



UNIVERSITÀ DI PARMA

Dipartimento di Ingegneria e Architettura
Corso di Laurea in Ingegneria Informatica, Elettronica e delle
Telecomunicazioni

Generazione di filtri ottimi per immagini tramite programmazione genetica

Generation of optimal filters for images through genetic
programming

Relatore:
Ch.mo. Prof. Stefano Cagnoni

Tesi di Laurea di:
Giuseppe Ricciardi

a mio fratello Giovanni, punto di riferimento ed esempio di forza e
dedizione anche quando si ha il mondo contro.

“Nessun limite eccetto il cielo”

Miguel Cervantes

Ringraziamenti

Ringrazio il professore Cagnoni per avermi dato la possibilità di poter svolgere questa attività di tesi, per il supporto e la disponibilità dimostratami.

Ringrazio la mia famiglia, sempre pronta ad esserci nel momento del bisogno.

Ringrazio mia madre, la spalla su cui cercare sostegno di cui tutti avrebbero bisogno, per aver coltivato la voglia di conoscere che mi ha spinto ad andare avanti su questa strada.

Ringrazio mio padre per tutte le volte che si è sacrificato per me, non sarò mai in grado di ripagarti ma spero che questo traguardo sia un qualcosa.

Ringrazio Giuseppe, Salvatore, Davide ed Helmi che sono compagni di vita più che semplici amici, conservo con cura ogni istante trascorso assieme con ognuno di voi che ha contribuito a diventare ciò che sono e ve ne sarò sempre grato.

Ringrazio Stefano, Cristian, Martina, Francesco, Michele, Gabriel, Mattia e tutte le splendide persone conosciute durante questo percorso. Avete reso più leggero il cammino e mi avete regalato un sorriso anche nei momenti in cui vi stressavo con le mie incessanti preoccupazioni.

Ringrazio Chiara per l'impegno e l'utilissimo aiuto che mi ha dato nel portare a termine questo scritto.

Infine ringrazio il 2022, descriverlo come “anno complicato” è un eufemismo ma se mi trovo a scrivere queste parole è anche grazie a lui nella sua interezza: al dolore, alle paure, alle lacrime, alle gioie, alle risate, ai bei momenti che mi hanno insegnato il valore della parola gratitudine.

Indice

Introduzione	1
1 Stato dell'arte	5
1.1 Evolutionary Computation	5
1.1.1 La struttura di un algoritmo evolutivo	6
1.1.2 Rappresentazione degli individui	7
1.1.3 Funzione fitness ed operatori di un algoritmo evolutivo	8
1.1.4 Le tipologie di algoritmi evolutivi	10
1.2 Genetic Programming	11
1.2.1 Caratteristiche generali	11
1.2.2 Rappresentazione degli individui	12
1.2.3 La funzione fitness	14
1.2.4 Popolazione iniziale e metodi per inizializzarla	15
1.2.5 Operatori di selezione, crossover e mutazione	16
1.3 Computer vision	19
1.3.1 Uso di algoritmi evolutivi nella computer vision	20
1.3.2 EA per il task di denoise	21
1.3.3 EA per il task di edge detection	24
2 Architettura del progetto	28
2.1 Librerie utilizzate	29
2.2 Dataset	31
2.3 Operazioni d'adattamento alle task	32
2.3.1 Operazioni preliminari sul dataset per il task di denoise	33

3 Realizzazione del progetto	36
3.1 Filtraggio convolutivo	36
3.2 Uso della GP per la creazione di filtri per la rimozione del rumore	37
3.2.1 Nodi funzione e nodi terminali	38
3.2.2 Fitness	41
3.2.3 Generazione ed evoluzione della popolazione	42
3.2.4 Parametri utilizzati ed alcuni esempi	44
3.3 Uso della GP per la creazione di filtri per l'edge detection	46
3.3.1 Nodi funzione e nodi terminali	47
3.3.2 La funzione fitness	48
3.3.3 Parametri utilizzati ed alcuni esempi	48
4 Risultati	51
4.1 Algoritmo per applicare i filtri ottenuti sul dataset	51
4.2 Risultati ottenuti con i filtri per il denoise	53
4.2.1 Risultati per Kernel di dimensioni 3x3	53
4.2.2 Risultati per Kernel di dimensioni 5x5	54
4.2.3 Statistiche riguardo l'evoluzione	55
4.2.4 Confronto con metodi esistenti di denoise	57
4.3 Risultati ottenuti con i filtri per l'edge detection	59
4.3.1 Risultati con Kernel 3x3	59
4.3.2 Risultati con Kernel 5x5	60
4.3.3 Statistiche riguardo l'evoluzione	61
4.3.4 Confronti con metodi esistenti di edge detection	63
4.4 Risultati ottenuti sul test set	65
Conclusioni	68
Bibliografia	70

Elenco delle figure

1	Foto scattata dal Ranger 7 dopo essere stata processata [1]	1
2	Rappresentazione matriciale di un'immagine in grayscale [2]	2
3	Schema rappresentativo sul funzionamento del DIP [3]	3
4	Applicazione di algoritmi di image processing su una radiografia [4]	3
1.1	Flowchart di un algoritmo evolutivo [5]	6
1.2	Rappresentazione del funzionamento della roulette wheel selection [6]	9
1.3	Esempio di un EA con una tournament selection [6]	10
1.4	Esempio d'esecuzione di un algoritmo GP [7]	12
1.5	Funzione rappresentata mediante una struttura ad albero	13
1.6	Esempio del funzionamento di un interprete [8]	15
1.7	Esempio del funzionamento dell'operatore di crossover [9]	17
1.8	Schema riassuntivo degli operatori di mutazione [10]	18
1.9	Numero delle pubblicazioni con le parole "evolutionary" e "computer vision" oppure "image analysis" nel titolo dal 2011 al 2021	20
1.10	Rappresentazione sintetica dell'algoritmo denoise di Chaudry [11]	22
1.11	Flowchart dell'algoritmo denoise di Yan [12]	22
1.12	Schema sul funzionamento dell'algoritmo denoise di Petrovic [13]	23
1.13	Risultati dell'algoritmo di denoise di Harding [14]	23

1.14	Schema sui vari metodi di edge detection con EC [15]	24
1.15	Flowchart del metodo ACO proposto da Kumar e Raheja per l'edge detection [16]	25
1.16	Risultati ottenuti dall'esecuzione del metodo proposto da Fu W. comparati con detectors tradizionali [17]	26
1.17	Schema sul funzionamento del metodo proposto da Basturk [18]	27
2.1	Schema sull'architettura generale del progetto	28
2.2	Confronto tra il numero di linee richieste per definire le componenti di un problema OneMax tra i principali framework di EC [19]	29
2.3	Un esempio del dataset BSDS500: a sinistra l'immagine originale mentre a destra l'immagine segmentata.	31
2.4	Un secondo esempio del dataset BSDS500	31
2.5	Schema riassuntivo sulle prime operazioni effettuate sul dataset	33
2.6	Rappresentazione grafica di una PDF Gaussiana	34
2.7	Schema sul funzionamento dell'algoritmo di aggiunta del rumore	34
2.8	Risultati algoritmo di aggiunta del rumore	35
3.1	Esempio di un filtro convolutivo applicato su un'immagine [20]	38
3.2	Codice sorgente di alcune funzioni implementate per l'algoritmo di denoise	40
3.3	Codice sorgente della funzione fitness per l'algoritmo generatore di filtri per il denoise	41
3.4	Schema sul funzionamento dell'algoritmo per la generazione di filtri per il denoise	43
3.5	Individui ottenuti dall'algoritmo GP di denoise per un Kernel 3x3	44
3.6	Individui ottenuti dall'algoritmo GP di denoise per un Kernel 5x5	45
3.7	Schema dell'algoritmo per la generazione di filtri per l'edge detection	46

3.8	Individui ottenuti dall'algoritmo GP di edge detection per un Kernel 3x3	49
3.9	Esempio di evoluzione di un individuo con algoritmo GP per l'edge detection	50
4.1	Schema sul funzionamento dell'algoritmo per l'image processing con i filtri ottenuti	52
4.2	Esempio di filtri GP 3x3 di denoise applicati sul training set .	53
4.3	Esempio di filtri GP 5x5 di denoise applicati sul training set .	54
4.4	Grafici sulle statistiche raccolte durante l'evoluzione dei filtri 3x3 di denoise	55
4.5	Grafici sulle statistiche raccolte durante l'evoluzione dei filtri 5x5 di denoise	56
4.6	Confronto dei risultati ottenuti con un filtro avente Kernel 3x3 con metodi di denoise esistenti	57
4.7	Confronto dei risultati ottenuti con un filtro avente Kernel 5x5 con metodi di denoise esistenti	58
4.8	Esempio di filtri GP 3x3 di edge detection applicati sul training set	59
4.9	Esempio di filtri GP 5x5 di edge detection applicati sul training set	60
4.10	Grafici delle statistiche raccolte durante l'evoluzione dei filtri 3x3 di edge detection	61
4.11	Grafici sulle statistiche raccolte durante l'evoluzione dei filtri 5x5 di edge detection	62
4.12	Confronto dei risultati ottenuti con un filtro per l'edge detection 3x3 con metodi esistenti	63
4.13	Confronto dei risultati ottenuti con un filtro per l'edge detection 5x5 con metodi esistenti	64
4.14	Schema sul funzionamento del test dei filtri generati	65
4.15	Confronto dei risultati sul test set con filtri 3x3	66
4.16	Confronto dei risultati sul test con filtri 5x5	67

Introduzione

Il contesto

Molte delle tecniche di elaborazione digitale delle immagini sono state sviluppate negli anni '60 presso i Bell Laboratories, il Jet Propulsion Laboratory, il Massachusetts Institute of Technology e l'Università del Maryland con applicazione a immagini satellitari. Lo scopo delle prime elaborazioni delle immagini era di migliorarne la qualità con il fine di fornire una maggiore comprensione alle persone che ne dovevano usufruire [21].

La prima applicazione di successo è stata realizzata dall'American Jet Propulsion Laboratory (JPL), in cui sono state utilizzate tecniche di elaborazione delle immagini come la correzione geometrica, la rimozione del rumore, ecc. sulle migliaia di foto lunari inviate dallo Space Detector Ranger 7 nel 1964, tenendo conto della posizione del sole e dell'ambiente lunare [22].

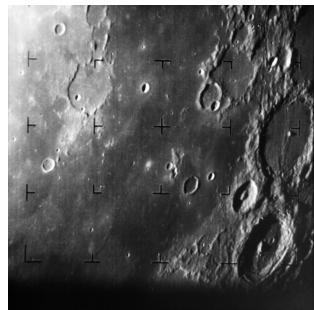


Figura 1: Foto scattata dal Ranger 7 dopo essere stata processata [1]

Un’**immagine digitale** è la conversione di un’immagine reale in una sequenza di numeri che può essere interpretata da un computer.

Per l’immagazzinamento in memoria nei computer, le immagini vengono definite come matrici bidimensionali di *pixel*. Ogni pixel rappresenta il livello di grigio (o l’intensità del colore nel caso di immagini a colori) dell’immagine, e può essere rappresentato come una funzione $F(x, y)$ con x ed y le coordinate matriciali che indicano la posizione del pixel nell’immagine. Nel caso di un’immagine in bianco e nero, un pixel di dimensione pari ad un *byte* (8 bit) può rappresentare 256 tonalità di grigio: da 0 (nero), fino a 255 (bianco).

