CHAPTER I – Methodology Application Classifier

I.1 Introduzione al Machine Learning e all'Intelligenza Artificiale

L'intelligenza artificiale (IA) è un ramo dell'informatica che si propone di progettare e sviluppare sistemi in grado di simulare comportamenti intelligenti, ossia capaci di percepire l'ambiente, apprendere, ragionare e prendere decisioni. Il concetto di "intelligenza" nelle macchine è stato introdotto formalmente da Alan Turing nel 1950, con il suo famoso articolo *"Computing Machinery and Intelligence"* in cui proponeva quello che oggi è noto come il *Test di Turing* per valutare la capacità di una macchina di esibire un comportamento indistinguibile da quello umano. [1, 2]

Con l'evoluzione tecnologica e l'incremento esponenziale della disponibilità di dati e potenza computazionale, l'IA ha conosciuto una crescita notevole, trovando applicazione in ambiti molto diversi tra loro: dalla robotica all'automazione industriale, dalla diagnostica medica al riconoscimento del linguaggio e delle immagini, fino alla finanza e al marketing predittivo. [3]

Uno dei sottocampi più rilevanti dell'intelligenza artificiale è il Machine Learning (ML), ovvero l'insieme di metodi e tecniche che permettono ai computer di apprendere da dati passati e migliorare le proprie prestazioni nel tempo senza essere stati esplicitamente programmati per ogni singola attività. A differenza dei programmi tradizionali, in cui le regole vengono definite manualmente, i sistemi di ML estraggono automaticamente modelli e correlazioni dai dati. [2]

Il Machine Learning può essere classificato in tre categorie principali:

1. Apprendimento supervisionato (*Supervised Learning*): l'algoritmo apprende da un dataset *etichettato*, cioè costituito da coppie input-output. È il caso tipico della classificazione o regressione.
2. Apprendimento non supervisionato (*Unsupervised Learning*): i dati *non* presentano etichette e l'algoritmo cerca di scoprirne la struttura, come ad esempio nel clustering.
3. Apprendimento per rinforzo (*Reinforcement Learning*): un agente apprende interagendo con un ambiente e ricevendo ricompense o penalità in base alle proprie azioni.

In questa tesi ci si concentra sul Machine Learning supervisionato, e in particolare sul problema della classificazione multiclasse. In questo contesto, l'obiettivo è addestrare un modello che, a partire da un insieme di osservazioni di input (*feature*), sia in grado di prevedere a quale delle *cinque classi* predefinite appartenga un nuovo dato. Questo tipo di problema è ricorrente in numerosi scenari reali, come ad esempio il riconoscimento di patologie in immagini mediche, alla categorizzazione automatica di articoli di notizie, o al riconoscimento di emozioni da segnali biometrici. [3]

Tra gli algoritmi maggiormente utilizzati per affrontare problemi di classificazione vi sono i modelli ad albero decisionale, come il *Decision Tree*, e le sue evoluzioni più sofisticate come la *Random Forest* [4] e *XGBoost* [5]. Questi metodi si basano su approcci *ensemble*, in cui più modelli deboli vengono combinati per ottenere un modello più robusto e accurato.

Parallelamente, con il progresso nel campo dell'hardware e del *deep learning*, le reti neurali artificiali (ANN) hanno assunto un ruolo di primaria importanza. In particolare, le *reti neurali convoluzionali* (CNN) hanno rivoluzionato il campo del riconoscimento delle immagini e dei segnali, grazie alla loro capacità di apprendere automaticamente caratteristiche rilevanti dai dati grezzi. [6]

L'approccio seguito in questo lavoro di tesi consiste nel confrontare tre differenti strategie di classificazione supervisionata:

- *RandomForestClassifier*: un metodo basato sull'aggregazione di numerosi alberi decisionali addestrati su sottoinsiemi diversi dei dati.
- *XGBClassifier*: un modello di boosting che costruisce sequenzialmente alberi in grado di correggere gli errori dei precedenti.
- *Convolutional Neural Network (CNN)*: un'architettura di deep learning ispirata alla corteccia visiva animale, particolarmente efficace per i dati ad alta dimensionalità come immagini o segnali strutturati.

Lo scopo è quello di valutare le prestazioni relative di ciascun modello nel contesto specifico del caso di studio considerato, utilizzando metriche di valutazione standard che verranno descritte nel dettaglio nella sezione dedicata.

Questo capitolo prosegue approfondendo i modelli di classificazione basati su alberi decisionali, per poi illustrare le reti neurali convoluzionali e infine presentare le tecniche di valutazione adottate per il confronto tra i modelli.

I.1.1 Decision Tree Classifier

Il *Decision Tree Classifier* è uno dei metodi più semplici e interpretabili per la classificazione supervisionata. Si tratta di un algoritmo che utilizza una struttura ad albero per prendere decisioni, basandosi su una serie di domande binarie (o multiclasse) poste in sequenza. Ogni nodo interno dell'albero rappresenta una condizione su una delle variabili (feature), ogni ramo rappresenta il risultato di quella condizione, e ogni nodo foglia rappresenta una previsione di classe. [2]

L'idea alla base di un albero decisionale è simile al processo decisionale umano: a partire da un problema complesso, si suddivide lo spazio decisionale in una serie di scelte semplici e intuitive. Questo lo rende particolarmente utile in contesti dove è richiesta trasparenza e interpretabilità del modello, come nella diagnostica clinica o nei sistemi decisionali aziendali.

