceve segnali di input, e se la somma dei segnali ponderata da specifici pesi supera una soglia di attivazione predeterminata, il neurone trasmette un segnale di output ai neuroni successivi.

A differenza delle tradizionali reti neurali, una rete è considerata "profonda" quando contiene numerosi strati di neuroni interni, come nei Multi Layer Perceptron (MLP) detti strati nascosti (hidden layers).

Nel corso del XXI secolo, il Deep Learning ha visto un'escalation notevole sia in termini di applicazioni pratiche sia per quanto riguarda l'interesse scientifico. Questi sviluppi sono iniziati con le reti neurali per il riconoscimento di immagini, come AlexNet [15], per poi evolversi verso architetture con miliardi di parametri che possono comprendere il linguaggio umano, generare articoli indistinguibili da quelli scritti da esseri umani, e risolvere complessi problemi di logica. Esempi notevoli includono i Language Models e le architetture basate su Transformers, come il famoso GPT-3 [16] di OpenAI.

### 1.2.1. Machine learning

Il machine learning rappresenta un campo specifico dell'intelligenza artificiale che aggrega una varietà di metodi, tecniche e algoritmi progettati per riconoscere automaticamente pattern e relazioni nei dati. Questi strumenti consentono alle macchine di apprendere dai dati e utilizzare queste informazioni acquisite per effettuare previsioni su dati futuri o prendere decisioni in scenari incerti. Per approfondire il concetto di "apprendimento dai dati", si fa riferimento a una definizione ben nota nell'informatica: "Un programma informatico è considerato in grado di apprendere dall'esperienza  $E$  rispetto a una classe di compiti  $T$  e una misura di prestazione  $P$ , se la sua efficacia nei compiti  $T$ , misurata da  $P$ , migliora con l'esperienza  $E$ ". Questo implica che l'apprendimento dai dati comporta un miglioramento nella capacità di svolgere determinati compiti man mano che si accumulano dati. Si procede ora con un'analisi più dettagliata di questa definizione.

Con  $T$  (task) si intendono tutte quelle attività complesse che richiedono un'intelligenza sofisticata per essere portate a termine con successo, ma che non includono l'apprendimento stesso dai dati; piuttosto, l'apprendimento è il mezzo attraverso il quale si acquisisce la capacità di eseguire tali task. Alcuni dei task più frequenti nel machine learning includono:

- **Classificazione**

L'algoritmo deve identificare la categoria di appartenenza di un dato input tra diverse opzioni possibili.

- **Regressione**

L'algoritmo è incaricato di prevedere un valore numerico basandosi su un input specifico, a differenza della classificazione, che produce un risultato categorico.

- **Trascrizione**

Il compito consiste nel trasformare input non strutturati in testo strutturato.

- **Traduzione**

Qui l'input è una serie di simboli in una lingua che l'algoritmo deve convertire in una serie equivalente in un'altra lingua.

La P (performance) è la metrica che quantifica le prestazioni di un algoritmo di machine learning, generalmente definita come l'accuratezza del modello, ovvero quante volte il modello fa previsioni corrette. La validità dell'algoritmo è misurata utilizzando un set di dati di test, separato da quello di addestramento (train set), perché la sfida principale del machine learning è quella di formulare previsioni valide su nuovi dati non ancora analizzati, utilizzando il modello sviluppato con dati correnti. Poiché i futuri dati non sono accessibili, si divide il set esistente in train set per l'addestramento e test set per la valutazione delle prestazioni.

Durante la fase di modellazione, come già visto, è cruciale evitare l'overfitting, che si verifica quando un modello è eccessivamente adattato ai dati di addestramento e non riesce a generalizzare su nuovi set di dati. Per prevenire questo, è necessario bilanciare accuratamente la varianza e il bias, due elementi inversamente proporzionali: aumentando l'uno si riduce l'altro.

### **1.2.2. Tecniche usate nel Deep Learning**

Il Deep Learning non si focalizza unicamente sulla profondità delle reti neurali, ma piuttosto sulla loro abilità di estrarre autonomamente le caratteristiche dai dati di input. Questa capacità, spesso descritta come un effetto "black box", presenta vantaggi e svantaggi: da un lato, permette alle reti di essere particolarmente efficaci nell'identificare tendenze in dati complessi, dall'altro, può rendere i risultati poco trasparenti e difficili da interpretare. Nonostante queste sfide, il deep learning continua a espandersi, sostenuto da prestazioni solide anche in contesti dove le tecniche convenzionali non riescono. Il suo sviluppo è ulteriormente accelerato dall'avanzamento tecnologico, come l'uso di unità di elaborazione grafica (GPU) e la disponibilità di dataset più ampi. Un prerequisito essenziale per ottenere risultati affidabili è l'utilizzo di un insieme di dati ampio e rappresentativo; in assenza di ciò, esiste il rischio che il modello diventi incapace di generalizzare, un fenomeno noto come overfitting [17]. Per ovviare a questo problema, si possono impiegare due approcci: la Data

Augmentation [18] e il Transfer Learning.

- **Data Augmentation**

Prendendo come esempio l'analisi di immagini, questo metodo consiste nel modificare leggermente le immagini attraverso rotazioni, inversioni, variazioni di intensità e ritagli, per creare molteplici immagini a partire da una sola.

- **Transfer Learning**

Questa strategia prevede l'utilizzo di reti pre-addestrate. Su grandi dataset (ad esempio, ImageNet), trasferendo la conoscenza insita nei pesi di tutti i livelli su un nuovo task d'interesse. Generalmente, tale approccio comporta il riaddestramento da zero degli ultimi livelli (ad un più alto livello di astrazione) con specifici dati di training: tramite questo approccio, le rappresentazioni a livelli di dettaglio molto basse (es. la capacità di distinguere i contorni degli oggetti) vengono trasferite dal modello pre-addestrato, mentre caratteristiche di più alto livello, specifiche del task, vengono apprese sfruttando la conoscenza di base trasferita.

I modelli prevalenti includono le reti neurali convoluzionali profonde (Deep Convolutional Neural Network, DCNN) e le reti neurali ricorrenti (Recurrent Neural Network, RNN).

### **1.2.3. Deep learning vs Machine Learning**

Come già menzionato, il deep learning rappresenta una branca specializzata del Machine Learning, da cui eredita l'obiettivo di utilizzare algoritmi progettati per apprendere e migliorare continuamente. La principale differenza rispetto al Machine Learning tradizionale sta nell'impiego delle reti neurali profonde, le quali sono concepite ispirandosi alla complessa architettura del cervello umano. La "profondità" di queste reti è definita dal numero di strati che compongono i modelli, consentendo ai dati di essere elaborati a diversi livelli di astrazione. Ciò facilita un'analisi dell'informazione sempre più raffinata e accurata. Gli algoritmi di deep learning sono stati introdotti per migliorare l'efficienza delle metodologie tradizionali di machine learning.



Le reti neurali profonde, tipiche del deep learning, sono particolarmente vantaggiose rispetto al machine learning convenzionale in diversi scenari:

- **Analisi di dati complessi**

Questi modelli sono capaci di assimilare rappresentazioni di alto livello dai dati, grazie soprattutto alle capacità delle reti neurali profonde.

- **Identificazione di pattern**

Le tecniche di deep learning si distinguono per la loro abilità nel rilevare e decodificare pattern intricati e caratteristiche latenti nei dati.

- **Gestione di grandi volumi di dati**

A differenza del machine learning tradizionale, che spesso richiede una selezione manuale delle caratteristiche dei dati, le reti neurali profonde possono elaborare automaticamente un'ampia quantità di variabili, semplificando notevolmente il processo grazie alla loro struttura.

- **Creazione di contenuti**

Il deep learning è largamente utilizzato anche nella generazione di nuovi dati e contenuti mediante l'utilizzo di Generative Adversarial Networks (GANs), che sono in grado di creare risultati più realistici rispetto alle tecniche tradizionali.

Nonostante questi vantaggi, il deep learning non è sempre la soluzione ideale. I modelli di deep learning danno il meglio di sé solo quando sono addestrati con dati abbondanti e di alta qualità. In contesti dove i dati sono scarsi, le risorse computazionali limitate, il problema da affrontare è relativamente semplice o è importante l'interpretabilità del modello, i metodi tradizionali di machine learning possono essere più adatti grazie alla loro semplicità e trasparenza.

### 1.2.4. Definizione e caratteristiche del Deep Learning

Il deep learning è un approccio avanzato all'intelligenza artificiale che simula, in modo semplificato, il funzionamento del cervello umano, permettendo ai computer di risolvere task complessi, come il riconoscimento di parole, volti o immagini.

Al cuore di questo approccio c'è la costruzione di una gerarchia di concetti, dove ogni livello si definisce attraverso legami con elementi più basilari. In questo modo, il computer visualizza il mondo come un esteso e dettagliato grafo.

Tale metodologia consente ai sistemi di analizzare e comprendere concetti complessi a partire da quelli più elementari, come l'identificazione di una persona tramite la composizione di occhi, naso e contorni.

Questa tecnica specifica è una ramificazione del machine learning, un campo più ampio che comprende vari metodi per l'apprendimento automatico dai dati. A differenza di approcci più tradizionali, che dipendono dalle caratteristiche definite dall'utente, questa tecnica identifica e impara autonomamente queste caratteristiche, organizzando l'input in una struttura di concetti interconnessi, indipendentemente dalla natura dell'input, sia esso un'immagine, un documento o un testo.

### 1.2.5. Quando utilizzare il Deep Learning?

Come discusso precedentemente, i modelli di deep learning dimostrano una notevole capacità nel riconoscere strutture complesse in una varietà di dati, come immagini, suoni e testi, generando informazioni e previsioni precise. Questa tecnologia permette di automatizzare compiti tradizionalmente legati all'intelligenza umana, inclusa l'identificazione e la classificazione di immagini, la conversione di file audio in testo, l'elaborazione del linguaggio naturale, la visione computazionale, l'analisi di testi, la bioinformatica e l'analisi di serie temporali. Considerando la potenza dei modelli di deep learning, potrebbe emergere il quesito se sia opportuno abbandonare altri metodi a favore esclusivamente di questi. La risposta è negativa. È fondamentale ricordare il principio del Rasoio di Occam, secondo il quale, tra due modelli che offrono prestazioni simili, è preferibile scegliere il più semplice. L'adozione di una rete neurale può risultare

complessa e non è pratico ricorrervi quando risultati equivalenti possono essere ottenuti con modelli più semplici e direttamente interpretabili. La scelta del modello adeguato richiede un'attenta valutazione del rapporto tra efficacia e complessità. Non necessariamente il modello più avanzato è il migliore per ogni scenario.

Inoltre, i modelli di deep learning raggiungono i migliori risultati quando addestrati su grandi quantità di dati di alta qualità. Anomalie o imprecisioni nei dati possono influenzare notevolmente la precisione del modello. Per minimizzare tali rischi, è essenziale effettuare una dettagliata pulizia e preparazione dei dati prima dell'addestramento. Questo processo richiede ampie capacità di memorizzazione dei dati. Pertanto, il deep learning si rivela una scelta vantaggiosa per le analisi quando il volume di dati è molto elevato e quando l'interpretazione dettagliata del modello non è una priorità.

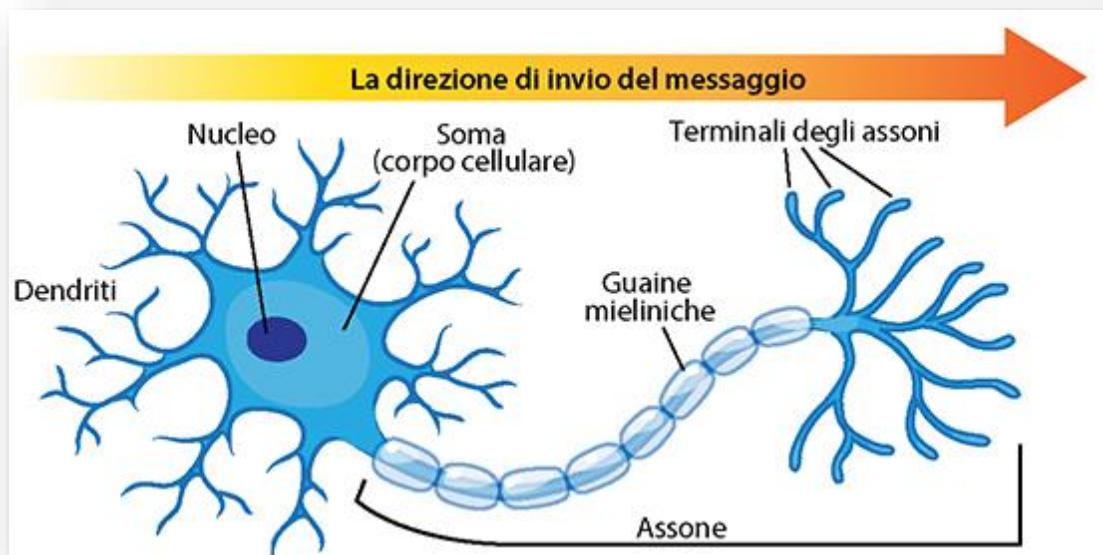
### **1.3. Reti neurali**

Una rete neurale è formata da un insieme densamente collegato di unità, che funzionano in modo simile ai neuroni biologici. Queste unità sono rappresentate come funzioni che possono essere addestrate, il che significa che si basano su parametri che possono essere modificati durante l'apprendimento. L'architettura delle reti neurali è strutturata in strati, ciascuno dei quali contiene un numero definito di unità logiche. Al di là del livello di input (che riceve il vettore  $x$ ) e del livello di output (che emette il vettore  $y$ ), ci sono i cosiddetti hidden layers, o livelli nascosti. Ogni livello è incaricato di elaborare il vettore di input destinato al livello successivo, conosciuto anche come vettore di attivazione.

La complessità di una rete può dipendere sia dal numero di unità logiche per ciascun livello sia dalla quantità di livelli stessi. Varie configurazioni architetturali sono adottate per soddisfare differenti requisiti computazionali. In alcuni casi, le reti neurali possono essere composte da un insieme di reti neurali più piccole, consentendo di suddividere il problema principale in vari sottoproblemi più semplici da affrontare. Questo approccio modulare offre un metodo efficace per decomporre e gestire complessità elevate in problemi più trattabili.

### 1.3.1. Neurone biologico

Una rete neurale è concepita per emulare il funzionamento dei neuroni, le cellule nervose che compongono il cervello umano. Ogni neurone è costituito da diverse componenti (Figura 1.1): i dendriti, che sono estensioni ramificate che ricevono segnali dalle sinapsi; il soma, o corpo cellulare, dove i segnali sono elaborati e generati; l'assone, il principale canale di trasmissione degli impulsi, protetto da una guaina mielinica; e il terminale assonico, che sono le estremità finali che connettono l'assone ad altre cellule come muscoli, ghiandole o altri neuroni.



*Figura 1.1 Struttura del neurone (Fonte: AskABiologist [19])*

La comunicazione tra neuroni avviene attraverso impulsi che si susseguono, scatenati da molecole chiamate neurotrasmettitori. Questi sono liberati nel terminale assonico (zona presinaptica) e captati dai dendriti del neurone ricevente (zona postsinaptica). Una volta nel nucleo, i neurotrasmettitori possono avere un effetto inibitorio, impedendo al neurone di attivarsi (tipicamente associato alla presenza di ioni di potassio), o eccitatorio, favorendo l'emissione di un nuovo impulso (generalmente correlato alla presenza di ioni di sodio). Nel cervello umano esistono diverse tipologie di neuroni, ognuna raggruppata in aree cerebrali specializzate in specifiche funzioni del corpo umano.

### 1.3.2. Neurone artificiale

Il primo modello di neurone artificiale venne proposto nel 1943 da Warren McCulloch e Walter Pitts. Questo modello era basato su input binari che producevano un output binario singolo. Frank Rosenblatt effettuò miglioramenti significativi al concetto originale di neurone artificiale nel 1958 con l'introduzione del Perceptron. Inoltre, questo prototipo non era confinato al trattamento di dati binari come il suo predecessore. Infatti, il Perceptron poteva processare input di valori reali e assegnare pesi variabili a ciascuno di essi. Un'importante innovazione introdotta era la regola di apprendimento basata sulla minimizzazione dell'errore, conosciuta come back-propagation. Questa regola consentiva al neurone di adattare automaticamente i pesi in modo ottimale.

Nonostante questi avanzamenti, le capacità del Perceptron restavano comunque limitate. Mentre era efficace nel riconoscere e classificare forme semplici, non aveva la capacità di gestire funzioni più complesse, come la funzione booleana XOR, evidenziando così i limiti intrinseci di questo modello nell'applicazione a problemi di classificazione più sofisticati. Ecco le varie somiglianze con il suo predecessore:

- **Zona presinaptica**

Nel perceptron, diversi segnali come  $x$ ,  $y$ ,  $z$  vengono ricevuti da neuroni antecedenti, ognuno dei quali è ponderato da un peso  $w_i$ .

- **Nucleo**

Alla combinazione pesata di questi segnali può essere aggiunto un bias  $b$ , una sorta di valore aggiunto che influenza l'output. Il totale viene poi elaborato da una funzione di attivazione  $f$ , che può essere sia lineare che non lineare, e il cui comportamento varia a seconda del tipo di funzione implementata.

- **Zona postsinaptica**

L'output della funzione di attivazione è inviato a tutti i neuroni nel livello

successivo.

