UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Meccanica

Analisi di metodi AI per creazione dataset e classificazione automatica di difetti del calcestruzzo

Analysis of AI methods for dataset creation and automatic classification of concrete defects

Relatore: Chiar.mo

Prof. Gian Marco Revel

Tesi di laurea di:

Maicol Mancinelli

Anno Accademico 2022-2023

"Artificial intelligence would be the ultimate version of Google.
The ultimate search engine that would understand everything on the Web.
It would understand exactly what you wanted,
and it would give you the right thing.
We're nowhere near doing that now.
However, we can get incrementally closer to that,
and that is basically what we work on."
Larry Page
Co-Founder of Google

Abstract

L’elaborato ha l’obiettivo di analizzare l'utilizzo delle reti neurali e di acquisire le conoscenze necessarie per sviluppare una rete neurale applicabile al caso studio del riconoscimento dei difetti del calcestruzzo tramite l’analisi di immagini.

La prima parte si concentra sull’analisi del mondo delle reti neurali e delle applicazioni dell’intelligenza artificiale, considerando studi già presenti in letteratura al fine di ottimizzare al meglio l’applicazione in esame.

La seconda parte è dedicata alla creazione del *dataset* per l’allenamento e alla scelta dei parametri per efficientare l’architettura della rete neurale costruita ad hoc per il caso studio.

Nella terza parte sono riportati e discussi i risultati delle fasi di allenamento e di test della rete neurale e vengono mostrati i valori di accuratezza restituiti per una valutazione finale delle potenzialità della rete.

Nella quarta ed ultima parte si dà una valutazione complessiva della rete neurale applicata al caso studio e si discutono i punti di forza degli strumenti supportati dall’intelligenza artificiale. Si pone particolare attenzione agli algoritmi di riconoscimento e classificazione dei difetti e alle implicazioni, presenti e future, che questi strumenti possono avere.

Indice

1 Introduzione

In questa sezione sarà trattato l'argomento delle reti neurali secondo lo stato dell'arte attualmente presente in letteratura. Si passa poi ad un'analisi della struttura di una rete neurale e dei i suoi parametri fondamentali per concludere con le applicazioni che questo tipo di strumenti possono avere in diversi ambiti ed in particolare alla classificazione tramite AI e al riconoscimento di difetti.

1.1 Intelligenza Artificiale

L'intelligenza artificiale, in sigla AI (*Artificial Intelligence*), è una disciplina dell'informatica che studia la possibilità di creare "meccanismi" in grado di replicare le capacità decisionali e di risoluzione dei problemi tipiche della mente umana.

Nel 1950 Alan Turing, considerato da molti il padre dell'informatica, pubblica la sua opera fondamentale "*Computing Machinery and Intelligence*". In questo documento pone la seguente domanda "Le macchine possono pensare?" e sviluppa un metodo, noto poi con il nome di "*Test di Turing*", che permette di capire se la macchina riesce a sviluppare un comportamento indistinguibile da quello umano.

Il test in questione è di grande rilevanza in quanto offre un metodo quantitativo per valutare l'abilità di una macchina nel manifestare comportamenti intelligenti. Nel campo dell'intelligenza artificiale, è stato ampiamente adottato come strumento per valutare i progressi nella ricerca e nello sviluppo dell'AI.

L'utilizzo di questi algoritmi apre un discorso più ampio e filosofico sull'etica, quali la trasparenza delle informazioni, il rispetto della privacy dei dati, la sicurezza ed altri temi legati alla responsabilità ed alla regolamentazione legale.

Rimanendo sull'aspetto tecnico, nella sua forma più semplice l'AI è un campo che combina l'informatica ed un gran numero di dati per consentire la risoluzione dei problemi tramite alcune tecniche di apprendimento che saranno trattate di seguito, come il *machine learning* ed il *deep learning*.

1.2 Machine Learning

Il *machine learning* come mostrato in Figura 1.1 è un sottoinsieme dell'intelligenza artificiale. In particolare, questo campo si occupa di addestrare gli algoritmi ad imparare dai dati in ingresso e a migliorare con l'allenamento, anziché essere appositamente programmati per riuscirci.

Analizzando i dati in ingresso, queste applicazioni cercano di identificare schemi, correlazioni o caratteristiche particolari che si ripetono e sulla base di queste formulano delle previsioni.

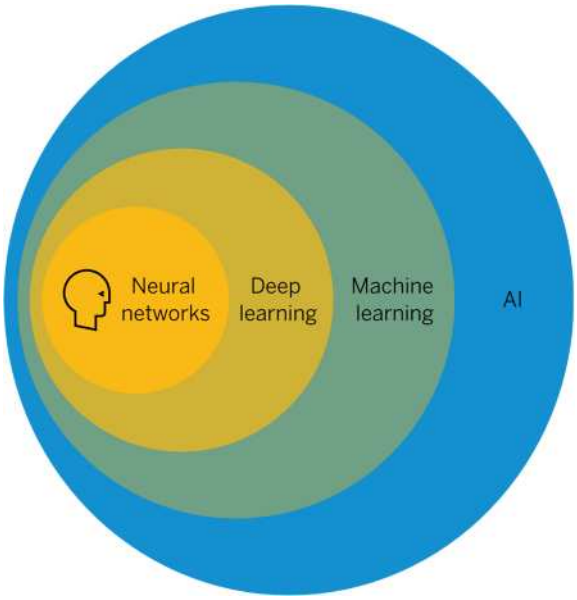

Figura 1.1 Diagramma del rapporto tra AI e machine learning

1.2.1 Deep Learning

Il *deep learning* è un sottoinsieme del *machine learning*. Il modello più semplice di rete neurale che si può costruire è composto solamente da tre strati (*layer*): il primo di *input* seguito da uno nascosto (*hidden*) ed infine uno strato di *output*. Le reti profonde (*deep*) sono caratterizzate invece da più di un livello nascosto, come si può vedere in Figura 1.2. Sebbene una rete neurale con un singolo livello nascosto sia comunque in grado di fare previsioni approssimative, si aggiungono ulteriori livelli per ottimizzare la rete ed aumentarne la precisione.

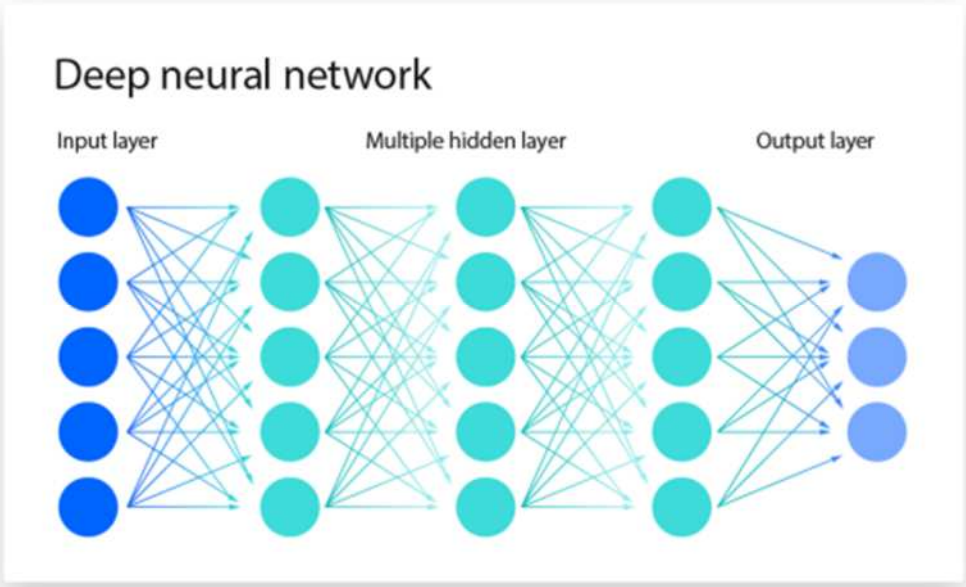

Figura 1.2 Schema di una rete neurale profonda

1.2.2 Supervised and Unsupervised Learning

Nell'ambito dell'intelligenza artificiale e dell'apprendimento automatico, esistono due approcci fondamentali: *supervised learning* e l'*unsupervised learning*. [1]

Nel *supervised learning* o apprendimento supervisionato i dati di input sono etichettati e fungono da target per la rete. Questi set di dati sono progettati per addestrare o "supervisionare" gli algoritmi nella classificazione o nella previsione accurata dei risultati. Utilizzando input e output etichettati il modello può misurare la propria accuratezza e apprendere nel tempo.

L'*unsupervised learning* o apprendimento non supervisionato differisce in quanto il *dataset* non include alcuna etichetta. Questi algoritmi sono in grado di scoprire *feature*, *pattern* o modelli nascosti nei dati senza la necessità dell'intervento umano (quindi non sono "supervisionati").

Tuttavia, è importante notare che la maggior parte delle attività di riconoscimento e classificazione tramite immagini utilizzano l'approccio dell'apprendimento supervisionato.

1.3 Reti Neurali

Le reti neurali, note anche come ANN (*Artificial Neural Network*), rappresentano l'elemento principale nel *machine learning*. Il nome stesso riconduce al cervello umano, infatti la loro struttura è progettata per imitare il modo in cui i neuroni biologici sono interconnessi tra loro e si scambiano segnali.

1.3.1 Layer e neuroni artificiali

In campo informatico la rete neurale è abbastanza simile al cervello umano, è composta infatti da nodi (neuroni) e dalle connessioni (sinapsi) che li collegano. I neuroni che compongono lo stesso livello formano un *layer*.

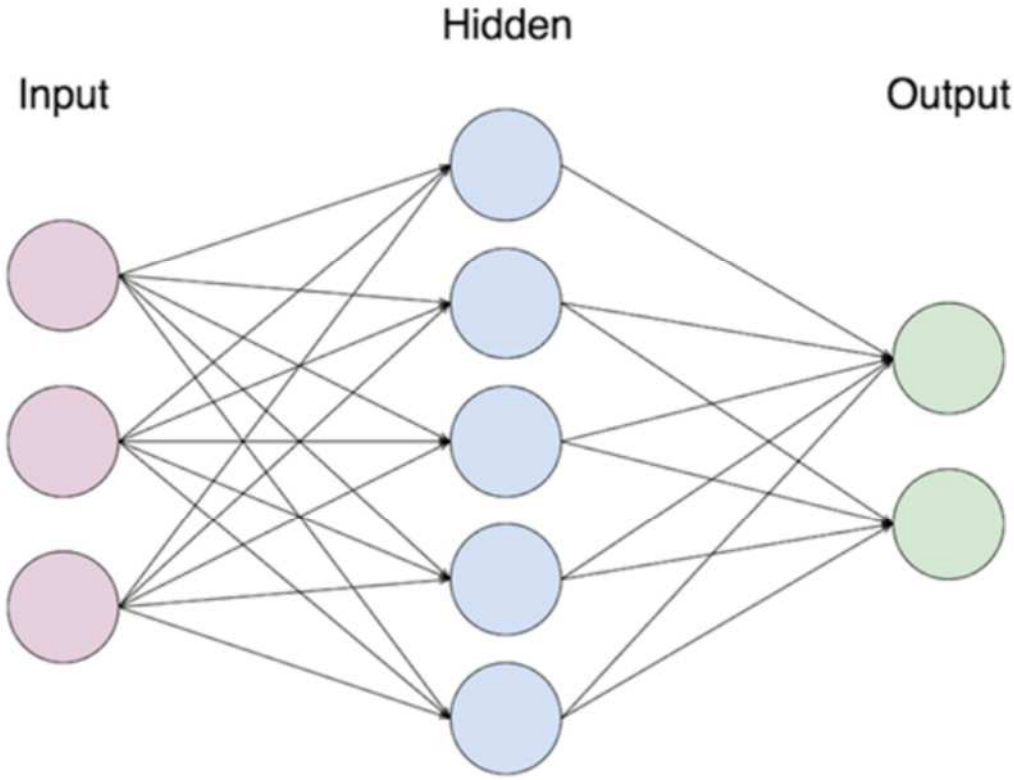

Figura 1.3 Schema di una rete neurale semplice

Il nodo, o neurone, è l'unità fondamentale della rete dove vengono effettuate le operazioni. Come evidenziato in Figura 1.3, esistono tre diverse tipologie di neuroni:

- Nodi di entrata (*input nodes*)

  Si trovano a sinistra dello schema, all'ingresso della rete neurale, e sono utilizzati per introdurre i dati di input.

- Nodi intermedi (*hidden nodes*)

  Sono i nodi posti nella parte centrale. Nella rete neurale più semplice questi sono organizzati in un unico *layer*

- Nodi di uscita (*output nodes*)

  Si trovano a destra nello schema, all'uscita della rete neurale e forniscono il risultato dell'elaborazione.