Un vantaggio chiave dei Decision Tree è la loro capacità di gestire sia variabili numeriche che categoriali senza necessità di normalizzazione o codifica complessa. Inoltre, sono in grado di modellare relazioni non lineari tra le variabili. [2]

La costruzione di un albero decisionale avviene attraverso un processo chiamato *ricorsione top-down*: a partire dall'intero dataset, si suddivide lo spazio in base alla feature che meglio separa le classi, ripetendo il processo sui sottoinsiemi risultanti. Questo procedimento continua finché non si raggiungono condizioni di *arresto*, come la profondità massima dell'albero o un numero minimo di campioni in un nodo. [7]

La scelta della feature ottimale per ogni split si basa su misure di purezza, ovvero criteri che quantificano quanto omogenee sono le classi nei sottoinsiemi generati. Le più comuni sono:

- *Entropia* e guadagno di informazione (*Information Gain*)**:** misura quanto si riduce l'entropia (cioè la disorganizzazione) dopo una divisione.
- *Indice di Gini*: misura la probabilità che un elemento scelto a caso venga classificato in modo errato, se assegnato casualmente in base alla distribuzione delle classi. [8]

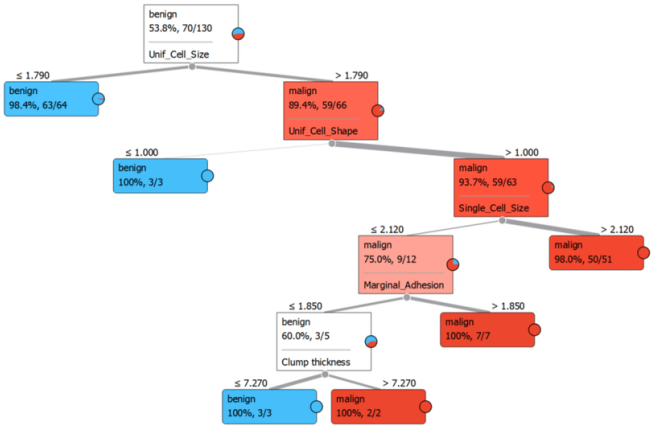

Figure 1 Esempio di albero decisionale applicato ad un caso di diagnosi medica - https://www.researchgate.net/figure/Figura-1-Esempio-di-albero-decisionale-applicato-ad-un-caso-di-diagnosi-medica_fig3_323339484

Un albero decisionale completamente cresciuto tende a memorizzare il dataset di training, creando una struttura molto complessa che cattura anche il rumore nei dati. Questo fenomeno, noto come *overfitting*, compromette la capacità del modello di generalizzare su nuovi dati.

Per contrastare l'overfitting, si ricorre a tecniche di potatura (*pruning*), che consistono nell'eliminazione di rami non significativi. [7, 8] Esistono due approcci principali:

- *Pre-pruning*: si impongono dei limiti durante la costruzione dell'albero (es. profondità massima, numero minimo di campioni per split).

- *Post-pruning*: l'albero viene costruito interamente e successivamente semplificato, eliminando nodi che non migliorano le prestazioni su un set di validazione.

Il Decision Tree è un algoritmo semplice che offre però molteplici vantaggi come la semplicità e la elevata interpretabilità, spesso non necessità di normalizzazione delle feature, funziona abbastanza bene su dati ridondanti o irrilevanti, essendo un modello molto semplice non è esenta da svantaggi che ne compromettono le prestazioni come l'elevato rischio di ricadere in overfitting, è sensibile a piccole variazioni nei dati che possono portare l'algoritmo all'instabilità, e generalmente è molto meno accurato rispetto ai metodi ensemble come Random Forest o XGBoost, se usato da solo.

Nonostante i suoi limiti, il Decision Tree rappresenta la base concettuale per molti altri algoritmi più avanzati, come appunto Random Forest e Gradient Boosting, che mirano a superarne le debolezze mantenendone i punti di forza.

I.1.2 Random Forest Classifier

Il Random Forest è un algoritmo di apprendimento supervisionato che rientra nella categoria dei *metodi ensemble*, ovvero modelli che combinano le predizioni di più modelli base (chiamati *weak learners* [10]) al fine di migliorare la precisione e la robustezza della classificazione. [4, 9]

L'idea alla base dei metodi ensemble è che un insieme di modelli deboli, che presi singolarmente possono essere poco accurati, possa produrre un modello molto più potente e generalizzabile se le loro predizioni vengono combinate in modo opportuno. Questo principio si basa sul concetto statistico della saggezza della folla (wisdom of the crowd), secondo cui la media di giudizi indipendenti tende ad essere più accurata del giudizio di un singolo esperto. [10]

I due approcci ensemble più comuni sono:

- Bagging (*Bootstrap Aggregating*): consiste nell'addestrare più modelli indipendenti su sottoinsiemi casuali del dataset (con ripetizione) e combinarne le predizioni. È il metodo su cui si basa la Random Forest.
- *Boosting*: costruisce modelli in sequenza, in cui ogni nuovo modello cerca di correggere gli errori commessi dai precedenti (ad es. AdaBoost, XGBoost, Gradient Boosting).