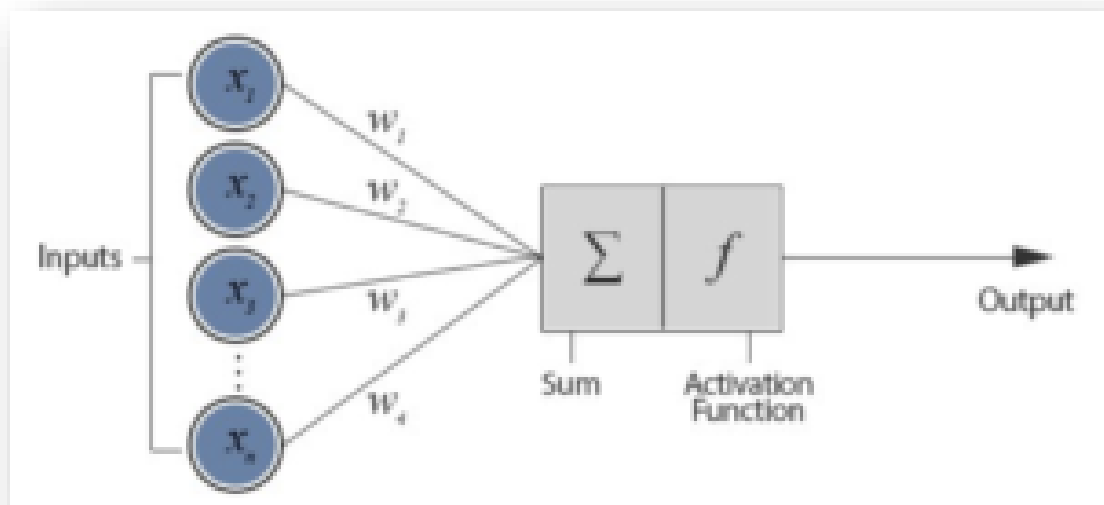


Figura 1.2 Neurone artificiale (Fonte: Domsoria [20])

Quando un input è introdotto nella rete, avviene quello che è noto come "forward pass", cioè le informazioni sono elaborate attraverso i vari strati in base alle operazioni descritte. Questo flusso può essere rappresentato dalla seguente equazione (Figura 1.2):

$$y = f(b + W \cdot X)$$

### 1.3.3. Funzioni di attivazione

Le funzioni di attivazione sono essenziali nelle reti neurali perché introducono elementi di non linearità, rendendo possibile l'apprendimento di compiti anche molto sofisticati. Se si utilizzassero funzioni di attivazione lineari, l'output del perceptron risulterebbe essere una mera funzione lineare, facile da computare ma incapace di gestire complessità elevate. Una proprietà fondamentale delle funzioni di attivazione è che devono essere differenziabili. Questo permette di applicare tecniche di ottimizzazione come la backpropagation per calcolare i gradienti della funzione di perdita e aggiornare di conseguenza i pesi nella rete.

Diverse funzioni di attivazione possono essere impiegate a seconda delle esigenze specifiche di un progetto, incluse la funzione Sigmoidale, la Tangente Iperbolica (*tanh*) e la ReLU (Rectified Linear Unit) [21]. Queste funzioni hanno caratteristiche uniche che possono essere sfruttate a seconda del contesto dell'applicazione. Va notato che la funzione ReLU, ha un punto di non differenziabilità quando  $x=0$ . Questo dettaglio può necessitare di considerazioni speciali durante l'implementazione degli algoritmi di apprendimento.

Nel processo di apprendimento delle reti neurali, le features in ingresso subiscono una somma pesata che viene successivamente normalizzata attraverso una funzione matematica per stabilizzare i valori di output. Qui di seguito sono descritte alcune delle funzioni di attivazione più comunemente impiegate per questa ricerca:

- **Funzione Sigmoidale**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Questa funzione accetta un valore reale come input e restituisce un output che varia tra 0 e 1. Specificamente ai valori altamente negativi corrisponde un output che si avvicina a zero, mentre valori fortemente positivi tendono verso 1. Questa funzione è stata molto usata in passato, dato che fornisce una rappresentazione intuitiva della percentuale di attivazione.

- **Funzione ReLU (Rectified Linear Unit)**

$$f(x) = \max(0, x)$$

Mantiene invariati gli input positivi mentre azzerava quelli negativi. Di conseguenza, il range di output di questa funzione varia da  $[0, +\infty]$ .

- **Funzione LeakyReLU**

$$f(x) = \max(0, x) + a \cdot \min(0, x)$$

A differenza della ReLU, questa funzione permette anche agli input negativi di contribuire all'output, seppur in misura ridotta, scalati da un coefficiente  $\alpha$ . Pertanto, il suo intervallo di output è  $[-\infty, +\infty]$ .

- **Funzione Tangente Iperbolica**

$$\tanh(x) = 2\sigma(2x) - 1 = \frac{\sinh(x)}{\cosh(x)}$$

Ha un codominio che si estende dall'intervallo  $[-1, 1]$ . Come la funzione sigmoide, anche la tangente iperbolica presenta una saturazione agli estremi del suo intervallo di output, ma si differenzia per il fatto che il suo output è centrato attorno allo zero, conferendole simmetria. In aggiunta, è interessante notare, come illustrato dalla formula che la funzione  $\tanh$  è essenzialmente una versione scalata della funzione sigmoide. Questa proprietà la rende particolarmente vantaggiosa in applicazioni dove è desiderabile un'attivazione che varia simmetricamente intorno allo zero, a differenza della sigmoide che è positivamente orientata.

- **Funzione Softmax**

È un'estensione della funzione sigmoide per casi in cui si hanno più classi. Formalmente è definita come segue:

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}$$

Questa funzione è particolarmente utilizzata nelle reti neurali per le classificazioni multiclasse. A differenza delle funzioni di attivazione precedenti che trattano ciascun input in modo indipendente, Softmax considera l'intero



vettore di input per produrre un vettore di output in cui ogni componente rappresenta la probabilità che l'input appartenga a una specifica classe. L'output di Softmax ha la proprietà di sommare a 1, rendendo le sue componenti interpretabili come probabilità. Essendo una generalizzazione della sigmoide a più dimensioni, è ideale per l'output delle reti neurali quando si vogliono categorizzare gli input in più di due categorie, garantendo che l'output sia una distribuzione di probabilità valida su tutte le classi possibili.

Recentemente, si è osservato un crescente interesse verso la funzione ReLU e le sue varianti, data la loro efficienza computazionale. Queste funzioni sono semplici da calcolare e anche le loro derivate sono facilmente ottenibili, il che velocizza significativamente i tempi di addestramento e di inferenza della rete. Un altro vantaggio significativo è la loro capacità di superare il problema dei gradienti che tendono a sparire, un inconveniente frequente con funzioni come la sigmoide.

### 1.3.4. Tipologie di reti neurali

Le **reti neurali convenzionali** sono caratterizzate da un'architettura basilare e sono comunemente utilizzate per la classificazione e l'elaborazione di dati strutturati. Sono ottimali per affrontare compiti che implicano l'analisi di dati organizzati in formati sistematici, poiché sono capaci di riconoscere schemi e correlazioni all'interno di tali dataset.

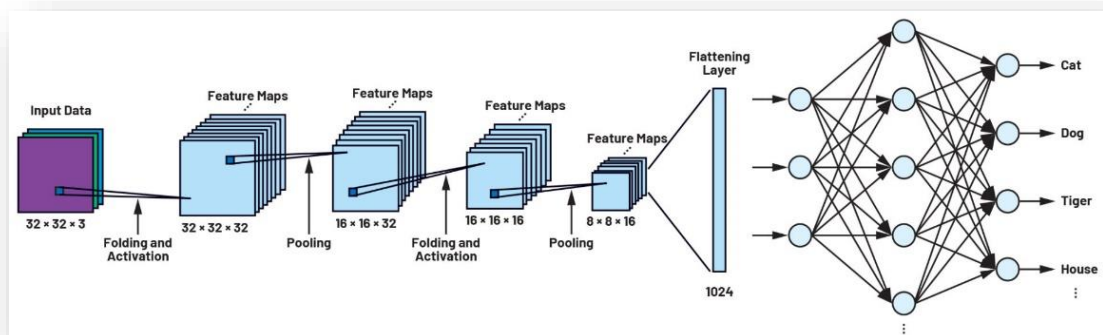


Figura 1.3 CNN (Fonte: ElettronicaNews [22])

Le CNN (Figura 1.3) sono ampiamente utilizzate nel campo della computer vision.

Queste reti approfittano della struttura spaziale delle immagini per elaborare i dati in modo più efficace. Grazie a questa capacità, sono particolarmente efficaci in applicazioni come il riconoscimento di immagini e l'analisi visiva.

D'altra parte, le **reti neurali ricorrenti** (Figura 1.4), o Recurrent Neural Networks (RNN), sono progettate per gestire dati che necessitano di tenere in considerazione le informazioni processate precedentemente. Tale caratteristica le rende adatte per applicazioni come i sistemi di conversione da parlato a testo o per il trattamento e l'interpretazione del linguaggio naturale. Le RNN, mantenendo una memoria delle interazioni passate, possono analizzare e comprendere sequenze di dati, essenziali in contesti dove è fondamentale la coerenza temporale delle informazioni.

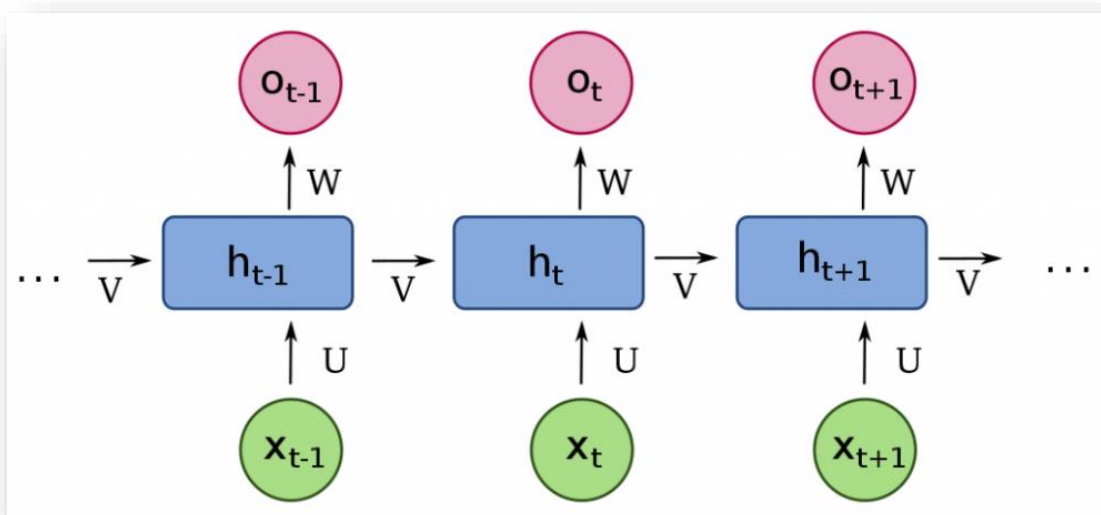


Figura 1.4 RNN (Fonte: DevelopersMaggioli [23])

Infine, le **reti neurali ibride** combinano diverse architetture o aggregano reti più semplici per creare modelli avanzati. Questi sono spesso impiegati in ambiti che richiedono soluzioni altamente specializzate, come nei sistemi di guida autonoma. In questi contesti, vari tipi di reti neurali vengono combinati per gestire e analizzare efficacemente molteplici tipi di dati e input sensoriali, facilitando così l'elaborazione necessaria per operazioni di navigazione autonoma sicure e affidabili.

### 1.3.5. Reti neurali convoluzionali

Le Reti Neurali Convoluzionali (CNN) rappresentano un pilastro fondamentale del deep learning, specializzate per l'elaborazione di dati strutturati in griglie, come le immagini digitali. Queste reti sono progettate per sfruttare le relazioni spaziali tra i dati attraverso un processo di apprendimento che evolve da semplici a complesse rappresentazioni.

Nella struttura delle CNN ogni neurone è collegato solo a un numero limitato di neuroni adiacenti nel livello precedente. Questo collegamento locale forma un campo recettivo che permette ai neuroni di concentrarsi su specifiche sotto-regioni dell'input. Questa caratterizzazione è rafforzata dalla condivisione dei pesi tra i neuroni dello stesso livello, una peculiarità che aiuta a ridurre il numero di parametri da addestrare, migliorando così l'efficienza computazionale.

I componenti chiave di una CNN includono:

- **Layer convoluzionali**

Impiegano filtri per estrarre le caratteristiche fondamentali dall'input.

- **Layer ReLU (Rectified Linear Unit)**

Introducono non-linearità nel modello, permettendo la cattura di pattern complessi.

- **Layer di pooling**

Condensano le dimensioni degli output convoluzionali, preservando le caratteristiche più rilevanti e mitigando l'overfitting.

- **Layer fully connected**

Elaborano le caratteristiche estratte per effettuare la classificazione finale.

Insieme, questi strati, che analizzeremo più dettagliatamente nel prossimo paragrafo, trasformano input visivi da forme e texture semplici a strutture intricate, come parti specifiche di un oggetto o interi scenari. Questo metodo di elaborazione imita in parte il modo in cui gli umani percepiscono visivamente il

mondo, riconoscendo oggetti basandosi su caratteristiche distinte.

Recentemente, le CNN hanno dimostrato eccellenti prestazioni in vari compiti di computer vision, tra cui la classificazione di immagini, il riconoscimento di oggetti e la segmentazione. Tali successi sottolineano la loro efficacia nel navigare e interpretare complessità visive, rendendole strumenti indispensabili per applicazioni avanzate in visione artificiale.

### 1.3.6. Architettura di una CNN: layer principali

#### 1.3.6.1. Layer convoluzionale

La convoluzione è una procedura lineare fondamentale nelle reti neurali, dove un filtro, o kernel, viene applicato a un tensore di ingresso, che è un array multidimensionale di numeri. Questo filtro è confinato in altezza e larghezza, ma copre completamente la profondità del volume di input, consentendo una completa elaborazione dei dati (Figura 1.5).

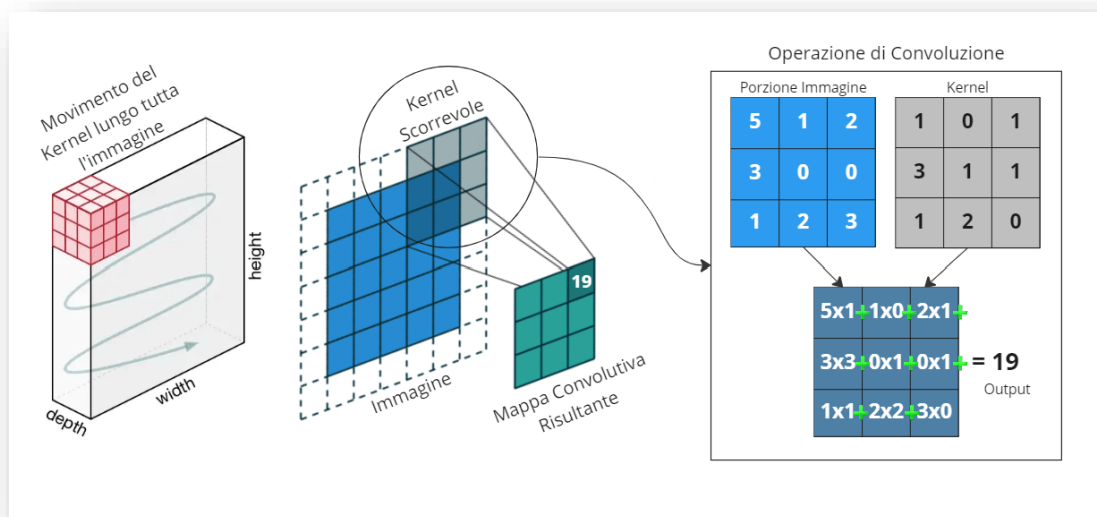


Figura 1.5 Operazione di convoluzione (Fonte: AI4Business [24])

Durante la convoluzione, il kernel scorre il tensore di ingresso sia verticalmente che orizzontalmente. In ogni posizione, viene calcolato un prodotto scalare tra il

kernel e la sezione del tensore sovrapposta, producendo così un valore di output specifico per quella posizione. Il risultato di questo processo è una mappa di attivazione o feature map, che rappresenta il volume di output ottenuto alla fine della convoluzione.

Questa operazione diventa particolarmente importante nel trattamento di immagini come input della rete. Per le immagini in scala di grigi, descritte dalla funzione  $I(x, y)$ , si utilizza un kernel bidimensionale  $K$  per implementare la convoluzione. Nel caso di immagini a colori, che incorporano i canali RGB (Red, Green, Blue) e sono quindi rappresentate da un tensore tridimensionale, anche il filtro utilizzato sarà tridimensionale per abbinarsi alla profondità dei dati. La convoluzione considera tutte e tre le dimensioni e, sommando gli elementi lungo la profondità del volume processato, produce una mappa di attivazione bidimensionale.

È possibile ripetere questo processo con vari filtri per creare diverse mappe di attivazione, ognuna rappresentando diverse caratteristiche estratte dal tensore di input usando filtri diversi. I pesi e il bias di ogni kernel sono i parametri apprendibili nello strato convoluzionale durante la fase di addestramento della rete.

Prima dell'addestramento, alcuni iper-parametri devono essere configurati dal programmatore:

- **Dimensione dei filtri**

L'altezza ( $h$ ) e la larghezza ( $w$ ) dei kernel, che determinano la dimensione del campo ricettivo di ogni neurone.

- **Numero di filtri**

Definisce la profondità di ciascun strato nascosto, e quindi il numero di feature maps prodotte.

- **Stride (passo)**

Gestisce la distanza tra due posizioni consecutive del kernel, influenzando quindi la sovrapposizione dei campi ricettivi.