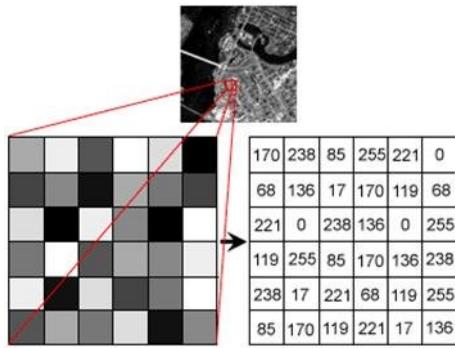


Figura 2: Rappresentazione matriciale di un’immagine in grayscale [2]

Per salvaguardare la genuinità delle informazioni da trasmettere viene utilizzata l’**elaborazione digitale** delle immagini, comunemente chiamata *DIP* (Digital Image Processing), che si basa sull’utilizzo di un computer per elaborare immagini digitali attraverso un algoritmo [23], rispetto all’elaborazione analogica presenta numerosi vantaggi: una grande quantità di algoritmi e soprattutto, l’eliminazione dei problemi d’introduzione di rumore e distorsione durante l’elaborazione che si ha nel caso delle immagini analogiche.

I **filtri** sono usati per modificare e migliorare le immagini digitali, ad esempio tramite la riduzione delle alte frequenze (tecniche di *smoothing*) o delle basse frequenze (tecniche di *edge detection*).

Le tecniche di filtraggio possono essere divise in due grandi categorie: *filtraggi spaziali* e *filtraggi nel dominio delle frequenze*: i primi modificano direttamente i valori dei singoli pixel mentre i secondi si occupano della rimozione di

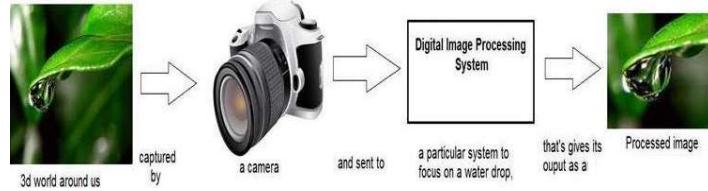


Figura 3: Schema rappresentativo sul funzionamento del DIP [3]

alte o basse frequenze [24].

Negli ultimi 50 anni sono stati compiuti numerosi progressi in questo ambito; un contributo importante è stato dato dall'utilizzo di algoritmi di intelligenza artificiale (IA) per l'elaborazione di immagini e dalla nascita della disciplina della Computer Vision, un campo di ricerca che si occupa di come i computer possano ottenere una comprensione ad alto livello di immagini o video digitali. Nell'ambito della CV esistono numerose task, come l'edge recognition, la riduzione del rumore, l'equalizzazione dell'immagine, la segmentazione, la feature selection, ecc. e basate su diversi approcci [25].

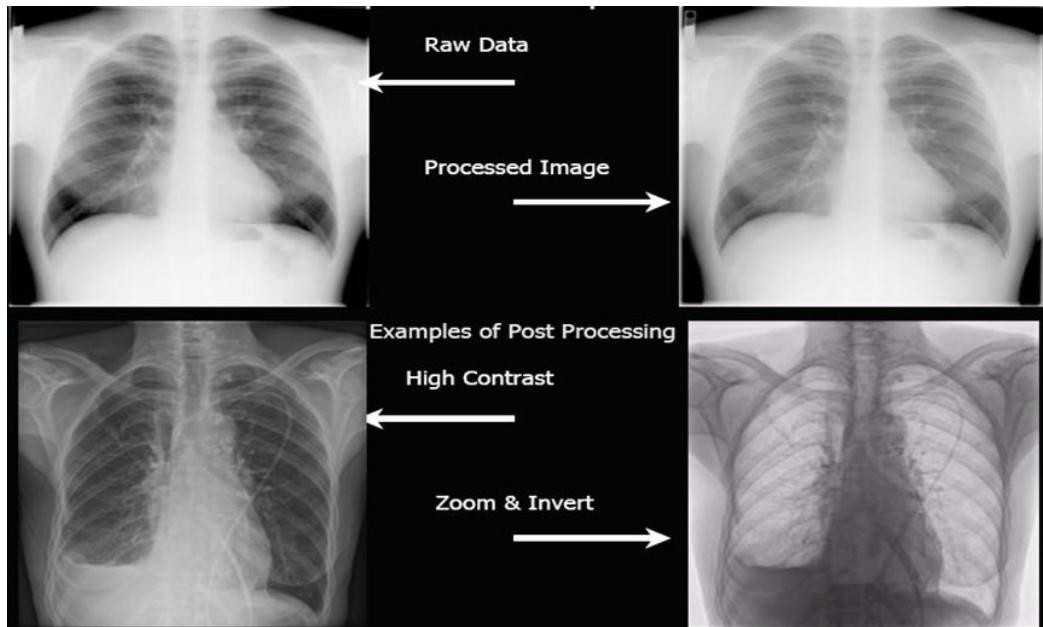


Figura 4: Applicazione di algoritmi di image processing su una radiografia [4]

Obiettivo dell'attività di tesi

L'attività di tesi ha come obiettivo l'utilizzo della **programmazione genetica (GP)** per la generazione di filtri ottimi per risolvere task di riduzione del rumore ed edge detection.

Nel primo capitolo vengono approfonditi i concetti di *computazione evolutiva* e *programmazione genetica* ed infine viene dato uno sguardo al punto in cui si è arrivati con il loro utilizzo nella Computer Vision. Nel secondo capitolo è illustrata l'architettura del progetto, analizzando in particolare le librerie utilizzate, il dataset e tutte le operazioni preliminari su di esso. Il terzo capitolo comprende una descrizione dettagliata della realizzazione dell'attività di tesi per ogni task sulla quale sono stati generati i filtri. Nel quarto capitolo vengono analizzati i risultati ottenuti; infine, nella conclusione, si fa il punto su quanto fatto e si discutono possibili sviluppi futuri.

Capitolo 1

Stato dell'arte

L’evoluzione è un processo d’ottimizzazione che ha lo scopo di migliorare le abilità di un organismo (o sistema) di sopravvivere a cambiamenti dinamici. Ci sono diversi tipi di evoluzione in base al contesto; nel caso dell’evoluzione biologica la *teoria della selezione naturale* di Charles Darwin ne è la legge cardine: in un mondo con risorse limitate e popolazioni stabili, ogni individuo compete con gli altri per sopravvivere; gli individui con le migliori caratteristiche (o tratti) sono quelli che hanno più possibilità di riprodursi e sopravvivere e queste caratteristiche saranno passate alla prole. Una seconda parte della teoria di Darwin afferma che, durante la produzione di un organismo figlio, eventi casuali, le mutazioni, possono generare cambiamenti casuali delle sue caratteristiche. Se queste caratteristiche creano un beneficio per l’organismo la probabilità di sopravvivenza di quest’ultimo aumenta e quindi aumenta la probabilità che vengano trasmesse alle generazioni future [26].

1.1 Evolutionary Computation

L’**evolutionary computation** (EC) si riferisce a sistemi di problem solving che usano modelli computazionali dei processi evolutivi come elementi prin-

cipali.

1.1.1 La struttura di un algoritmo evolutivo

Un algoritmo evolutivo è una *ricerca stocastica* di una soluzione ottimale ad un problema dato [10]. È caratterizzato dai seguenti componenti:

- Un *encoding* di soluzioni al problema come cromosomi
- Una funzione per valutare la bontà della soluzione, chiamata *fitness*.
- Un'*inizializzazione* della popolazione iniziale.
- *Operatori di selezione* per scegliere gli individui migliori da far riprodurre.
- *Operatori di riproduzione* per generare nuovi individui.

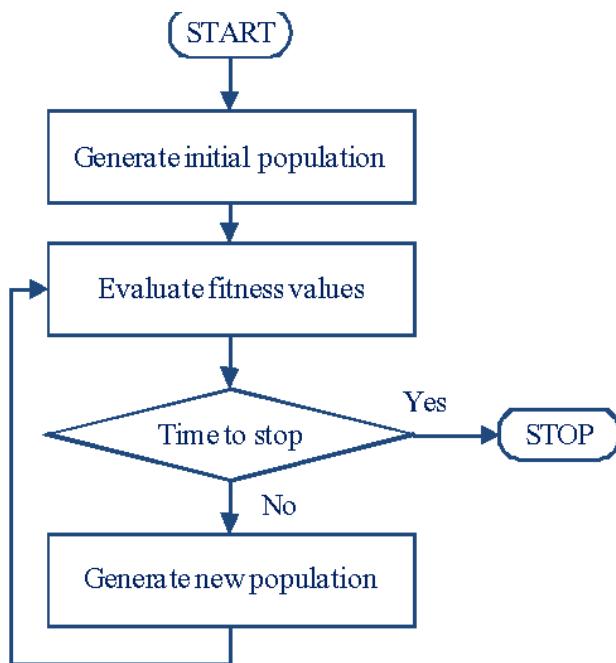


Figura 1.1: Flowchart di un algoritmo evolutivo [5]

Come si può vedere dalla figura 1.1, dopo aver generato una popolazione iniziale l'algoritmo ripete i seguenti passaggi fino a quando non raggiunge una condizione di terminazione:

1. valutazione della fitness di ogni individuo della popolazione
2. selezione dei migliori individui (i genitori) per effettuare la riproduzione
3. generazione dei nuovi individui (la prole) attraverso l'operazione di *crossover* e mutazione.

1.1.2 Rappresentazione degli individui

Nel contesto dell'EC, ogni individuo rappresenta un candidato come soluzione al problema d'ottimizzazione. Le caratteristiche di un individuo sono rappresentate dai *cromosomi*, che compongono il suo genoma: queste caratteristiche codificano gli elementi di una soluzione al problema d'ottimizzazione per i quali è richiesta una assegnazione ottimale. Ogni elemento che deve essere ottimizzato prende il nome di *gene*. Un *allele* è un'assegnazione di valori permessa per il gene a cui è riferito. Un passo importante per il progetto di un algoritmo evolutivo (EA) è trovare un'adeguata rappresentazione delle soluzioni candidate (quindi degli individui). L'efficienza e la complessità dell'algoritmo dipendono strettamente dallo schema rappresentativo; molti algoritmi evolutivi rappresentano soluzioni come vettori di un certo tipo di dato. Uno dei metodi più comuni per generare una popolazione iniziale è assegnare un valore random dal dominio (ad esempio un numero reale se il dominio è \mathbb{R}) ad ogni gene di ciascun cromosoma. L'obiettivo dell'assegnazione casuale è quello di assicurare che la popolazione iniziale sia una rappresentazione con distribuzione uniforme dell'intero spazio di ricerca. La scelta delle dimensioni della popolazione incide in termini di complessità di calcolo e abilità esplorative dell'algoritmo. Incrementare il tasso di mutazioni della popolazione può essere utile nel caso di piccole popolazioni per forzare l'algoritmo ad esplorare meglio lo spazio di ricerca [10] (*ivi, p.129-134*).

1.1.3 Funzione fitness ed operatori di un algoritmo evolutivo

Per poter stabilire quanto una soluzione trovata dall'algoritmo sia efficace per il problema viene introdotta la funzione di *fitness*, che mappa un cromosoma in un valore scalare e rappresenta la funzione obiettivo del problema d'ottimizzazione. Spesso la funzione fitness rappresenta una metrica universale per il problema; esistono casi in cui occorrono più misure relative per misurare la performance individuale in relazione con le altre di una popolazione.