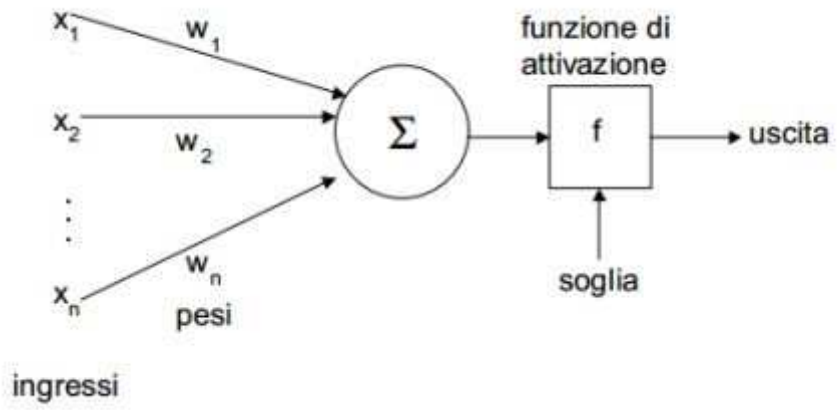

Figura 1.4 Schema di un neurone artificiale

Il neurone, come riportato in [2] e nella Figura 1.4, riceve gli ingressi $(x_1, x_2, \dots, x_n)$ che vengono moltiplicati per un opportuno peso (*weight*) caratteristico di ogni connessione $(w_1, w_2, \dots, w_n)$. Il risultato delle moltiplicazioni viene sommato e se la somma supera una certa soglia il neurone si attiva attivando la sua uscita.

Il peso serve a quantificare l'importanza dell'ingresso, in particolare una connessione con un peso elevato avrà una maggiore importanza rispetto ad un ingresso meno utile all'algoritmo. Aggiustando i pesi possiamo ottenere l'output desiderato per uno specifico input.
Al primo avvio della rete i pesi vengono inizializzati con dei valori casuali, e saranno poi aggiornati e ottimizzati durante l'addestramento.

In base alla struttura delle connessioni, le reti possono essere raggruppate in due categorie:

- *Feed-forward network*

  Rappresenta lo schema più utilizzato, in Figura 1.5, la propagazione delle informazioni avviene in avanti, da destra verso sinistra

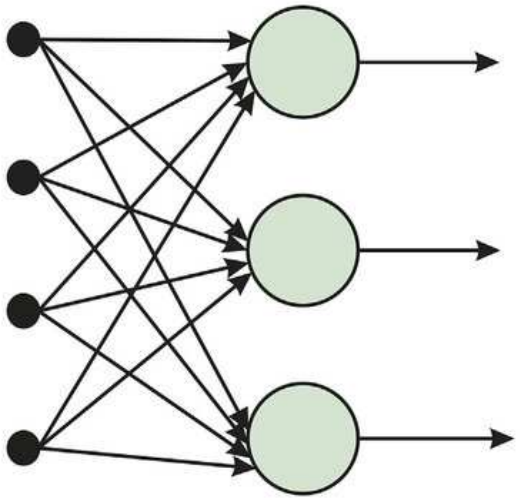

Figura 1.5 Schema di una rete feedforward

- *Feedback network*

  Nelle reti con *feedback* le informazioni non viaggiano solamente in avanti, ma come mostrato in Figura 1.6, l'output viene trasmesso all'indietro e rappresenta una delle caratteristiche principali delle RNN (*Recurrent Neural Network*)

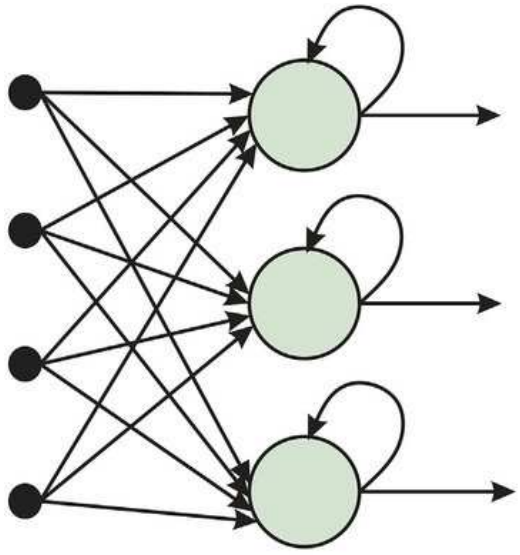

Figura 1.6 Schema di una rete feedback

1.3.2 Activation

La rete neurale calcola la somma dei prodotti degli input e i loro pesi corrispondenti e infine applica una funzione di attivazione per ottenere l'output di quel particolare livello e fornirlo come input allo strato successivo. Come sottolineato in [3] esistono diversi tipi di funzioni di attivazione e l'accuratezza della rete neurale dipende anche dalla scelta della funzione più efficace. Non esiste un manuale standard sulle funzioni di attivazione da utilizzare per ogni applicazione, tuttavia sono presenti alcuni esempi sulle funzioni più utilizzate per alcune tipologie di caso studio, in quanto saranno poi i test a verificarne l'effettiva funzionalità. Si evidenziano di seguito le tipologie più utilizzate.

1.3.2.1 Sigmoide

La sigmoide, mostrata in Figura 1.7 è una delle funzioni non lineari più utilizzate. Essa restituisce valori compresi tra 0 e 1. Inoltre, la funzione sigmoidea non è simmetrica rispetto all'asse x quindi tutti i valori di output dei neuroni avranno segno solamente positivo. La sua espressione matematica è descritta nella Equazione 1.1

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

Equazione 1.1 Funzione sigmoide

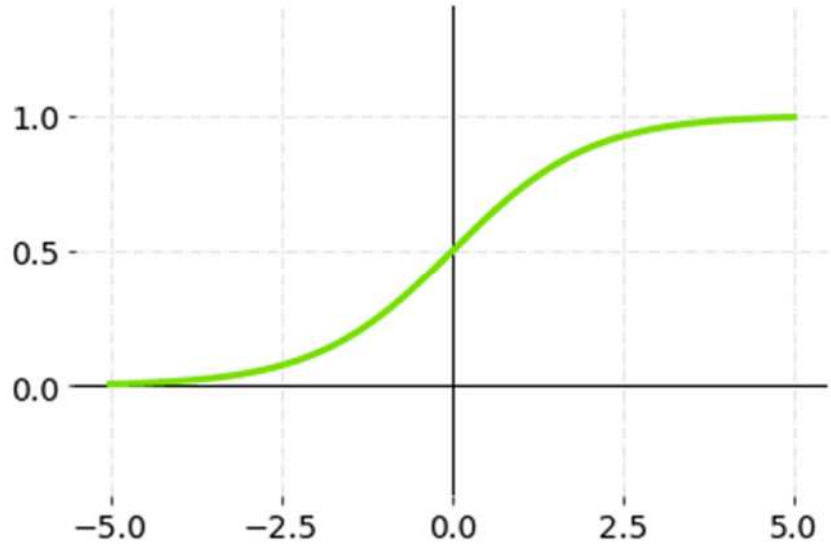

Figura 1.7 Grafico della funzione sigmoide

1.3.2.2 Softmax

La funzione *softmax*, mostrata in Figura 1.8, è una combinazione di più funzioni sigmoidali e può essere utilizzata per esprimere la probabilità relativa che il dato in ingresso appartenga ad una determinata classe. Rappresenta infatti una delle funzioni più utilizzate nei problemi di classificazione multiclasse. In questo caso, lo strato di output avrà lo stesso numero di neuroni delle classi da riconoscere in fase di classificazione e in uscita avremo la probabilità di appartenenza a ciascuna classe. La classe con la probabilità più alta rappresenterà la previsione della rete. Matematicamente può essere definita dalla Equazione 1.2

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad per\ j = 1, \dots, K$$

Equazione 1.2 Funzione softmax

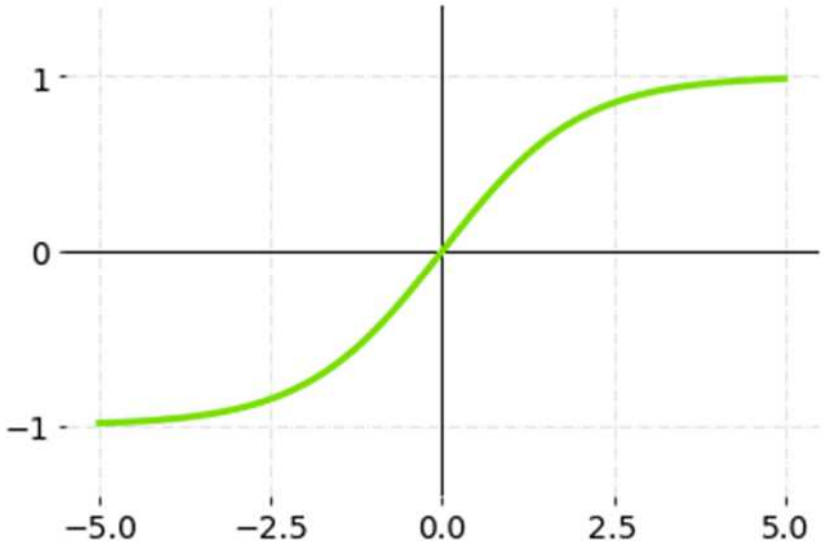

Figura 1.8 Grafico della funzione softmax

1.3.2.3 ReLU

La funzione ReLU, mostrata in Figura 1.9, sta per *Rectified Linear Unit* ed i valori che assume sono compresi tra 0 e infinito. Settando a zero tutti i valori negativi, solo un certo numero di neuroni viene attivato contemporaneamente e risulta più efficiente dal punto di vista computazionale. Si utilizza principalmente nei livelli nascosti della rete neurale. La sua espressione matematica è descritta nella Equazione 1.3

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

Equazione 1.3 Funzione ReLU

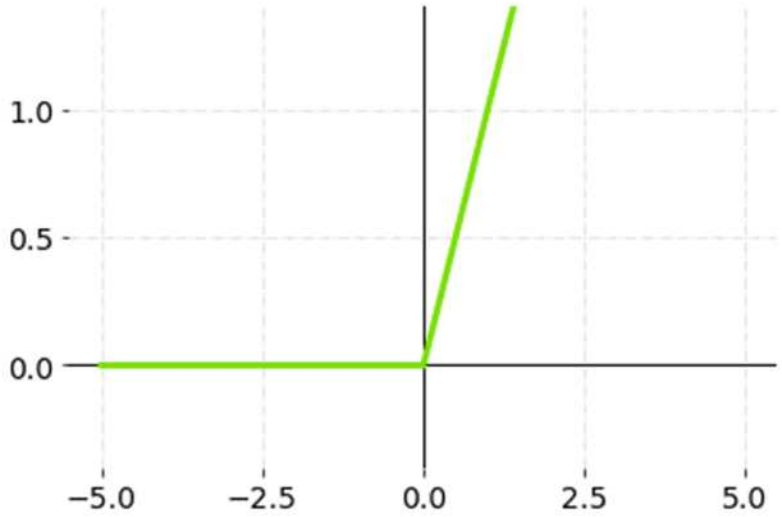

Figura 1.9 Grafico della funzione ReLU

1.3.3 Loss function

In *machine learning* e in particolare nelle reti neurali, la funzione di perdita o funzione di costo (*loss function*) è una misura della discrepanza tra la previsione della rete e l'output atteso. L'obiettivo della rete neurale durante la fase di addestramento è quello di minimizzare la funzione di perdita per avvicinarsi il più possibile all'output atteso.

Nell'apprendimento supervisionato, esistono due tipi principali di funzioni di perdita, correlate ai rispettivi modelli di reti neurali:

- Funzioni di perdita di regressione

Si utilizzano nelle reti neurali di regressione. Dato un valore di ingresso, il modello predice un valore di uscita corrispondente. Tra le funzioni di perdita si evidenziano l'errore quadratico medio (*Mean Squared Error*) e l'errore assoluto medio (*Mean Absolute Error*)

- Funzioni di perdita di classificazione

Nelle reti neurali di classificazione dato un input, la rete neurale produce un vettore di probabilità di appartenenza a varie categorie preimpostate e si può quindi selezionare la categoria con la più alta probabilità di appartenenza. Tra le *loss function* più utilizzate si trovano la *Binary Cross-Entropy* e la *Categorical Cross-Entropy*

1.3.3.1 Cross Entropy Loss

Ogni probabilità di classe prevista viene confrontata con l'effettivo risultato desiderato di classe 0 o 1 e viene calcolato un punteggio/perdita che penalizza la probabilità in base alla distanza dal valore effettivo previsto.