- **Padding**

Aggiungendo un bordo al tensore di ingresso in varie modalità (come lo zero padding), si può modificare l'altezza e la larghezza del volume di output della convoluzione. Ad esempio, lo zero padding aggiunge un bordo di zeri per mantenere le dimensioni originali del tensore di ingresso in output.

Questi parametri hanno un impatto diretto sull'efficacia della rete nel catturare e processare le informazioni visive.

### 1.3.6.2. Layer di pooling

Lo strato di pooling gioca un ruolo essenziale nelle reti neurali, riducendo le dimensioni spaziali dell'immagine per semplificare l'output dello strato convoluzionale. Questo strato si occupa di comprimere la mappa delle caratteristiche, trasformando un'immagine più grande in una versione riassuntiva più piccola.

Due tecniche prevalenti per il pooling sono il **max pooling** e l'**average pooling** (Figura 1.6). Il max pooling seleziona il valore massimo all'interno di ogni regione della bitmap, evidenziando la caratteristica predominante di quella porzione dell'immagine. In contrasto, l'average pooling calcola la media dei valori all'interno di ciascun blocco, offrendo una rappresentazione più uniforme e meno accentuata delle caratteristiche.

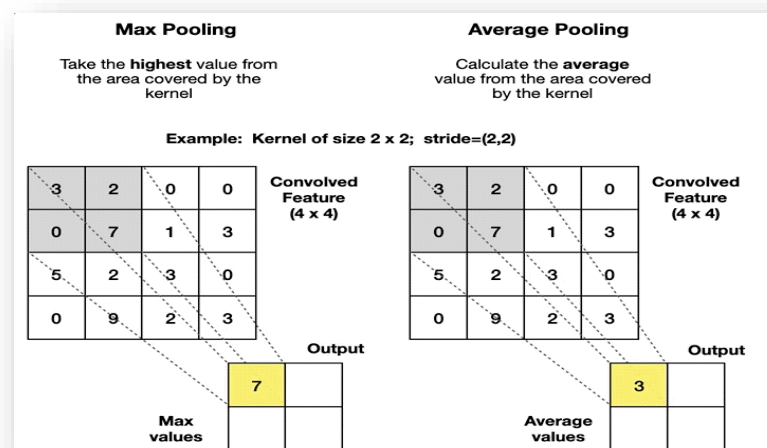


Figura 1.6 Operazione di pooling (Fonte: TowardsDataScience [25])

Il pooling è fondamentale anche per mantenere l'invarianza delle caratteristiche rispetto alla loro posizione originale. Benché la posizione esatta di una caratteristica possa diventare meno precisa, la relazione spaziale tra le caratteristiche viene preservata, mantenendo così l'importanza relativa delle caratteristiche all'interno dell'immagine compressa. Importante notare che il numero di canali dell'immagine rimane invariato, poiché il pooling opera solo sulle dimensioni altezza e larghezza dell'immagine.

Le dimensioni del blocco di pooling e lo stride, ovvero il passo con cui il blocco si sposta per coprire la prossima area di pooling, sono definiti in anticipo. Tipicamente si utilizzano blocchetti di dimensioni 2x2 o 3x3, con lo stride che può variare a seconda che si desideri o meno una sovrapposizione tra le aree di pooling. Se si preferisce che le regioni di pooling non si sovrappongano, lo stride viene impostato pari alla dimensione del blocco.

### 1.3.6.3. Layer fully connected

Dopo l'ultimo blocco dedicato all'estrazione delle caratteristiche, le mappe di attivazione vengono convertite in un vettore che viene poi collegato a uno o più strati completamente connessi (fully connected layers, FC layers). In questi strati, ogni elemento del vettore di input è collegato a tutti gli elementi dello strato successivo (Figura 1.7). L'ultimo strato completamente connesso ha il compito di produrre un vettore di dimensione  $K$ , dove  $K$  rappresenta il numero delle classi target per la classificazione dell'input.

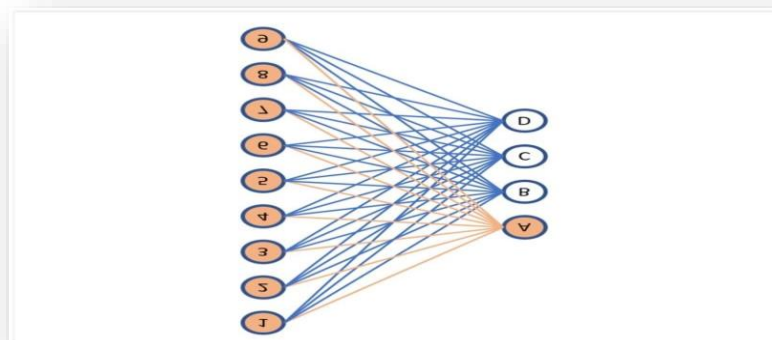


Figura 1.7 Fully connected layer (Fonte: BuiltIn [26])

La funzione di attivazione impiegata nell'ultimo strato FC solitamente varia rispetto a quelle usate nei livelli precedenti, scelta in base alle esigenze specifiche della rete. Per esempio, in contesti di apprendimento multi-classe si fa uso della funzione softmax, che trasforma i  $K$  valori reali prodotti dall'ultimo strato FC in una distribuzione di probabilità che rappresenta la probabilità di appartenenza alle  $K$  classi analizzate. Invece, nel contesto trattato nel capitolo 3, verrà adottata la funzione sigmoid, indicata per casi di classificazione binaria, dove il risultato deve distinguere tra due sole possibili categorie.

### 1.3.7. Layer aggiuntivi

Il layer di **dropout** e il layer di **batch normalization** sono due componenti cruciali nelle moderne architetture di reti neurali, utilizzati per migliorare la generalizzazione e l'efficienza del training. Il layer di dropout agisce durante la fase di addestramento della rete, "disattivando" casualmente alcuni neuroni con una certa probabilità. Questo processo aiuta a prevenire il fenomeno dell'overfitting, poiché impedisce che la rete diventi eccessivamente dipendente da qualsiasi input particolare o configurazione neuronale, promuovendo così una maggiore robustezza. D'altra parte, il layer di batch normalization normalizza gli output di un layer precedente sottraendo la media del batch e dividendo per la deviazione standard, stabilizzando così il processo di apprendimento. Questa normalizzazione aiuta a mantenere una distribuzione più uniforme dei valori di attivazione attraverso la rete, accelerando la convergenza durante l'addestramento e migliorando le prestazioni complessive della rete. Entrambi questi strati contribuiscono significativamente all'efficacia delle reti neurali nelle applicazioni pratiche, riducendo il tempo necessario per il training e aumentando la loro capacità di generalizzare bene su nuovi dati.



### 1.3.8. Considerazioni finali

Le reti neurali convoluzionali (CNN) sono caratterizzate da un alto numero di parametri da addestrare, il che può comportare una significativa richiesta di tempo e risorse computazionali. Una strategia per superare queste sfide è l'utilizzo di modelli pre-addestrati. Questi modelli sono già stati allenati su vasti set di dati per affrontare problemi simili a quelli di interesse. Importando una CNN già addestrata con i suoi pesi, è possibile sottoporre il modello ad un riaddestramento di fino su dati specifici, il che può coinvolgere solo i layer finali o limitare l'entità degli aggiornamenti eseguiti sui pesi tramite tecniche di fine-tuning.

Questo metodo, noto come transfer learning, non serve solo a diminuire i requisiti computazionali, ma aiuta anche a compensare la possibile scarsità di dati disponibili per l'addestramento. Il transfer learning consente di applicare conoscenze pregresse a nuove sfide, accelerando il processo di apprendimento e migliorando l'efficacia del modello su nuovi compiti con meno sforzo nella fase di addestramento e nella raccolta di dati.



## 2. Proprietà ed applicazioni dei watermark

Il digital watermarking [27], noto anche come "filigrana digitale", rappresenta un'innovativa tecnologia progettata per garantire la protezione dei diritti d'autore e confermare l'autenticità dei contenuti digitali quali documenti, immagini, audio e video. Questo metodo implica l'incorporamento di una sequenza di bit, nota come watermark, direttamente all'interno del file. Questa sequenza, di solito invisibile nell'uso quotidiano del documento, può essere estratta o identificata per assicurare che il contenuto sia integro e autentico.

Le procedure fondamentali del digital watermarking includono:

- **L'inserimento del watermark**

Questo passaggio dettaglia come integrare la sequenza di bit nel documento senza alterarne l'uso o la percezione, ad esempio modificando leggermente i pixel di un'immagine in modo che le modifiche siano invisibili all'occhio umano.

- **Il recupero del watermark**

Questa fase descrive il processo attraverso il quale il watermark viene estratto o rilevato per verificare l'integrità e l'autenticità del contenuto.

Strettamente collegato alla steganografia [28], che si occupa della dissimulazione delle informazioni all'interno di altri dati, il digital watermarking si focalizza specificatamente sulla salvaguardia dei diritti d'autore e sulla verifica dell'autenticità. La sua importanza si estende attraverso vari ambiti, inclusi la sicurezza informatica e il settore dei media digitali, permettendo ai creatori di proteggere e rivendicare la loro proprietà intellettuale in maniera efficace e discreta.

### 2.1. Cenni storici

Nel contesto attuale, fortemente influenzato dall'Internet, i contenuti digitali giocano un ruolo cruciale in molteplici aree, inclusi affari, industria e aspetti quotidiani della vita. Questo è dovuto al fatto che documenti e media digitali possono essere duplicati e diffusi con estrema facilità e a costi minimi; perciò,

questioni di sicurezza e pirateria sono diventate criticamente importanti.

Un tempo, la distribuzione pirata di questi dati necessitava di una copia fisica, limitando così la sua circolazione illegale. Tuttavia, con l'aumento dei trasferimenti dati tramite reti digitali e un accesso più facile ai contenuti, questa barriera fisica è stata superata.

Originariamente si pensava che la crittografia fosse una soluzione adeguata a proteggere le comunicazioni, ma si è dimostrata insufficiente nella pratica. Pur garantendo la sicurezza dei dati durante il trasferimento da un punto all'altro, una volta decodificati, i dati rimangono esposti e vulnerabili. Inoltre, la presenza di dati crittografati può attirare attenzione e generare sospetti sulle informazioni che contengono. Benché le tecniche di crittografia siano state sviluppate fin dal Rinascimento, già nel 1641 si evidenziavano casi in cui era preferibile nascondere informazioni segrete all'interno di messaggi inoffensivi piuttosto che cifrarle [29]. Questo approccio è ancora utilizzato oggi, mostrando come lo studio sulla sicurezza delle comunicazioni debba estendersi oltre la crittografia per includere la protezione del flusso di dati, che si concentra sul mascheramento delle informazioni.

Il focus accademico e industriale sulle metodologie di mascheramento delle informazioni ha preso forma nei primi anni '90, e la prima conferenza accademica dedicata a questo argomento si tenne nel 1996 [30]. Gli approcci iniziali erano orientati alla salvaguardia dei diritti d'autore, essenziali quando il materiale protetto, come film, musica o foto, viene rappresentato digitalmente. I vantaggi di questa digitalizzazione sono evidenti, ma porta con sé il rischio di riproduzioni illegali illimitate, che minacciano i diritti dei detentori dei contenuti originali. La pirateria informatica rappresenta una minaccia seria per le industrie che producono contenuti digitali come software e media. Si prenda ad esempio una società che sviluppa un software: dopo aver investito notevoli risorse nella creazione di un programma, la società lo commercializza, vendendo il prodotto finito insieme alle sue licenze. Tuttavia, durante il processo di distribuzione, può accadere che un pirata informatico intercetti una copia del software e inizi a duplicarla illegalmente per rivenderla su canali non autorizzati, spesso online. Questi pirati possono anche alterare la copia originale per ingannare sia i consumatori che le autorità legali.

Questo fenomeno non solo danneggia l'azienda originaria, riducendo le sue vendite ufficiali e limitando la sua capacità di finanziare lo sviluppo di nuovi prodotti, ma ha anche un impatto negativo sull'economia globale. Si stima che la pirateria di software generi perdite annuali per circa 16 miliardi di dollari. Per contrastare tale minaccia, i produttori di software stanno esplorando metodi per disincentivare la pirateria e facilitare l'azione legale contro i pirati. Un passo critico in questo processo è dimostrare che le copie in possesso dei pirati sono state acquisite illegalmente.

In risposta a queste sfide, sono state sviluppate tecniche come il "watermarking" digitale. Il watermarking serve a marciare il software o altre tipologie di contenuti con un segno distintivo invisibile, che può essere rivelato in seguito per provare l'origine legittima del programma. Questo tipo di marcatura è cruciale perché fornisce una prova tangibile in tribunale, dimostrando la proprietà dei diritti d'autore. La capacità di identificare risorse piratate attraverso il watermark è fondamentale per prevenire l'alterazione o la distruzione del marchio da parte dei pirati, garantendo così una protezione continua al prodotto originale.

Per contrastare efficacemente la pirateria, gli autori dei contenuti implementano tecniche avanzate di watermarking progettate per rendere estremamente difficile la rimozione del segno distintivo del prodotto. Se alterato, il file potrebbe diventare inutilizzabili, dissuadendo così gli attacchi.

Per quanto riguarda i software, ad esempio, esistono due approcci principali al watermarking digitale utilizzati nel settore del software:

- Il primo metodo consiste nell'incorporare un watermark identico in ogni versione distribuita del prodotto. Questo approccio serve principalmente a prevenire le false rivendicazioni di autorevolezza sul software, poiché il watermark inserito stabilisce chiaramente i diritti di proprietà intellettuale.
- Il secondo metodo, noto come watermarking seriale o fingerprinting, è considerato più robusto. In questo caso, ogni copia venduta del software contiene un watermark unico che non solo identifica il creatore del software ma anche l'acquirente. Al momento della vendita, il produttore registra il

contenuto del watermark insieme ai dettagli dell'acquirente. Se si sospetta che una copia del software sia stata acquisita illegalmente, verificare il watermark può immediatamente rivelare sia l'identità dell'acquirente originale sia la provenienza della copia illecita. Questo sistema facilita la raccolta di prove utili per supportare un'azione legale contro coloro che distribuiscono o possiedono copie non autorizzate del software.

## **2.2. Possibili scenari applicativi**

### **2.2.1. Rivendicazione di proprietà**

Il watermark di paternità è un tipo specifico di watermark destinato a identificare l'autore di un software o di un contenuto digitale, rivelandosi cruciale per garantire il riconoscimento e la protezione dei diritti intellettuali degli sviluppatori o degli autori coinvolti. Le sue caratteristiche principali, la visibilità e la robustezza, giocano un ruolo fondamentale. Deve essere facilmente riconoscibile e visibile agli utenti finali per servire come garanzia di qualità, suggerendo che il contenuto è genuino e prodotto da una fonte affidabile. Questa visibilità non solo facilita l'attribuzione chiara e diretta all'autore ma serve anche come un segno di qualità e autenticità del prodotto.

Per quanto riguarda la robustezza, è essenziale che il watermark resista a tentativi di rimozione o alterazione per proteggere efficacemente la paternità del contenuto. Un watermark robusto è progettato per resistere a varie forme di manipolazione digitale e tentativi di oscuramento, garantendo che il nome dell'autore rimanga associato al contenuto nonostante le modifiche. Inoltre, può essere personalizzato per riflettere la paternità individuale o collettiva; per esempio, in caso di un unico sviluppatore, un singolo watermark sarà sufficiente, mentre per progetti di gruppo o collaborazioni, possono essere usati contrassegni multipli per indicare tutti gli autori coinvolti.

Complessivamente, il watermark di paternità aiuta a prevenire l'uso non autorizzato del software o del contenuto digitale, fungendo da deterrente legale contro la violazione dei diritti d'autore e assicurando che gli autori ricevano il giusto riconoscimento e credito per il loro lavoro, essenziale per il supporto della

loro attività professionale e creativa. In questo modo, si rivela un elemento chiave nella gestione dei diritti digitali.

### 2.2.2. Watermark di validità

Il watermark di validità è un tipo di watermark specificamente progettato per confermare l'autenticità di documenti digitali o software. Al cuore di questo sistema si trova un compendio crittografico, essenzialmente un'identificazione unica derivata dall'essenza del documento originale, che funge da impronta digitale del contenuto. Questo compendio viene creato tramite una funzione di hash o aggregazione, che può variare nella sua complessità da metodi semplici come la concatenazione a tecniche più elaborate, per garantire che il compendio sia strettamente legato al documento.

Una volta generato, il compendio crittografico viene integrato nel documento attraverso un processo di aggregazione ben definito, assicurando così che l'identità digitale del documento rimanga coerente e sicura. Per verificare l'autenticità del documento, i verificatori utilizzano uno strumento pubblico per separare il compendio dal resto del documento e applicano un processo di estrazione per calcolare nuovamente il compendio dal documento potenzialmente modificato, confrontandolo con quello originale.

Il watermark di validità è progettato per essere estremamente sensibile a modifiche, rendendolo un eccellente indicatore di integrità. Qualsiasi alterazione del contenuto originale cambierà il risultato del compendio, segnalando una possibile manomissione. Nonostante la sua fragilità rispetto a modifiche del contenuto, il watermark deve mantenere una certa robustezza per resistere a cambiamenti accidentali o irrilevanti che non compromettono l'integrità fondamentale del documento.

Questo tipo di watermark è fondamentale per assicurare che il software o il documento non sia stato alterato dopo la sua creazione, offrendo agli utenti la certezza della fonte e fornendo un mezzo efficace per combattere la distribuzione non autorizzata e la falsificazione di prodotti software, tutelando i diritti di creatori e sviluppatori.