L'operatore di selezione è uno degli operatori principali di un algoritmo evolutivo, si occupa di scegliere le migliori soluzioni e prevede la selezione di una nuova popolazione partendo dalla soluzione candidata da utilizzare come nuova popolazione; questa fase prende il nome di *selection*. La riproduzione avviene successivamente alla selezione degli individui che saranno usati come genitori attraverso operatori di *crossover*, un processo di creazione di uno o più individui attraverso la combinazione di materiale genetico selezionato da due o più genitori, o di *mutazione*, in cui i cambiamenti sono individuali e casuali e si applicano ai valori dei geni in un cromosoma. Gli individui migliori avranno più possibilità di riprodursi per assicurare che la prole contenga le loro caratteristiche. Tra gli operatori di selezione più comuni abbiamo:

- *random selection*, in cui ogni individuo ha probabilità $\frac{1}{n}$ di essere selezionato, dove n rappresenta la dimensione della popolazione. La probabilità non deriva dalla *fitness*, di conseguenza sia gli individui migliori che quelli peggiori hanno la stessa probabilità di essere selezionati.
- *fitness-proportional selection*, viene creata una distribuzione di probabilità in base alla fitness, e gli individui sono selezionati campionando la distribuzione:

$$\varphi_s(x_i(t)) = \frac{f_\gamma(x_i(t))}{\sum_{l=1}^{n_s} f_\gamma(x_l(t))} \quad (1.1)$$

dove n_s è il numero totale degli individui di una popolazione e $\varphi_s(x_i(t))$ è la probabilità che l'individuo $x_i(t)$ sia selezionato, $f_\gamma(x_i(t))$ è il valore

della fitness di x_i [27].

Uno dei modi più comuni utilizzato per campionare la popolazione per la selezione proporzionale alla fitness è la *roulette wheel selection*: la ruota è divisa in n fette, dove n rappresenta il numero degli individui della popolazione, ogni individuo ottiene una porzione della ruota in modo proporzionale al suo valore di fitness. Si sceglie un *fixed point* sulla ruota e la fetta che gli si trova di fronte rappresenterà l'individuo genitore, per il secondo genitore si ripete lo stesso processo. Lo spazio occupato dalla fetta (e il rispettivo valore di fitness dell'individuo) è direttamente proporzionale alla probabilità di essere scelto secondo la formula 1.1 [6].

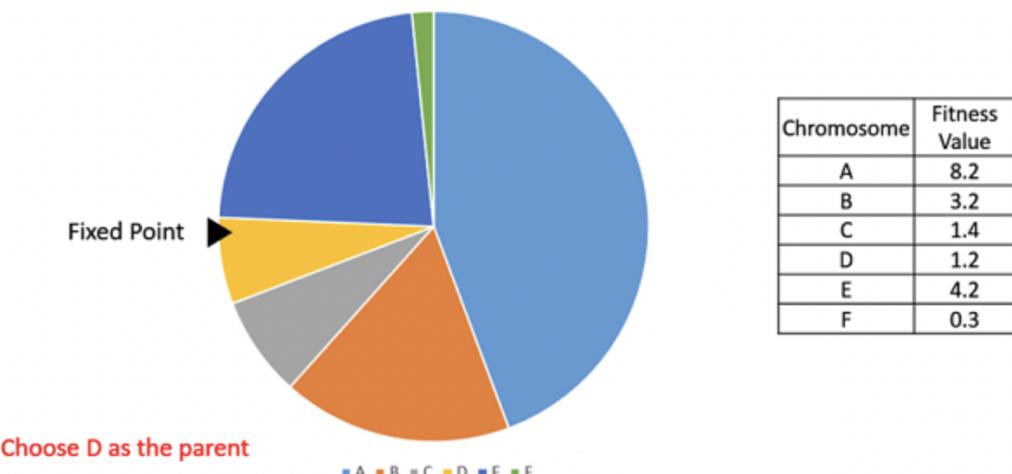


Figura 1.2: Rappresentazione del funzionamento della roulette wheel selection [6]

- *tournament selection*, seleziona un gruppo n_{ts} di individui a caso dalla popolazione dove $n_{ts} << n_s$. Viene confrontata tra loro la performance degli individui e il migliore individuo del gruppo viene selezionato. Per il *crossover* tra due genitori la selezione viene eseguita due volte, una per ogni genitore.

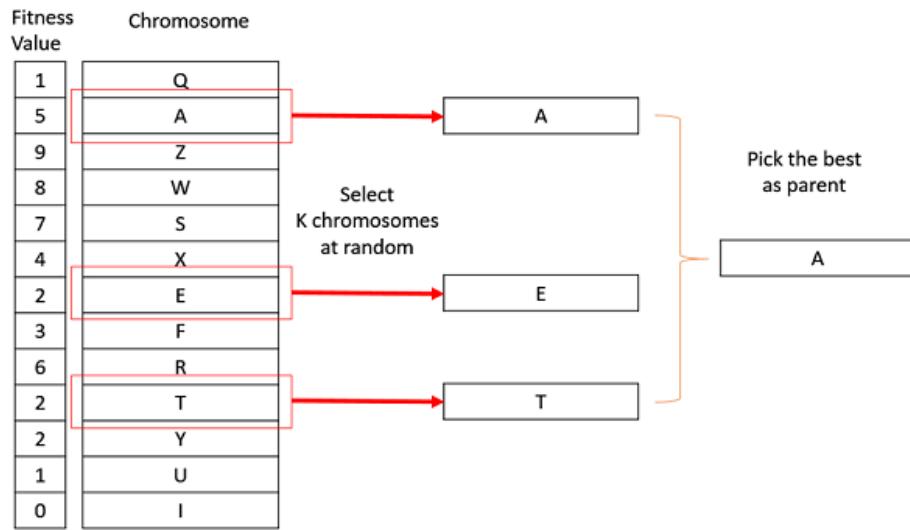


Figura 1.3: Esempio di un EA con una tournament selection [6]

Gli operatori evolutivi sono applicati iterativamente negli algoritmi EA fino a quando le condizioni di stop vengono soddisfatte. La condizione di stop più semplice è limitare il numero di generazioni che l'EA può eseguire. In alternativa viene posto un limite al numero di valutazioni della fitness. In aggiunta al limite del tempo d'esecuzione viene usato un criterio di convergenza. La convergenza avviene quando non ci sono cambiamenti nella popolazione; altri criteri di convergenza possono essere definiti in base alle proprie esigenze.[10] (*ivi, p.140*)

1.1.4 Le tipologie di algoritmi evolutivi

Esistono diversi tipi di algoritmi evolutivi, che differiscono per la rappresentazione degli individui e per il tipo di problemi cui vengono applicati:

- *Genetic Algorithm* (GA): è l'algoritmo EA più famoso, viene spesso utilizzato per problemi di ottimizzazione. Le soluzioni di un problema sono rappresentate come stringhe di simboli (spesso bit) a cui vengono appli-

cati operatori di riproduzione e mutazione, con una *fitness-proportional parent selection*.

- *Evolution strategy*: in questo algoritmo la rappresentazione delle soluzioni avviene mediante vettori di numeri reali, ed è basata su sole mutazioni e selezioni.
- *Genetic programming*(GP): le soluzioni in questo problema sono dei programmi, rappresentati come alberi sintattici. Il crossover avviene mediante la ricombinazione di sotto-alberi mentre la mutazione tramite modifiche casuali ai singoli nodi o rami dell'albero. La fitness viene calcolata proporzionalmente in base all'abilità di un programma di risolvere il problema.
- *Evolutionary programming*: deriva dall'emulazione del comportamento adattativo di un individuo in un'evoluzione, è simile alla programmazione genetica ma la struttura dei programmi è prestabilita e i suoi parametri numerici possono evolversi.

1.2 Genetic Programming

1.2.1 Caratteristiche generali

La **genetic programming**(GP) è un EA per la risoluzione automatica di problemi che parte da istruzioni di un linguaggio ad alto livello che definiscono le operazioni. È un metodo indipendente dal dominio, che crea geneticamente una popolazione di programmi (in inglese *computer programs*, abbreviato in CP) per risolvere il problema. La GP trasforma iterativamente una popolazione di programmi in una nuova popolazione applicando le operazioni genetiche viste prima, ovvero: *crossover*, mutazioni, riproduzione, duplicazione o eliminazione dei geni [28].

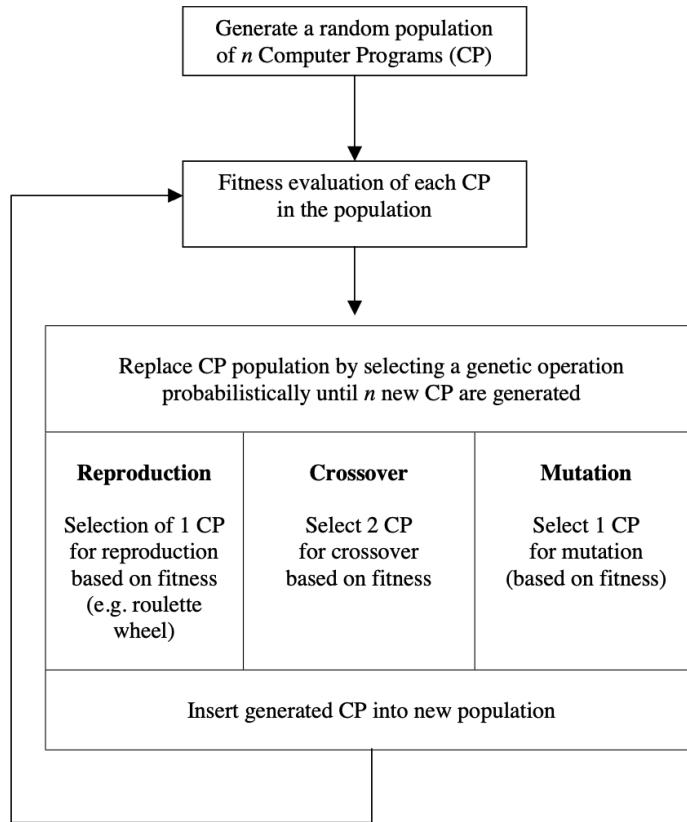


Figura 1.4: Esempio d'esecuzione di un algoritmo GP [7]

Il *modus operandi* (figura 1.4) di questo algoritmo è simile a quello dei GA, teorizzati da Holland (1968, 1973), ma presenta alcune differenze:

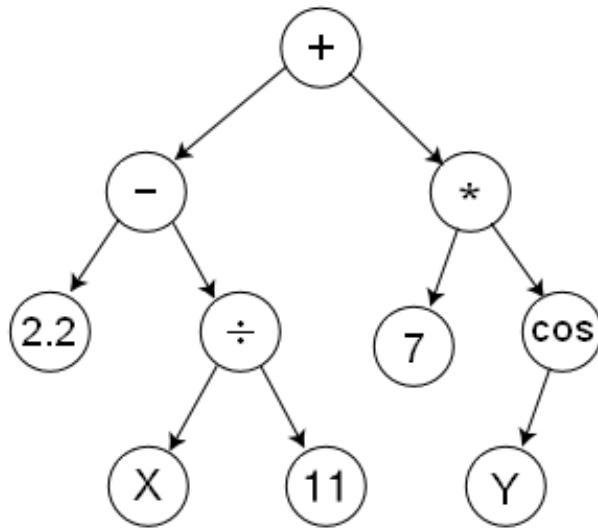
1. la rappresentazione del problema e delle relative soluzioni
2. la valutazione della fitness per l'introduzione dell'inserimento di parametri in una funzione prefissata all'esecuzione di un programma (o una funzione)

1.2.2 Rappresentazione degli individui

La GP per la rappresentazione degli individui utilizza dei programmi espressi come alberi. Nella GA le dimensioni di un individuo sono prefissate (basti pensare ad un *array* di numeri di dimensione n), mentre in una popolazione

di un algoritmo GP gli individui possono presentare dimensioni diverse, dove per dimensione in questo caso si intende la profondità di un albero. Inoltre bisogna definire attentamente una grammatica composta da un insieme di *funzioni F* ed un insieme di *terminali T* [29].