La penalizzazione è di tipo logaritmico e dà luogo a un punteggio elevato per le grandi differenze vicine a 1 e a un punteggio ridotto per le piccole differenze che tendono a 0. L'obiettivo è quello di minimizzare la perdita, vale a dire che quanto più piccola è la perdita tanto migliore è il modello. Un modello perfetto ha una *cross entropy loss* pari a 0.

1.3.3.2 Binary Cross Entropy

Come evidenziato nelle conclusioni in [4] la *binary cross entropy* è una *loss function* utilizzata nei modelli di classificazione binaria. Risulta molto efficace per allenare un modello a risolvere più problemi di classificazione allo stesso tempo, se ogni classificazione piò essere ridotta ad una scelta binaria. La sua espressione matematica è descritta nella Equazione 1.4

$$Cross\ Entropy\ Loss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)$$

Equazione 1.4 Cross Entropy Loss

Nella classificazione binaria, ci sono solo due possibili valori effettivi di y (0 o 1), quindi per calcolare la perdita tra i valori effettivi e quelli previsti, è necessario confrontare il valore effettivo con la probabilità che l'input si allinei a quella categoria

p_i = probabilità che la categoria sia 1

$1 - p_i$ = probabilità che la categoria sia 0

1.3.3.3 Categorical Cross Entropy

Nei casi in cui il numero di classi è superiore a due, utilizziamo la *categorical cross entropy*, descritta dalla Equazione 1.5, che segue un processo molto simile alla *binary cross entropy*

$$Cross\ Entropy\ Loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot \log(p_{ij})$$

Equazione 1.5 Categorical Cross Entropy Loss

La *binary cross entropy* è un caso speciale della *categorical cross entropy* dove M = 2 (il numero di categorie è 2)

1.3.4 Backpropagation

La *backpropagation* è oggi lo strumento più utilizzato nel campo delle reti neurali. Come evidenziato in [5] al centro della *backpropagation* c'è un metodo per calcolare le derivate in modo esatto ed efficiente in qualsiasi grande sistema costituito da sottosistemi o calcoli elementari che sono rappresentati da funzioni note e differenziabili.

La *backpropagation* viene utilizzata durante l'addestramento dei modelli di rete neurale per calcolare il gradiente per ciascun peso nel modello di rete. Il gradiente viene quindi utilizzato da un algoritmo di ottimizzazione per aggiornare i pesi del modello.

1.3.5 Optimizer

Gli algoritmi di ottimizzazione, definiti anche *optimizer*, sono un aspetto critico nell'addestramento delle reti neurali. Il loro obiettivo principale è infatti quello di minimizzare la *loss function* che, come visto in precedenza, rappresenta la discrepanza tra i valori previsti dalla rete e i valori desiderati. Si intuisce quindi che una buona ottimizzazione migliora l'accuratezza della rete.

1.3.5.1 SGD

La discesa stocastica del gradiente, o SGD (*Stochastic Gradient Descent*) è uno degli algoritmi di ottimizzazione più utilizzato per l'allenamento delle reti neurali profonde. Come evidenziato in [6] l'algoritmo si comporta bene in molte varietà di applicazioni ma ha anche solide basi teoriche.

Addestrare la rete neurale significa risolvere il problema nella Equazione 1.6

$$\min_{w \in \mathbb{R}^n} f(w)$$

Equazione 1.6 Matematica dell'addestramento di una rete neurale

dove *f* è la *loss function*.

Le iterazioni possono essere descritte dalla Equazione 1.7

$$w_k = w_{k-1} - \alpha_{k-1} \nabla f(w_{k-1})$$

Equazione 1.7 Iterazioni

dove w_k denota la k-esima iterazione, α_k è la dimensione del passo, definita anche tasso di apprendimento o *learning rate* e $\nabla f(w_k)$ rappresenta il gradiente calcolato in w_k.

L'algoritmo di ottimizzazione stima il gradiente dell'errore per lo stato corrente del modello e aggiorna poi i pesi attraverso un algoritmo di retropropagazione dell'errore. Il *learning rate* stabilisce di quanto debbano essere aggiornati i pesi (*weights*). Più nello specifico, questo iper-parametro ha un valore scalare positivo, spesso compreso tra 0 e 1.
Durante l'apprendimento, l'algoritmo di retropropagazione dell'errore calcola di quanto ciascun nodo della rete sia responsabile dell'errore sulla previsione, ed invece di aggiornare i pesi con il valore totale dell'errore, questo è scalato dal *learning rate*. Ad esempio, significa che con un valore di 0.1 i pesi sono aggiornati di un ammontare pari al 10% dell'errore calcolato.

Uno svantaggio dell'SGD è che scala il gradiente in modo uniforme in tutte le direzioni: il *learning rate* è un parametro che va fissato dall'utente. Questo può essere dannoso per problemi di piccola portata ed inoltre il processo di regolazione del suo valore può essere particolarmente complicato.

1.3.5.2 Adam

Sono stati proposti dei metodi alternativi, chiamati adattivi, per sopperire al problema del *learning rate*. Questi metodi scalano il gradiente diagonalmente tramite un vettore di tassi di apprendimento, uno per ogni parametro, che vengono adattati man mano che il training avanza. Tra questi metodi troviamo l'Adam (*Adaptive Moment Estimation*).

1.3.6 Regularization – Overfitting

Uno dei problemi più comuni che si possono incontrare durante l'addestramento delle reti neurali è il cosiddetto *overfitting*. Quando si è in presenza di *overfitting* si applicano delle tecniche, chiamate tecniche di regolarizzazione, che apportano leggere modifiche all'algoritmo di addestramento in modo che il modello riesca a generalizzare meglio le informazioni sui dati in ingresso e restituisca risultati migliori in fase di test.

Come analizzato in [7] un problema nell'apprendimento supervisionato può essere dato dal fatto che l'algoritmo si adatti troppo ai dati di addestramento, ottenendo degli ottimi risultati in fase di *train*, ma quando viene testato su dati che non ha mai visto fornisce dei pessimi risultati. Questo fenomeno prende il nome di *overfitting* e può dipendere da diversi fattori:

- Il training set potrebbe essere troppo piccolo o poco vario; quindi, il modello non riesce a trovare una regola sufficientemente generalizzata.
- Il modello è troppo complesso, quindi trova più facile "aderire" ai singoli esempi piuttosto che determinare una regola di riconoscimento generalizzata.

Il modello memorizza quindi i vari aspetti dei dati in ingresso, anche gli eventuali rumori, e non coglie effettivamente le caratteristiche principali delle varie categorie. Tutto ciò si riconduce alla fase di test, quando saranno forniti input mai analizzati prima e l'algoritmo non sarà in grado di classificarli con precisione.

1.3.6.1 Dropout

Il *dropout*, come descritto in [8], è una tecnica di regolarizzazione che si basa sull'esclusione di alcuni neuroni durante l'addestramento, costringendo l'algoritmo a diventare più "intelligente" e a non adattarsi ai soliti dati in ingresso. La probabilità di scegliere quanti nodi eliminare è il parametro da settare della funzione di *dropout*. Lo schema di una rete con e senza applicazione di *dropout* è mostrato nella Figura 1.10

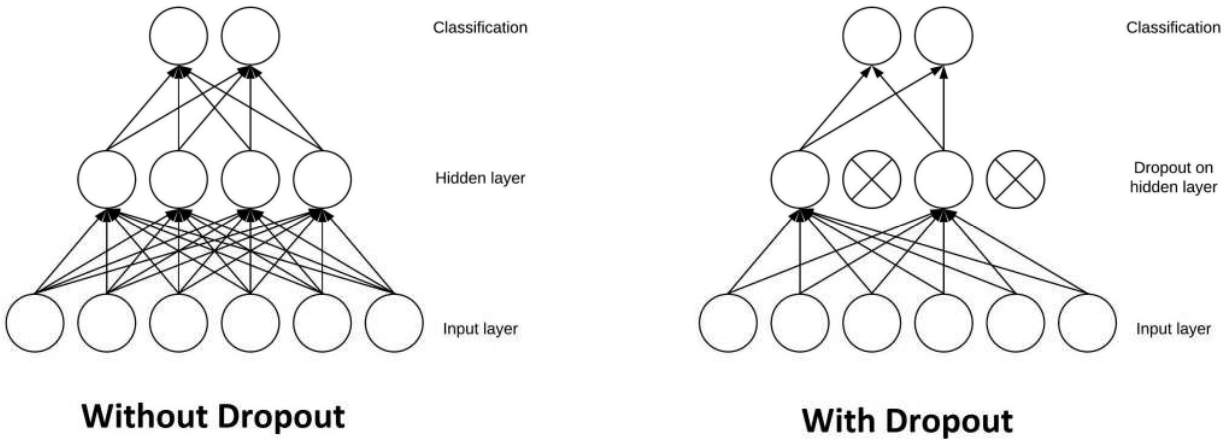

Figura 1.10 Schema di una rete neurale senza e con applicazione del dropout

1.4 Reti Neurali Convoluzionali

Le reti neurali convoluzionali (*Convolutional Neural Network*) sono analoghe alle ANN tradizionali in quanto sono costituite da neuroni che si auto-ottimizzano attraverso l'apprendimento. Riprendendo l'articolo [1] la differenza sostanziale è che le reti convoluzionali vengono utilizzate principalmente nel campo del riconoscimento di *pattern* all'interno delle immagini.

I neuroni che ricoprono gli strati all'interno della CNN sono organizzati in tre dimensioni, la dimensione spaziale dell'input (altezza e larghezza) e la profondità. A differenza delle reti standard, i neuroni all'interno di un dato strato si connetteranno solo a una piccola regione dello strato che li precede.

1.4.1 Architettura CNN

Le reti convoluzionali sono composte da tre tipologie di *layer* caratteristici in successione:

- ***Convolutional Layer***

Il *layer* convoluzionale è il più importante e definisce anche il nome della tipologia di rete. Il livello di input contiene i valori dei pixel dell'immagine; lo strato convoluzionale, come mostrato in Figura 1.11, attraverso l'applicazione di uno o più filtri, o *kernels*, di dimensione fissata, restituirà una nuova immagine. I pixel della nuova immagine, definita *feature map*, conterranno la somma della moltiplicazione dei valori dei pixel nell'immagine di input con i valori nel filtro. Ogni filtro crea una *feature map* che evidenzia una particolare caratteristica dell'immagine. L'output di ogni filtro rappresenta la presenza o l'assenza di quella caratteristica nell'immagine.
Come mostrato nella Figura 1.12 si utilizza solitamente una funzione di attivazione prima dell'output.

I parametri che vanno fissati per la costruzione di un *layer* convoluzionale sono:

1. *filters* = 24 numero di filtri

2. *kernel_size* = (3,3) dimensione del filtro 3x3
3. *strides* = (1, 1) rappresenta il passo con cui spostiamo il filtro sull'immagine
4. *padding* = "same" il riempimento del bordo, con eventuali zeri, serve per controllare la dimensione di uscita, "same" significa che avremo la stessa dimensione in uscita
5. *activation* = 'ReLU' eventuale funzione di attivazione

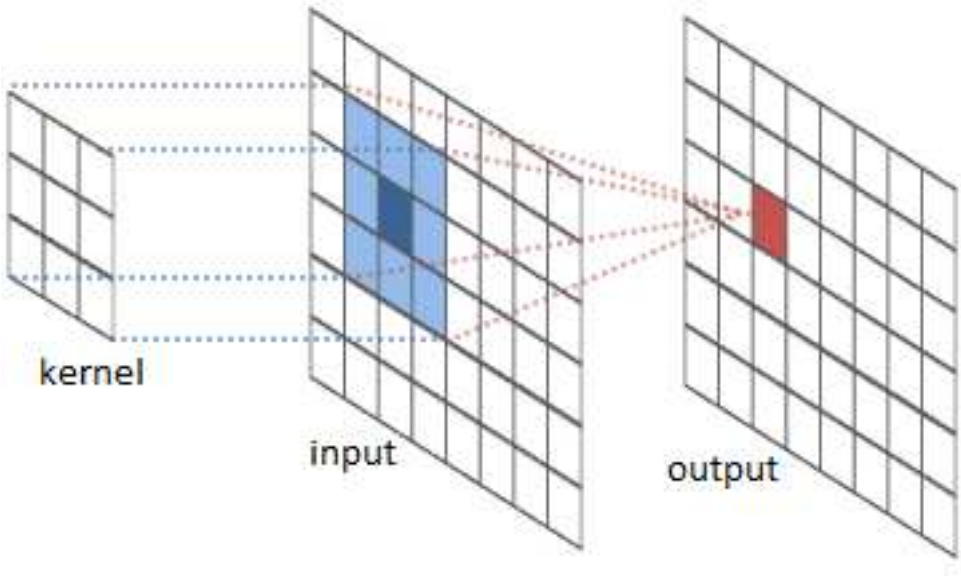

Figura 1.11 Applicazione del filtro sull'input ed evidenziazione del pixel di uscita della feature map