### 2.2.3. Fingerprint

Il watermark di fingerprinting è un tipo di watermark utilizzato nel software per inserire dati univoci come numeri seriali o informazioni sugli acquirenti, essenziale per il tracciamento e la sicurezza dei diritti di proprietà intellettuale. Implementando il fingerprinting, gli sviluppatori possono seguire il percorso di una copia del software dalla sua origine fino all'utente finale, il che è particolarmente utile per rilevare e agire contro la distribuzione non autorizzata e il pirataggio. Questa tecnica offre versatilità nella gestione delle informazioni tracciate, permettendo di configurare il sistema per riconoscere sia un singolo distributore sia vari attori all'interno di una rete di distribuzione.

Una qualità critica di questo tipo di watermark è la sua invisibilità, che assicura che l'utente finale non percepisca la presenza del watermark, mantenendo inalterata l'esperienza d'uso e prevenendo manipolazioni da parte di chi potrebbe voler eliminare o alterare il watermark. Inoltre, la robustezza è fondamentale per garantire che il watermark resista a tentativi di rimozione o di alterazione, rimanendo intatto anche in situazioni di sospetta violazione delle licenze d'uso per fornire prove affidabili dell'origine del software.

Attraverso un efficace sistema di fingerprinting, gli sviluppatori non solo possono dissuadere la distribuzione non autorizzata ma anche identificare e perseguire i responsabili grazie alle tracce lasciate dal watermark, rendendo il watermark di fingerprinting uno strumento prezioso per proteggere i diritti degli sviluppatori e garantire che il valore generato dal software sia correttamente attribuito ai suoi legittimi creatori. Questa tecnica rappresenta una soluzione di watermarking discreta ma potente, cruciale per la salvaguardia dei diritti di proprietà intellettuale.



## 2.2.4. Watermark di licenza

L'ultima categoria di watermark nel settore software comprende i contrassegni di licenza, che sono cruciali per regolamentare e monitorare l'uso legittimo di un software, incorporando informazioni chiave sui diritti di utilizzo concessi agli utenti. I contrassegni di licenza portano dati crittografati che specificano le condizioni d'uso del software, integrate direttamente nel codice del programma, associando così le politiche di licenza al software stesso. Spesso, questi watermark includono anche la chiave necessaria per decrittografare e avviare il software, la cui validità è strettamente legata all'integrità del watermark; se il watermark è alterato o rimosso, la chiave perde efficacia, rendendo il software inutilizzabile.

È fondamentale che i contrassegni di licenza siano estremamente fragili, così che qualsiasi tentativo di alterazione comprometta non solo il watermark stesso ma anche la funzionalità del software. Questa fragilità serve a scoraggiare la pirateria e l'uso illecito del software. Inoltre, la caratteristica dell'invisibilità di questi watermark è progettata per evitare che individui malintenzionati possano facilmente individuare e manomettere il watermark, aiutando a mantenere sicure e intatte le informazioni sulla licenza e sull'uso del software.

La combinazione di fragilità e invisibilità rende i contrassegni di licenza strumenti efficaci contro la contraffazione, proteggendo il software da manipolazioni non autorizzate e garantendo che le restrizioni di licenza siano rispettate. Questi watermark permettono agli sviluppatori di assicurarsi che il software venga utilizzato secondo le modalità previste dalla licenza, tutelando i diritti legali e commerciali legati al prodotto. In sintesi, i contrassegni di licenza rappresentano una componente di sicurezza essenziale nei programmi software, fungendo da custodi delle informazioni di licenza e dei diritti d'uso, la cui progettazione accurata e la loro implementazione nel software sono cruciali per mantenere il controllo sulla distribuzione e sul corretto utilizzo del software.

### **2.2.5. Altri possibili usi**

In certi contesti, il proprietario di un'immagine potrebbe voler limitare l'uso del contenuto a un determinato periodo di tempo. In questi casi, il watermark non solo identifica il proprietario, ma può anche definire le restrizioni temporali di utilizzo. Queste limitazioni possono essere comunicate ai dispositivi di visualizzazione o riproduzione, garantendo che l'immagine non venga utilizzata oltre il periodo consentito.

Oltre a questo, il watermark può arricchire l'immagine incorporando dati aggiuntivi, come la data, il luogo dello scatto, e il nome del fotografo. Queste informazioni non solo valorizzano il contenuto fornendo contesto, ma facilitano anche l'uso appropriato dell'immagine da parte degli utenti, aiutandoli a rispettare i diritti di utilizzo e a riconoscere il lavoro dell'autore. Questi dati arricchiscono l'esperienza visiva e aumentano la trasparenza e la tracciabilità del contenuto.

## **2.3. Classificazione di watermark**

### **2.3.1. Per robustezza**

I watermark fragili operano in modo sostanzialmente indipendente rispetto all'immagine che ospitano, implicando che qualsiasi intervento malevolo sull'immagine inevitabilmente implica una manipolazione del watermark stesso. Durante la verifica, le distorsioni provocate dal watermark vengono analizzate per localizzare le modifiche effettuate sull'immagine. Tuttavia, distinguere tra modifiche malevole e innocue, come la compressione dell'immagine che non altera il contenuto visivo, rappresenta una sfida significativa con questo approccio, poiché tali azioni possono essere erroneamente interpretate come manipolazioni del watermark dai più comuni algoritmi fragili.

D'altra parte, un algoritmo di watermarking semifragile è capace di distinguere tra interventi malevoli sull'immagine, come l'aggiunta o l'eliminazione di parti significative, e azioni che, pur modificando l'immagine, ne conservano la semantica, come la compressione o il ridimensionamento. Questi algoritmi sono specificamente progettati per identificare alterazioni che compromettono

l'integrità del watermark inserito, permettendo una distinzione efficace tra diversi tipi di modifiche [31].

Infine, i watermark robusti sono progettati per resistere a vari tipi di manipolazioni legate all'elaborazione del segnale. L'integrazione di una firma digitale all'interno del segnale rende estremamente difficile modificarla o eliminarla senza causare un deterioramento della qualità del segnale stesso. Questi tipi di watermark sono generalmente impiegati per garantire la protezione del copyright, assicurando che il diritto d'autore sia rispettato e che il contenuto originale rimanga intatto nonostante le manipolazioni.

### 2.3.2. Per rilevamento

Le tecniche di inserimento di watermark nei documenti digitali possono essere raggruppate in base alla loro modalità di rilevamento, distinguendosi in quattro principali approcci: watermarking pubblico, watermarking privato, watermarking semi-privato e watermarking asimmetrico.

Il watermarking pubblico non necessita né del documento originale né di una copia del watermark per verificare la presenza di una marcatura digitale. L'identificazione del watermark avviene attraverso l'analisi dell'autocorrelazione dei coefficienti utilizzati per la marcatura, rendendo questa metodologia molto robusta e adatta a un vasto numero di applicazioni.

Nel caso del watermarking privato, il processo di verifica del watermark richiede l'accesso al documento originale e talvolta anche al watermark stesso. Il rilevamento si basa sul confronto tra il documento marcato e l'originale, offrendo una maggiore precisione e riducendo il rischio di errori. Questo metodo, però, è limitato agli utenti che detengono il documento originale.

Per il watermarking semi-privato, si utilizza il documento marcato insieme al watermark per analizzare la loro correlazione. Il processo restituisce un valore booleano che conferma la presenza o l'assenza del watermark, con l'efficacia di questa tecnica che dipende dalla segretezza dell'algoritmo utilizzato per inserire il watermark, al fine di prevenire la manipolazione o l'estrazione del watermark stesso.

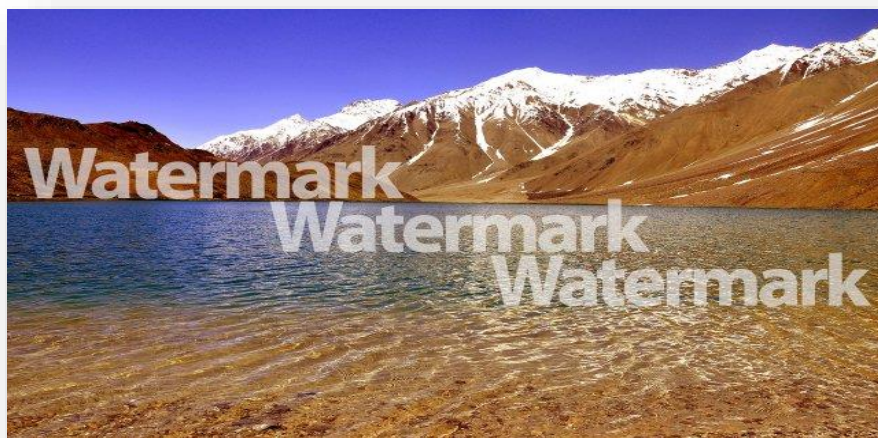
Infine, il watermarking asimmetrico impiega una chiave pubblica per il rilevamento del watermark, consentendo a tutti di verificare la sua presenza, mentre la chiave privata per l'inserimento rimane confidenziale. Questo sistema protegge il watermark da rimozioni o falsificazioni, anche se finora non si sono consolidati metodi completamente affidabili che prevengano sia la lettura non autorizzata sia la rimozione del watermark.

Questi metodi variano significativamente in termini di sicurezza, accessibilità e applicazioni pratiche, rendendo ogni tecnica adatta a specifici scenari di utilizzo.

### 2.3.3. Per visibilità

I watermark applicati ai documenti digitali si distinguono principalmente per la loro visibilità, essendo classificabili in visibili e invisibili, a seconda delle intenzioni dell'autore.

I watermark visibili (Figura 2.1) sono utilizzati principalmente per identificare il creatore o il detentore dei diritti di un'opera e si manifestano come loghi, firme o altri simboli che attestano l'originalità e la proprietà intellettuale di un'immagine, video o documento. Questi watermark fungono da deterrente visivo contro l'uso non autorizzato del contenuto, legando inequivocabilmente l'opera al suo legittimo proprietario, e sono spesso usati per mantenere una presenza visibile che conferma la proprietà.



*Figura 2.1 Esempio di Watermark multiplo visibile (Fonte: Fastweb [32])*

Dall'altra parte, i watermark invisibili sono progettati per rimanere occultati agli occhi dell'osservatore e non alterano visivamente il documento originale, permettendo di mantenere intatta l'estetica del contenuto. Questi vengono impiegati per codificare informazioni confidenziali all'interno del documento, come dettagli sull'autore, tracce di distribuzione o autenticazioni digitali che non sono rilevabili senza strumenti specializzati. L'accesso alle informazioni celate richiede tecniche specifiche, che variano in base al tipo di algoritmo di watermarking utilizzato e possono includere metodi di analisi avanzati per decifrare i dati incorporati.

La decisione di utilizzare un watermark visibile o invisibile dipende dagli obiettivi specifici dell'autore, sia che si tratti di proteggere visibilmente i diritti di un'opera o di immettere discretamente informazioni utili per la sicurezza e la tracciabilità del documento. In ogni caso, è essenziale che il watermark, sia esso visibile che invisibile, non danneggi in alcun modo la qualità visiva dell'immagine, garantendo così che il contenuto originale mantenga il suo valore estetico e informativo.

### **2.3.4. Per reversibilità**

La reversibilità dei watermark è un aspetto cruciale nella gestione dei diritti digitali, consentendo al proprietario originale del contenuto di eliminare il watermark a piacimento. Questa caratteristica può però contrastare con la necessità di mantenere la robustezza del watermark, particolarmente importante in contesti dove la resistenza a tentativi di alterazione è prioritaria.

I watermark reversibili permettono al detentore dei diritti di rimuoverli completamente, restituendo il documento al suo stato originale senza alcun segno della precedente marcatura. Tuttavia, la robustezza di un watermark, ovvero la sua capacità di resistere a manipolazioni o deterioramenti, spesso si riduce se il watermark è facilmente reversibile. Ad esempio, una firma digitale fortemente integrata potrebbe non essere rimovibile senza danneggiare l'immagine ospitante. Per quanto riguarda la quasi reversibilità, questo termine descrive una situazione in cui il watermark può essere eliminato in modo che il documento risultante sia

solo superficialmente simile all'originale, potenzialmente con minime alterazioni visive. La reversibilità può anche implicare la possibilità di generare un documento o un watermark fittizio che appaia identico all'originale, una pratica utile per verificare la robustezza dei sistemi di watermarking.

Sul fronte del copyright, per garantire che il watermark serva efficacemente come prova del diritto d'autore, è spesso necessario che esso sia permanente o quasi permanente. Questo garantisce che il watermark non possa essere facilmente eliminato o alterato, fornendo una protezione continua e affidabile del diritto d'autore.

In conclusione, mentre la reversibilità di un watermark offre al proprietario del contenuto una significativa flessibilità, può richiedere compromessi in termini di sicurezza e integrità del contenuto. Equilibrare la facilità di rimozione con la necessità di proteggere i diritti d'autore è quindi essenziale nella scelta di un sistema di watermarking.

## **2.4. Caratteristiche principali**

### **2.4.1. Watermark con bassa probabilità di errore**

Per un watermark efficace, è fondamentale garantire che l'identificazione del proprietario del contenuto sia precisa e indubbia, al fine di prevenire problemi di attribuzione errata dei diritti d'autore. L'importanza di minimizzare gli errori nella rilevazione dei watermark è cruciale per mantenere l'integrità dei diritti digitali. Il sistema di watermarking deve essere progettato in modo da identificare con certezza e precisione il proprietario del contenuto, assicurando che i dettagli incapsulati nel marchio siano distintivi e irripetibili. È essenziale che il processo di estrazione del watermark eviti completamente i falsi positivi, situazioni in cui il sistema riconosce erroneamente la presenza del watermark specifico in contenuti non marchiati. Allo stesso modo, è critico precludere i falsi negativi, che si verificano quando un watermark esistente non viene individuato dal sistema, mettendo a rischio il riconoscimento dei diritti del legittimo titolare.

La progettazione di un watermark deve includere elementi unici e personalizzati

per ciascun contenuto, difficilmente duplicabili, che assicurino l'unicità e la specificità del marchio applicato. La precisione di un watermark a bassa probabilità di errore dipende significativamente dalle tecnologie impiegate. L'uso di tecniche avanzate come l'analisi crittografica, schemi di codifica complessi e personalizzazione dettagliata sono vitali per potenziare la sicurezza e l'affidabilità del sistema.

In sintesi, progettare un sistema di watermarking che minimizzi gli errori sia di falsi positivi che di falsi negativi è essenziale per un efficace tutela dei diritti d'autore, garantendo che ogni uso del contenuto sia legittimamente autorizzato e tracciato correttamente.

## **2.4.2. Watermark multipli**

L'adozione di numerosi watermark in un unico documento è una tattica di sicurezza avanzata che mira a rafforzare la protezione del contenuto, tuttavia presenta sfide specifiche, specialmente quando i file sono condivisi su piattaforme che alterano i dati, come i social network.

Il documento può ospitare molteplici watermark, ciascuno associato a una chiave privata distinta. Questo metodo non solo aumenta la sicurezza ma permette anche una gestione differenziata dei diritti associati a diverse parti del contenuto. Ogni singolo watermark può essere individuato e autenticato attraverso la sua chiave dedicata, facilitando una verifica accurata dei diritti di proprietà e utilizzo del documento.

L'inserimento ripetuto di watermark utilizzando diversi algoritmi accresce la resistenza del documento a modifiche non autorizzate, potenziando la protezione contro attacchi informatici. Tuttavia, la qualità e l'integrità dei watermark possono degradarsi significativamente se il documento è soggetto a compressione o altre modifiche tipiche dei social network, rendendo complesso il recupero delle informazioni codificate.

È fondamentale utilizzare tecnologie avanzate per l'inserimento e il rilevamento dei watermark, capaci di gestire la sovrapposizione di molteplici firme digitali in maniera efficiente, minimizzando l'impatto sulla qualità visiva del documento. Deve essere mantenuto un equilibrio ottimale tra il rafforzamento della sicurezza

tramite multipli watermark e la conservazione della qualità visuale del contenuto, soprattutto in vista della condivisione su piattaforme digitali.

In conclusione, inserire più watermark in un singolo documento richiede una strategia ben ponderata che consideri le complesse dinamiche tra sicurezza incrementata e potenziale degradazione della qualità. La progettazione e l'implementazione di questi sistemi devono quindi equilibrare con cura robustezza, visibilità, e recupero delle firme digitali, assicurando che il contenuto rimanga protetto senza compromettere la sua fruibilità.

### 2.4.3. Watermark codificati

L'impiego di chiavi private nel processo di watermarking digitale svolge un ruolo critico nella protezione e nell'autenticazione dei documenti digitali. Queste chiavi, composte da sequenze di bit uniche, sono fondamentali sia per l'incorporazione che per la verifica dei watermark all'interno di un file.

Le chiavi private non sono solo responsabili per generare il segnale di watermark che viene integrato nel documento, ma sono anche indispensabili per la sua successiva identificazione. Solo i detentori della chiave possono confermare la presenza del watermark, assicurando così l'autenticità e la legittimità del contenuto. Ogni chiave è specificamente legata al suo proprietario, offrendo una certificazione sicura dell'origine e della proprietà del documento. Questa caratteristica è cruciale per prevenire l'uso improprio del materiale e facilitare il suo tracciamento.

Per quanto riguarda la robustezza dell'algoritmo di watermarking, è essenziale che esso sia progettato per difendersi efficacemente contro intrusioni. L'algoritmo deve includere un numero adeguato di chiavi complesse per sostenere attacchi esterni. La sicurezza del watermark rimane intatta finché non vengono decifrate tutte le chiavi, rendendo estremamente difficile per gli aggressori modificare o eliminare il watermark senza permesso. Inoltre, l'uso di molteplici chiavi private crea uno strato di sicurezza aggiuntivo che complica notevolmente ogni tentativo di accesso non autorizzato. Questa barriera crittografica multilivello protegge efficacemente sia il watermark sia il contenuto a esso associato.