L'insieme delle funzioni F contiene le operazioni di base, ad esempio le funzioni matematiche (somma, prodotto, differenza, divisione etc.), inoltre può includere operazioni o funzioni definite appositamente per il problema. Le funzioni sono rappresentate dai nodi interni di un albero e bisogna definirne la cardinalità (il numero di argomenti richiesti). L'insieme dei terminali T contiene tutte le variabili e le costanti di un programma, rappresentate dalle foglie di un albero.



$$\left(2.2 - \left(\frac{X}{11} \right) \right) + \left(7 * \cos(Y) \right)$$

Figura 1.5: Funzione rappresentata mediante una struttura ad albero

Nonostante questo tipo di rappresentazione possa essere implementata in tutti i linguaggi di programmazione, conviene introdurre la rappresentazione *LISP* [30]: i programmi ed i dati hanno la stessa forma (*S-expressions*); in questo modo un CP può essere trattato come se fosse un dato e valutato

immediatamente per accedere al risultato. L'espressione in figura 1.5 può essere riscritta in notazione LISP in questo modo:

$$(+(-(/xy)2.2)(*(cosy)7)) \quad (1.2)$$

L'espressione deve essere letta da sinistra a destra applicando ricorsivamente ogni funzione alle sue sotto-S-expressions (attraversamento simmetrico dell'albero).

Sia l'insieme delle funzioni che quello dei terminali devono rispettare due proprietà importanti [7] per ottenere una rappresentazione utile alla risoluzione di un problema:

- *Closure property*: ogni funzione dell'insieme delle funzioni deve essere in grado di elaborare tutti gli argomenti generati da altre funzioni o dai terminali. Ad esempio la divisione deve essere "protetta" dalle divisioni per zero, perché un nodo funzione non sarebbe in grado di processare il risultato ottenuto da tale operazione.
- *Sufficiency property*: il problema deve essere risolvibile usando solamente i terminali e le funzioni proposte, ovvero il programmatore deve scegliere un insieme di funzioni e terminali che possa essere rilevante per la risoluzione del problema. Il programmatore deve quindi avere un livello adeguato di conoscenza riguardo al problema da risolvere.

1.2.3 La funzione fitness

La *fitness* di un individuo (programma GP) dipende dal tipo di problema; il calcolo della fitness richiede che il programma venga valutato rispetto ad un numero di casi di test. Le sue prestazioni sui test vengono utilizzate per quantificare la fitness dell'individuo [10]. La fitness inoltre può anche penalizzare individui che hanno proprietà strutturali non conformi a quelle desiderate (anche se hanno buoni risultati rispetto al problema), ad esempio si può pensare di penalizzare gli individui con una profondità dell'albero maggiore di una soglia x . Spesso viene utilizzato un interprete per valutare i programmi

evoluti. Interpretare un albero significa percorrere i nodi seguendo un ordine che garantisca che i nodi non vengano eseguiti prima che il valore dei loro argomenti (se presenti) sia noto ed è possibile farlo ricorsivamente in diversi modi [8].

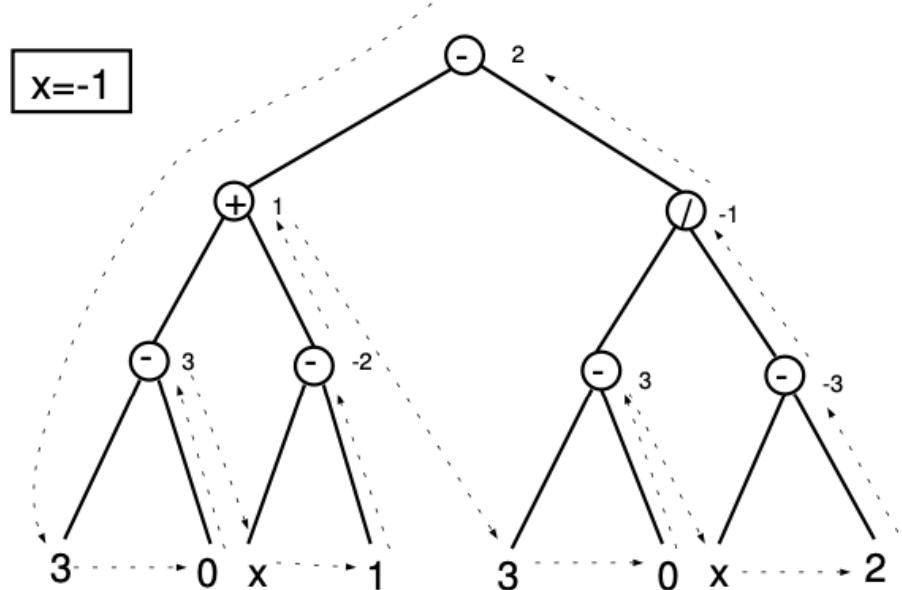


Figura 1.6: Esempio del funzionamento di un interprete [8]

Dalla figura 1.6 è possibile vedere un esempio sul funzionamento dell'interprete: partendo dal terminale 3 si passa al terminale 0 si effettua la funzione definita dal nodo padre, in questo caso una sottrazione; il numero alla destra del nodo è il risultato dell'interpretazione del *sub-tree* che ha come radice quel nodo. La stessa operazione viene ripetuta ricorsivamente per percorrere interamente l'albero.

1.2.4 Popolazione iniziale e metodi per inizializzarla

La popolazione iniziale della GP è costituita da programmi generati casualmente composti dalle funzioni e i terminali definiti dall'utente e può essere vista come una "ricerca alla cieca" [31] nello spazio di ricerca rappresentato dai programmi. La maggior parte degli individui appartenenti a questa gene-

razione avranno un valore di fitness molto basso, alcuni individui avranno dei valori migliori rispetto ad altri e queste differenze di valori verranno sfruttate per la creazione di nuove generazioni attraverso gli operatori di *crossover* e *mutazione*. I due metodi principali per generare la popolazione iniziale sono: il metodo *full* e il metodo *grow*, in entrambi i metodi gli individui iniziali sono generati in modo da non superare una profondità massima definita dall'utente.

La profondità di un nodo è il numero di archi che devono essere attraversati per raggiungere il nodo che inizia dalla radice dell'albero, mentre la profondità di un albero è data dalla profondità più grande di un suo ramo. Nel *full method* i nodi sono presi dall'insieme delle funzioni F fino a quando non viene raggiunta la profondità massima, dopo quella profondità vengono aggiunti solo nodi terminali. Il *full method* genera alberi in cui i rami hanno tutte la stessa profondità, il *grow method* invece permette la creazione di con rami di profondità diversa: i nodi sono selezionati sia dall'insieme dei terminali T che dall'insieme delle funzioni F.

Iohu Koza propose il metodo *ramped half and half* [29] che si basa sul generare metà della popolazione mediante il *full method* e l'altra metà con il *grow method*, inserendo dei vincoli che garantiscono una varietà di forme e profondità degli individui.

1.2.5 Operatori di selezione, crossover e mutazione

Per quanto riguarda la selezione degli individui per generare una nuova prole vengono utilizzati i metodi illustrati nella sezione 1.1.3: bisogna sottolineare come il processo di selezione non sia *greedy*: individui con un basso valore di fitness possono essere comunque selezionati così come non è certo che venga selezionato l'individuo migliore.

L'operatore di *crossover* sceglie un nodo a caso dai suoi genitori per creare dei *sub-tree* che vengono scambiati per creare due figli a partire dai propri genitori. Gli operatori di mutazione sono spesso sviluppati secondo le necessità del problema da risolvere, tuttavia i più utilizzati [10] sono:

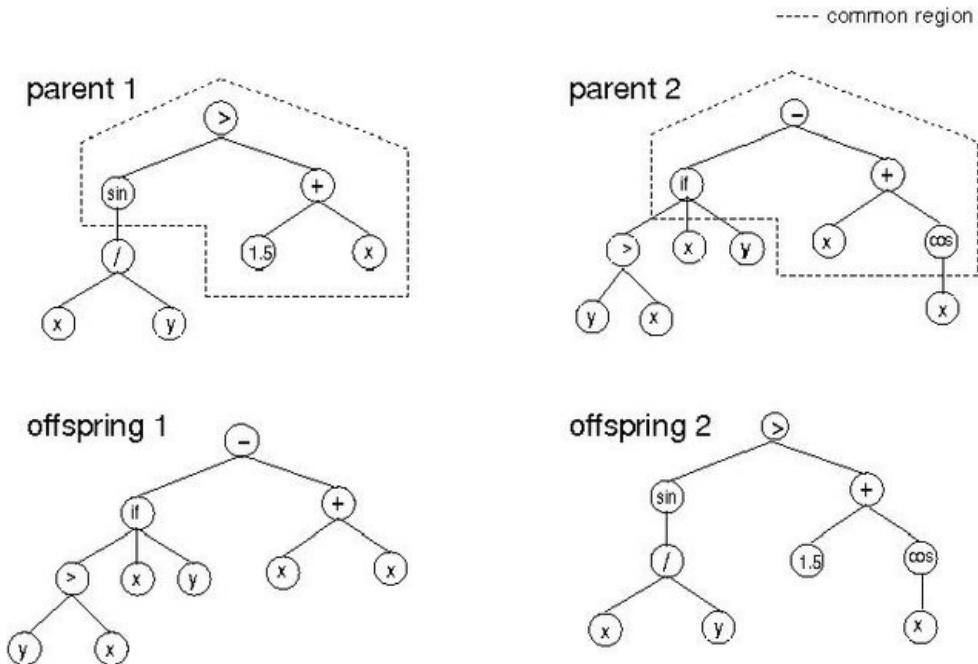


Figura 1.7: Esempio del funzionamento dell'operatore di crossover [9]

- Mutazione di un nodo funzione: un nodo funzione è scelto a caso e sostituito con un altro nodo funzione con la stessa cardinalità.
- Mutazione di un nodo terminale: sostituzione di un nodo terminale con un altro.
- *Swap mutation*: viene scelto a caso un nodo funzione e ne viene cambiato l'ordine degli argomenti.
- *Grow mutation*: un nodo scelto casualmente è sostituito da un *sub-tree*.
- *Gaussian mutation*: un nodo terminale viene selezionato e mutato aggiungendo una variabile Gaussiana casuale.
- *Trunc mutation*: un nodo funzione scelto a caso è sostituito da un nodo terminale, avviene quindi un troncamento dell'albero.