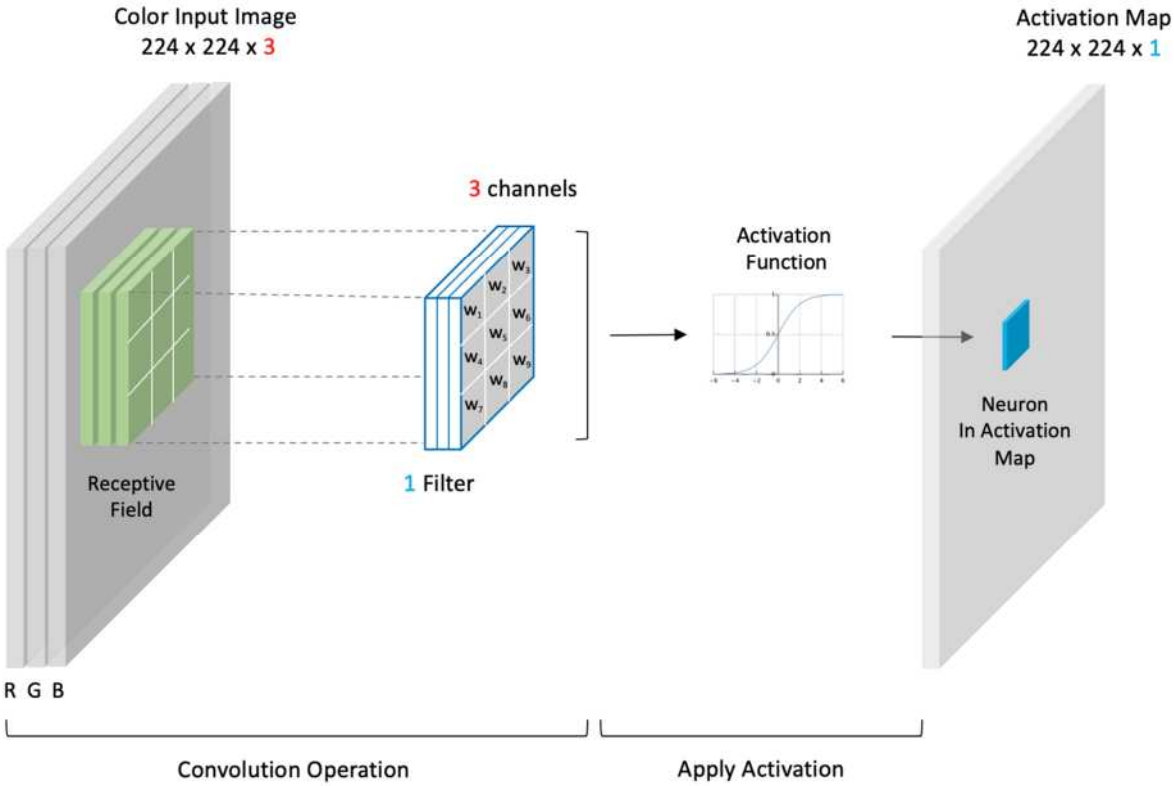

Figura 1.12 Operazione di convoluzione in RGB con funzione di attivazione

- ***Pooling Layer***

Dopo un *layer* convoluzionale si utilizza un *layer* di *pooling*. Questa operazione ha diverse ragioni importanti, in particolare riduce le dimensioni dell'immagine in input, riducendo quindi la complessità computazionale della rete e il numero di parametri da allenare, il che rende il modello più efficiente.

Si può così accelerare il processo di apprendimento del modello, poiché le informazioni diventano più concentrate ed esso apprende più rapidamente le relazioni tra le caratteristiche, riducendo il rumore o le variazioni minori che potrebbero non essere significative per il problema di classificazione o analisi dell'immagine.

Il *pooling* introduce inoltre una certa invarianza spaziale, il che significa che il modello sarà in grado di riconoscere caratteristiche rilevanti anche se si trovano in posizioni leggermente diverse nell'immagine, importante per la capacità di generalizzazione del modello.

I tipi di *pooling* più comuni sono il *max pooling* e *l'average pooling*, che estraggono rispettivamente il valore massimo o la media da una regione di input. Questi metodi sono spesso utilizzati in successione ai *layer* convoluzionali per estrarre progressivamente le caratteristiche rilevanti dalle immagini.

L'esempio di *max pooling* riportato in Figura 1.13 utilizza una matrice 2x2 e scansiona le *feature map* estraendo ogni volta per ognuno dei 4 pixel che analizza contemporaneamente (2x2), il valore massimo. Inserisce questo valore in un pixel di output e poi si sposta del passo impostato ed effettua una nuova scansione fino a coprire tutta l'immagine di input. Il passo (*stride*) impostato è 2x2, quindi il filtro si sposterà di due pixel a destra e due in basso, senza creare sovrapposizioni.

Si genererà una nuova matrice di dimensione dimezzata, riducendo quindi la matrice 4x4 in una 2x2 senza perdere però le *features* identificate dal *layer* convoluzionale.

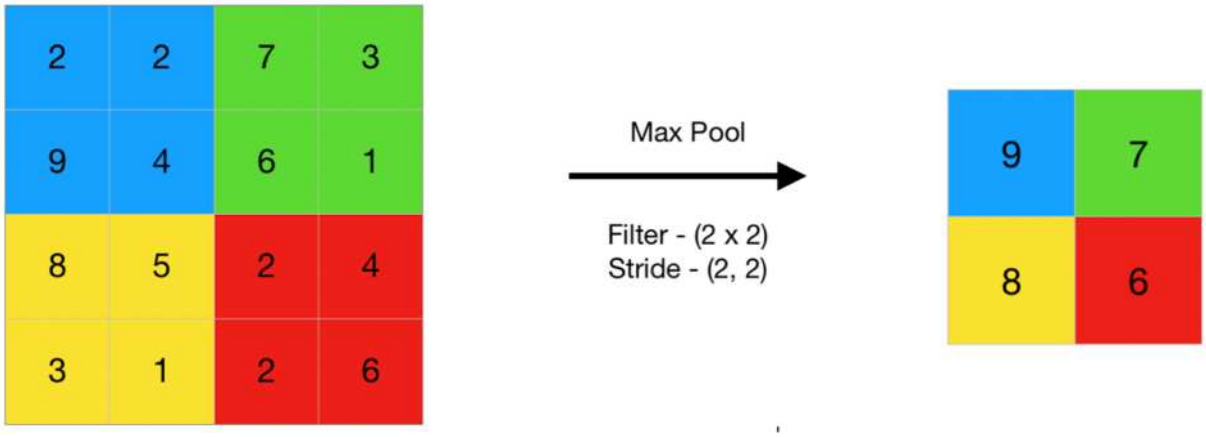

Figura 1.13 Operazione di max pooling

- ***Flatten & Dense/Fully-Connected Layers***

Il risultato della convoluzione e del *pooling* viene appiattito utilizzando un *flatten layer* collegato poi a uno o più *fully-connected layers*, o *dense layers*, che restituiranno le probabilità per ogni classe da utilizzare per la classificazione. Si suggerisce inoltre di utilizzare la ReLU in questi strati, in modo da migliorare le prestazioni.

1.5 Applicazioni

L'intelligenza artificiale ha un'ampia gamma di campi di applicazione in diversi settori come illustrato nell'articolo [9]:

- **Medicina**

L'AI può aiutare nella diagnosi di malattie, nell'analisi delle immagini mediche (come la radiologia e la tomografia computerizzata), nella gestione dei dati dei pazienti e nello sviluppo di farmaci. Può migliorare la gestione delle cure sanitarie, la programmazione delle risorse e l'ottimizzazione dei flussi di lavoro ospedalieri.

- **Automotive**

Fondamentale per veicoli autonomi e sistemi avanzati di assistenza alla guida (ADAS), che includono funzioni come il riconoscimento dei segnali stradali, la guida autonoma e la prevenzione degli incidenti.

- **Finanza**

Utilizzata per l'analisi dei mercati finanziari, la gestione del rischio, la prevenzione delle frodi e l'automazione di alcune attività bancarie come l'analisi dei dati finanziari, la gestione dei portafogli e l'automazione dei servizi al cliente.

- **E-commerce**

Molte piattaforme di e-commerce utilizzano l'AI per migliorare le raccomandazioni di prodotti, l'analisi dei dati sugli acquisti dei clienti e la gestione delle scorte.

- **Ricerca scientifica**

L'AI è impiegata in una vasta gamma di discipline scientifiche per l'analisi dei dati, la simulazione di processi complessi e la scoperta di nuovi materiali o farmaci.

- **Industria manifatturiera**

Utilizzata per migliorare la produzione, l'automazione delle fabbriche e il controllo della qualità dei prodotti.

- **Trasporti e logistica**

I dati dei sistemi di trasporto sono molto spesso ampi e complessi.

Gli algoritmi di intelligenza artificiale permettono la simulazione del comportamento dei conducenti dei veicoli, la manutenzione della pavimentazione stradale, il rilevamento e la classificazione dei veicoli, l'analisi dei modelli di traffico e la previsione del flusso ma anche il controllo del trasporto aereo, delle metropolitane e la navigazione automatica delle navi.

- **Linguaggio naturale e assistenti virtuali**

L'AI è alla base di assistenti virtuali come Siri di Apple, Alexa di Amazon e Google Assistant, che comprendono e rispondono alle domande degli utenti.

- **Gaming**

L'AI è utilizzata per creare personaggi più intelligenti e per migliorare l'esperienza di gioco nei videogiochi.

- **Agricoltura**

Si utilizza per l'ottimizzazione della produzione agricola, il monitoraggio delle colture e la gestione delle risorse idriche.

- **Sicurezza**

Implementata nella videosorveglianza per il riconoscimento facciale, il rilevamento di comportamenti sospetti e la prevenzione dei crimini.

- **Informazioni**

Molte volte le informazioni a disposizione sono incomplete ed il processo di analisi di grandi quantità di dati può essere molto complicato. Le reti neurali hanno la capacità di fornire diagnosi automatiche e risolvere problemi. Gli algoritmi complessi e con la capacità di auto-organizzarsi sono in grado di lavorare anche in condizioni di scarsa connessione e sono ampiamente utilizzati ad esempio apparecchiature elettroniche di sistemi militari (sistemi di tracciamento e monitoraggio automatici, sistemi di guida, diagnosi dei guasti e sistemi di allarme)

- ***Pattern recognition***

Il riconoscimento di modelli è il processo di descrizione, identificazione, classificazione ed interpretazione di fenomeni tramite l'elaborazione e l'analisi di varie forme di informazioni che li caratterizzano.

Negli ultimi anni il riconoscimento dei modelli tramite reti neurali ha sostituito gradualmente i metodi tradizionali. Dopo anni di ricerca e sviluppo è diventato l'attuale tecnologia più avanzata ed è applicato ad esempio al riconoscimento dei caratteri, vocale, facciale, delle impronte digitali, al riconoscimento delle immagini.

Si nota che sono davvero tanti i campi in cui sono applicate le tecniche di intelligenza artificiale, molti già con anni di esperienza alle spalle, alcuni di recente sviluppo e altrettanti in fase di ricerca anche grazie agli ingenti investimenti delle grandi aziende hi-tech.

Ricollegandoci all'ultimo punto della *pattern recognition*, una delle applicazioni che riguarda il caso studio è la classificazione di immagine tramite metodi AI ed in particolare la rilevazione di difetti, che saranno trattati di seguito.

1.5.1 AI Classification & Defect Detection

L'intelligenza artificiale è molto utilizzata nell'analisi di immagini. Lo studio in [10] riporta ad esempio lo sviluppo della diagnostica assistita da computer basata sull'intelligenza artificiale (AI) per la diagnosi del cancro della pelle. Grazie ai molti *dataset* sulle lesioni cutanee presenti pubblicamente online i ricercatori hanno sviluppato algoritmi di *deep learning* per distinguere le lesioni maligne da quelle benigne tramite l'analisi delle diverse tipologie di immagini a disposizione.
Sempre in campo medico, lo studio in [11] tratta le tecniche AI utilizzate nel rilevamento e nella classificazione delle immagini mediche della malattia da coronavirus (COVID-19) in termini di valutazione e analisi comparativa.

In riferimento al caso studio invece, si riportano diverse pubblicazioni sulla *defect detection* per entrare poi più nel dettaglio con l'analisi dei difetti del calcestruzzo.