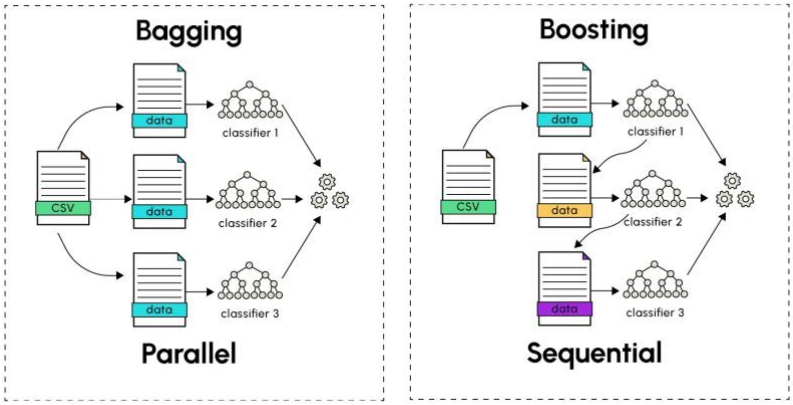

Figure 2 schema comparativo tra bagging e boosting - https://datascientest.com/en/bagging-vs-boosting

Il Random Forest è un ensemble di alberi decisionali, in cui ciascun albero viene costruito utilizzando una porzione casuale del dataset di addestramento tramite il metodo del bootstrap. Oltre al sottocampionamento dei dati, il Random Forest introduce un'ulteriore componente di casualità: in ogni split dell'albero viene considerato solo un sottoinsieme casuale delle feature, piuttosto che tutte. Questa doppia randomizzazione (dei dati e delle feature) riduce la *varianza* del modello e ne migliora la capacità di generalizzazione. [2, 4, 9]

Il processo di classificazione in una Random Forest funziona nel seguente modo:

1. Vengono generati *N* alberi decisionali, ciascuno addestrato su un campione diverso del dataset (scelto con sostituzione).
2. Ogni albero produce una predizione di classe per un'istanza di input.
3. La predizione finale è ottenuta tramite *voto di maggioranza* tra le predizioni degli alberi (in caso di classificazione) oppure *media* (in caso di regressione).

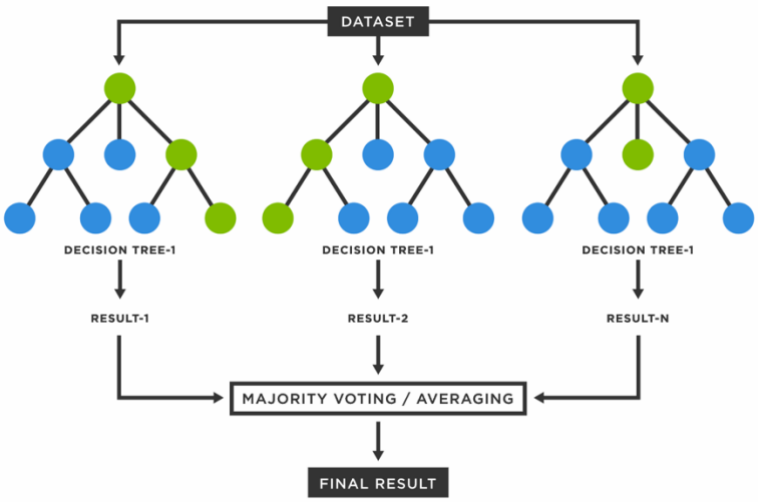

Figure 3 diagramma della struttura di una Random Forest con diversi alberi addestrati su subset differenti e il processo di voto finale - https://www.diariodiunanalista.it/posts/algoritmi-di-machine-learning-guida-introduttiva-per-comprendere-i-principi-e-le-app

Il Random Forest è apprezzato per una serie di motivi principali, come ad esempio l'essere robusto al rumore e all'overfitting, grazie alla media delle predizioni è meno incline a sovradattarsi ai dati rispetto a un singolo albero di decisione, ha una buona capacità di gestire dati ad alta dimensionalità con molte features non informative, è adattabile gestendo sia dati numerici che categoriali.

Nonostante i suoi punti di forza rendano il Random Forest un algoritmo che produce spesso buone classificazioni non è esente da limitazioni, rispetto a un Decision Tree è molto bene interpretabile poiché la predizione deriva dall'aggregazione di numerosi alberi, in più è richiesto un maggiore sforzo computazionale sia in termini di addestramento che di memoria, specialmente per un elevato numero di alberi, spesso inoltre è necessario bilanciare correttamente le classi per non avere risultati sub-ottimali.

In sintesi, il Random Forest è una tecnica potente e versatile, ampiamente utilizzata in ambito industriale e accademico per problemi di classificazione e regressione. Le sue capacità predittive elevate, combinate alla relativa semplicità di implementazione, lo rendono una scelta eccellente per molti problemi pratici, inclusa la classificazione multiclasse trattata in questo progetto di tesi.

I.1.3 XGBClassifier

L'XGBClassifier è una delle implementazioni più popolari e performanti dell'algoritmo *Gradient Boosting*, introdotta con la libreria XGBoost (Extreme Gradient Boosting) da Chen e Guestrin nel 2016. [5] Questo algoritmo ha rivoluzionato il campo del machine learning per la sua efficienza, accuratezza e capacità di gestione di grandi dataset. XGBoost è stato ed è tuttora largamente impiegato in competizioni internazionali come quelle ospitate su *Kaggle*, dove spesso risulta vincente.