Un passaggio molto importante nella realizzazione di un'applicazione GP è la scelta dei parametri di controllo. Non esistono dei valori ottimali per

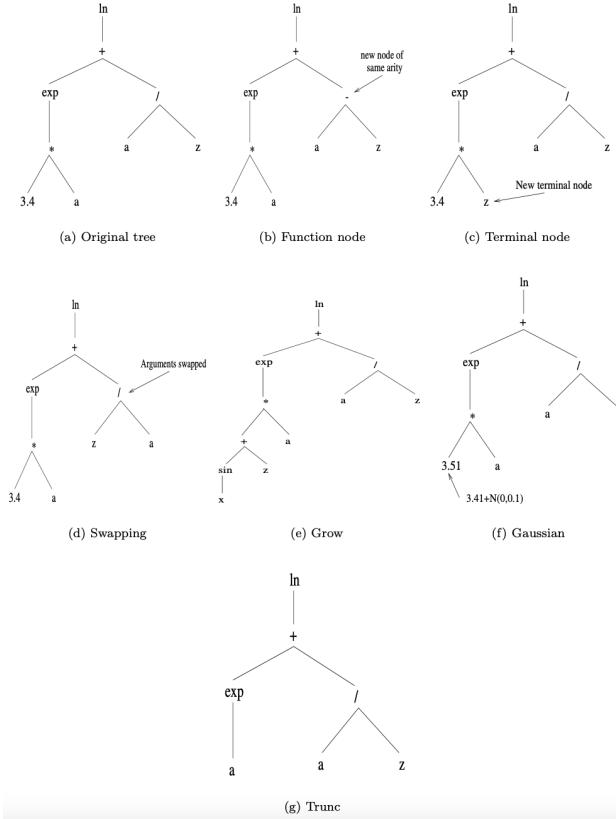


Figura 1.8: Schema riassuntivo degli operatori di mutazione [10]

la risoluzione di ogni problema, ogni parametro dipende molto dai dettagli dell'applicazione. Il parametro più importante è la dimensione della popolazione, altri parametri sono: la probabilità di *crossover*, la probabilità di mutazione, la profondità massima di un individuo e altri dettagli sull'esecuzione del programma.

È sconsigliato utilizzare popolazioni molto grandi in quanto il tempo dedicato al calcolo della fitness dei singoli individui potrebbe aumentare drasticamente: il numero di generazioni è tipicamente più limitato rispetto a un GA; le ricerche della soluzione ottima del problema da risolvere ottengono risultati migliori nelle prime generazioni, se non è stata trovata nessuna soluzione allora è molto probabile che non verrà trovata in un periodo temporale ragionevole. La dimensione degli alberi della popolazione iniziale dipende strettamente dalla dimensione del problema.

tamente dal tipo di funzioni e dal numero di terminali stabiliti dall'utente [8].

1.3 Computer vision

Negli ultimi venti anni l'utilizzo della Computer Vision (CV) si è sempre più diffuso, partendo come disciplina accademica focalizzata esclusivamente sulla ricerca o per l'utilizzo in settori estremamente specializzati, fino a trovare utilizzo nelle catene di montaggio delle industrie più comuni.

Così come il numero di settori in cui viene applicata oggi la CV, anche le task di cui si occupa sono variegate e in continuo aumento. Una possibile suddivisione delle task [32] può essere fatta in:

1. Visione di basso livello: comprende i primi stage dell'*image processing*, ne fanno parte le task di *image filtering*, *smoothing*, *denoising*, *lightness computation*, *edge detection*. Nonostante questi siano argomenti di ricerca trattati da diversi anni, rimangono una sfida interessante in quanto i risultati di queste operazioni sono fondamentali per eseguire le task di livelli più avanzati. Da un punto di vista matematico queste task vengono spesso risolte riconducendole a problemi d'ottimizzazione, rendendo utile ed interessante l'approccio tramite algoritmi evolutivi.
2. Visione di medio livello: provvede alla connessione delle task di basso livello con quelle di alto livello. L'obiettivo è quello di tradurre la percezione dell'immagine in una descrizione simbolica, che verrà poi utilizzata dagli algoritmi di alto livello per estrarne il contesto. Il task più importante di questo livello è la *segmentazione* che si occupa di dividere le immagini in regioni d'interesse; altre task sono *feature extraction* e *texture classification*.
3. Visione di alto livello: si occupa di studiare come implementare un approccio cognitivo in un computer; in questo livello rientrano le task

di *content-based image retrieval, recognition, identification, 3D-scene analysis.*

1.3.1 Uso di algoritmi evolutivi nella computer vision

La caratteristica di avere algoritmi basati su una popolazione rende l'EC una scelta interessante per la risoluzione delle task CV; ad esempio problemi come la *feature selection* sono caratterizzati da un grande spazio di ricerca per il quale potrebbero risultare poco efficaci le soluzioni che richiedono una conoscenza specifica del dominio.

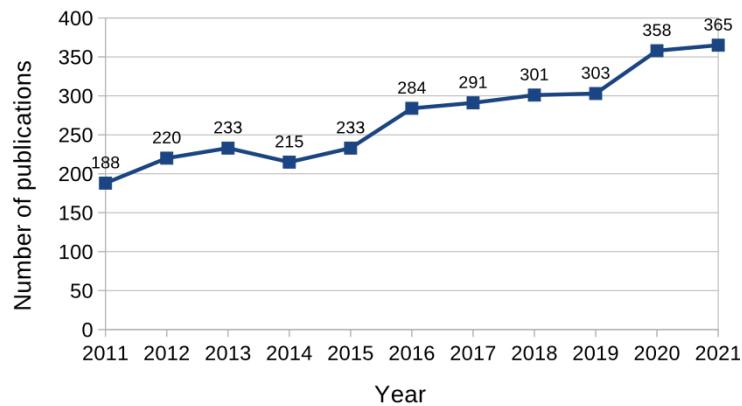


Figura 1.9: Numero delle pubblicazioni con le parole "evolutionary" e "computer vision" oppure "image analysis" nel titolo dal 2011 al 2021

La ricerca nell'ambito di algoritmi evolutivi e computer vision è in continua crescita, lo si denota facilmente dall'andamento del grafico in figura 1.9 [15], nel 2016 Gustavo Olague pubblica il primo libro in cui viene esposto il concetto di *Evolutionary Computer Vision* (ECV) [33]. Da un punto di vista pratico gli algoritmi GEC (Generative and Evolutive algorithms) forniscono strumenti utili per la soluzione di problemi in cui non si ha mai una conoscenza completa del dominio del problema, altrimenti risulterebbe semplice trovare un'eventuale legge risolutiva per esso.

Visualizzando i problemi di CV come problemi d'ottimizzazione è possibile, attraverso gli algoritmi EC, esplorare lo spazio di ricerca per trovare punti

”notevoli” di un problema (come il massimo o il minimo) mediante una *random search*¹ e *greedy search*² simultaneamente grazie al lavoro svolto dagli operatori genetici (sezione 1.2.5), rendendo gli algoritmi EC una soluzione flessibile a questo tipo di problemi [32].

Per non risultare eccessivamente prolissi, dato il grande numero di pubblicazioni presenti riguardo l’argomento, saranno analizzati soltanto alcune applicazioni di metodi EC e GP alle task di denoise ed edge detection .

1.3.2 EA per il task di denoise

L’utilizzo degli algoritmi evolutivi per rimuovere il rumore da un’immagine digitale è un argomento trattato da molti ricercatori. Il significato più generale di rumore è ”segnalet non voluto”, in un’immagine rappresenta una variazione di colore o di luminosità che può essere introdotta da un *sensore*³ o dai circuiti di uno scanner o di una fotocamera digitale [34].

Alcuni metodi implementati per il task di rimozione rumore tramite GP [35] sono, ad esempio:

- l’algoritmo GP proposto da Chaudhry [11] per ripristinare le immagini degradate sviluppando una funzione ottimale che stima l’intensità del pixel. La soluzione proposta prevede di mappare l’immagine assegnando un valore tra 0 ed 1 ad ogni singolo pixel per decidere quali pixel debbano essere modificati (*Fuzzy logic*), successivamente la GP viene usata per sviluppare la funzione ottimale.
- Il metodo di *denoise* di Yan [12] durante il training prevede l’utilizzo del clustering per classificare le immagini in base alle loro strutture locali, successivamente viene applicata la GP per determinare un filtro ottimale. L’insieme delle funzioni dell’algoritmo GP è composto da operazioni aritmetiche, filtri Gaussiani e filtri bilaterali.

¹Scelta casuale di un punto dello spazio di ricerca

²Analisi singolare di ogni punto di un intervallo dello spazio di ricerca

³Sensore che rileva e trasmette le informazioni utilizzate per creare un’immagine

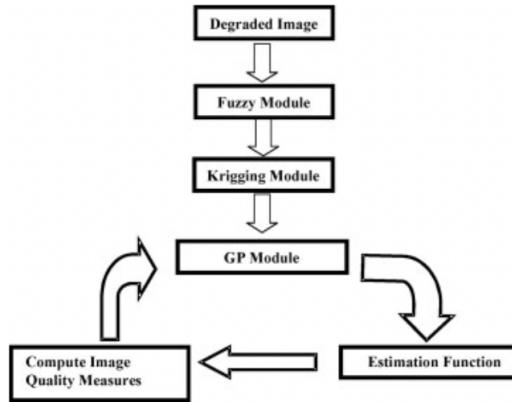


Figura 1.10: Rappresentazione sintetica dell'algoritmo denoise di Chaudry [11]

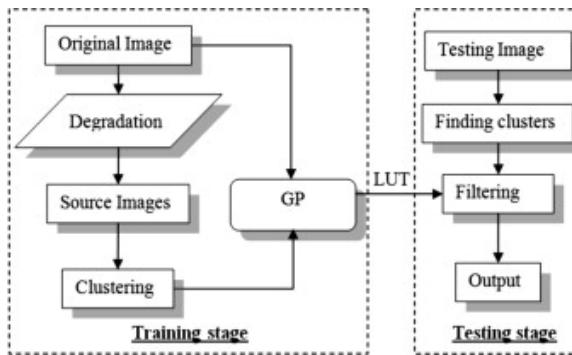


Figura 1.11: Flowchart dell'algoritmo denoise di Yan [12]

- Petrovic e Crnojevic [13] propongono un filtro GP-based a due step per rimuovere il rumore gaussiano ognuno avente il proprio *estimator*; il secondo estimator si occupa di rimuovere i pixel rumorosi tralasciati dal primo. Lo schema in figura 1.12 rappresenta una commutazione con rilevamento a due stadi.

Sulla base delle decisioni binarie di due rilevatori, l'uscita del filtro viene scelta tra tre possibilità: il pixel originale viene preservato oppure viene utilizzato uno dei due risultati degli *estimator*.

- Harding [14] usa la *Cartesian GP* per evolvere filtri di immagini e valutare la loro funzione di fitness su un'unità di processo grafica (GPU), la fitness è pari all'errore medio su ogni pixel.

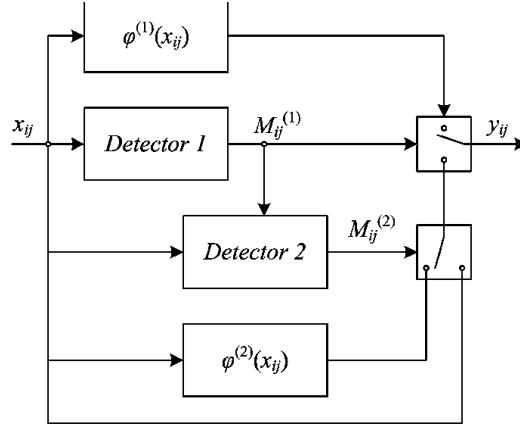


Figura 1.12: Schema sul funzionamento dell'algoritmo denoise di Petrovic [13]

- Majid [36] presenta un algoritmo GP per stimare il valore ottimo dei pixel rumorosi per rimuovere l'*impulse noise*. I pixel rumorosi sono determinati attraverso le derivate direzionali, successivamente il loro nuovo valore viene stimato tramite GP aggiungendo dei pixel senza rumore.