In sintesi, l'utilizzo di chiavi private nel watermarking digitale non solo rafforza la sicurezza ma stabilisce anche una barriera chiara contro la manipolazione dei documenti digitali, preservando i diritti d'autore e mantenendo l'integrità del contenuto originale.



### 3. Identificazione di watermarks mediante reti neurali convoluzionali

Il capitolo è strutturato per dimostrare l'applicazione pratica delle tecniche di Deep Learning, già esposte nei capitoli precedenti, al task di watermark identification. L'analisi, che comprende l'utilizzo di tre diversi dataset, è stata condotta tramite l'implementazione di una rete neurale convoluzionale (CNN). A tale scopo sono stati utilizzati il linguaggio di programmazione Python e le librerie di deep learning Keras e Tensorflow, unitamente all'ambiente di esecuzione Google Colab per l'uso di risorse di calcolo virtualizzate.

#### 3.1. Ambienti di sviluppo e librerie impiegati nella ricerca

Nel contesto del progetto, l'impiego di Python 3 e delle sue librerie scientifiche e di machine learning è cruciale per assicurare efficienza, precisione e flessibilità. Tra le librerie fondamentali figurano TensorFlow, NumPy, SciKit-Learn, Keras Tuner e Pandas, ciascuna con un ruolo specifico nel facilitare e ottimizzare il lavoro di programmazione e di modellazione.

**TensorFlow**, sviluppata da Google, è una libreria open source progettata per il calcolo numerico e tensoriale che semplifica la costruzione di reti neurali. Questa libreria è estremamente efficace nella creazione e nell'addestramento di modelli di deep learning, permettendo di gestire operazioni complesse sui tensori con grande efficienza. Inoltre, TensorFlow è compatibile con hardware dedicato come le GPU e le TPU, accelerando notevolmente i tempi di training dei modelli.

**NumPy** si afferma come il framework di riferimento per la manipolazione di array e matrici in Python. È indispensabile nel deep learning per la gestione delle operazioni sui tensori, essenziali per la costruzione e l'ottimizzazione di modelli di rete neurale. La sua capacità di effettuare calcoli numerici ad alta performance è fondamentale per il pre-processing dei dati e per le trasformazioni matematiche.

**SciKit-Learn**, nella sua versione 0.24, rappresenta una delle librerie più utilizzate per il machine learning in Python. Questa libreria offre un'ampia varietà di algoritmi di machine learning, funzioni di pre-processing, tecniche di riduzione della dimensionalità e metriche di valutazione. Pur non essendo specificamente progettata per il deep learning, è frequentemente impiegata per preparare i dati prima di addestrarli su modelli più complessi realizzati con TensorFlow o Keras, o per valutare le metriche di performance post-training.

**Keras Tuner** emerge come uno strumento essenziale per l'ottimizzazione degli iperparametri nei modelli costruiti con Keras, facilitando la ricerca automatica della configurazione ottimale dei parametri, il che può migliorare significativamente le prestazioni dei modelli.

**Pandas** è una libreria cruciale per la manipolazione e l'analisi dei dati, particolarmente utile per gestire dati tabulari con le sue strutture DataFrame e Series che offrono funzionalità avanzate per l'indice, il slicing e l'aggregazione dei dati.

L'integrazione di queste librerie con Google Colab, un ambiente di sviluppo basato su Jupyter notebook, permette di sfruttare risorse computazionali gratuite come GPU e TPU. Google Colab mette a disposizione GPU NVIDIA gratuite che accelerano il training dei modelli rispetto all'uso delle sole CPU. Le TPU (Tensor Processing Units), sviluppate da Google, sono processori specializzati che ottimizzano i carichi di lavoro di TensorFlow. La TPU v2, in particolare, offre prestazioni elevate per il training e l'inferenza, supportando operazioni complesse e grandi set di dati a velocità impressionanti.

L'uso di **Google Colab** con queste unità di calcolo può ridurre drasticamente i tempi di addestramento e migliorare l'efficacia del progetto di tesi, rendendo possibili esperimenti che altrimenti richiederebbero risorse hardware proibitive.

### **3.2. Preparazione e pre-elaborazione del dataset**

### 3.2.1. Introduzione ai dataset utilizzati

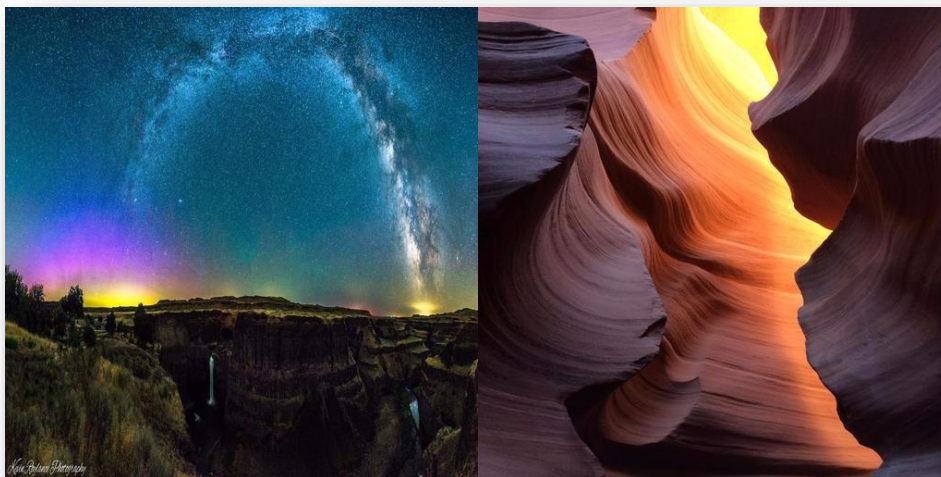
Nel lavoro di ricerca presentato, sono stati selezionati tre dataset specifici per esplorare la classificazione delle immagini con e senza watermark, ciascuno scelto per la sua pertinenza rispetto agli obiettivi dello studio.

Il primo dataset (Figura 3.1), denominato "**Scenery\_Watermark**" [33], consiste in 22,762 immagini di paesaggi, tutte con dimensioni di 512x512 pixel. Ogni immagine è stata accuratamente etichettata a mano, con il 44% di esse che presenta un watermark. Questa cura nell'etichettatura assicura la precisione necessaria per gli esperimenti di classificazione.



*Figura 3.1 Immagini rispettivamente con e senza watermark di "Scenary\_Watermark" (Fonte: Kaggle [33])*

Il secondo dataset (Figura 3.2) utilizzato è "**The Watermark Challenge**" [34]. Qui si è impiegata esclusivamente la porzione di training, che comprende 15,933 immagini, anch'esse di dimensioni 512x512 pixel. Similmente al primo dataset, anche in questo caso il 44% delle immagini è watermarkato. È stata presa la decisione di escludere i dati non etichettati per mantenere la coerenza e affidabilità nell'analisi dei risultati.



*Figura 3.2 Immagini rispettivamente con e senza watermark di “The Watermark Challenge”  
(Fonte: Kaggle [34])*

Il terzo dataset (Figura 3.3), **“Watermarked / Not Watermarked Image”** [35], include 31,575 immagini ottenute dal portale Pexels, noto per le sue fotografie di alta qualità disponibili gratuitamente. Questo dataset varia in dimensione e presenta una distribuzione bilanciata delle etichette, con esattamente il 50% delle immagini con watermark, offrendo così un ideale scenario di equilibrio per l'analisi comparativa.



*Figura 3.3 Immagini rispettivamente con e senza watermark di “Watermarked / Not Watermarked Image”  
(Fonte: Kaggle [35])*

Tutti questi dataset sono stati acquisiti attraverso la piattaforma **Kaggle**, che è un riferimento essenziale per il campo del machine learning, fornendo accesso a una vasta quantità di dati utili per diversi tipi di analisi computazionale.

In termini di caratteristiche comuni, le dimensioni delle immagini nei primi due dataset sono uniformemente di 512x512 pixel, mentre il terzo dataset presenta una varietà nelle dimensioni. La colorazione gioca un ruolo fondamentale in queste collezioni, con la maggior parte delle immagini presentate a colori, un fattore critico per le tecniche di classificazione visiva, in particolare per la rilevazione di watermark in contesti vari.

### 3.2.2. Preprocessing dei dati

Il preprocessing dei dati è un passaggio cruciale per preparare i dataset al fine di ottenere risultati ottimali dal modello di machine learning. Questa sezione descrive le diverse fasi e le tecniche adottate per assicurare che i dati siano adatti ed efficaci per l'addestramento.

#### 3.2.2.1. Normalizzazione

Il primo passo nel pre-processing è la normalizzazione dei dati. Per le immagini, questo significa scalare il valore dei pixel in un intervallo compreso tra 0 e 1. Ciò si ottiene dividendo ciascun pixel per 255, che è il valore massimo possibile per un pixel in un'immagine a colori standard. Questa operazione è fondamentale poiché riduce la varianza dei dati di input e facilita il processo di apprendimento, rendendo il training più veloce e più stabile.

#### 3.2.2.2. Ridimensionamento delle immagini

Data la diversità nelle dimensioni delle immagini del dataset e le notevoli dimensioni del dataset stesso, si è optato per un approccio di ridimensionamento per uniformare le dimensioni di tutte le immagini a 224x224 pixel. Questa decisione è stata presa per due motivi principali: primo, per velocizzare

l'addestramento riducendo il numero di pixel da processare, e secondo, per uniformare le dimensioni d'input per il modello di machine learning.

Il ridimensionamento è stato effettuato tramite uno script jsx in Photoshop, sfruttando la tecnica di interpolazione bicubica. Questa scelta è dovuta alla capacità della bicubica di offrire risultati superiori in termini di qualità dell'immagine rispetto ad altre tecniche di interpolazione, come la bilineare. L'interpolazione bicubica considera i pixel adiacenti in una griglia più ampia (4x4), risultando in immagini finali più nitide e dettagliate.

### **3.2.2.3. Gestione delle immagini corrotte**

Durante il processo di ridimensionamento, è stata anche effettuata una verifica per identificare e scartare le immagini corrotte. Questo è essenziale per mantenere l'integrità del dataset, assicurando che il modello di machine learning non venga addestrato su dati errati o incompleti, il che potrebbe portare a errori durante la fase di training o a una ridotta accuracy del modello.

### **3.2.2.4. Conclusioni sul preprocessing**

Le tecniche di preprocessing adottate per questo progetto assicurano che il dataset sia non solo uniforme in termini di formato delle immagini, ma anche di qualità. L'uso di Photoshop per il ridimensionamento tramite interpolazione bicubica garantisce la conservazione dei dettagli critici, importanti per il riconoscimento accurato di elementi sottili come i watermark, specialmente quando questo è molto piccolo in proporzione all'immagine. Questa attenzione ai dettagli nel preprocessing è vitale per il successo dell'analisi finale e per la performance del modello di machine learning.

### **3.2.3. Suddivisione del dataset: training, validation e test set**

Nella ricerca descritta, il processo di creazione e suddivisione dei dataset utilizzati per la classificazione delle immagini con e senza watermark è stato



curato per garantire l'efficacia dell'allenamento, della validazione e del test del modello. Tre distinti dataset sono stati combinati, formando un insieme totale di 70,270 immagini. Seguendo la proporzione standard nel campo del machine learning, i dati sono stati distribuiti in 70% per il training, 15% per la validation e 15% per il test.

Per assicurare una distribuzione uniforme delle caratteristiche provenienti da ciascuna fonte di dati, è stato essenziale miscelare bene il dataset prima di procedere con la suddivisione. Ciò è stato realizzato utilizzando la funzione `shuffle()` fornita dalle librerie di manipolazione dei dati come TensorFlow o Keras. Questa pratica si rivela critica, specialmente quando si lavora con dataset compositi, per evitare qualsiasi forma di bias nella distribuzione dei dati tra i vari set.

Il **Training Set** costituisce il 70% dell'intero insieme, corrispondente a 49,189 immagini. Questo set è cruciale per l'addestramento del modello, permettendo di apprendere a identificare e distinguere le caratteristiche rilevanti relative alle categorie di interesse, in questo caso, immagini con e senza watermark. La bilanciatura delle classi è stata assicurata tramite la funzione di filtraggio di Keras, mantenendo una rappresentazione equa del 50% per ciascuna categoria.

Il **Validation Set** comprende il 15% del totale, con 10,541 immagini. Serve per affinare i parametri del modello e per prevenire l'overfitting. Durante il training, questo set agisce come un controllo per monitorare le prestazioni del modello su dati non utilizzati nell'apprendimento diretto.

Il **Test Set** rappresenta anch'esso il 15% del totale, inclusivo di 10,541 immagini, utilizzato per valutare le prestazioni finali del modello. Questo set offre una valutazione imparziale dell'efficacia del modello su un campione di immagini che non sono state impiegate né nel training né nella validation, fornendo un'indicazione realistica del comportamento del modello in condizioni operative reali o su nuovi dati.

Ulteriori considerazioni sulla suddivisione del dataset includono la scelta della divisione 70-15-15, motivata dalla necessità di massimizzare i dati disponibili per l'addestramento mantenendo al contempo sufficienti risorse per una valida verifica e valutazione delle capacità generalizzative del modello. Per i set di validation e test, le immagini sono state selezionate casualmente per mantenere la variabilità e assicurare che la performance del modello rifletta fedelmente il suo comportamento su una varietà di dati reali.

Questa metodologia, attenta alla preparazione e alla suddivisione dei dataset, garantisce che il modello sviluppato sia robusto, affidabile e capace di generalizzare da un ambiente controllato a scenari di utilizzo reale, mitigando il rischio di overfitting e assicurando che le prestazioni siano replicate efficacemente su nuovi dati.

### 3.2.4. Utilizzo di caching e prefetching per l'ottimizzazione del caricamento dei dati

Per potenziare l'efficienza e la velocità del processo di addestramento dei modelli di deep learning, sono state implementate le tecniche di caching e prefetching dei dati attraverso le funzioni disponibili in Keras. L'adozione di queste pratiche è fondamentale per ottimizzare l'uso delle risorse durante l'addestramento, specialmente nel contesto di dataset di grandi dimensioni.

La tecnica di **caching** dei dati implica la memorizzazione dei dati che sono stati già caricati e preprocessati, permettendo così al modello, nelle epoche successive dell'addestramento, di accedere rapidamente a questi dati senza la necessità di ricaricarli e ripreprocessarli. Questo approccio, reso possibile dalla funzione `cache()` di Keras, è estremamente utile quando i dati di addestramento sono invarianti tra le diverse epoche. L'effetto diretto è una riduzione significativa dei tempi di caricamento dei dati e della gestione dell'I/O, aspetti che spesso rappresentano un collo di bottiglia durante il training.

Il **prefetching** dei dati, d'altra parte, è una strategia che prepara anticipatamente i dati per essere subito disponibili non appena il modello ne ha bisogno. Grazie

alla funzione `prefetch()` di Keras, è possibile caricare in memoria il prossimo batch di dati mentre il modello elabora il batch corrente. Questo garantisce che il tempo di attesa tra la fine dell'elaborazione di un batch e l'inizio del successivo sia ridotto al minimo, poiché i dati successivi sono già pronti per l'elaborazione. Tale meccanismo è vitale per mantenere costantemente attivo l'hardware di addestramento, sia esso CPU o GPU, eliminando sostanzialmente i tempi morti e massimizzando l'efficienza operativa dell'addestramento.

Nel progetto in questione, l'implementazione di caching e prefetching ha notevolmente migliorato l'efficienza dell'addestramento. Queste strategie hanno permesso di velocizzare il processo di allenamento del modello e di ottimizzare l'utilizzo delle risorse computazionali, come la TPU. In definitiva, l'introduzione di queste tecniche non solo ha ridotto i tempi di addestramento, ma ha anche contribuito a stabilizzare il processo di apprendimento, risultando in un modello più robusto e affidabile.

### 3.2.5. Ampliamento del dataset con tecniche di data augmentation

Per accrescere la capacità di generalizzazione dei modelli di deep learning, è cruciale disporre di un ampio volume di dati di addestramento. Le reti neurali convoluzionali, che tendono ad avere milioni di parametri apprendibili, richiedono un numero di esempi di addestramento proporzionale per ottenere prestazioni ottimali. In questo contesto, la data augmentation rappresenta una tecnica particolarmente utile.

La **data augmentation** arricchisce il dataset esistente mediante l'applicazione di una serie di trasformazioni ai dati, aumentando così il valore delle informazioni di base. Studi numerosi hanno validato questo approccio, dimostrando che anche semplici tecniche di manipolazione delle immagini come il cropping, la rotazione e il flipping possono migliorare notevolmente la capacità di generalizzazione di un modello.

Questa pratica può essere realizzata in due modi principali: online e offline, ognuno con specifici vantaggi e svantaggi.

### **Tecnica Online**

Nell'approccio online, le trasformazioni vengono applicate in tempo reale durante il training del modello. Ogni immagine nel batch subisce modifiche casuali prima di ogni iterazione, che possono includere rotazioni, traslazioni, ridimensionamenti, e molto altro. Le immagini trasformate non vengono salvate sul disco, evitando così l'occupazione di spazio di archiviazione aggiuntivo. Questo metodo introduce una variazione casuale continua, prevenendo la memorizzazione delle specifiche immagini da parte del modello e promuovendo una migliore generalizzazione. Tuttavia, può prolungare i tempi di addestramento poiché le trasformazioni devono essere applicate al volo.