L'articolo [12] presenta la sogliatura automatica (*automatic thresholding*), una tecnica ampiamente utilizzata nel settore della visione artificiale per l'ispezione automatizzata dei difetti. Si legge tra le conclusioni che la tecnica di soglia utilizzata, il metodo Otsu, fornisce risultati soddisfacenti.
Lo studio [13] si occupa invece dei metodi tramite *machine learning* e immagini 3D per il rilevamento delle crepe sulla pavimentazione stradale, data l'importanza della manutenzione e della sicurezza del traffico.

Il problema del riconoscimento dei difetti del calcestruzzo tramite metodi AI è molto trattato in letteratura, in particolare in [14], [15], [16], [17], [18] si discute l'analisi di immagini tramite reti neurali per il riconoscimento e la classificazione dei difetti, con metodi di acquisizione ed elaborazione dei dati differenti.

In [19] si affronta un altro problema, sempre legato alle infrastrutture e alla viabilità, ed è quello della classificazione dei difetti del calcestruzzo durante l'ispezione dei ponti, che rimane un compito soggettivo e laborioso.
Secondo i dati, se diversi ispettori valutano lo stesso difetto, il rischio di ottenere un risultato falso è di circa il 50%. Anche qui l'importanza della rilevazione dei difetti è enorme, dato che la mancata manutenzione – con l'invecchiamento delle infrastrutture ed i budget sempre più ridotti – può portare ad eventi catastrofici, come il crollo del ponte nel 2018 a Genova.

1.6 Difetti del calcestruzzo

La rete neurale dovrà imparare a classificare le immagini che riceve in ingresso tra tre classi preimpostate:

- Immagini senza difetti
- Immagini in cui è presente la *fessurazione*
- Immagini in cui è presente il *pitting*

Si fornisce di seguito una breve descrizione dei due difetti analizzati.

1.6.1 Cracks

Le crepe, come nell'esempio in Figura 1.14, sono uno dei danni caratteristici delle strutture in calcestruzzo, che possono ridurre la capacità portante, la tenuta della struttura e portare a cedimenti. Fessurazioni eccessive ed incontrollate possono causare inoltre la corrosione e l'indebolimento dei rinforzi di acciaio presenti all'interno, oltre ad influire negativamente sull'estetica.

L'articolo [20] fornisce una revisione approfondita delle problematiche legate alla formazione e allo sviluppo di danni e fessurazioni nelle strutture in calcestruzzo. Si concentra sulle cause dell'innesco delle cricche e ne caratterizza le tipologie fondamentali.
Negli elementi in cemento armato le crepe dipendono dalla tipologia dei carichi applicati, mentre le micro-fessure sono correlate ai casi specifici. Va notato che l'argomento delle crepe nel calcestruzzo è importante sotto molti aspetti in quanto influenza la durabilità delle infrastrutture e degli edifici e di conseguenza la sicurezza delle persone.

Figura 1.14 Crepa superficiale nel calcestruzzo

1.6.2 Pitting

Il *pitting* o *bugholes*, in Figura 1.15, è un difetto caratterizzato da vuoti superficiali che si formano dalla migrazione dell'aria intrappolata nella cassaforma di cemento verso l'esterno. Sono analizzati in [21] i fattori che ne influenzano la formazione, come le proporzioni del mix e la tecnologia di costruzione. I risultati mostrano che la quantità di buchi e il diametro massimo possono essere ridotti controllando il rapporto W/C (acqua-cemento), il contenuto di super fluidificante, il rapporto della sabbia e il contenuto di ceneri volanti.

Rispetto alla cassaforma in legno, le superfici del calcestruzzo che utilizzano casseforme in acciaio e in PVC hanno una quantità maggiore di buchi e un rapporto delle aree inferiore, ma l'influenza sul diametro massimo dei buchi non è significativo. I calcestruzzi con diverse caratteristiche necessitano di tempi di vibrazione differenti per ridurre la formazione di pitting sulla superficie, ma anche con vibrazioni prolungate, il fenomeno è difficile da eliminare completamente.

Figura 1.15 Superficie con pitting

2 Materiali e Metodi

In questa sezione si descrive il primo approccio alla programmazione in *Python* tramite l'utilizzo di *Google Colab* e delle librerie *TensorFlow* e *Keras* per la costruzione di una rete neurale. Si prosegue poi con lo studio per la creazione del *dataset* e l'ultimo step prevede la definizione dei parametri della rete e la sua struttura finale.

2.1 Google Colab e TensorFlow

Colaboratory, o in breve "*Colab*", è un prodotto di *Google Research*. *Colab* permette a chiunque di scrivere ed eseguire codice *Python* tramite il browser ed è particolarmente adatto per *machine learning*, analisi dei dati e formazione. Più tecnicamente *Colab* è un servizio di blocchi note *Jupyter* ospitato che fornisce l'accesso a potenze di calcolo su cloud, incluse GPU e TPU, che si rendono necessarie quando aumenta la complessità del modello.

TensorFlow è una libreria *open source* sviluppata da *Google*, molto utilizzata per applicazioni di *deep learning*, ma supporta anche l'apprendimento automatico tradizionale. Accetta dati sotto forma di *array* multidimensionali di dimensioni superiori chiamati tensori, che sono molto utili nella gestione di grandi quantità di dati.

Keras è l'*API* di alto livello scritta in *Python* della piattaforma *TensorFlow*. Fornisce un'interfaccia accessibile e altamente produttiva per la risoluzione di problemi di *machine learning*, con particolare attenzione al moderno *deep learning*. *Keras* copre ogni fase del flusso di lavoro di *machine learning*, dall'elaborazione dei dati all'ottimizzazione degli iper-parametri, fino alla distribuzione. È stato sviluppato con l'obiettivo di consentire una rapida sperimentazione.

2.2 Dataset

L'apprendimento della rete neurale si basa sull'analisi dei dati forniti in ingresso, che verranno poi elaborati dalla rete per ottenere il risultato desiderato. Si può capire facilmente quindi come i dati in ingresso siano di fondamentale importanza.
Tali dati vengono raccolti in un *dataset* che, nel caso studio, è una cartella contenente tutte le immagini raccolte.

2.2.1 Raccolta delle immagini

Il caso studio si concentra sulla classificazione dei difetti del calcestruzzo tra tre categorie principali:

- Crepe (*Cracks*)
- *Pitting*
- Immagini senza difetti

Le immagini raccolte all'interno del *dataset* sono principalmente immagini scattate con diversi dispositivi mobili, immagini presenti in alcuni *dataset* pubblici in letteratura e immagini scaricate da internet con l'ausilio di *SerpAPI.*

SerpAPI (*Search Engine Results Page*) è una API (*Application Programming Interface*) fornita da *Google* che permette di fare *scraping* attraverso la *Google Search*, ovvero permette di estrarre dati dalle pagine web per poi raccoglierli in database locali. Lo *script*, eseguito tramite *Colab* salva tutti i risultati della ricerca tramite una o più parole chiave (*query*) nella cartella di destinazione selezionata. Alcune delle immagini scaricate potrebbero essere bianche o generare file corrotti, sono necessarie quindi ulteriori operazioni prima di fornire il *dataset* alla rete.
In Figura 2.1, Figura 2.2 e Figura 2.3 si riportano gli esempi di alcune immagini raccolte per ognuna delle tre categorie:

Figura 2.1 Esempio di 15 immagini della categoria cracks

Figura 2.2 Esempio di 15 immagini della categoria no_defects

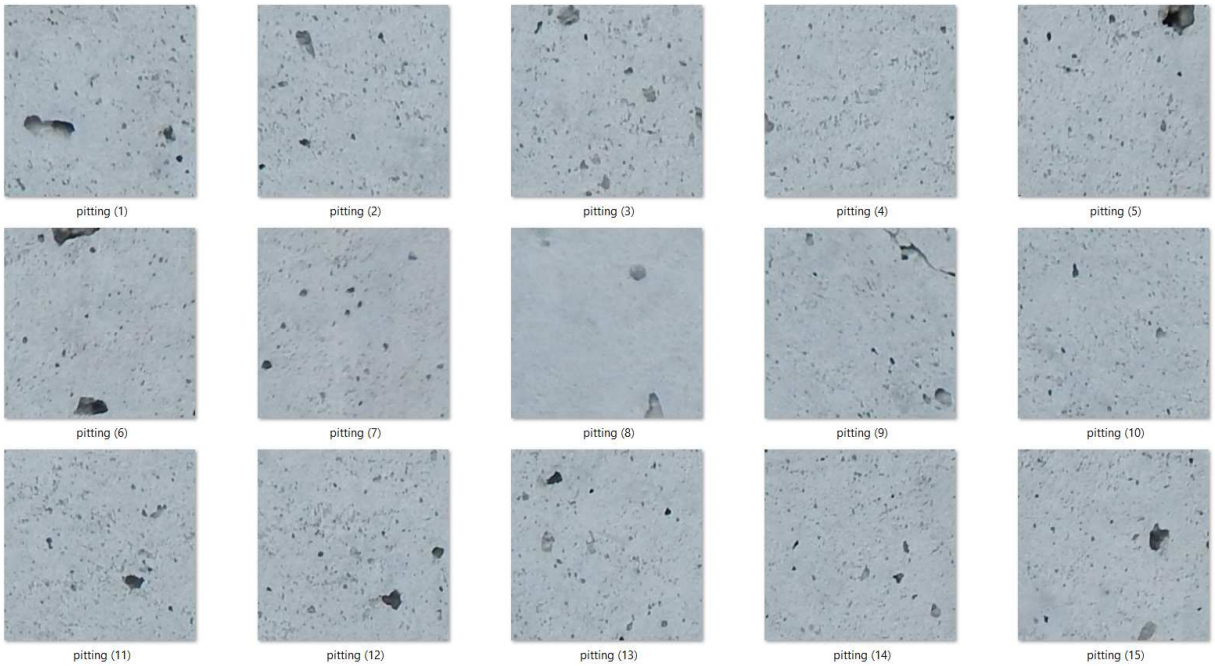

Figura 2.3 Esempio di 15 immagini della categoria pitting

2.2.2 Pulizia e controllo del dataset

Le immagini presenti all'interno del *dataset* possono essere di diversi formati, dimensioni ed è importante che non contengano errori o file corrotti. La presenza di questi tipi di file, infatti, restituisce degli errori durante l'esecuzione dell'algoritmo.
Si rende quindi necessaria una fase di pulizia prima di fornire il *dataset* alla rete per l'addestramento. Questa fase viene eseguita con un piccolo algoritmo che riceve le immagini in ingresso ed esegue il controllo delle estensioni per far sì che non ci siano problemi durante l'utilizzo del *dataset* nella rete. Se durante l'analisi si evidenziano problemi con alcuni file, l'algoritmo ne segnala il percorso così da permettere l'eliminazione da parte dell'utente.

2.2.3 *Patchify*

Talvolta si può avere la necessità di suddividere le immagini ad alta risoluzione in porzioni più piccole, tutte della stessa dimensione. Questo può accadere sia per aumentare il numero di immagini a disposizione per l'allenamento, sia per migliorare le prestazioni della rete fornendo immagini più piccole. In *python* esiste una libreria denominata *patchify* che permette di dividere le immagini in piccole porzioni (*patch*) fornendo la dimensione della patch ed il passo, come nell'esempio in Figura 2.4.

Il passo (*step*) definisce la distanza tra una patch e quella successiva (in verticale e in orizzontale). Se il passo è maggiore delle dimensioni della patch non ci sarà sovrapposizione. La libreria permette anche di effettuare l'operazione opposta, ovvero unire le patch e ritornare all'immagine originale. Alcune delle immagini presenti nei *dataset* in letteratura sono ottenute tramite questa tecnica.

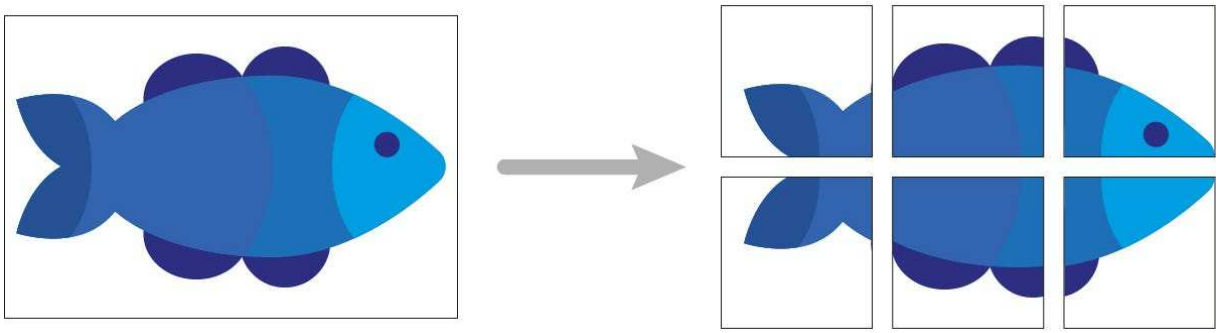

Figura 2.4 Esempio di suddivisione in patch senza sovrapposizione

2.2.4 Split Dataset

Prima di fornire il *dataset* alla rete, l'ultima fase è quella della suddivisione delle immagini in tre categorie: *train (allenamento)*, *validation* (validazione) *e test* (verifica). Questa fase può essere eseguita tramite uno script esterno alla rete che carica il *dataset* e lo restituisce suddiviso in tre gruppi in una nuova *directory*. I parametri da impostare sono le percentuali per definire le proporzioni per generare la divisione. Nell'esempio si prendono come riferimento le percentuali utilizzate in letteratura, che rappresentano un buon bilanciamento. Si è utilizzata la terna (0.8, 0.1, 0.1) che indica su un totale di 11.461 immagini presenti nel *dataset*, l'80% assegnate per l'allenamento, il 10% per la validazione ed il restante 10% per il *test* restituendo i valori per ogni classe mostrati in Figura 2.5.