Figura 1.13: Risultati dell'algoritmo di denoise di Harding [14]

La prima colonna della figura 1.13 rappresenta l'immagine originale, la seconda l'immagine con rumore, la terza il risultato ottenuto dal filtro crea-

to tramite GP e la quarta l'effetto di un filtro mediano generico applicato all'immagine rumorosa.

1.3.3 EA per il task di edge detection

L'*edge detection* è una delle task essenziali per la Computer vision, utile soprattutto ai fini di task di livello più alto come la *feature detection* e la *feature extraction*. Lo scopo dell'*edge detection* è quello di identificare i punti di una immagine in cui i valori dell'intensità dei pixel cambiano nettamente o irregolarmente. Quasi tutti i metodi risolutivi per questa task si basano su punti di discontinuità, come il *Sobel detector* o il *Laplacian detector*. Un'alternativa a questa tipologia di filtri è offerta dai filtri Gaussiani, che sfruttano la derivata di una funzione Gaussiana, ad esempio i *Canny detectors* [37].

Le tecniche di EC applicate all'*edge detection* [15] possono generare autonomamente dei rivelatori di bordi oppure ottimizzare dei parametri di algoritmi già esistenti. Per quanto riguarda i metodi che si occupano di costruire dei filtri partendo da zero i più importanti sono ACO⁴ e GP, mentre per i metodi che ottimizzano i parametri di metodi già esistenti va citato il DE⁵.

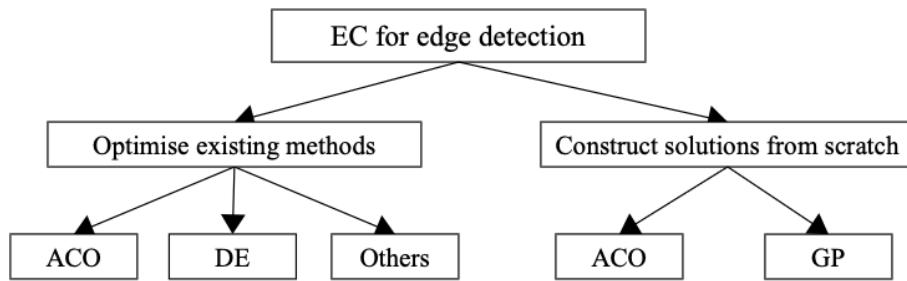


Figura 1.14: Schema sui vari metodi di edge detection con EC [15]

⁴Ant Colony Optimization

⁵Differential Evolution

Di seguito un elenco di alcune metodologie sviluppate:

- Nei metodi ACO l'immagine viene rappresentata come un grafo dove ogni nodo rappresenta un pixel, le “formiche” (dalla parola “ant”) possono muoversi da un nodo ad un altro e segnare il nodo incrementando la corrispondente cella nella “matrice dei feromoni”. I bordi possono essere calcolati impostando un valore di soglia sulla quantità di feromoni rilasciato dal passaggio di più formiche per quel punto.

I metodi ACO sono usati per trovare direttamente dei bordi sulle immagini, ne è un esempio quello di Kumar e Raheja [16] schematizzato in figura 1.15.

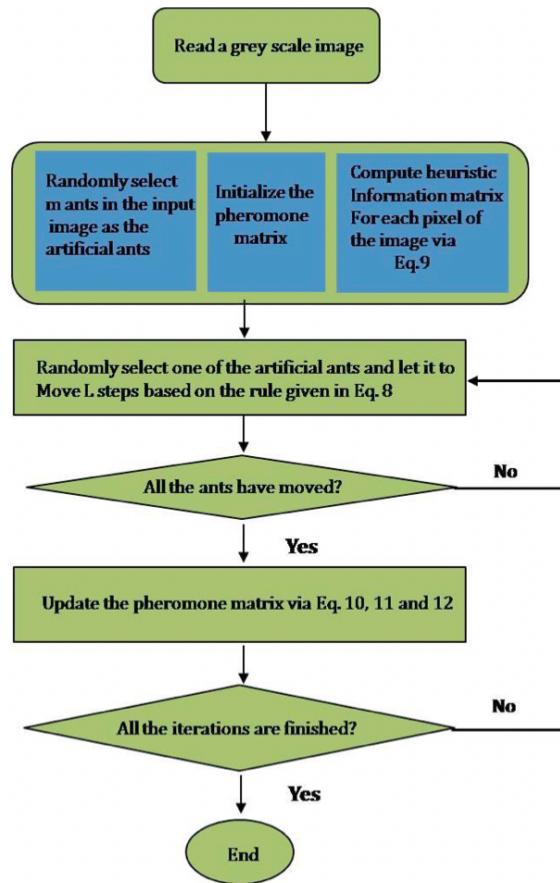


Figura 1.15: Flowchart del metodo ACO proposto da Kumar e Raheja per l'edge detection [16]

L'ACO è stato impiegato anche per modificare filtri già esistenti, Sudhiriti [38] ha realizzato un metodo ACO che identifica lesioni della pelle partendo dai risultati dell'elaborazione di filtri generati da un Canny detector.

- L'utilizzo della GP permette di creare dei modelli che classificano autonomamente i pixel che sono dei bordi da quelli che non lo sono.

Wenlong Fu [17] propone come metodo l'utilizzo dell'intera immagine come input per classificare direttamente i bordi senza operazioni di *pre-processing* o *post-processing*, utilizzando come operatori sia operazioni standard che operazioni di *shifting* e come funzioni di fitness la *precision*, la *recall* e il *true negative rate*. Questo approccio ha portato risultati che competono con quelli di un classico Sobel detector.

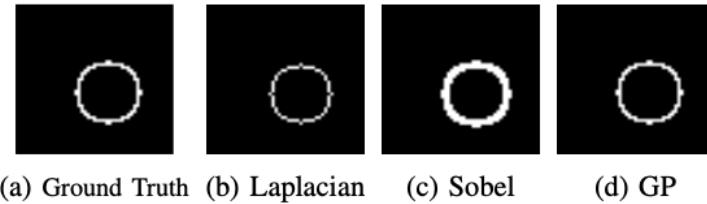


Figura 1.16: Risultati ottenuti dall'esecuzione del metodo proposto da Fu W. comparati con detectors tradizionali [17]

Un altro approccio interessante proposto da Fu W. [39] è l'utilizzo della GP per evolvere *Bayesian detectors* per aumentarne l'accuratezza. L'evoluzione è basata sull'uso di operatori e funzioni Bayesiane combinati ad operatori algebrici semplici.

- Servirsi di algoritmi di DE (differential evolution) per migliorare metodi di edge detection preesistenti. Zheng (2019) [40] ridefinisce gli input di una *generative adversarial network*(GAN) mediante un algoritmo ED con una funzione di fitness valutata da un *discriminator*.

Basturk [18] utilizza la DE per migliorare le prestazioni di una *cellular neural network* (CNN).

Lo studio dimostra come, nonostante la semplicità di una struttura come quella di una CNN (caratterizzata da equazioni differenziali di primo ordine) sia possibile, grazie ai miglioramenti offerti dalla DE, raggiungere performance migliori rispetto ai metodi proposti dalla letteratura.

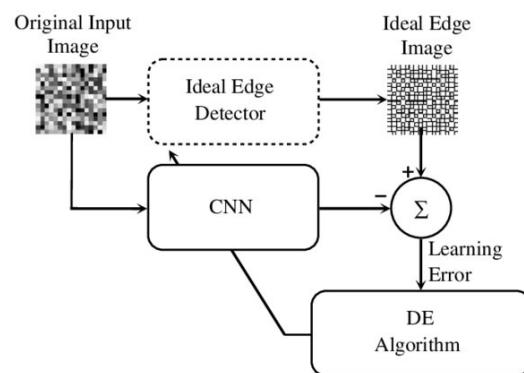


Figura 1.17: Schema sul funzionamento del metodo proposto da Basturk [18]

Capitolo 2

Architettura del progetto

La figura 2.1 mostra una rappresentazione dell’architettura dell’attività di tesi: i riquadri azzurri rappresentano le immagini input dell’algoritmo, provenienti dal dataset (sezione 2.2), in arancione sono rappresentate le operazioni preliminari effettuate sul dataset (sezione 2.3), in viola i filtri risultanti degli algoritmi di GP (capitolo 3) ed infine in verde i risultati (capitolo 4).

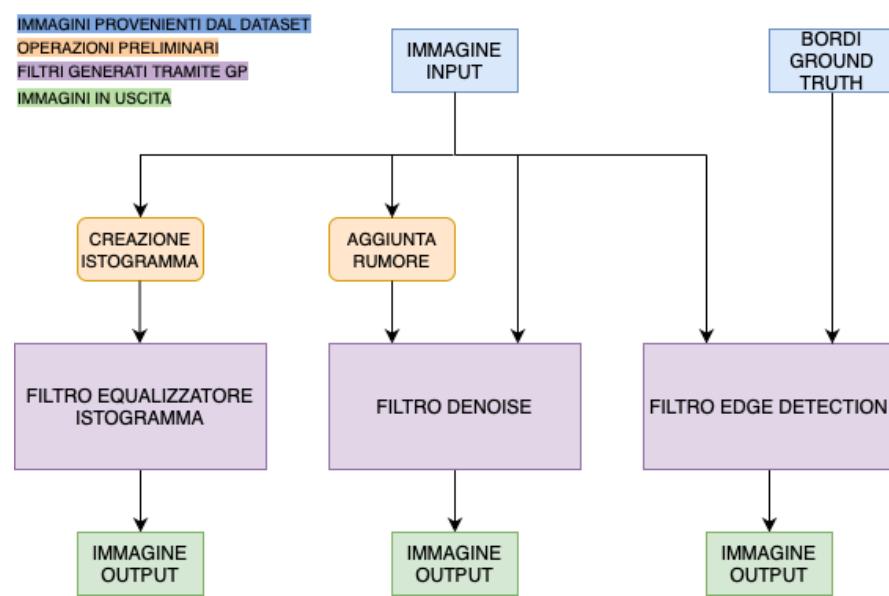


Figura 2.1: Schema sull’architettura generale del progetto

2.1 Librerie utilizzate

Il linguaggio di programmazione scelto per la realizzazione del progetto è *Python* che oltre ad avere un’infinità di librerie disponibili (alcune delle quali verranno analizzate in questa sezione) permette un approccio semplice, diretto e *responsive*.

DEAP (Distributed Evolutionary Algorithms in Python) [19] è la libreria utilizzata per implementare l’algoritmo di GP.

Sviluppata nel *Computer Vision and Systems Laboratory* presso l’Università Laval (Canada), si tratta di un *evolutionary algorithm framework* che si contraddistingue da altri framework dello stesso tipo perché offre più libertà nella creazione degli algoritmi sotto vari aspetti, ad esempio per la definizione e inizializzazione di una popolazione o per la vasta scelta di operatori da utilizzare (oppure creandone anche dei propri).

La potenzialità creativa di questo framework è data dall’utilizzo di due moduli: il *creator* e la *toolbox*. Il *creator* permette la produzione in run-time di classi attraverso l’ereditarietà e la composizione, ovvero la creazione di popolazioni per gli algoritmi evolutivi utilizzando qualsiasi tipo di strutture dati (liste, dizionari, alberi etc.). La *toolbox* contiene tutti gli operatori che l’utente vuole implementare nel proprio algoritmo evolutivo, altre funzionalità sono fornite dalla presenza di numerosi moduli come il *tools module* che offre operatori evolutivi di base per l’inizializzazione, il crossover, le mutazioni etc. Questi operatori possono essere aggiunti direttamente ad una toolbox per essere usati.