Per comprendere XGBoost, è essenziale prima capire cos'è il Gradient Boosting, una tecnica di ensemble che combina molti modelli deboli (di solito alberi decisionali a bassa profondità) in un modello forte. A differenza del *bagging*, che costruisce modelli indipendenti in parallelo (come nel Random Forest), il boosting costruisce i modelli in *sequenza*, dove ogni nuovo modello cerca di *correggere gli errori* commessi dai modelli precedenti. [10]

Nel Gradient Boosting, l'addestramento avviene in modo iterativo [11]:

1. Si inizia con una previsione iniziale (spesso la media nel caso di regressione o una log-odds costante nel caso di classificazione).
2. Ad ogni iterazione, viene costruito un nuovo albero che predice i residui (cioè gli errori) del modello corrente.
3. Le predizioni vengono aggiornate sommando una frazione (learning rate) della predizione dell'albero ai risultati precedenti.

4. Questo processo si ripete fino al raggiungimento di un numero prefissato di alberi o al soddisfacimento di una condizione di arresto.

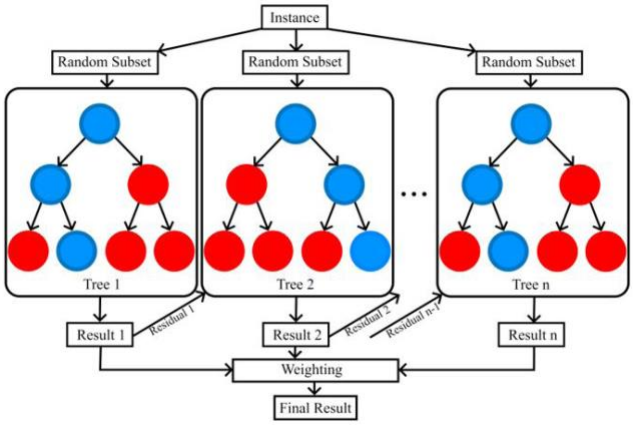

Figure 4 schema sequenziale di addestramento in Gradient Boosting - https://www.researchgate.net/figure/Simplified-structure-of-XGBoost_fig5_381857634

XGBoost estende il concetto di Gradient Boosting con una serie di miglioramenti volti a rendere l'algoritmo più veloce, più preciso, e più robusto [5, 11]:

- Regularizzazione *L1* e *L2*: a differenza del Gradient Boosting standard, XGBoost introduce una regolarizzazione esplicita per evitare l'overfitting, migliorando così la generalizzazione. [5]
- Sparsity-aware algorithm: XGBoost è in grado di gestire in maniera efficiente dati sparsi e con valori mancanti, sfruttando pattern di sparsità.
- Parallelizzazione interna: l'algoritmo è ottimizzato per sfruttare il parallelismo durante la costruzione degli alberi.
- Pruning post-anticipato: XGBoost adotta una strategia di "pruning posticipato" (a contrario di quello preemptive del Gradient Boosting tradizionale), cioè esplora completamente l'albero prima di decidere se potarlo, portando a modelli più ottimizzati.

L' XGBoost può essere visto come una tecnica di ottimizzazione dove l'obiettivo è minimizzare una funzione di costo definita come:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{k=1}^{t} \Omega(f_k) \qquad (1)$$

Dove l è la funzione di perdita (ad es. log-loss per classificazione). $\hat{y}_i^{(t)}$ rappresenta la predizione all'iterazione t mentre $\Omega(f_k)$ è il termine di regolarizzazione [5], che penalizza la complessità del modello:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {w_j}^2 \quad (2)$$

Nell'equazione (2) T è il numero di foglie e w_j rappresenta il peso di ciascuna foglia.

I vantaggi dell'XGBClassifier sono nelle prestazioni predittive riuscendo a competere sempre alla pari con algoritmi come Random Forest soprattutto in dataset ben strutturati dove potrebbe superarne le prestazioni, l'elevato numero di iperparametri come il Learning Rate il numero di Estimators o il Max Depth lo rendono un algoritmo controllabile, in più ha una elevata efficienza computazionale supportando contemporaneamente classificazione binaria multiclasse e regressione. I suoi svantaggi sono nella creazione di una maggiore complessità nella configurazione, rischio di ricadere in overfitting se non viene correttamente regolarizzato o se il numero di alberi è troppo alto, spesso offre prestazioni inferiori su dataset non tabulari come immagini o sequenze, dove spesso vengono applicate le reti neurali.

" Nel contesto di questo progetto di tesi, l'XGBClassifier viene utilizzato per affrontare un problema di classificazione multiclasse a cinque etichette. Grazie alla sua capacità di modellare relazioni complesse tra le feature e alla sua robustezza agli outlier e ai valori mancanti, rappresenta un'ottima scelta per confrontare le prestazioni rispetto a modelli come Random Forest o le reti neurali convoluzionali. " Può essere che lo togliamo

I.2 Reti Neurali Artificiali (Artificial Neural Networks - ANN)

Le reti neurali artificiali (ANN – Artificial Neural Networks) sono modelli computazionali ispirati al funzionamento del cervello umano, progettati per riconoscere pattern e apprendere relazioni complesse nei dati attraverso una struttura stratificata di nodi interconnessi, detti *neuroni*.