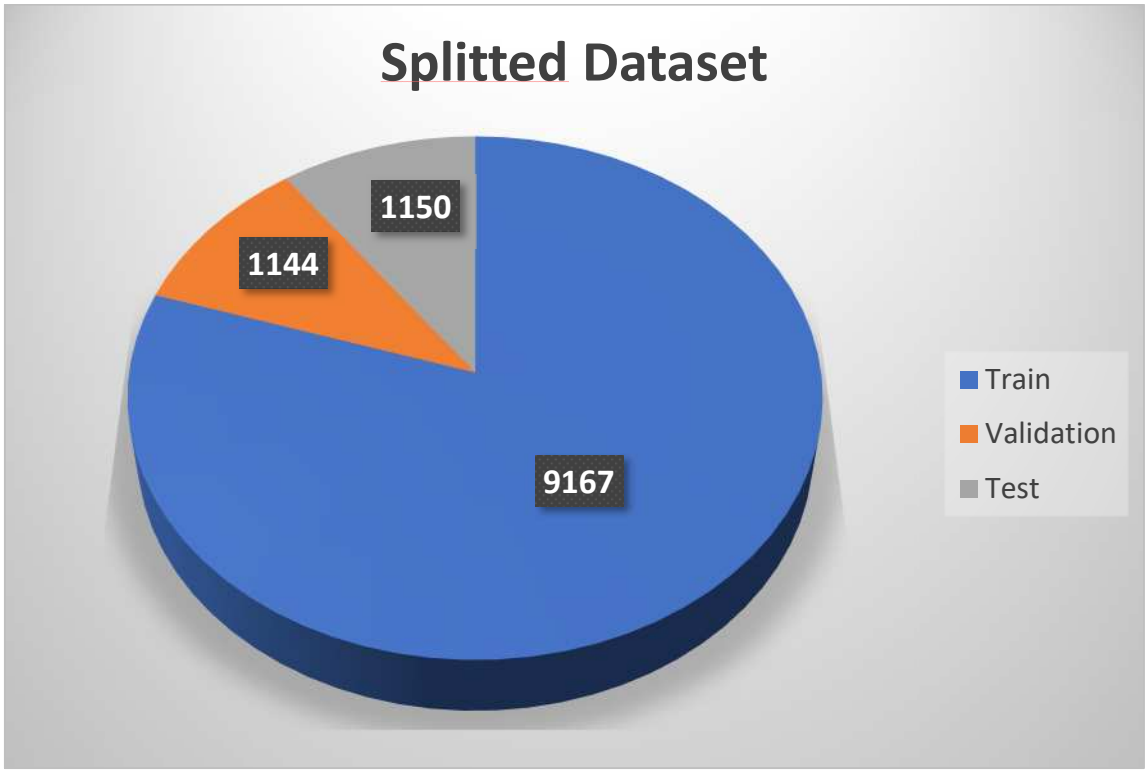

Figura 2.5 Composizione del dataset splittato tramite script esterno

Una seconda possibilità prevede di far eseguire lo split direttamente all'interno dell'algoritmo della rete. Si carica il *dataset* in formato zip tramite il collegamento a Google Drive e l'algoritmo esegue prima l'unzip e poi suddivide il *dataset* sempre seguendo i parametri impostati dall'utente.

2.3 Struttura Rete Neurale

L'algoritmo completo per la costruzione della rete neurale è composto da diverse operazioni in successione, di seguito si analizzano le fasi principali:

1. Importazione librerie
2. Importazione *dataset*
3. Definizione architettura
4. Compilazione del modello
5. Allenamento
6. Test
7. Predizione

2.3.1 Librerie

Nella prima fase si importano tutte le librerie necessarie, in particolare *Tensorflow*, *Keras*, *Numpy*, *Matplotlib* e si collega il *Drive* di *Google* per l'accesso al *dataset.*

2.3.2 Importazione dataset

La seconda fase è quella di importare il *dataset* all'interno della rete e di suddividerlo. Questa fase, come visto in precedenza, può essere eseguita sia tramite script esterno alla rete, sia all'interno dell'algoritmo. L'obiettivo comune è quello di ottenere le immagini suddivise in *train dataset* per l'allenamento, *validation dataset* per la validazione ed il *test dataset* per il test finale.

2.3.3 *Definizione architettura*

La terza fase è quella della definizione vera e propria dell'architettura. Si aggiungono al modello in sequenza i *layer* che faranno parte della rete. Per analizzare la sequenza di *layer* completa utilizzata possiamo dividerla in due blocchi:

1. Il primo blocco, mostrato in Figura 2.6, è quello adibito all'estrazione delle *features* dalle immagini, ed è composto da tre sotto blocchi convoluzione-pooling in successione. Passando attraverso i tre *layer* convoluzionali si mantengono gli stessi parametri e si aumenta solamente il numero dei filtri, rispettivamente 16, 32 e 64.

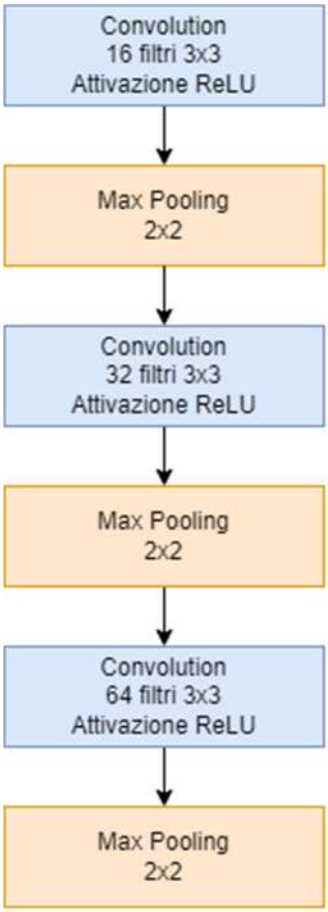

Figura 2.6 Blocco di apprendimento delle features

2. Il secondo blocco, mostrato in Figura 2.7, è quello di classificazione. Si fornisce il risultato della convoluzione ad un *layer* di *dropout*, il cui parametro 0.2 definisce che si lavorerà ogni volta con il 20% di neuroni in meno all'interno dello strato, "spenti" in maniera casuale. Si appiattisce poi il risultato ottenuto dai *layer* convoluzionali attraverso il *flatten layer*, si ha poi un *layer dense* da 128 neuroni con attivazione ReLU ed infine l'ultimo *layer dense* di output con 3 neuroni (uno per ogni categoria) con la funzione di attivazione S*oftmax*.

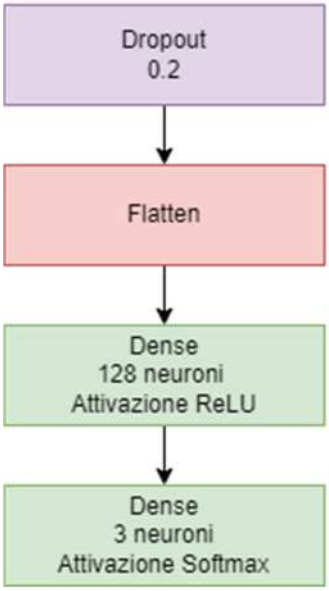

Figura 2.7 Blocco di classificazione

I due blocchi sono poi messi in successione ottenendo così l'architettura completa della rete neurale, mostrata in Figura 2.8.

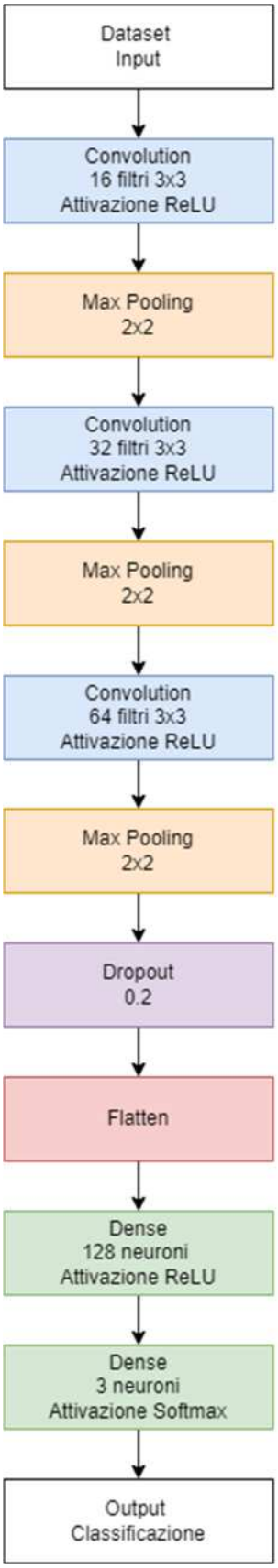

Figura 2.8 Architettura completa della rete neurale

2.3.4 Compilazione modello

Nella fase di compilazione del modello si definiscono gli *optimizer* e la *loss function*, il *learning rate* e si indica la funzione per valutare le performance del modello. Per valutare la bontà dei risultati si misura la distanza del risultato ottenuto dal sistema rispetto al risultato atteso. Per fare questo entrano in gioco le metriche, degli strumenti di natura matematica e/o statistica che consentono di valutare quanto la risposta del sistema sia lontana o vicina alla risposta corretta. La scelta della giusta metrica è un fattore importante per ottenere un risultato attendibile ed anche per velocizzare il processo di addestramento.

Una delle metriche più utilizzate, la *accuracy*, viene in genere determinata dopo che i parametri sono stati appresi. Si contano gli esempi correttamente elaborati e viene calcolata l'accuratezza come rapporto tra le immagini correttamente elaborate e il totale delle immagini inviate al modello.

2.3.5 Training

La fase di allenamento è la parte più importante della costruzione del modello in uscita dalla rete neurale. Qui vengono fornite all'architettura le seguenti informazioni: le immagini del *dataset* che devono essere utilizzate per l'allenamento (*train_images*), le immagini per la validazione (*val_images*), il numero di epoche (*initial_epochs*) che rappresenta quante volte l'intero training set è sottoposto al modello ed il *batch_size*, che definisce il numero di immagini selezionate da analizzare prima di ogni aggiornamento dei pesi.

La fase di addestramento solitamente è costituita da più epoche, a volte anche molto numerose, ma non sempre aumentandone il numero il modello migliora.

Solitamente il *train dataset* è troppo grande per essere elaborato tutto in una volta dalla rete; quindi, si suddivide in sottogruppi chiamati *batch*. Il numero di input contenuti in ogni *batch* è chiamato *batch size*. Il *batch size* indica quindi il numero di immagini che vengono fornite alla rete prima di aggiornare il modello, ovvero viene aggiornato più volte durante un'epoca.

Il numero di epoche ed il *batch size* influiscono sulla velocità di addestramento di un modello, ma anche sul suo modo di perfezionarsi. Sia un *batch size* troppo piccolo sia troppo grande possono creare problemi, sarà quindi compito dell'utente regolare correttamente questi parametri. In letteratura, sono indicate solamente delle regole empiriche generali e questi parametri posso differire in base al tipo di applicazione, quindi una buona scelta dei valori sarà possibile solamente a seguito di numerosi test sperimentali.

2.3.6 Test

La fase di test restituisce i valori definitivi di *loss* e *accuracy* sulla parte restante del *dataset* che non è ancora stato mostrato alla rete. Si utilizza lo stesso *batch size* impostato in precedenza.

2.3.7 Prediction

L'ultima fase è stata inserita per eseguire una previsione manuale. Lo script permette di caricare all'interno della rete un'immagine qualsiasi presente in locale e dopo averla mostrata esegue una previsione. La rete restituirà la percentuale relativa di appartenenza ad ogni classe. Selezionando quindi la classe con la maggiore probabilità, si restituisce a schermo il risultato. Questo rappresenta un ulteriore test per verificare l'accuratezza della rete e la capacità di classificare correttamente l'immagine fornita.
Si riporta a titolo di esempio la previsione eseguita dalla rete sull'immagine di una crepa in Figura 2.9 che non è mai stata mostrata alla rete.