### **Tecnica Offline**

Nell'approccio offline, l'aumento dei dati viene eseguito prima dell'inizio dell'addestramento. Le trasformazioni desiderate vengono applicate a ciascuna immagine del dataset originale e le varianti vengono salvate sul disco. Questo processo crea un dataset espanso, permettendo al modello di allenarsi su molte più immagini di quelle originalmente disponibili, migliorando la robustezza e riducendo il rischio di overfitting. Il vantaggio principale è che il preprocessing viene eseguito una sola volta, risparmiando tempo durante le iterazioni successive di training, ma richiede maggiore spazio di archiviazione.

### **Scelta tra Online e Offline**

La scelta tra le due tecniche dipende da vari fattori come le risorse di calcolo e di storage disponibili, la dimensione del dataset originale e la variabilità necessaria. L'approccio online è preferibile per evitare l'espansione eccessiva del numero di file e per offrire una variabilità illimitata, mentre l'offline è più efficiente se lo storage non rappresenta un problema e le risorse di calcolo sono limitate.

Nel progetto attuale, nonostante il dataset sia già considerevole, non si è optato per implementare tecniche di data augmentation a causa della necessità di preservare i watermark nelle immagini. Trasformazioni distruttive come il cropping potrebbero eliminare parti cruciali delle immagini contenenti i watermark, compromettendo l'accuratezza del modello. Tuttavia, se fossero state disponibili più risorse computazionali, si sarebbero potute considerare trasformazioni non distruttive, che non alterano la completezza delle informazioni visive essenziali, come il flip orizzontale o verticale, fornendo così variazioni utili senza influenzare negativamente la capacità del modello di riconoscere i watermark.

### **3.3. Architettura della rete neurale: costruzione e configurazione**

Nel contesto di questo progetto di tesi, l'applicazione dei modelli di deep learning è stata realizzata utilizzando Keras, un potente modulo incorporato nella libreria TensorFlow. TensorFlow, introdotta da Google nel 2015, è una libreria open source ampiamente adottata per lo sviluppo di algoritmi complessi di machine learning. Originariamente sviluppata come una libreria autonoma da François Chollet, Keras è stata integrata come API ufficiale all'interno di TensorFlow nel 2017, al fine di semplificare la creazione e l'addestramento di reti neurali. Quest'integrazione ha reso Keras uno strumento indispensabile per lo sviluppo di modelli di deep learning, offrendo un approccio semplice e intuitivo alla costruzione di reti neurali.

#### **3.3.1. Caratteristiche e vantaggi dell'algoritmo di Ottimizzazione ADAM**

Per quanto riguarda l'ottimizzazione durante il training dei modelli di rete neurale, il progetto ha adottato l'algoritmo **ADAM** [36] (Adaptive Moment Estimation). ADAM è una raffinazione dell'algoritmo di discesa del gradiente stocastico (SGD) e rappresenta un miglioramento significativo nella gestione del tasso di apprendimento [37]. A differenza di SGD, che utilizza un tasso di apprendimento fisso, ADAM regola dinamicamente il tasso di apprendimento per

ciascun parametro del modello durante l'addestramento, riducendo significativamente il rischio di rimanere bloccati in minimi locali non ottimali e accelerando la convergenza verso il minimo globale.

ADAM migliora l'efficacia del processo di addestramento regolando il tasso di apprendimento in base all'importanza relativa di ciascun coefficiente del modello. Questo si traduce in aggiustamenti più precisi dei pesi della rete, ottimizzando la capacità del modello di adattarsi efficacemente ai dati di addestramento senza soffrire di fluttuazioni eccessive. In sintesi, ADAM non solo accelera il processo di addestramento ma contribuisce anche a migliorare la stabilità e l'affidabilità della convergenza del modello, elementi cruciali per la realizzazione di reti neurali robuste e performanti.

Durante il processo di addestramento del modello di deep learning, la selezione accurata dell'algoritmo di ottimizzazione rappresenta solo una parte dell'equazione. Altri parametri critici devono essere configurati attentamente per garantire l'efficacia del training e l'ottimizzazione del modello. Questi includono la funzione di perdita, la metrica di valutazione delle prestazioni, la dimensione del batch e il numero di epoche di addestramento. Ognuno di questi parametri gioca un ruolo fondamentale nel modellare la capacità del modello di apprendere in modo efficiente e accurato.

### 3.3.2. La Funzione di perdita “Binary Cross-Entropy”

Nel contesto della classificazione binaria, la funzione di perdita utilizzata è la **Binary Cross-Entropy**. Questa funzione è ideale per problemi di classificazione in cui le etichette sono mutuamente esclusive. Misura la performance del modello producendo un valore di loss tra 0 e infinito e punisce le previsioni errate con una perdita significativamente alta, incentivando così il modello a fare previsioni più accurate.

### 3.3.3. Scelta della dimensione del batch: equilibrio tra precisione e velocità

Nel processo di addestramento delle reti neurali per questo progetto di tesi, è stata adottata una strategia specifica per la gestione dei batch di dati, fondamentale quando si lavora con dataset di grandi dimensioni. Invece di alimentare l'intero dataset alla rete in una sola volta, il set è stato suddiviso in gruppi più piccoli, o batch, che sono stati poi propagati attraverso la rete durante l'addestramento. Questa tecnica è essenziale per ottimizzare l'uso della memoria e accelerare il processo di addestramento, mantenendo al contempo la gestione dei dati gestibile ed efficiente.

Nel progetto, si è adottato diverse dimensioni di batch per ottimizzare l'apprendimento del modello sviluppato partendo da zero. Per favorire un apprendimento rapido nelle fasi iniziali, il modello ha iniziato con un batch di dimensioni maggiori, specificatamente 128. Man mano che il processo di addestramento avanzava e si rendeva necessario individuare miglioramenti più sottili e dettagliati, si è progressivamente ridotto la dimensione del batch, passando prima a 64 e successivamente a 32. Questo approccio ha permesso di adattare la velocità di apprendimento e l'efficienza del modello alle diverse fasi del suo sviluppo.

Una pratica comune per ottimizzare l'uso della memoria nella maggior parte delle piattaforme di computing è adottare una potenza di due come dimensione del batch, il numero di osservazioni del train set selezionate casualmente per calcolare il gradiente della funzione di perdita durante ogni iterazione dell'algoritmo di discesa. Un batch size maggiore aiuta a velocizzare il processo di training garantendo al contempo che l'algoritmo abbia abbastanza informazioni per fare aggiornamenti informati dei parametri del modello senza essere né troppo lento né troppo propenso a errori causati da varianze troppo ampie nei dati di un batch troppo piccolo. La scelta di queste dimensioni di batch ha permesso di bilanciare efficacemente la velocità di addestramento e l'accuratezza della stima del gradiente.

Dopo aver configurato l'architettura della rete e i batch con l'aiuto di Keras Tuner, che ha ottimizzato ulteriormente gli iperparametri, si è proceduto all'addestramento effettivo del modello.

### 3.3.4. Impiego di Keras Tuner nella Ricerca degli Iperparametri Ottimali

Nella sezione dedicata all'implementazione delle reti neurali convoluzionali (CNN) per questo progetto di tesi, si è fatto ricorso alla classe *“Sequential”* del modulo *“keras.models”*. La classe *“Sequential”* facilita la costruzione di modelli in maniera lineare, aggiungendo strati uno dopo l'altro in una sequenza ben definita, il che è particolarmente adatto per lo sviluppo di CNN dove ogni strato si basa sull'output del precedente.

Il processo di addestramento di una rete neurale, in particolare la selezione e l'ottimizzazione degli iper-parametri, gioca un ruolo cruciale nel determinare sia l'efficienza computazionale che la performance finale del modello. La scelta accurata di questi parametri influisce non solo sui tempi di addestramento ma anche sulla capacità del modello di generalizzare su dati non visti.

Per identificare la configurazione ottimale degli iper-parametri, è essenziale condurre una serie di test variando sistematicamente i parametri e osservando l'effetto sulle prestazioni del modello. Questo processo iterativo prevede la modifica di variabili come il numero di strati, la dimensione dei filtri, il numero di filtri per strato, il tasso di apprendimento e molti altri.

#### 3.3.4.1. Random Search e Grid Search con Keras Tuner

Keras Tuner è una libreria Python progettata per ottimizzare i modelli di deep learning di TensorFlow attraverso la ricerca automatizzata degli iperparametri più adatti. Questa libreria consente ai ricercatori e agli sviluppatori di definire lo spazio di ricerca degli iperparametri e di selezionare automaticamente la configurazione migliore basata sulle prestazioni del modello. Tra le strategie di ottimizzazione degli iperparametri più utilizzate in Keras Tuner ci sono il Random Search e il Grid Search.



Il **Random Search** testa una selezione casuale di valori per ciascun iperparametro. Questo metodo si rivela spesso più efficiente del Grid Search, particolarmente in contesti con uno spazio di ricerca ampio o quando certi iperparametri hanno un impatto minimo sulle prestazioni del modello. Il Random Search esplora una vasta gamma di combinazioni e può identificare rapidamente configurazioni efficaci, anche se non garantisce l'eshaustività della ricerca o la scoperta del set ottimale di iperparametri.

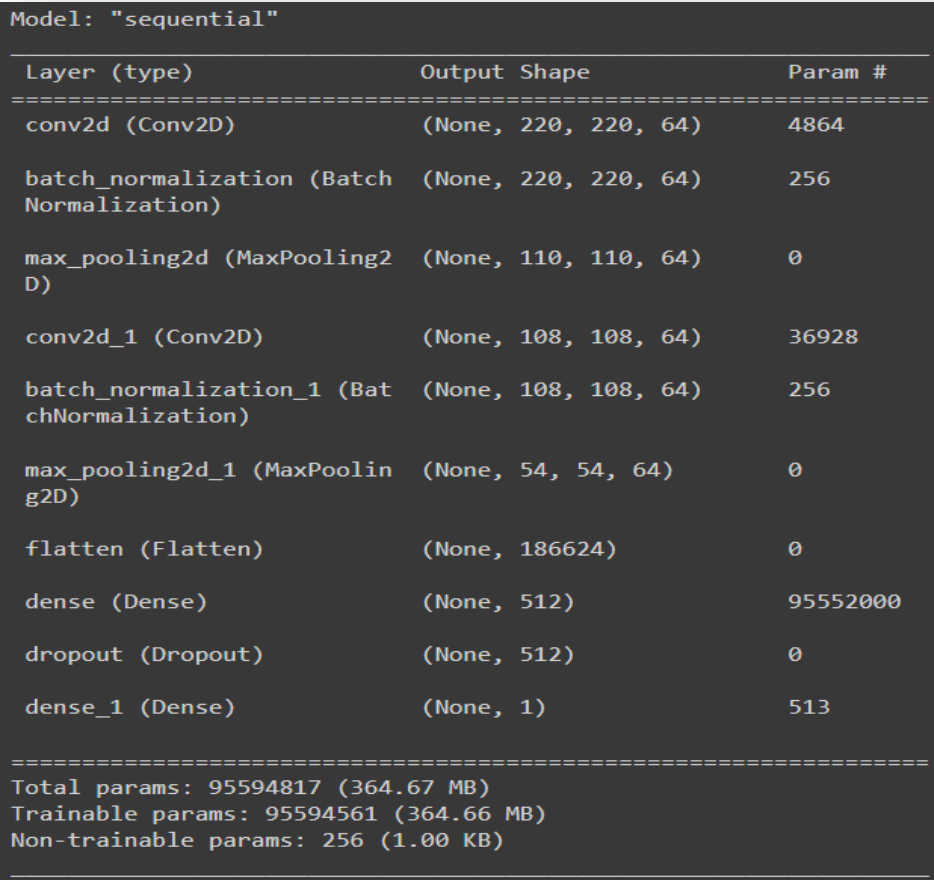
Il **Grid Search**, invece, procede con un'esplorazione sistematica di tutte le combinazioni possibili di valori degli iperparametri definiti in una griglia preimpostata. Questo metodo assicura che la migliore combinazione possibile tra quelle specificate sarà identificata, ma può risultare molto lento e richiedere un notevole dispendio di risorse computazionali, specialmente in presenza di un ampio spazio di iperparametri o di valori variabili.

In termini di efficienza, il Random Search tende ad essere più efficace in grandi spazi di ricerca perché non richiede la valutazione di tutte le combinazioni possibili. Tuttavia, il Grid Search offre una completa verifica di tutte le combinazioni entro i limiti specificati, garantendo così la scoperta della migliore configurazione possibile. Da un punto di vista pratico, il Random Search è generalmente più adatto per la maggior parte delle applicazioni di deep learning, data la sua velocità e la sua capacità di trovare buone soluzioni anche in spazi ampi di iperparametri.

La scelta tra Random Search e Grid Search dipenderà dalle specifiche esigenze del progetto, incluse le dimensioni dello spazio di ricerca, l'importanza di identificare la configurazione ottimale e le risorse computazionali a disposizione.

### 3.3.4.2. Risultati della Selezione degli Iperparametri mediante Keras Tuner

Dopo un'ampia fase di sperimentazione, sono stati selezionati gli iper-parametri illustrati nella Figura 3.4, che mostra la configurazione del modello del progetto. La progettazione ha coinvolto l'impiego di strati convoluzionali e di pooling, alternati da tecniche di regolarizzazione come la batch normalization [38] e il dropout [39] per migliorare la generalizzazione e prevenire l'overfitting.



```
Model: "sequential"
Layer (type)                Output Shape                Param #
=====
conv2d (Conv2D)              (None, 220, 220, 64)       4864
batch_normalization (Batch Normalization) (None, 220, 220, 64)       256
max_pooling2d (MaxPooling2D) (None, 110, 110, 64)       0
conv2d_1 (Conv2D)            (None, 108, 108, 64)       36928
batch_normalization_1 (Batch Normalization) (None, 108, 108, 64)       256
max_pooling2d_1 (MaxPooling2D) (None, 54, 54, 64)       0
flatten (Flatten)            (None, 186624)             0
dense (Dense)                 (None, 512)                95552000
dropout (Dropout)            (None, 512)                0
dense_1 (Dense)              (None, 1)                  513
=====
Total params: 95594817 (364.67 MB)
Trainable params: 95594561 (364.66 MB)
Non-trainable params: 256 (1.00 KB)
```

Figura 3.4 Configurazione del modello del progetto

Per affinare ulteriormente la configurazione del modello, è stato impiegato Keras Tuner. Questo processo ha incluso l'analisi di diverse combinazioni di iperparametri cruciali, come il kernel size, il kernel\_regularizer e il numero di filtri per ogni strato convoluzionale. Durante le varie iterazioni, combinando

Random Search e Grid search, sono state testate diverse configurazioni, scartando quelle che producevano i risultati peggiori in termini di accuratezza, come mostrato nella Figura 3.5.

2	3
<pre> "values": {   "conv_1_filter": 128,   "conv_1_kernel": 3,   "conv_1_l2": 1.4978936438265577e-05,   "conv_2_filter": 64,   "conv_2_kernel": 5,   "conv_2_l2": 3.1491614102652946e-05,   "dense_units": 512,   "dense_l2": 3.90962615769036e-05,   "learning_rate": 0.0001   "loss": {     0.5041153430938721   "accuracy": {     0.7721633911132812   "val_loss": {     0.43309760093688965   "val_accuracy": {     0.8393632173538208   "lr": {     9.999999747378752e-05   "score": 0.8393632173538208,   "best_step": 3, </pre>	<pre> "values": {   "conv_1_filter": 64,   "conv_1_kernel": 5,   "conv_1_l2": 8.32209105099796e-05,   "conv_2_filter": 64,   "conv_2_kernel": 3,   "conv_2_l2": 3.164057177704157e-05,   "dense_units": 512,   "dense_l2": 5.012702030057976e-05,   "learning_rate": 0.0001   "loss": {     0.4497268497943878   "accuracy": {     0.8205631971359253   "val_loss": {     0.4053977131843567   "val_accuracy": {     0.8658948540687561   "lr": {     9.999999747378752e-05   "score": 0.8658948540687561,   "best_step": 3, </pre>
7	8
<pre> "values": {   "conv_1_filter": 128,   "conv_1_kernel": 5,   "conv_1_l2": 7.360816470781754e-05,   "conv_2_filter": 32,   "conv_2_kernel": 5,   "conv_2_l2": 3.928633850073882e-05,   "dense_units": 128,   "dense_l2": 2.5115802886377302e-05,   "learning_rate": 0.0001   "loss": {     0.47224873304367065   "accuracy": {     0.7709022760391235   "val_loss": {     0.41697385907173157   "val_accuracy": {     0.8218041658401489   "lr": {     9.999999747378752e-05   "score": 0.8218041658401489,   "best_step": 3, </pre>	<pre> "values": {   "conv_1_filter": 64,   "conv_1_kernel": 3,   "conv_1_l2": 1.445235436956808e-05,   "conv_2_filter": 32,   "conv_2_kernel": 5,   "conv_2_l2": 1.850579979754401e-05,   "dense_units": 512,   "dense_l2": 3.424437941990688e-05,   "learning_rate": 0.0001   "loss": {     0.44497185945510864   "accuracy": {     0.8108667135238647   "val_loss": {     0.3861723244190216   "val_accuracy": {     0.8597202301025391   "lr": {     9.999999747378752e-05   "score": 0.8597202301025391,   "best_step": 3, </pre>

Figura 3.5 Confronto di alcune delle combinazioni di iperparametri trovate

Il primo strato convoluzionale del modello è specificato con la dimensione dei dati in input, che nel caso delle immagini a colori processate è di 224x224 pixel, con 3 canali corrispondenti ai colori RGB. Il parametro input\_shape del primo strato convoluzionale è quindi impostato su (224, 224, 3).