Nel contesto del machine learning, le ANN rappresentano un'evoluzione significativa rispetto agli algoritmi tradizionali, soprattutto per la loro capacità di apprendere feature gerarchiche e non lineari direttamente dai dati grezzi, senza bisogno di una complessa ingegnerizzazione manuale delle feature. [12]

Una rete neurale artificiale è composta da tre tipi fondamentali di strati (*layers*):

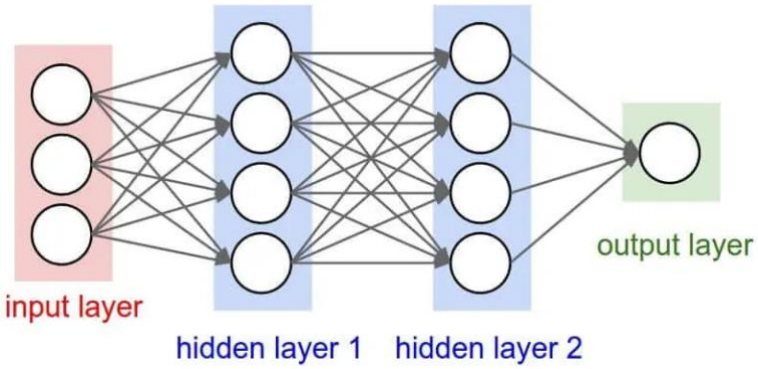

Figure 5 rappresentazione di una rete neurale semplice - https://www.ai4business.it/intelligenza-artificiale/dal-mit-un-progetto-che-spiega-le-reti-neurali/

1. *Input Layer*: lo strato iniziale, dove i dati grezzi vengono inseriti nel modello.
2. *Hidden Layers*: strati intermedi che elaborano i dati, applicando trasformazioni non lineari e apprendendo rappresentazioni sempre più astratte.
3. *Output Layer*: lo strato finale che produce la predizione del modello.

Ogni nodo (o *neurone artificiale*) in un layer riceve in input una combinazione pesata dei segnali in ingresso, vi applica una funzione di attivazione, e trasmette il risultato ai neuroni del layer successivo. [13]

Matematicamente, l'attivazione di un neurone j è espressa da:

$$y_j = \sigma\left(\sum_{i=1}^{n} w_{ij} x_i + b_j\right) \tag{3}$$

Nella (3) x_i rappresentano i dati di input, w_{ij} rappresentano i pesi sinaptici, b_j è il bias del neurone, mentre σ rappresenta la funzione di attivazione.

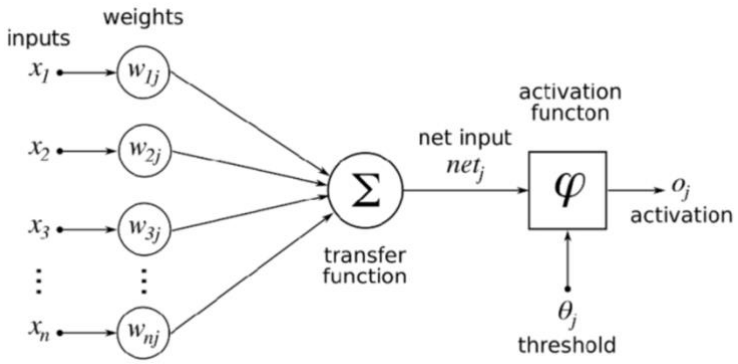

Figure 6 flusso del calcolo in un singolo neurone artificiale con evidenza dei pesi, bias e funzione di attivazione - https://www.diariodiunanalista.it/posts/introduzione-alle-reti-neurali-pesi-bias-e-attivazione/

Le funzioni di attivazione sono fondamentali per introdurre non linearità nel modello. [14] Alcune delle più comuni includono:

- *Sigmoid*: utile per output binari ma suscettibile al problema del gradiente che scompare.
- *Tanh*: simile alla sigmoid ma centrata in zero.
- *ReLU* (Rectified Linear Unit): oggi la più usata, semplice e computazionalmente efficiente.

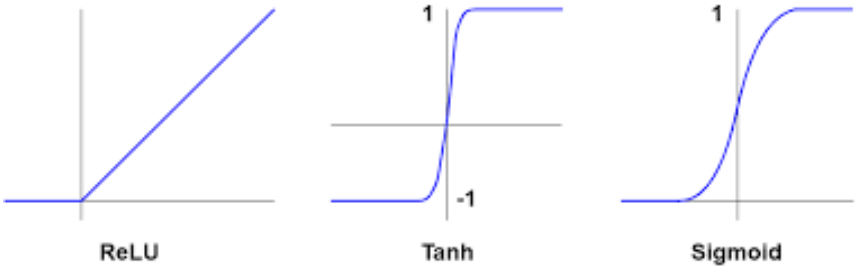

Figure 7 confronto grafico tra le principali funzioni di attivazione - https://amslaurea.unibo.it/id/eprint/24979/1/binzoni_andrea_tesi.pdf

Il processo di apprendimento delle reti neurali avviene mediante un algoritmo chiamato *backpropagation*, [12] che calcola il gradiente dell'errore rispetto ai pesi della rete e li aggiorna usando una tecnica di ottimizzazione, solitamente la discesa del gradiente (*gradient descent*) o una sua variante (es. *Adam*, *RMSprop*, *Momentum*).