Figura 2.9 Immagine di prova con una crepa per la previsione

La Figura 2.10 mostra come risulta l'immagine caricata nella rete

```
# Plottiamo l'immagine caricata
plt.imshow(img)
plt.grid(True)
```

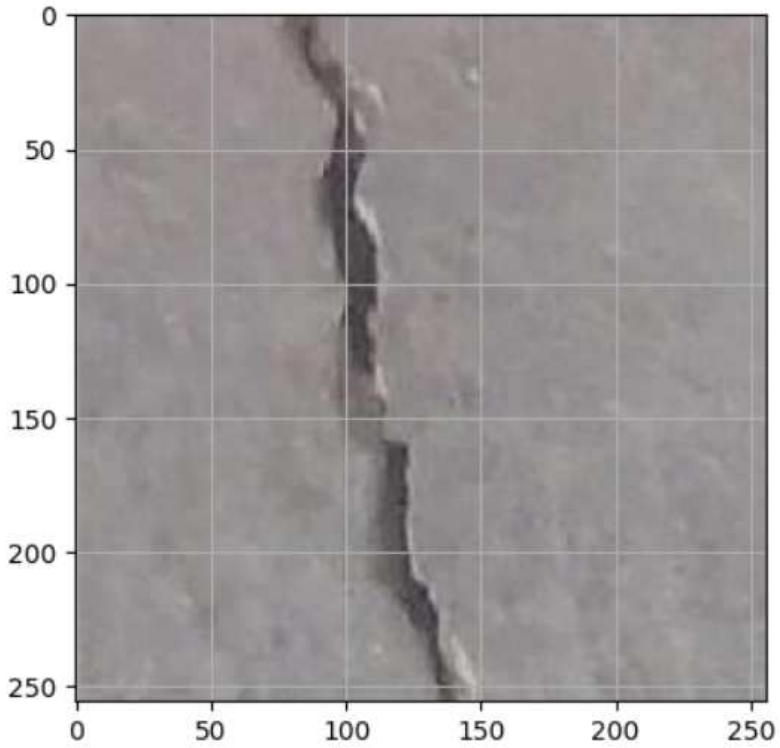

Figura 2.10 Immagine di prova caricata nella rete

Si prosegue con l'esecuzione della previsione, in Figura 2.11

```
# Eseguiamo la previsione
predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
1/1 [==============================] - 0s 19ms/step
[[0.57338333 0.2125185  0.21409819]]
```

```
# Prendiamo il valore massimo, che corrisponderà alla più probabile classe di appartenenza (da 0 a 2)
np.argmax(predictions_single[0])
```

```
0
```

Figura 2.11 Esecuzione della previsione

Si può notare come la previsione restituisce tre valori di probabilità di appartenenza ad ognuna delle tre classi. Tramite il comando successivo si seleziona la classe con la più alta probabilità, ovvero la classe 0 (le tre classi sono numerate da 0 a 2).
Il passo successivo sarà quello di stampare a schermo il risultato della previsione, come mostrato in Figura 2.12.

```
# Evidenziamo le tre categorie
print(train_dataset.class_names)
```

```
['cracks', 'no_defects', 'pitting']
```

```
# Stampiamo a schermo il risultato della previsione
if np.argmax(predictions_single[0]) == 0:
  print('''Nell'immagine si evidenziano crepe''')

if np.argmax(predictions_single[0]) == 1:
  print('''Nell'immagine non si evidenziano difetti''')

if np.argmax(predictions_single[0]) == 2:
  print('''Nell'immagine si evidenziano difetti superficiali legati al pitting''')
```

```
Nell'immagine si evidenziano crepe
```

Figura 2.12 Risultato della previsione

La classe 0 corrisponde alla categoria *cracks* quindi la rete ci restituisce, correttamente, che nell'immagine si evidenziano delle crepe, in quanto viene stampato il messaggio "Nell'immagine si evidenziano crepe".

3 Risultati

In questa sezione si riportano i risultati ottenuti durante la fase dei test necessari per trovare la migliore regolazione dei parametri e la sequenza di *layer* più efficace per il caso studio.

3.1 Architetture

Nel corso della sperimentazione sono state costruite e testate diverse architetture ottenute aggiungendo o togliendo *layer*, tuttavia non tutte hanno eseguito correttamente i processi infatti hanno restituito valori di *loss* in alcuni casi molto superiori ad 1. Si riportano quindi le tre architetture che hanno evidenziato risultati soddisfacenti per il problema di classificazione del caso studio mostrate in Figura 3.1.

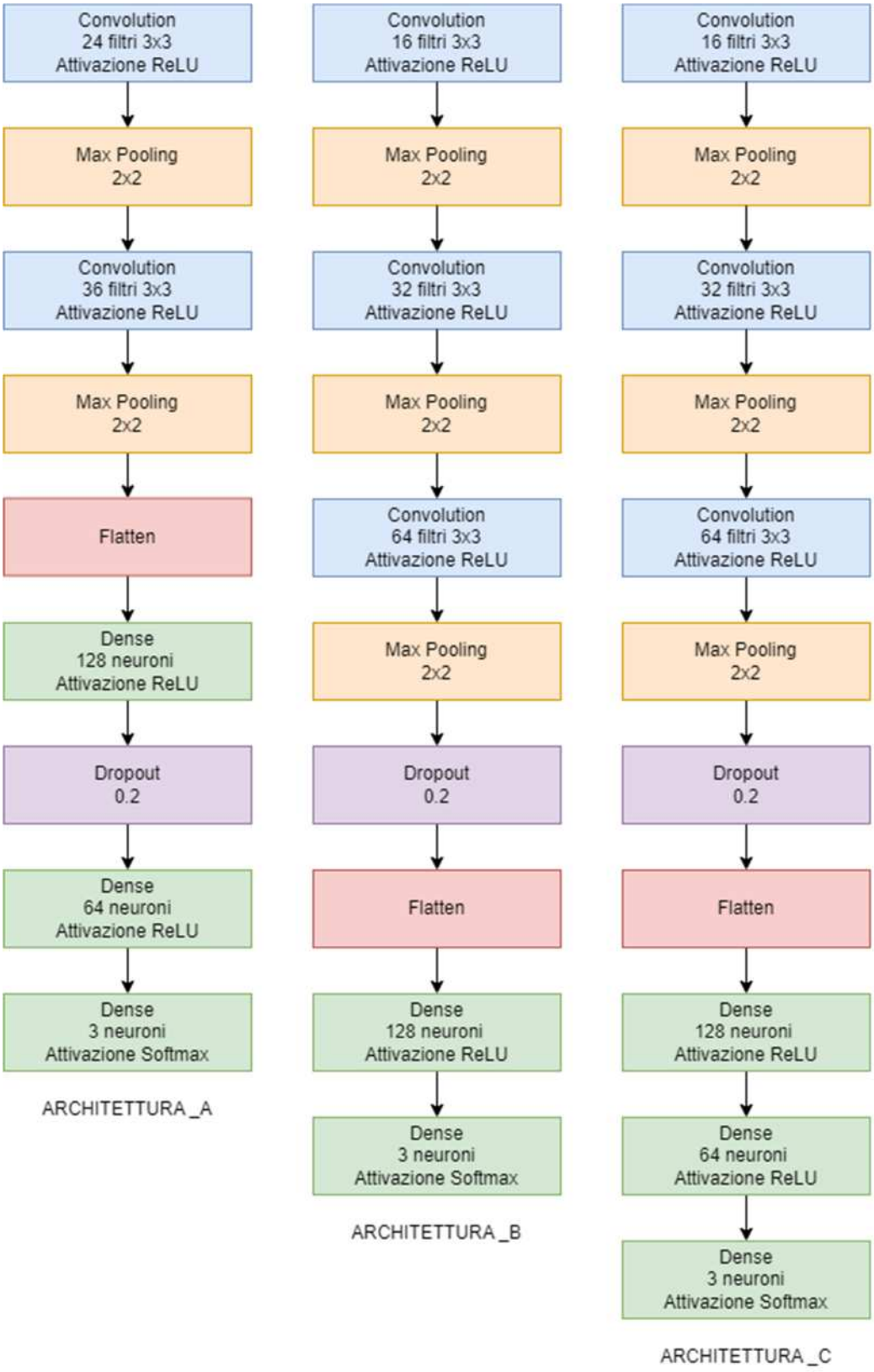

Figura 3.1 Architetture A, B e C

3.2 Risultati Test

Una volta definite le architetture si modificano i parametri di *batch size* ed il numero di epoche, per verificare come questi influiscono sulle performance della rete. I test effettuati sulle architetture ed i relativi parametri sono riassunti nella Tabella 3.1.

Tabella 3.1 Risultati Test

Test	Architettura	*batch_size*	*epochs*	*Test Loss*	*Test Accuracy*
1	A	32	15	0,6862	0,7899
2	A	32	20	0,5829	0,7917
3	A	32	30	1,2423	0,7769
4	A	64	30	0,7274	0,8121
5	A	16	30	1,0942	0,3715
6	B	32	10	0,4260	0,8220
7	B	32	10	0,4388	0,8212
8	B	32	15	0,3638	0,8620
9	B	32	20	0,4385	0,8377
10	C	32	10	0,4259	0,8255
11	C	32	15	0,4546	0,8403
12	B	32	15	0,4066	0,8394
13	B	32	30	0,3959	0,8646

Analizzando i risultati ottenuti si evidenzia il Test 8 sull'architettura B che ha ottenuto il valore di *loss* più basso in assoluto e il secondo miglior risultato in termini di *accuracy*.

Si riportano per quest'ultimo il risultato del test restituito direttamente dallo script in Figura 3.2 e si aggiungono anche i risultati dell'allenamento in Figura 3.3 ed i grafici di *loss* e *accuracy* restituiti dalla rete tramite la libreria matplotlib in Figura 3.4 e Figura 3.5.

TEST

```
[12] test_loss, test_acc = cnn_model.evaluate(test_images)

     print("Test accuracy:", round(test_acc*100,4),"%")
     print("Test Loss :", round(test_loss*100,4),"%")
```

```
36/36 [==============================] - 12s 283ms/step - loss: 0.3638 - accuracy: 0.8620
Test accuracy: 86.1979 %
Test Loss : 36.3817 %
```

Figura 3.2 Risultato del Test 8

```
Epoch 1/15
287/287 [==============================] - 140s 436ms/step - loss: 0.8517 - accuracy: 0.5979 - val_loss: 0.7205 - val_accuracy: 0.6825
Epoch 2/15
287/287 [==============================] - 99s 335ms/step - loss: 0.6710 - accuracy: 0.7211 - val_loss: 0.6792 - val_accuracy: 0.6702
Epoch 3/15
287/287 [==============================] - 99s 340ms/step - loss: 0.6046 - accuracy: 0.7542 - val_loss: 0.5495 - val_accuracy: 0.7711
Epoch 4/15
287/287 [==============================] - 98s 336ms/step - loss: 0.5357 - accuracy: 0.7854 - val_loss: 0.5130 - val_accuracy: 0.7798
Epoch 5/15
287/287 [==============================] - 93s 316ms/step - loss: 0.4871 - accuracy: 0.7979 - val_loss: 0.4962 - val_accuracy: 0.7781
Epoch 6/15
287/287 [==============================] - 94s 320ms/step - loss: 0.4534 - accuracy: 0.8186 - val_loss: 0.4727 - val_accuracy: 0.8088
Epoch 7/15
287/287 [==============================] - 99s 339ms/step - loss: 0.4190 - accuracy: 0.8328 - val_loss: 0.4637 - val_accuracy: 0.7904
Epoch 8/15
287/287 [==============================] - 97s 330ms/step - loss: 0.3920 - accuracy: 0.8498 - val_loss: 0.4215 - val_accuracy: 0.8219
Epoch 9/15
287/287 [==============================] - 97s 331ms/step - loss: 0.3533 - accuracy: 0.8701 - val_loss: 0.4314 - val_accuracy: 0.8228
Epoch 10/15
287/287 [==============================] - 94s 322ms/step - loss: 0.3263 - accuracy: 0.8786 - val_loss: 0.4250 - val_accuracy: 0.8289
Epoch 11/15
287/287 [==============================] - 99s 339ms/step - loss: 0.3056 - accuracy: 0.8892 - val_loss: 0.4262 - val_accuracy: 0.8219
Epoch 12/15
287/287 [==============================] - 99s 337ms/step - loss: 0.2825 - accuracy: 0.8985 - val_loss: 0.3994 - val_accuracy: 0.8325
Epoch 13/15
287/287 [==============================] - 97s 331ms/step - loss: 0.2742 - accuracy: 0.8994 - val_loss: 0.3603 - val_accuracy: 0.8482
Epoch 14/15
287/287 [==============================] - 98s 333ms/step - loss: 0.2375 - accuracy: 0.9194 - val_loss: 0.3900 - val_accuracy: 0.8430
Epoch 15/15
287/287 [==============================] - 97s 330ms/step - loss: 0.2254 - accuracy: 0.9240 - val_loss: 0.3850 - val_accuracy: 0.8421
```