Il modello ottimizzato si distingue per la sua architettura particolarmente efficace. Inizia con uno strato convoluzionale che utilizza la classe Conv2D di Keras, contenente 64 filtri di dimensioni 5x5, ciascuno con la funzione di

attivazione ReLU. Questo strato è seguito immediatamente da uno strato di pooling che esegue il max pooling su blocchetti 2x2 dei dati in uscita dallo strato convoluzionale. Questo riduce la dimensionalità spaziale dell'output, preservando le caratteristiche più salienti. Tra lo strato convoluzionale e quello di pooling è inserito uno strato di batch normalization, che, come già accennato precedentemente, normalizza gli output del precedente strato convoluzionale per migliorare la stabilità e l'efficienza dell'addestramento.

Il modello replica la configurazione del primo blocco, ma con filtri di dimensione ridotta a 3x3, consentendo di focalizzarsi su dettagli più fini nelle immagini. Dopo questi due blocchi convoluzionali e di pooling, un layer Flatten trasforma l'output multidimensionale dei blocchi precedenti in un vettore unidimensionale, che alimenta un denso strato di 512 neuroni. Questo strato serve come collegamento critico per l'integrazione delle caratteristiche apprese in previsioni complesse. Durante l'addestramento, lo strato di dropout disattiva casualmente alcuni neuroni della rete.

Questa tecnica di regolarizzazione, come già discusso durante il primo capitolo, impedisce al modello di affidarsi eccessivamente a qualsiasi neurone singolo, promuovendo la robustezza e riducendo il rischio di overfitting. Infine, un ultimo strato completamente connesso, contenente un singolo neurone con una funzione di attivazione sigmoid, produce la previsione finale di classificazione binaria del modello.

La **regolarizzazione L2** e il **Dropout** sono state tecniche fondamentali nel modellare la robustezza della rete. La regolarizzazione L2 penalizza i pesi più grandi nella funzione di perdita del modello, spingendo il modello a preferire pesi più piccoli e distribuendo così l'apprendimento su molteplici caratteristiche piuttosto che lasciare che pochi pesi dominino il processo decisionale. Il Dropout funziona "ignorando" casualmente alcuni neuroni durante l'allenamento, impostando a zero l'output di alcuni neuroni selezionati casualmente. Questo impedisce alla rete di diventare eccessivamente dipendente da qualsiasi neurone o percorso specifico, costringendo il modello a imparare percorsi ridondanti per le informazioni e, di conseguenza, a creare una rete più robusta che generalizza

meglio su nuovi dati.

La **funzione di perdita** utilizzata è la “**Binary cross-entropy**”, ideale per compiti di classificazione binaria dove le classi sono mutuamente esclusive. La metrica principale per valutare la performance del modello è l'accuratezza, che misura la percentuale di etichette correttamente predette rispetto al totale. Il tasso di apprendimento iniziale è stato impostato a 0,001, riducendolo gradualmente per ottimizzare la convergenza e minimizzare le oscillazioni.

### 3.3.5. L'Impatto delle Callbacks nella Gestione dell'Addestramento dei Modelli

Durante l'addestramento, si è implementato tre callbacks fondamentali per migliorare l'efficacia del processo e prevenire l'overfitting: **ReduceLROnPlateau**, **EarlyStopping** e **ModelCheckpoint**.

#### 3.3.5.1. ReduceLROnPlateau

La callback **ReduceLROnPlateau** di Keras è uno strumento avanzato per ottimizzare il processo di addestramento dei modelli di machine learning, specialmente quando si affrontano sfide legate a plateau nella riduzione della loss o al miglioramento della performance del modello. Questa callback monitora una metrica specifica, generalmente la loss di validazione, e se non osserva miglioramenti per un numero predefinito di epoche, procede a ridurre il tasso di apprendimento (learning rate).

L'uso di **ReduceLROnPlateau** è particolarmente utile in scenari dove i modelli di deep learning tendono a beneficiare di un tasso di apprendimento elevato inizialmente, che permette rapidi progressi e una convergenza veloce verso un minimo locale. Tuttavia, man mano che l'addestramento procede, un tasso di apprendimento troppo elevato può causare instabilità e rendere difficile per il modello convergere verso il minimo globale della funzione di perdita. Qui entra in gioco **ReduceLROnPlateau**, che, rilevando la mancanza di progressi nella riduzione della loss, automaticamente abbassa il learning rate, permettendo al

modello di fare passi più piccoli e più precisi verso il minimo (Figura 3.6).

```
val_loss: 0.1385 - val_accuracy: 0.9931 - lr: 7.2900e-08  
val_loss: 0.1384 - val_accuracy: 0.9931 - lr: 7.2900e-08  
val_loss: 0.1384 - val_accuracy: 0.9931 - lr: 7.2900e-08  
val_loss: 0.1384 - val_accuracy: 0.9931 - lr: 2.1870e-08  
val_loss: 0.1384 - val_accuracy: 0.9931 - lr: 6.5610e-09
```

*Figura 3.6 Esempio di riduzione del Learning Rate nel progetto*

Il funzionamento della callback è regolato da parametri configurabili, tra cui `monitor`, che specifica la metrica da monitorare, `factor`, che determina di quanto ridurre il learning rate, `patience`, il numero di epoche da attendere dopo l'ultimo miglioramento prima di ridurre il learning rate, e `min_lr`, che stabilisce un limite inferiore al quale il learning rate non può scendere. Questa flessibilità permette agli sviluppatori di adattare il comportamento della callback alle specifiche esigenze del loro modello e dataset.

In conclusione, `ReduceLROnPlateau` è un'aggiunta preziosa alla suite di strumenti di Keras per il tuning dei modelli di machine learning, offrendo un metodo pratico ed efficace per migliorare la convergenza dei modelli e ottimizzare le loro performance, specialmente in fasi avanzate dell'addestramento dove ogni piccolo aggiustamento può fare la differenza in termini di risultati finali.

### 3.3.5.2. Early Stopping e Numero di Epoche: Bilanciare Apprendimento e Overfitting

Il numero di epoche di addestramento è una misura di quante volte l'intero set di dati viene processato dall'algoritmo. Un'epoca è completa quando ogni osservazione del set di addestramento è stata utilizzata una volta per l'aggiornamento dei parametri del modello. Determinare il numero giusto di epoche è cruciale: un numero eccessivamente alto di epoche può causare overfitting, dove il modello si adatta troppo specificamente ai dati di addestramento, perdendo la capacità di generalizzare su dati nuovi o non visti. Al contrario, un numero troppo basso di epoche può risultare in un modello sottoaddestrato, che non ha appreso sufficientemente dai dati di addestramento per fare previsioni accurate.

La tecnica di **EarlyStopping**, d'altra parte, monitora le prestazioni del modello su un set di validazione e interrompe l'addestramento non appena tali prestazioni cessano di migliorare, evitando così un inutile consumo di risorse computazionali e prevenendo l'overfitting. Per questo progetto, si è scelto di allenare le reti fino a quando non si è verificato l'intervento dell'EarlyStopping (Figura 3.7), l'addestramento, quindi, ha terminato la sua esecuzione quando non si sono verificati miglioramenti sul set di validazione per 3 epoche consecutive (**patience=3**).

```
Epoch 23/10000
756/756 [=====] - 1641s 2s/step - loss: 0.1377 - accuracy: 0.9940
Epoch 24/10000
756/756 [=====] - 1645s 2s/step - loss: 0.1383 - accuracy: 0.9937
Epoch 25/10000
756/756 [=====] - 1652s 2s/step - loss: 0.1367 - accuracy: 0.9940
Epoch 26/10000
756/756 [=====] - 1643s 2s/step - loss: 0.1371 - accuracy: 0.9944
Epoch 27/10000
756/756 [=====] - 1647s 2s/step - loss: 0.1370 - accuracy: 0.9942
Epoch 27: early stopping
162/162 [=====] - 71s 438ms/step - loss: 0.1377 - accuracy: 0.9924
162/162 [=====] - 70s 431ms/step
```

*Figura 3.7 Esempio di EarlyStopping nel progetto*

### 3.3.5.3. ModelCheckpoint: Protezione e Ripristino dei Modelli in Google Colab

Utilizzare Google Colab per addestrare modelli di deep learning presenta vantaggi significativi grazie alla sua accessibilità e potenza computazionale, inclusa la possibilità di utilizzare le TPU gratuitamente. Tuttavia, una delle sfide più comuni durante l'utilizzo di Colab è la gestione delle sessioni, che possono disconnettersi dopo un periodo di inattività o superamento del limite di tempo di utilizzo. Questo comporta il rischio di perdere i progressi dell'addestramento se i dati non sono stati salvati adeguatamente. Per mitigare questo rischio, è essenziale impiegare strategie di salvataggio dei progressi, come l'uso della callback **ModelCheckpoint** di Keras per salvare automaticamente i modelli durante l'addestramento.

La callback **ModelCheckpoint** in Keras è uno strumento estremamente potente e versatile usato per salvare il modello o i pesi del modello a intervalli specifici. Questo può essere particolarmente utile in molti scenari, come durante addestramenti prolungati o quando si svolgono esperimenti su configurazioni di modelli che richiedono un lungo tempo per essere addestrati. **ModelCheckpoint** assicura che i progressi non vengano persi e permette di riprendere l'addestramento da un punto specifico in caso di interruzione involontaria.

Offre funzionalità avanzate per la gestione efficace del salvataggio dei modelli durante l'addestramento. Una delle opzioni più utili è il salvataggio condizionale, che permette di salvare il modello solo quando si verifica un miglioramento in termini di performance su una metrica specificata, come la accuracy di validazione. Questo approccio aiuta a conservare solo la versione migliore del modello, ottimizzando lo spazio di archiviazione e la gestione delle risorse.

In aggiunta, **ModelCheckpoint** permette di configurare la frequenza di salvataggio del modello, offrendo la possibilità di salvare ad ogni epoca o dopo un numero definito di epoche. Questa flessibilità è particolarmente utile per adattarsi alle diverse esigenze del training e alla durata delle epoche.

Per quanto riguarda le opzioni di salvataggio, gli utenti possono scegliere di salvare l'intero modello, inclusa la sua architettura, o solo i pesi del modello.



Salvare solo i pesi può essere vantaggioso se l'architettura del modello rimane invariata, permettendo di ridurre i requisiti di spazio di archiviazione.

Un'ulteriore strategia per gestire il salvataggio dei modelli, specialmente utile quando si utilizza Colab, è integrare la funzione ModelCheckpoint con **Google Drive**. Montando Google Drive nell'ambiente Colab, è possibile impostare ModelCheckpoint per scrivere i file direttamente nella propria area di storage su Drive. Questo metodo non solo protegge i dati salvati, ma facilita anche la ripresa dell'addestramento da dove era stato interrotto in caso di disconnessioni.

In particolare, se si verifica una disconnessione durante l'uso di Colab, è possibile ricaricare l'ultimo checkpoint salvato da Google Drive e continuare l'addestramento senza perdere progressi significativi. Questo è particolarmente rilevante quando si utilizzano le TPU in Colab, dato che le sessioni possono essere soggette a disconnessioni frequenti, specialmente con un utilizzo prolungato. Implementare una strategia robusta di checkpoint è quindi essenziale per assicurare che ogni batch di addestramento contribuisca al progresso complessivo del modello, riducendo il lavoro duplicato e aumentando l'efficienza computazionale.

### 3.4. Risultati e Considerazioni Finali

#### 3.4.1. Metriche di valutazione Scikit-Learn

Nel contesto del machine learning, valutare l'efficacia di un modello di classificazione richiede l'impiego di metriche precise. Iniziando con la precisione, questa metrica esprime la proporzione di identificazioni corrette tra le previsioni positive, essenzialmente calcolata come il numero di veri positivi diviso la somma dei veri positivi e dei falsi positivi. Questa misura assume un ruolo critico quando è importante minimizzare l'impatto dei falsi positivi.

$$Precision = \frac{Veri\ positivi}{Veri\ positivi + Falsi\ positivi}$$

Passando al recall, o sensibilità, questa misura calcola quanto efficacemente il modello è in grado di identificare tutti i casi positivi all'interno di un dataset, definita come il rapporto tra i veri positivi e la somma di veri positivi e falsi negativi.

$$Recall = \frac{Veri\ positivi}{Veri\ positivi + Falsi\ negativi}$$

L'indice F1 armonizza precisione e recall in un singolo punteggio, attraverso la media armonica di entrambe le metriche, risultando utile quando è necessario un equilibrio tra il riconoscimento completo dei positivi e una bassa incidenza di falsi positivi.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

L'accuratezza, d'altra parte, offre una visione d'insieme dell'efficacia del modello, calcolando la proporzione totale delle previsioni corrette (sia positive che negative) rispetto all'intero insieme di osservazioni.

$$Accuratezza = \frac{Veri\ positivi + Veri\ Negativi}{Elementi\ totali}$$

Per quanto riguarda l'analisi delle prestazioni attraverso diverse classi, la media macro e la media ponderata forniscono approcci differenziati. La media macro calcola la media aritmetica delle metriche come precisione, recall e F1 per ogni classe, considerando tutte le classi ugualmente senza tener conto della loro

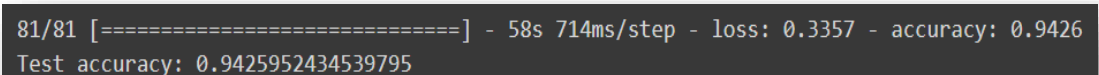
distribuzione numerica nel dataset.

Al contrario, la media ponderata tiene conto delle dimensioni relative delle classi, calcolando le metriche per ciascuna classe proporzionalmente al numero di casi veri per quella classe, il che aiuta a riflettere l'importanza numerica di ciascuna classe nel modello generale.

Queste metriche sono integralmente supportate da *scikit-learn*, una libreria di Python dedicata al machine learning. *Scikit-learn* facilita il calcolo di queste metriche attraverso funzioni come *classification\_report* permettendo di derivare direttamente le prestazioni del modello dai dati di test e dalle previsioni generate.

### 3.4.2. Analisi dei Risultati

Nel progetto, si sono osservati miglioramenti significativi nelle prestazioni del modello man mano che venivano le dimensioni del batch durante l'addestramento. Inizialmente, utilizzando un batch size di 128, si è raggiunta un'accuratezza del 94% (Figura 3.8). Questo risultato era già promettente, ma si è continuato a sperimentare con le dimensioni del batch per cercare di migliorare ulteriormente le prestazioni.



```
81/81 [=====] - 58s 714ms/step - loss: 0.3357 - accuracy: 0.9426
Test accuracy: 0.9425952434539795
```

*Figura 3.8 Testing del modello del progetto*

Abbassando la dimensione del batch a 64, si è verificato un notevole incremento dell'accuratezza, che è salita al 99,3% (Figura 3.9). Questo aumento significativo può essere attribuito a una migliore stima del gradiente durante il processo di apprendimento, dato che batch più piccoli possono offrire una convergenza più rapida e accurata in alcuni contesti.

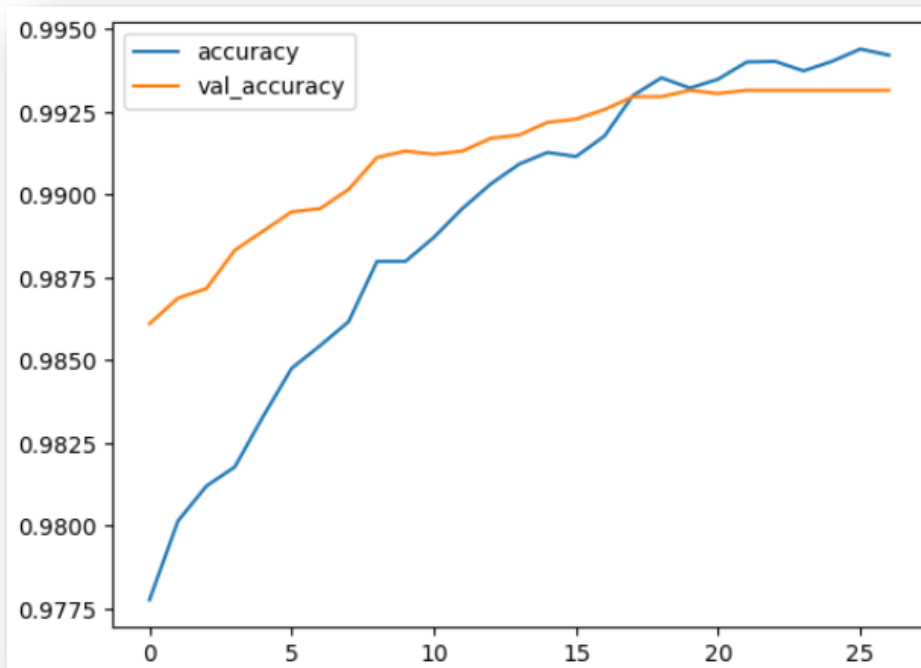


Figura 3.9 Grafico accuracy (train-validation)

Non fermi ai successi precedenti, abbiamo ulteriormente ridotto la dimensione del batch a 32. Questo adattamento ha portato a una performance ancora migliore, con un'accuratezza che ha raggiunto il 99,7% (Figura 3.10). Questo livello di accuratezza dimostra l'efficacia di utilizzare batch più piccoli per perfezionare questo specifico modello e set di dati, ottimizzando l'aggiornamento dei pesi e la precisione dell'algoritmo di apprendimento.

	precision	recall	f1-score	support
no_watermark	0.9978	0.9975	0.9976	6350
watermark	0.9960	0.9965	0.9963	4015
accuracy			0.9971	10365
macro avg	0.9969	0.9970	0.9970	10365
weighted avg	0.9971	0.9971	0.9971	10365

Figura 3.10 Classification-report finale

### 3.4.3. Caratteristiche e analisi della matrice di confusione

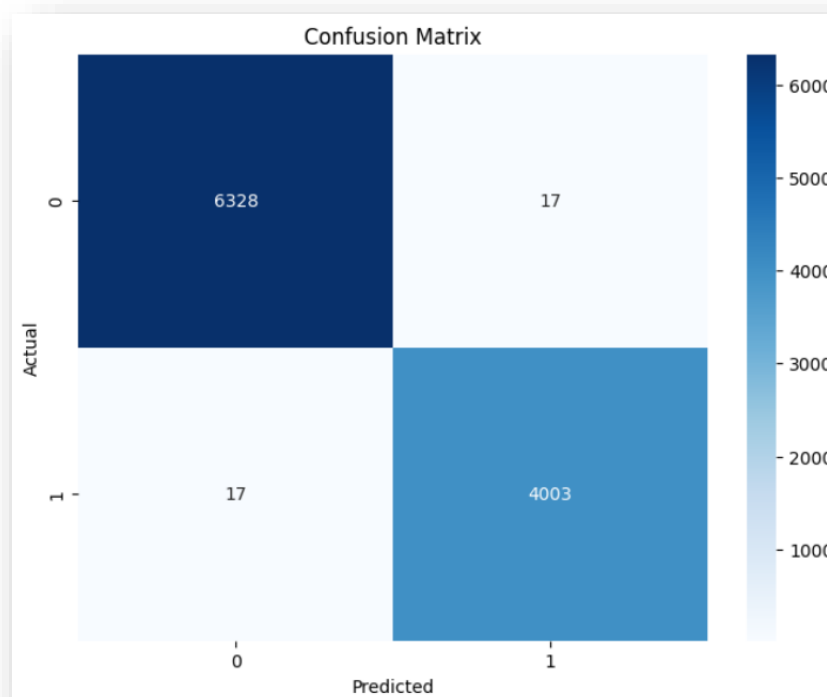
La matrice di confusione è uno strumento fondamentale per analizzare le prestazioni dei modelli di classificazione. Questa matrice, nota anche come matrice di errore, offre una visualizzazione chiara e diretta delle performance di un modello attraverso le classificazioni corrette e incorrette che esso effettua, rendendola particolarmente utile sia in contesti di classificazione binaria che multiclasse.