L'ottimizzazione in una rete neurale consiste nell'aggiornamento dei pesi per minimizzare una funzione di costo, in genere una *loss function* (come MSE, Cross-Entropy, ecc.). Questo processo è reso possibile grazie a metodi di *ottimizzazione numerica* basato sul *calcolo del gradiente*, ovvero su quanto e come modificare i pesi per migliorare la prestazione del modello. [12]

L'algoritmo di Gradient Descent consiste nel calcolare il gradiente della funzione di costo rispetto ai pesi e aggiornare questi ultimi nella direzione opposta al gradiente. Matematicamente:

$$w := w - \eta \nabla \mathcal{L}(w) \quad (4)$$

In questa formula (4) w rappresentano i pesi, η è il learning rate (tasso di apprendimento) che è un parametro critico e può essere fissato o essere scelto dinamicamente durante l'addestramento, $\nabla \mathcal{L}(w)$ è il gradiente della loss function rispetto ai pesi. Ci sono diverse varianti di gradient descent, ciascuna con vantaggi e svantaggi. [15]

1. Batch Gradient Descent:

Calcola il gradiente sull'intero dataset prima di aggiornare i pesi, il vantaggio è una convergenza stabile, ma lo svantaggio è su un costo in termini computazionali significativo specialmente per dataset grandi e il rischio di essere bloccato in minimi locali.

2. Stochastic Gradient Descent (SGD)

Aggiorna i pesi dopo ogni esempio del dataset, il vantaggio si traduce in una maggiore velocità per dataset grandi e l'introduzione di rumore per sfuggire a minimi locali, ma soffre di una convergenza rumorosa e meno stabile.

3. Mini-Batch Gradient Descent

Compromesso tra i due precedenti: aggiorna i pesi dopo un piccolo batch di esempi (es. 32, 64, 128), offrendo vantaggi nella convergenza più veloce rispetto al batch puro, migliorando la stabilità rispetto all'SGD. Mini-batch è oggi lo standard de facto nell'addestramento delle reti neurali profonde.

4. Momentum

Il metodo Momentum migliora la convergenza accumulando *velocità* nel gradiente:

$$v_t = \beta v_{t-1} + (1 - \beta) \nabla \mathcal{L}(w_t)$$

$$w_{t+1} = w_t - \eta v_t$$

Dove β controlla il contributo delle iterazioni precedenti. Aiuta ad accelerare in direzioni utili e a ridurre oscillazioni. [12]

5. RMSProp

RMSProp adatta dinamicamente il learning rate per ciascun peso, tenendo conto della media esponenziale dei quadrati dei gradienti:

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla \mathcal{L}(w_t))^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \varepsilon}} \nabla \mathcal{L}(w_t)$$

Stabilizza la convergenza anche in presenza di funzioni di perdita molto complesse, è ottimo per reti neurali ricorrenti e dati rumorosi. [15]

6. Adam (Adaptive Moment Estimation)

L'ottimizzatore Adam combina i punti di forza del Momentum e dell'RMSProp, la sua formulazione matematica è espressa come:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla\mathcal{L}(\theta_t)$$

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla\mathcal{L}(\theta_t))^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t} + \varepsilon} m_t$$

È veloce ed è stabilile. Generalmente fornisce ottimi risultati senza molti aggiustamenti manuali. [14, 16]

Algoritmo	Stabilità	Velocità	Adattività	Utilizzo
Batch GD	Alta	Lenta	No	Dataset piccoli
SGD	Bassa	Alta	No	Dataset grandi
Mini-Batch GD	Alta	Alta	No	Standard
Momentum	Alta	Media	No	Gradienti Oscillanti
RMSProp	Alta	Alta	Si	Dati rumorosi / RNN
Adam	Alta	Alta	Si	Default

Table 1 Tabelle riassuntiva degli algoritmi di ottimizzazione

Le fasi principali nell'addestramento delle reti neurali prevedono:

1. Forward pass: i dati vengono propagati dalla rete fino all'output.
2. Calcolo della loss: viene misurata la discrepanza tra la predizione della rete e il valore reale.
3. Backward pass: tramite il teorema della catena (chain rule), l'errore viene "retropropagato" per aggiornare i pesi.
4. Aggiornamento dei pesi: in base al learning rate e all'algoritmo di ottimizzazione scelto.

Uno dei problemi più comuni nelle reti neurali è l'overfitting, ovvero quando la rete apprende troppo bene i dati di addestramento, perdendo capacità di generalizzazione. Per contrastare questo fenomeno vengono usate diverse tecniche:

- *Dropout*: disattivazione casuale di neuroni durante l'addestramento.
- *Early Stopping*: interruzione dell'addestramento quando la performance sul validation set peggiora.
- *Regularizzazione L1/L2*: penalizzazione dei pesi eccessivamente grandi. [14]
- *Data augmentation* (soprattutto per immagini): generazione di variazioni artificiali dei dati per ampliare il dataset.

Le reti neurali sono state alla base di grandi avanzamenti nell'IA moderna e sono utilizzate ampiamente nella visione artificiale, come il riconoscimento facciale, la segmentazione e le diagnosi mediche, vengono usate nell'elaborazione del linguaggio naturale come per esempio nella traduzione automatica. [6]

Nel contesto di questa tesi, le reti neurali sono utilizzate come base per la rete neurale convoluzionale (CNN) presentata di seguito, ottimizzata per trattare input strutturati, come immagini o dati con correlazioni spaziali.