Figura 3.3 Risultati dell'allenamento del Test 8

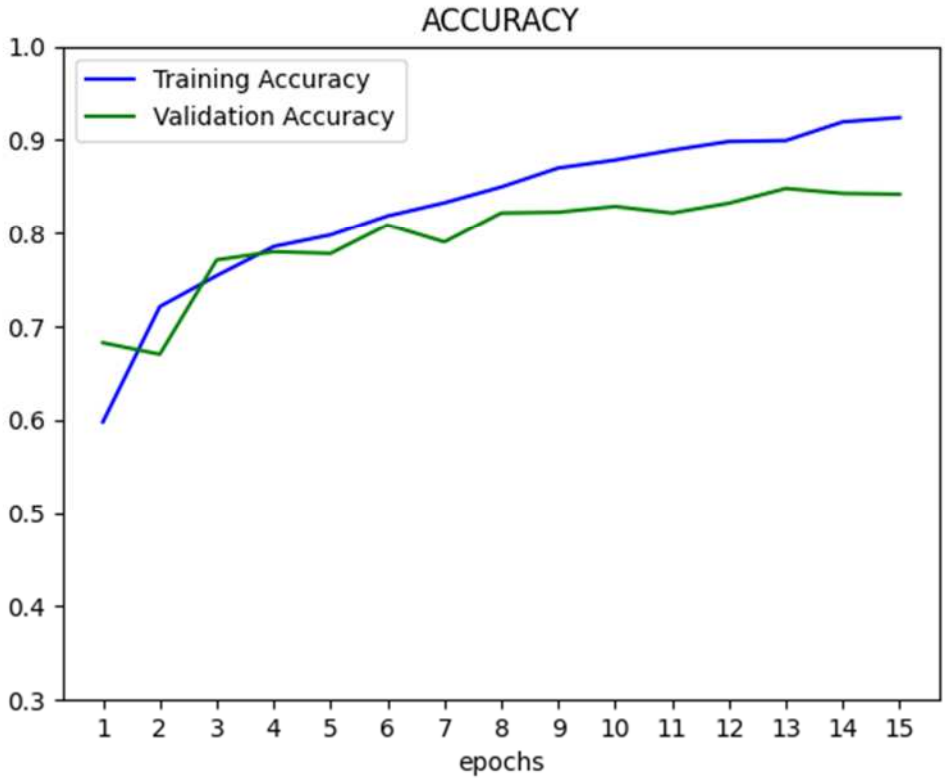

Figura 3.4 Grafico di Training e Validation Accuracy per il Test 8

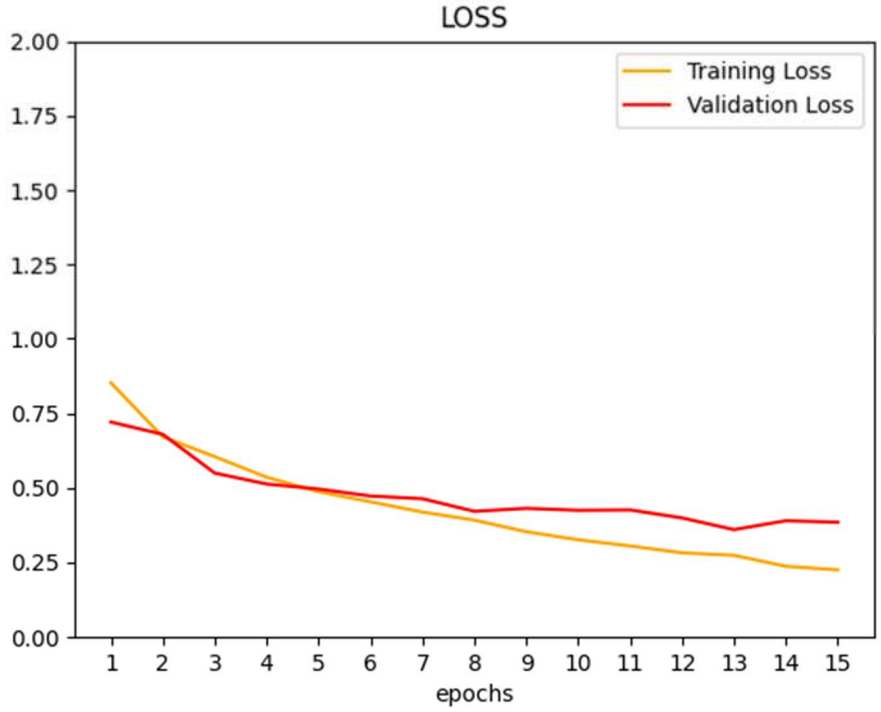

Figura 3.5 Grafico di Training e Validation Loss per il Test 8

Prendendo come riferimento il Test 7 si può notare quanto discusso nell'elaborato: aumentando le epoche da 10 a 15 (Test 8) si ha un miglioramento globale delle performance, ma testando l'allenamento per 20 epoche (Test 9) e per 30 epoche (Test 13) i risultati peggiorano. Questo a dimostrazione del fatto che non esiste una regola universale per configurare i parametri correttamente, ma vanno fatte delle considerazioni in funzione dell'applicazione e del tipo di rete costruita.
Si segnalano inoltre per i Test 3 e 5 dei valori di *loss*, seppur di poco, superiori ad 1.

4 Conclusioni

Il modello costruito è un insieme di algoritmi per l'identificazione dei difetti quindi anche i suoi risultati possono non essere perfetti. Tuttavia, permette di capire il funzionamento di questi strumenti che sono comunque la base degli algoritmi più complessi.

Le tecniche AI per il riconoscimento e la classificazione dei difetti - in particolare del calcestruzzo - rappresentano un passo avanti significativo nel campo dell'ispezione e della manutenzione delle strutture. Questi sistemi offrono un'efficienza e una precisione senza precedenti, consentendo di identificare difetti in modo tempestivo e accurato, riducendo il rischio di danni strutturali e garantendo una maggiore sicurezza delle infrastrutture.

Grazie all'uso di algoritmi di *machine learning* e reti neurali, è possibile analizzare grandi quantità di dati in modo rapido ed efficace, consentendo un monitoraggio costante delle condizioni delle strutture. I modelli potranno analizzare i dati raccolti in tempo reale dai sensori posti sulle strutture e monitorare costantemente le condizioni del calcestruzzo. Ciò permetterà di identificare e affrontare i difetti prima che possano causare danni significativi.
L'AI potrà essere utilizzata per sviluppare sistemi di manutenzione predittiva che prevedono quando e dove saranno necessari interventi di riparazione o sostituzione, riducendo i costi di manutenzione. Si apriranno così nuove prospettive per la manutenzione delle infrastrutture e la sicurezza pubblica.

Resta il fatto che queste tecniche richiedono una continua formazione e addestramento dei modelli per mantenere la loro precisione nel tempo ed è essenziale l'intervento umano nell'analisi dei risultati per agire poi sulla manutenzione e sulla riparazione delle strutture.

Esistono moltissime ricerche mirate a migliorare le performance delle reti neurali per tutti i campi di applicazione. In particolare, in base alle singole esigenze, si studiano nuove funzioni di attivazione, *loss function* e *optimizer* o combinazioni di essi per migliorare l'accuratezza.

L'evoluzione continua di queste tecnologie promette di rivoluzionare tantissimi settori ed in particolare, il settore della costruzione e dell'ispezione delle strutture, contribuendo a garantire la durabilità e l'affidabilità delle infrastrutture nel futuro.

Un'ultima riflessione sui dati, che grazie anche a nuovi sistemi di raccolta e acquisizione sono ormai definiti "il nuovo oro" a sottolinearne l'importanza nell'era digitale e tecnologica odierna, oltre all'alto valore economico.
Si può concludere da quanto visto nell'elaborato, infatti, che per ottenere dei buoni risultati con gli algoritmi di intelligenza artificiale si ha bisogno di una grandissima quantità e varietà di dati.

Questo solleva importanti questioni sulla privacy e la sicurezza, ovvero su come i nostri dati vengono raccolti, protetti ed utilizzati, ma si intuisce facilmente che la capacità di raccogliere, elaborare e utilizzare i dati in modo efficace può offrire enormi vantaggi in termini di potere e competitività.

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] An Introduction to Convolutional Neural Networks, Keiron O'Shea and Ryan Nash

[2] Artificial Neural Network, Neha Gupta, Institute of Engineerinf and Technology,DAVV,Indore

[3] Activation Functions in Neural Networks, International Journal of Engineering Applied Sciences and Technology, 2020 Vol. 4, Issue 12, ISSN No. 2455-2143, Pages 310-316 Published Online April 2020 in IJEAST

[4] Binary cross entropy with deep learning technique for Image classification, International Journal of Advanced Trends in Computer Science and Engineering, Volume 9, No.4, July – August 2020, Dr.A.Usha Ruby, Prasannavenkatesan Theerthagiri, Dr.I.Jeena Jacob, Dr.Y.Vamsidhar

[5] Backpropagation Through Time: What It Does and How to Do It, PAUL J. WERBOS

[6] Improving Generalization Performance by Switching from Adam to SGD, Nitish Shirish Keskar, Richard Socher

[7] Overfitting and Undercomputing in Machine Learning, Tom Dietterich, Department of Computer Science, Oregon State University, Corvallis

[8] Understanding Dropout, part of Advances in Neural Information Processing Systems 26 (NIPS 2013), Authors Pierre Baldi, Peter J. Sadowski

[9] Development and Application of Artificial Neural Network, Yu-chen Wu, Jun-wen Feng, Published online: 30 December 2017, Springer Science Business Media, LLC, part of Springer Nature 2017

[10] Artificial intelligence-based image classification methods for diagnosis of skin cancer: Challenges and opportunities, Manu Goyal, Thomas Knackstedt, Shaofeng Yan, Saeed Hassanpour

[11] Systematic review of artificial intelligence techniques in the detection and classification of COVID-19 medical images in terms of evaluation and benchmarking: Taxonomy analysis, challenges, future solutions and methodological aspects, O.S. Albahria, A.A. Zaidana, A.S. Albahrib, B.B. Zaidana, Karrar Hameed Abdulkareemc, Z.T. Al-qaysid, A.H. Alamoodia, A.M. Aleesae, M.A. Chyada, R.M. Alesae, L.C. Kema, Muhammad Modi Lakulua, A.B. Ibrahima, Nazre Abdul Rashida

[12] Automatic thresholding for defect detection, Hui-Fuang Ng, Department of Computer Science and Information Engineering, Asia University, No. 500, Liufeng Road, Wufong, Taichung 41354, Taiwan, ROC

[13] Review of Pavement Defect Detection Methods, Wenming Cao, Qifan Liu, and Zhiquan He

[14] Anomaly detection of defects on concrete structures with the convolutional autoencoder, J.K. Chow, Z. Su, J. Wu, P.S. Tan, X. Mao, Y.H. Wang

[15] A Method for Crack Detection on a Concrete Structure, Yusuke Fujita Yamaguchi University, Yoshihiro Mitani Ube National College of Technology, Yoshihiko Hamamoto Yamaguchi University

[16] Artificial intelligence-empowered pipeline for image-based inspection of concrete structures, Jun Kang Chowa, Zhaoyu Sua, Jimmy Wua, Zhaofeng Lib, Pin Siang Tana, Kuan-fu Liuc, Xin Mao, Yu-Hsing Wang

[17] Concrete surface defect detection using deep neural network based on lidar scanning, Nasrollahi M., Bolourian N. and Hammad A.

[18] Performance Comparison of Multiple Convolutional Neural Networks for Concrete Defects Classification, Palisa Arafin, Anas Issa, and A. H. M. Muntasir Billah

[19] Multi-classifier for reinforced concrete bridge defects, Philipp Hüthwohl, Ruodan Lu, Ioannis Brilakis

[20] The Phenomenon of Cracking in Cement Concretes and Reinforced Concrete Structures: The Mechanism of Cracks Formation, Causes of Their Initiation, Types and Places of Occurrence, and Methods of Detection—A Review, Grzegorz Ludwik Golewski

[21] Factors influencing bugholes on concrete surface analyzed by image processing technology, Baoju Liu, Tengyu Yang, Youjun Xie