Composta da righe e colonne che rappresentano rispettivamente le etichette osservate e le previsioni fatte dal modello, la matrice include termini quali veri positivi, dove il modello ha correttamente predetto la classe positiva, e falsi positivi, dove il modello ha erroneamente predetto la stessa. Allo stesso modo, si annotano i veri negativi e i falsi negativi per rappresentare le accuratezze e gli errori nelle previsioni della classe negativa.

L'utilizzo della matrice di confusione si rivela vantaggioso per vari motivi. Offre una chiara visualizzazione delle aree di efficacia e di errore del modello, costituisce la base per calcolare altre importanti metriche di performance come la precisione, la recall e l'indice F1, e si dimostra particolarmente utile in situazioni dove le classi di output sono sbilanciate.

In sostanza, la matrice di confusione aiuta a identificare se un modello manifesta confusione in specifiche tipologie di classificazioni, fornendo indicazioni preziose per miglioramenti nel processo di training.

L'analisi della matrice di confusione ha rivelato un'omogeneità notevole (Figura 3.11) nel modo in cui il modello ha gestito le classificazioni, mostrando che non vi sono state discrepanze significative tra le previsioni corrette e quelle errate. Questo indica che il modello ha mantenuto una coerenza elevata nelle sue previsioni attraverso diverse classi, riuscendo a mantenere un equilibrio tra veri positivi e veri negativi e limitando sia i falsi positivi che i falsi negativi.



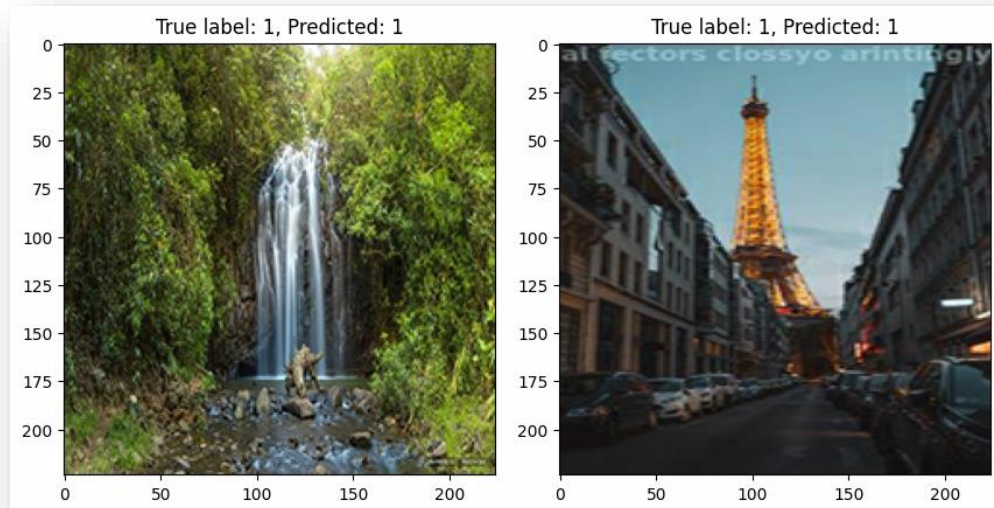
*Figura 3.11 Confusion matrix*

Questa uniformità nelle prestazioni del modello suggerisce che il processo di addestramento e le scelte metodologiche impiegate hanno efficacemente minimizzato il rischio di bias verso una particolare classe o di overfitting su specifiche caratteristiche dei dati, soprattutto grazie alla notevole mole di immagini nel dataset di train. Tale risultato è indicativo di un modello ben calibrato e robusto, che si traduce in previsioni affidabili e applicabili in modo trasversale a tutti i segmenti del dataset.

L'assenza di discrepanze significative nella matrice di confusione è particolarmente rassicurante in contesti dove la precisione e la recall equilibrate sono cruciali, come nelle applicazioni mediche o finanziarie, dove gli errori di classificazione possono avere conseguenze gravi. Inoltre, questo scenario fornisce una solida base per la fiducia nell'impiego del modello in situazioni reali, promettendo prestazioni stabili anche quando viene introdotto a nuovi dati o contesti di uso.

Di seguito vengono mostrate una serie di immagini classificate correttamente dal modello.

### Watermarked (Figura 3.12)



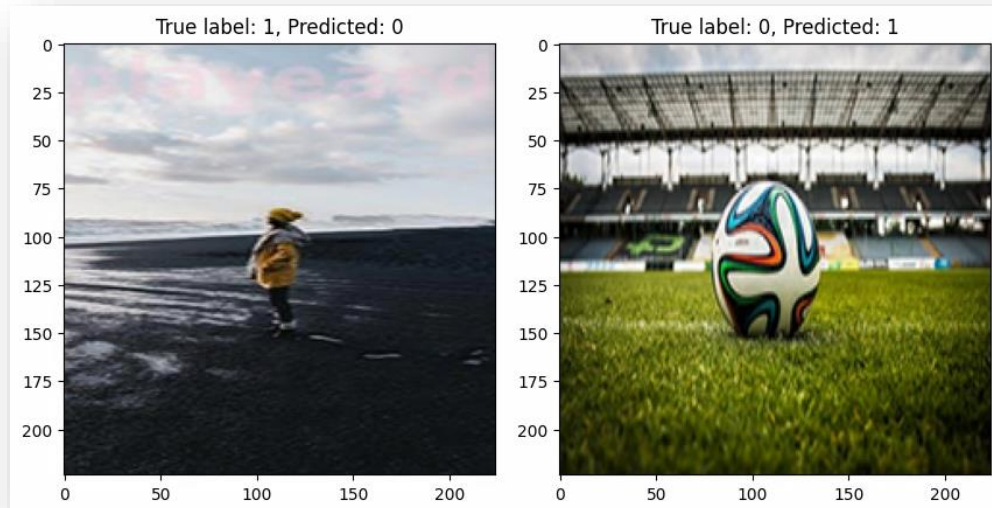
*Figure 3.12 Immagini watermarked identificate*

### Not watermarked (Figura 3.13)



*Figure 3.13 Immagini non watermarked identificate*

L'analisi delle immagini che il modello ha classificato in modo errato (Figura 3.14) rivela aspetti interessanti e cruciale per l'ottimizzazione del processo di machine learning.



*Figura 3.14 Esempio di possibile predizione errata*

In molti casi, si scopre che queste immagini presentano caratteristiche tali per cui nemmeno l'occhio umano sarebbe riuscito a classificarle correttamente. Questo può essere dovuto a vari fattori, come una qualità dell'immagine particolarmente bassa, una composizione visiva ambigua o elementi di distrazione che confondono sia l'osservatore umano che il modello. Altre volte, le immagini errate sono il risultato di una cattiva etichettatura iniziale, dove l'etichetta fornita non corrisponde effettivamente al contenuto visivo dell'immagine. Questo problema di etichettatura può portare a un addestramento inadeguato del modello, che 'impara' da informazioni errate, influenzando negativamente la sua capacità di generalizzare da esempi correttamente etichettati. Identificare queste anomalie è fondamentale, poiché fornisce indicazioni preziose su come migliorare la raccolta dati, la preparazione del dataset e le tecniche di pre-elaborazione per migliorare le prestazioni future del modello.



## 4. Conclusioni

La ricerca presentata in questa tesi ha dimostrato l'efficacia nell'identificazione di watermark digitali di un modello di rete neurale convoluzionale (CNN). Il modello in questione, addestrato da zero, ha raggiunto una notevole accuratezza, il che conferma non solo la validità dell'approccio adottato ma apre anche la strada a interessanti sviluppi futuri. Primo fra tutti, l'introduzione del transfer learning potrebbe ulteriormente affinare la prestazione del modello. Il fine tuning di un modello convoluzionale pre-addestrato su larga scala potrebbe in effetti migliorare le capacità di generalizzazione su nuovi dataset.

Focalizzandosi invece sui possibili scenari applicativi del modello realizzato nel presente lavoro di tesi, esso potrebbe essere integrato in piattaforme social come Pinterest, Instagram, Telegram o Facebook, all'interno di meccanismi di protezioni dei diritti d'immagine. In particolare, rilevando il caricamento di un'immagine watermarked, il sistema potrebbe richiedere all'utente di dimostrare la proprietà dell'immagine, prevenendo così la diffusione non autorizzata di contenuti online. Inoltre, tale modello potrebbe aiutare le aziende a proteggere il proprio brand da danni d'immagine, consentendo l'identificazione e la segnalazione delle immagini del brand che sono state diffuse senza autorizzazione, e permettendo dunque azioni tempestive per la loro rimozione. In conclusione, il sistema di identificazione di watermark sviluppato nel corso di questa tesi rappresenta un significativo passo avanti nella lotta contro la violazione dei diritti digitali e l'abusivo uso di immagini protette. Esso non solo evidenzia il potenziale dell'intelligenza artificiale nell'ambito della sicurezza digitale ma sottolinea anche l'importanza di continuare a esplorare e sviluppare tecnologie che possono essere facilmente integrate in contesti quotidiani.

Per una visione completa e dettagliata del progetto, incluso il codice sorgente, si invita a visitare la repository disponibile su GitHub<sup>1</sup>, dove è possibile esaminare, scaricare e contribuire allo sviluppo del progetto stesso.

---

<sup>1</sup> [https://github.com/AlessandroPata/CNN\\_Watermark\\_Detection](https://github.com/AlessandroPata/CNN_Watermark_Detection)

## 5. Bibliografia

- [1] R. Bellman, *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, 1978.
- [2] D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*, Jan. 1998.
- [3] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, Wiley, New York, Jun. 1949.
- [4] A. M. Turing, "I.—computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433-460, Oct. 1950.
- [5] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the Dartmouth summer research project on artificial intelligence," *AI Magazine*, vol. 27, p. 12 ss, 2006.
- [6] J. Weizenbaum, "Eliza—a computer program for the study of natural language communication between man and machine," 1966.
- [7] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386 ss, 1958.
- [8] H. J. Kelley, "Gradient theory of optimal flight paths," *ARS Journal*, vol. 30, pp. 947-954, 1960.
- [9] S. Dreyfus, "The numerical solution of variational problems," *Journal of Mathematical Analysis and Applications*, vol. 5, no. 1, pp. 30-45, 1962.
- [10] P. L. Frana and M. J. Klein, *Encyclopedia of Artificial Intelligence*, ABC-CLIO, California, 2021, p. 144.
- [11] D. W. Patterson, *Introduction to Artificial Intelligence and Expert Systems*, PHI Learning, New Delhi, 2008, p. 327.
- [12] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115 ss, 1943.
- [13] S. Papert and M. Minsky, *A Review of Perceptrons: An Introduction to Computational Geometry*, The M.I.T. Press, Cambridge, Mass, 1969.
- [14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 ss, 1998.
- [15] G. E. Hinton, A. Krizhevsky, and I. Sutskever, "Imagenet classification with deep convolutional neural networks," 2012.

- [16] N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Aspell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, T. B. Brown, and B. Mann, "Language models are few-shot learners," OpenAI, 2020.
- [17] D. C.-E. Lin, "8 Simple Techniques to Prevent Overfitting," 2020. [Online]. Available: <https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d>.
- [18] L. Perez and J. Wang, "The Effectiveness of Data Augmentation in Image Classification using Deep Learning," ArXiv e-prints, Dec. 2017. [Online]. Available: arXiv:1712.04621 [cs.CV].
- [19] "Anatomia dei Neuroni," Ask A Biologist, Arizona State University. [Online]. Disponibile: <https://askabiologist.asu.edu/italian/anatomia-dei-neuroni>.
- [20] D. Soriano, "Come è fatta una rete neurale (terza parte)," Domsoria, Apr. 2018. [Online]. Disponibile: <https://www.domsoria.com/2018/04/come-e-fatta-una-rete-neurale-terza-parte/>.
- [21] A. F. Agarap, "Deep learning using rectified linear units (relu)," arXiv preprint arXiv:1803.08375, 2018.
- [22] O. Dreessen, "Reti neurali convoluzionali: cos'è il machine learning," Elettronica News, *Online*. Disponibile: <https://www.elettronicanews.it/reti-neurali-convoluzionali-cos-e-il-machine-learning/>.
- [23] C. Casadei, "Le reti neurali ricorrenti," Developers Maggioli, *Online*. Disponibile: <https://www.developersmaggioli.it/blog/le-reti-neurali-ricorrenti/>.
- [24] G. Nardini, "Rete neurale convolutive: cosa è," AI4Business, *Online*. Disponibile: <https://www.ai4business.it/intelligenza-artificiale/rete-neurale-convolutive-cosa-e/>.
- [25] S. Dobilas, "Convolutional Neural Networks Explained: How to Successfully Classify Images in Python," Towards Data Science. *Online*. Disponibile: <https://towardsdatascience.com/convolutional-neural-networks-explained-how-to-successfully-classify-images-in-python-df829d4ba761>.
- [26] D. Unzueta, "Fully Connected Layer," Built In. *Online*. Disponibile: <https://builtin.com/machine-learning/fully-connected-layer>.
- [27] M. M. Yeung, "Digital Watermarking," *Communications of the ACM*, vol. 41, no. 7, 1998.
- [28] F. Petitcolas, *Steganography and Digital Watermarking*. The Information Hiding, 1996. [Online]. Available: <http://www.petitcolas.net/fabien/steganography/>.

- [29] J. Wilkins, *Mercury: Or the Secret and Swift Messenger: Shewing, How a Man May with Privacy and Speed Communicate His Thoughts to a Friend at Any Distance*, 2nd ed., Rich Baldwin, London, U.K., 1694.
- [30] R. J. Anderson, Ed., *IH'96: Proceedings of the 1st International Workshop on Information Hiding*, vol. 1174. Berlin, Germany: Springer-Verlag, 1996.
- [31] C. Y. Lin and S. F. Chang, "Semifragile watermarking for authenticating JPEG visual content," in *Electronic Imaging*, 2000, pp. 140-151.
- [32] Cultur-e, "Come applicare il watermark alle foto," Fastweb, *Online*. Disponibile: <https://www.fastweb.it/fastweb-plus/digital-magazine/come-applicare-il-watermark-alle-foto/>.
- [33] R. Dalsaniya, "The Watermark Challenge," Kaggle, 2023. [Online]. Disponibile: <https://www.kaggle.com/datasets/rajdalsaniya/the-watermark-challenge>.
- [34] "Scenery Watermark Detection," Kaggle, 2023. [Online]. Disponibile: <https://www.kaggle.com/datasets/qwertyforce/scenery-watermarks>.
- [35] F. Pollano, in Watermarked not Watermarked Images, Kaggle, [Data di pubblicazione]. [Online]. Disponibile: <https://www.kaggle.com/datasets/felicepollano/watermarked-not-watermarked-images/discussion/372069>.
- [36] V. Bushaev, "Adam — latest trends in deep learning optimization," 2018. [Online]. Available: <https://towardsdatascience.com/adam-latest-trends-in-deeplearning-optimization-6be9a291375c>.
- [37] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
- [38] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in International Conference on Machine Learning, 2015, pp. 448-456. [Online]. Available: <http://jmlr.org/proceedings/papers/v37/ioffe15.html>.
- [39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014. [Online]. Available: <http://jmlr.org/papers/v15/srivastava14a.html>.