I.2.2 Convolutional Neural Networks (CNN)

Le *Convolutional Neural Networks* (CNN) rappresentano una classe di reti neurali artificiali particolarmente efficace per l'elaborazione di dati con struttura spaziale, come le immagini. L'architettura delle CNN si è affermata soprattutto nell'ambito della computer vision, grazie alla sua capacità di catturare pattern locali e gerarchici nei dati visivi. [6]

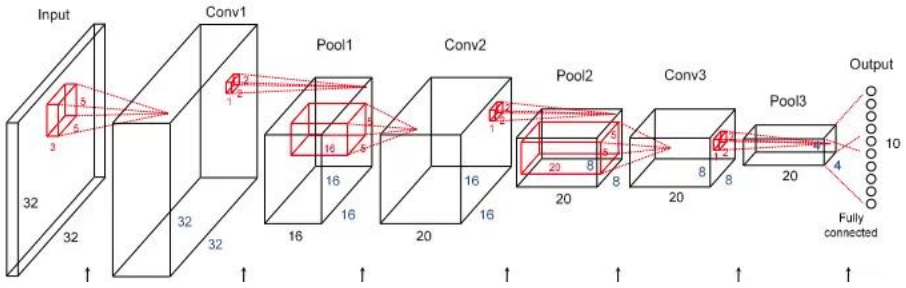

Figure 8 architettura CNN con convoluzione, pooling, fully connected - https://www.domsoria.com/2020/04/tipologie-e-comportamenti-di-reti-neurali-convoluzionali/

Il primo concetto di rete convoluzionale può essere fatto risalire al *Neocognitron* di Fukushima (1980), [17] ma il vero salto di qualità è arrivato con l'architettura *LeNet-5* sviluppata da Yann LeCun nel 1998 per il riconoscimento delle cifre scritte a mano (dataset *MNIST*). [18]

Il boom dell'utilizzo delle CNN è arrivato nel 2012 grazie alla rete *AlexNet*, che ha rivoluzionato la competizione ImageNet, portando l'errore top-5 da ~26% a ~15%. [6, 19]

A differenza delle reti dense (*fully connected*), le CNN si basano su *strati convoluzionali* che operano con filtri (*kernel*) capaci di catturare relazioni locali nei dati:

Ogni convoluzione consiste nel far scorrere un filtro sopra l'immagine (o l'output dello strato precedente), effettuando un *prodotto scalare* locale. I filtri imparano a rilevare bordature, texture, forme o pattern astratti. L'output di questo processo è una *feature map*.

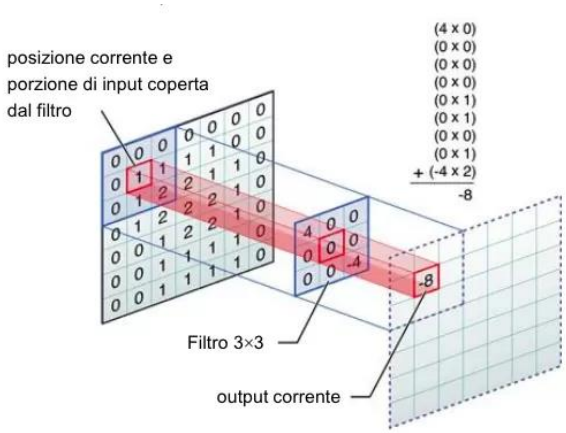

Figure 9 esempio di convoluzione su immagine 3x3 con filtro - https://www.domsoria.com/2019/10/come-funziona-una-rete-neurale-cnn-convolutional-neural-network/

Dopo la convoluzione, si applica una funzione di attivazione non lineare, tipicamente ReLU perché aiuta ad introdurre non linearità e riducendo il rischio di vanishing gradient.

Gli strati di *pooling* riducono la dimensionalità spaziale delle feature map, migliorando l'efficienza computazionale e la generalizzazione. Il più comune è il *max pooling*, che prende il valore massimo in una finestra (es. 2x2). Permette di mantenere le informazioni salienti riducendo la complessità.

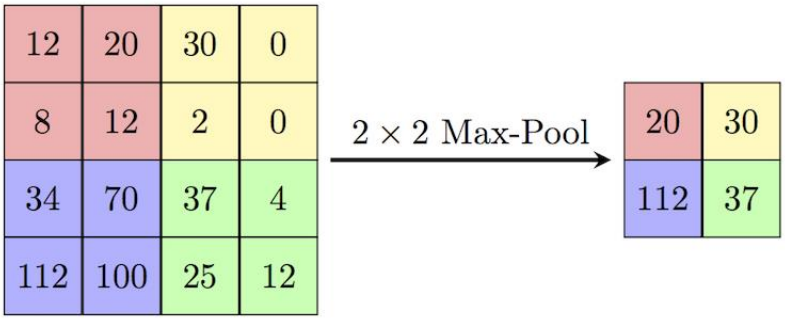

Figure 10 max pooling illustrato su matrice con evidenziazione dei valori massimi - https://paperswithcode.com/method/max-pooling

Alla fine della parte convoluzionale, l'output tridimensionale viene "appiattito" (*flatten*) in un vettore e passato a uno o più strati *fully connected*, come nelle reti classiche, che producono il risultato finale della classificazione.

Le CNN vengono addestrate come le altre reti neurali tramite backpropagation e ottimizzazione dei pesi con algoritmi come SGD o Adam visti nella sezione I.2.1.

Tuttavia, le CNN richiedono grandi quantità di dati e potenza di calcolo. Per questo si utilizzano spesso:

- Data augmentation (rotazioni, zoom, traslazioni delle immagini)
- Transfer learning (partendo da modelli pre-addestrati come ResNet, VGG, MobileNet, ecc.)

Le CNN offrono molti vantaggi riescono a catturare pattern spaziali, hanno una maggiore efficienza rispetto a reti fully connected su immagini mostrando un'elevata accuratezza.

I.3 Tecniche di Valutazione dei Modelli di Classificazione

Nel contesto del machine learning supervisionato, la fase di valutazione riveste un'importanza fondamentale, poiché consente di verificare la capacità generalizzativa del modello e la *qualità delle sue predizioni*. In particolare, per problemi di *classificazione*, esistono diverse metriche e strategie di valutazione, ciascuna adatta a specifici contesti e caratteristiche del dataset (es. distribuzione sbilanciata delle classi).

La *accuracy* rappresenta la metrica più intuitiva: misura la *percentuale di predizioni corrette* rispetto al totale.

$$Accuracy = \frac{numero\ di\ predizioni\ corrette}{numero\ totale\ di\ osservazioni}$$

Tuttavia, l'accuracy può essere *fuorviante* nei casi in cui il dataset sia sbilanciato (ad es. una classe è molto più frequente delle altre). In questi casi, è utile ricorrere alla confusion matrix, una tabella che consente di analizzare in dettaglio le prestazioni del classificatore per ciascuna classe.

Una confusion matrix per problemi multiclasse ha dimensione $C \times C$, dove C è il numero di classi. Le diagonali rappresentano le predizioni corrette, mentre gli elementi fuori diagonale indicano gli errori. [14, 19]

Per valutare meglio le prestazioni in scenari complessi, si utilizzano metriche derivate dalla confusion matrix, particolarmente efficaci per dataset sbilanciati [19]:

- *Precision*: quanti dei campioni predetti positivi sono effettivamente positivi.

$$Precision = \frac{TP}{TP + FP}$$

- Recall (Sensibilità): quanti dei campioni effettivamente positivi sono stati predetti come tali.

$$Recall = \frac{TP}{TP + FN}$$

- F1-score: media armonica tra precision e recall, utile per trovare un equilibrio tra i due.

$$F1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

La validazione incrociata (*cross-validation*) è una tecnica di valutazione che consente di stimare in maniera più robusta le prestazioni del modello, evitando il rischio di overfitting al dataset di test.

Il metodo più comune è la *K-Fold Cross Validation*, che consiste nel suddividere il dataset in K parti (*folds*), addestrando il modello su $K - 1$ parti e testandolo sull'ultima. Questo processo viene ripetuto K volte, cambiando il fold di test a ogni iterazione, e le metriche vengono poi medie. [14, 20]

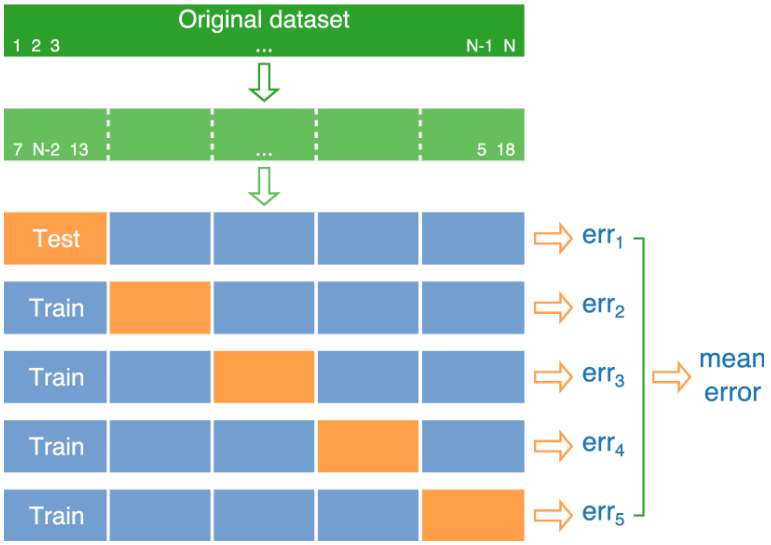

Figure 11 : schema visivo della K-Fold Cross Validation - https://www.researchgate.net/figure/A-schematic-illustration-of-K-fold-cross-validation-for-K-5-Original-dataset-shown-in_fig5_311668395

[1] Turing, A. M. (1950). Computing machinery and intelligence.

[2] Mitchell, T. M. (1997). *Machine learning*. McGraw Hill.

[3] Russell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.

[4] Breiman, L. (2001). Random forests. *Machine learning*.

[5] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD*.

[6] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553).

[7] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*.

[8] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group.

[9] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*. Springer.

[10] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning* (2nd ed.). Springer.

[11] Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*.

[12] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[13] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

[14] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.).

[15] Ruder, S. (2016). *An overview of gradient descent optimization algorithms*.

[16] Kingma, D. P., & Ba, J. (2015). *Adam: A method for stochastic optimization*.

[17] Fukushima, Kunihiko. (2012). *Artificial vision by multi-layered neural networks: Neocognitron and its advances. Neural networks : the official journal of the International Neural Network Society*.

[18] Lecun, Yann & Bottou, Leon & Bengio, Y. & Haffner, Patrick. (1998). *Gradient-Based Learning Applied to Document Recognition*. Proceedings of the IEEE.

[19] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. In Advances in neural information processing systems.

[20] Raschka, S. (2015). *Python Machine Learning*. Packt Publishing.