Framework	Type	Configuration	Algorithm	Example	Total
ECJ	202	35	65	26	328
EO	43	n/a	67	68	178
Open BEAGLE	256	41	116	64	477
Pyevolve	42	n/a	336	25	378
inspyred	23	n/a	143	24	190
DEAP	n/a	n/a	n/a	59	59

Figura 2.2: Confronto tra il numero di linee richieste per definire le componenti di un problema OneMax tra i principali framework di EC [19]

Altre librerie utilizzate sono:

- *Numpy*, libreria per l'*array programming*. È stata implementata per esportare le immagini in file di testo con una matrice di numeri (come in figura 2), per generare il rumore sulle immagini del dataset (sezione 2.3.1) e più in generale come strumento ai fini di calcoli ed elaborazioni bidimensionali.
- *OpenCV* (Open-source Computer Vision Library) è una libreria che comprende più di 2500 algoritmi ottimizzati per la computer vision, creata dal dipartimento di ricerca Intel. Ai fini del progetto è stata implementata per tutte le operazioni preliminari sul dataset per l'adattamento alle task affrontate (sezione 2.3). Un esempio di funzione utilizzata nel progetto è *cvtColor* per convertire le immagini a colore in grayscale.
- *PIL*: si tratta di una libreria sviluppata per l'*image processing*, fornisce un vasto supporto per i vari formati delle immagini ed una rappresentazione interna efficiente. Nel progetto viene usata in numerosi processi d'elaborazione delle immagini del dataset.
- *Matplotlib* è una libreria per rappresentazioni grafiche 2D, utilizzata soprattutto in Data Science ed IA. Nel progetto gestisce il plotting del grafico sul valore della fitness (capitolo 4).
- *Graphviz* è un software di visualizzazione per i grafi. È stato usato per la rappresentazione delle soluzioni migliori, ovvero gli alberi generati dall'algoritmo di programmazione genetica.

2.2 Dataset

Il dataset utilizzato per addestrare i vari algoritmi è il *Berkeley Segmentation Data Set and Benchmarks 500 (BSDS500)*, creato dal dipartimento di Computer Vision dell'Università della California [41]. È composto da 500 immagini naturali a colori, con soggetti di ogni tipo e di dimensioni varie. Il dataset propone una suddivisione di 300 immagini per il *training set* e di 200 per il test set; ogni immagine ha inoltre una versione segmentata manualmente che rappresenta la *ground truth* per il problema di *supervised learning* dell'edge detection.



Figura 2.3: Un esempio del dataset BSDS500: a sinistra l'immagine originale mentre a destra l'immagine segmentata.

Bisogna sottolineare che il dataset è stato pensato per essere usato nell'addestramento di algoritmi di *segmentazione*; un'immagine con dei bordi più evidenti implica che più soggetti hanno definito quel bordo come ben visibile e lo stesso ragionamento si applica per i bordi meno evidenti nella fase di annotazione.

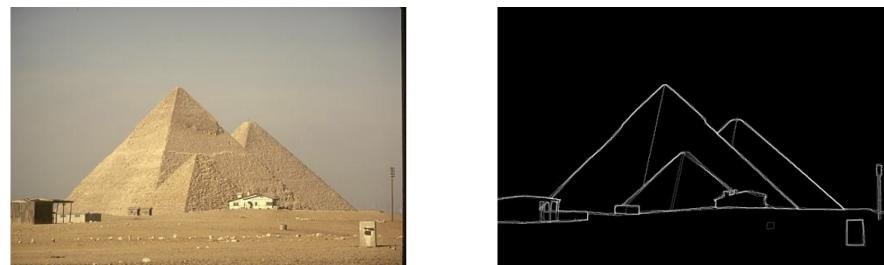


Figura 2.4: Un secondo esempio del dataset BSDS500

2.3 Operazioni d'adattamento alle task

Per adattare al meglio il dataset agli scopi dell'attività di tesi e per ottenere migliori prestazioni in termini di tempo e performance sono state effettuate alcune modifiche:

1. Riduzione del numero di immagini e suddivisione del *training set* per le varie task. Dalle 300 immagini proposte dal dataset per il training set ne sono state selezionate 60, suddivise tra le due task affrontate: 30 per il *denoise* e 30 per l'*edge detection*. Il test set è composto da 15 immagini.
2. Conversione in grayscale. Attraverso l'utilizzo del metodo *cvtColor* di openCV (sezione 2.1) le immagini a colori sono state convertite in grayscale con valori che vanno da 0 a 255; lavorare in grayscale oltre a ridurre i tempi d'elaborazione dell'algoritmo (basti pensare che per immagini RGB sono necessari 3 canali di tonalità distinte su cui lavorare) facilita le operazioni computazionali degli algoritmi GP.
3. Resize delle immagini mediante uno script con Numpy. L'algoritmo riceve in ingresso un'immagine di dimensione variabile, ne estrae la matrice di pixel e ne effettua un *resize* portando l'immagine ad una dimensione predefinita di 128x128 pixel. L'algoritmo infine salva l'array bidimensionale dell'immagine ridimensionata in un file di testo che verrà utilizzato come input dagli algoritmi di programmazione genetica.

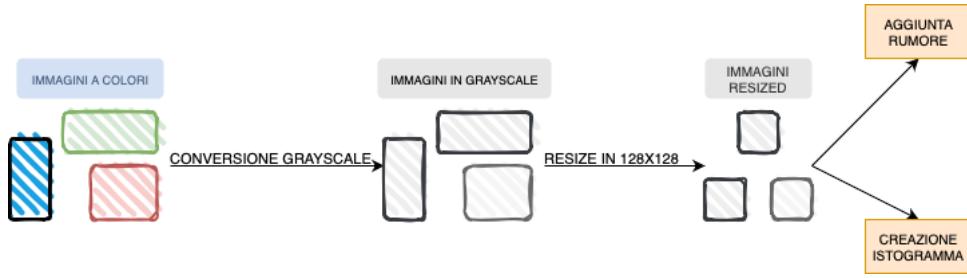


Figura 2.5: Schema riassuntivo sulle prime operazioni effettuate sul dataset

2.3.1 Operazioni preliminari sul dataset per il task di denoise

Per il task di *denoise* l'algoritmo di GP richiede in input l'immagine senza rumore e l'immagine con il rumore, quest'ultima viene generata mediante un algoritmo creato appositamente per l'attività di tesi. L'algoritmo apre una directory passata dall'utente, ad esempio '*trainset/tran_denoise/*', ed iterativamente aggiunge del rumore Gaussiano ad ogni immagine salvando il risultato in una sub-directory, riprendendo l'esempio '*trainset/tran_denoise/noise/*'. Il rumore Gaussiano è un tipo di rumore che ha una *funzione di densità di probabilità* (PDF) con *distribuzione gaussiana*. La densità di probabilità di una variabile Gaussiana random z è data da:

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(z-\mu)^2}{2\sigma^2}} \quad (2.1)$$

con z che rappresenta il livello di grigio, μ la *media* dei valori di grigio e infine σ la *deviazione standard*. [42]

Il rumore gaussiano nelle immagini digitali si presenta durante la fase di acquisizione, può essere causato da un sensore per via della scarsa illuminazione o per una temperatura eccessivamente alta (o bassa).

Per la creazione dell'immagine rumorosa viene generata una distribuzione gaussiana mediante la funzione *normal()* di Numpy. L'array bidimensionale ottenuto da queste operazioni viene sommato, tramite Numpy, all'immagine originale.

La funzione utilizzata per la creazione della PDF gaussiana richiede come

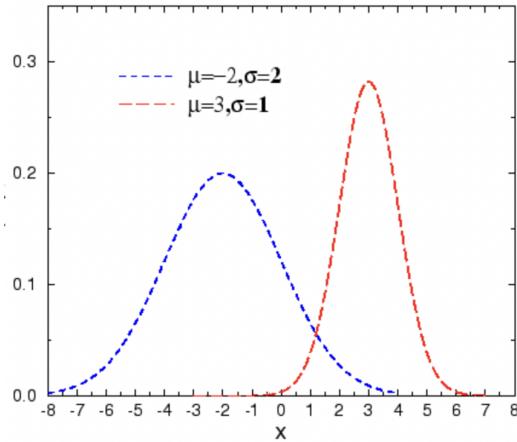


Figura 2.6: Rappresentazione grafica di una PDF Gaussiana

parametri la media e la deviazione standard, μ e σ nell'equazione 2.1; questi parametri vengono generati mediante la funzione *uniform* che restituisce un numero float in un intervallo definito dall'utente. L'intervallo utilizzato per l'attività di tesi è $[1.5, 30]$ estremi inclusi, il float sorteggiato viene arrotondato alle prime quattro cifre decimali mediante la funzione di python *round()*.

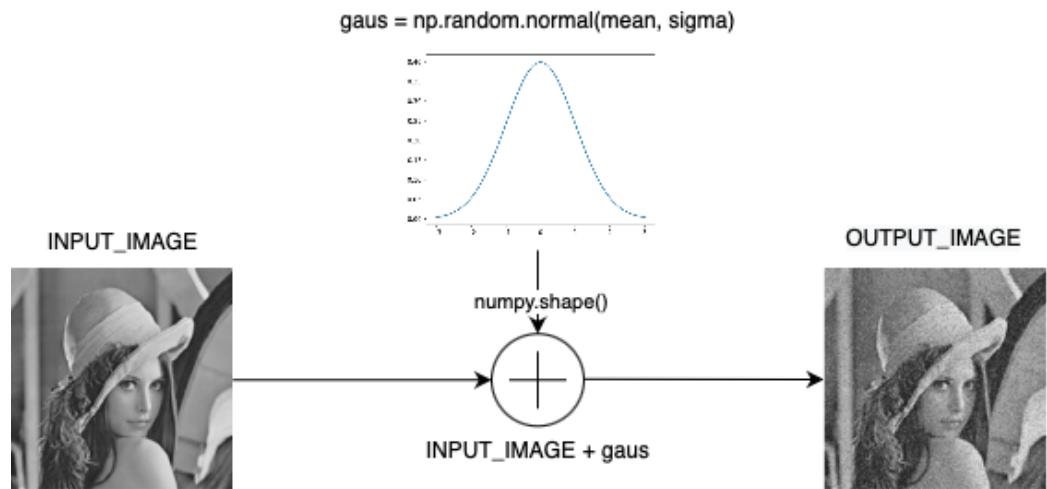


Figura 2.7: Schema sul funzionamento dell'algoritmo di aggiunta del rumore

All'immagine appena modificata viene applicata una trasformazione lineare

per mantenere tutti i suoi pixel su valori compresi nel dominio [0, 255]:

$$x_{(i,j)} = 255 \cdot \frac{(x_{(i,j)} - \min(X))}{\max(X) - \min(X)} \quad (2.2)$$

$x_{(i,j)}$ è un pixel in posizione (i, j) dell'immagine modificata e $\min(X)$ e $\max(X)$ rappresentano il minimo ed il massimo valore ottenuti dall'immagine modificata. Di seguito alcuni esempi sull'applicazione dell'algoritmo d'aggiunta del rumore sul training set:

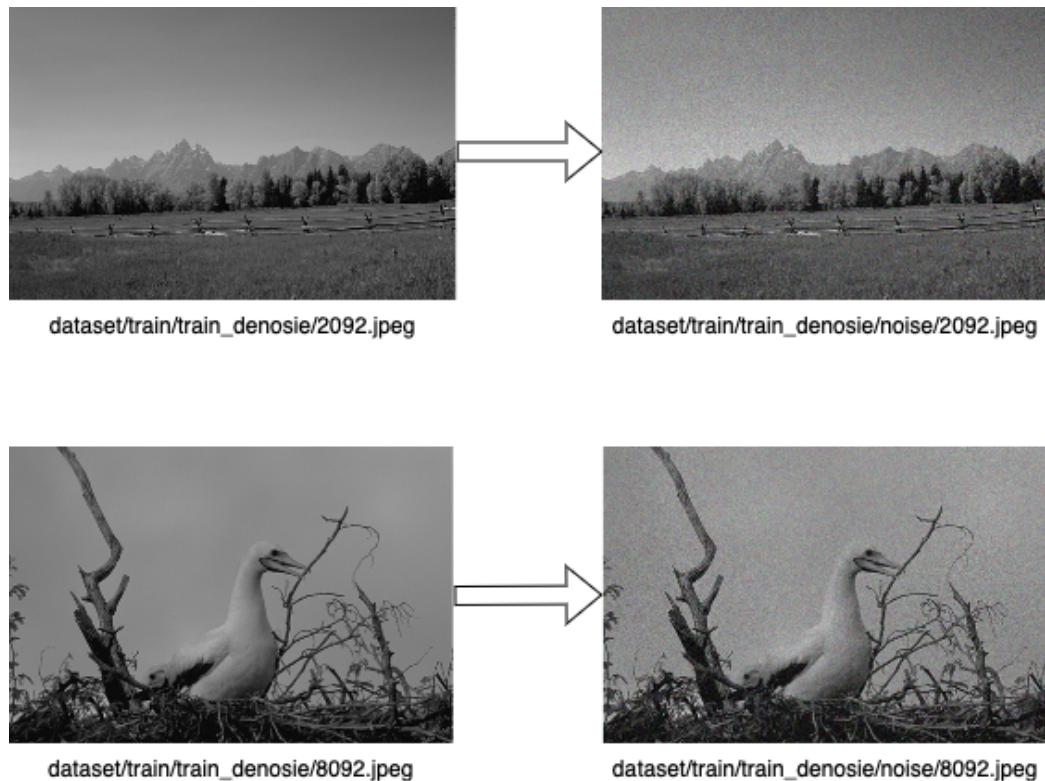


Figura 2.8: Risultati algoritmo di aggiunta del rumore

Capitolo 3

Realizzazione del progetto

Dopo aver effettuato le operazioni preliminari sul dataset illustrate nella sezione 2.3, inizia lo sviluppo degli algoritmi di programmazione genetica per risolvere le task. In questo capitolo verrà chiarito il concetto di *convolution filtering* e sarà analizzato nel dettaglio tutto quello che è stato fatto per la creazione di filtri convolutivi mediante la programmazione genetica.

3.1 Filtraggio convolutivo

Il filtraggio convolutivo è usato per modificare le caratteristiche delle frequenze spaziali di un’immagine, avviene mediante l’operazione matematica di convoluzione con un *Kernel* (o *mask*) usato per filtri *general purpose* (*blurring, denoise, edge detection* etc.). Un Kernel è una matrice numerica intera di piccole dimensioni, solitamente 3x3 o 5x5; le dimensioni differenti del Kernel permettono di contenere pattern numerici diversi e di conseguenza diversi risultati durante l’operazione di convoluzione [20].

La convoluzione del Kernel con un’immagine consiste nel determinare il valore di un pixel centrale rispetto ad una finestra di dimensioni pari a quelle del Kernel sommando fra loro i valori dei pixel vicini pesati mediante moltiplicazione per il corrispondente valore del Kernel.

Da un punto di vista matematico la convoluzione di un’immagine con un Kernel 3x3 può essere espressa in questo modo:

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2, 2] = (a \cdot 1) + (b \cdot 2) + (c \cdot 3) + (d \cdot 4) + (e \cdot 5) + (f \cdot 6) + (g \cdot 7) + (h \cdot 8) + (i \cdot 9) \quad (3.1)$$

La matrice con i coefficienti letterali rappresenta l’immagine mentre quella con i coefficienti numerici il Kernel. I valori di un determinato pixel posto al centro di una finestra di dimensione pari a quella del Kernel nell’immagine di output sono calcolati moltiplicando ogni valore del Kernel con il corrispondente valore del pixel dell’immagine in input e sommando poi i risultati.

Per quanto riguarda la gestione dei bordi di un’immagine possono essere applicate diverse tecniche per gestirli, le più comuni sono:

- estendere i bordi più vicini tra loro quanto basta per provvedere i valori necessari alla convoluzione. I pixel che si trovano sugli angoli sono estesi su più posti su due vertici a 90° fra loro.
- L’immagine viene ”avvolta” su se stessa , prendendo i valori dall’angolo o bordo opposto.
- Utilizzare valori costanti per i pixel all’esterno delle immagini, spesso i valori rappresentano il nero o il grigio.

3.2 Uso della GP per la creazione di filtri per la rimozione del rumore

Il problema di riduzione del rumore è stato formulato in questo modo: “*Data l’immagine a livelli di grigio $I(x, y)$ e una sua versione rumorosa $O(x, y) = I(x, y) + N$, con N che rappresenta il rumore gaussiano, trovare una funzione $f(x, y)$ tale che: $I(x, y) = f(O(x, y))$ sia il più possibile simile a $I(x, y)$* ”.

Un dato da ricordare è che il dominio su cui l’algoritmo deve operare è composto da tutti i numeri naturali compresi tra 0 e 255, ovvero le possibili

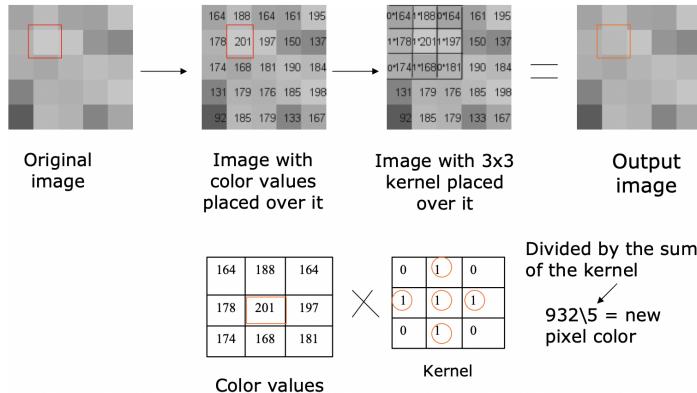


Figura 3.1: Esempio di un filtro convolutivo applicato su un’immagine [20]

tonalità di grigio che può assumere un pixel.

Nell’attività di tesi si affronta il problema creando dei Kernel 3x3 e 5x5 mediante l’uso della programmazione genetica prendendo in input delle finestre 3x3 o 5x5 campionate da punti casuali delle immagini del training set grazie all’utilizzo della libreria Numpy.

3.2.1 Nodi funzione e nodi terminali

Le funzioni da far utilizzare all’algoritmo di GP per evolversi sono:

- *safeadd* è l’operazione algebrica della somma resa sicura limitando i possibili risultati ad un valore massimo di 255, ha una cardinalità pari a 2. In linguaggio matematico:

$$\text{safeadd}(a, b) = \begin{cases} a + b & \text{se } (a + b) < 255 \\ 255 & \text{altrimenti} \end{cases} \quad (3.2)$$

- *safesub* è una sottrazione resa sicura ponendo pari a 0 ogni possibile risultato minore di 0. La cardinalità è 2. In linguaggio matematico:

$$\text{safesub}(a, b) = \begin{cases} 0 & \text{se } (a - b) < 0 \\ a - b & \text{altrimenti} \end{cases} \quad (3.3)$$

- *safemul* è una moltiplicazione i cui risultati sono limitati superiormente ad un valore pari a 255 ed inferiormente a 0. Ha cardinalità 2, in linguaggio matematico può essere vista come:

$$safemul(a, b) = \begin{cases} 0 & se (a \cdot b) < 0 \\ 255 & se (a \cdot b) \geq 255 \\ a \cdot b & altrimenti \end{cases} \quad (3.4)$$

- *safediv* è una divisione che restituisce come risultato 0 se il denominatore o il numeratore sono pari a 0 o se il risultato della divisione è minore di 0. Se il risultato della divisione è maggiore di 255 viene restituito 255. La cardinalità è 2, in linguaggio matematico:

$$safediv(a, b) = \begin{cases} 0 & se \frac{a}{b} < 0 \\ 0 & se a = 0 oppure b = 0 \\ 255 & se \frac{a}{b} \geq 255 \\ \frac{a}{b} & altrimenti \end{cases} \quad (3.5)$$

- *avg2* esegue la media di due numeri, se il numeratore è pari a 0 viene restituito 0 come risultato. In linguaggio matematico:

$$avg2(a, b) = \frac{a + b}{2} \quad (3.6)$$

- *avg3* stessa operazione di *avg2* ma con 3 numeri, la cardinalità cambia da 2 a 3. In linguaggio matematico:

$$avg3(a, b, c) = \frac{a + b + c}{3} \quad (3.7)$$

Il *PrimitiveSet* è inizializzato passando il numero di argomenti pari a 9 o 25 in base alla dimensione del Kernel che si vuole ottenere. Mediante la funzione *addPrimitive()* si aggiungono le funzioni appena mostrate al *PrimitiveSet*. Gli elementi appartenenti all'insieme dei nodi terminali dell'algoritmo sono rappresentati dal valore dei pixel dell'immagine in una finestra quadrata di

```

def safediv(a, b):
    if b == 0:
        return 0
    try:
        s = a / b
    except:
        return 0
    if s < 0:
        return 0
    if s > 255:
        return 255

def avg2(a, b):
    try:
        return (a + b) / 2
    except:
        return 0

```

Figura 3.2: Codice sorgente di alcune funzioni implementate per l'algoritmo di denoise

dimensioni $N \times N$.

Nel caso di una matrice 3x3 possono essere rappresentati in questo modo:

$$\begin{bmatrix} I(x-1, y-1) & I(x, y-1) & I(x+1, y-1) \\ I(x-1, y) & I(x, y) & I(x+1, y) \\ I(x-1, y+1) & I(x, y+1) & I(x+1, y+1) \end{bmatrix} \quad (3.8)$$

per un Kernel 5x5:

$$\begin{bmatrix} I(x-2, y-2) & I(x-1, y-2) & I(x, y-2) & I(x+1, y-2) & I(x+2, y-2) \\ I(x-2, y-1) & I(x-1, y-1) & I(x, y-1) & I(x+1, y-1) & I(x+2, y-1) \\ I(x-2, y) & I(x-1, y) & I(x, y) & I(x+1, y) & I(x+2, y) \\ I(x-2, y+1) & I(x-1, y+1) & I(x, y+1) & I(x+1, y+1) & I(x+2, y+1) \\ I(x-2, y+2) & I(x-1, y+2) & I(x, y+2) & I(x+1, y+2) & I(x+2, y+2) \end{bmatrix} \quad (3.9)$$

In DEAP è possibile aggiungere i nodi terminali mediante la funzione *renameArguments()*. Oltre a questi argomenti sono state aggiunte tre costanti *effimere*; le costanti effimere sono delle funzioni senza argomenti che restituiscono un valore casuale. La prima volta che viene invocata, genera un numero casuale in un intervallo prefissato. Tutte le altre volte restituisce

sempre il numero generato la prima volta. Le costanti effimere utilizzate sono:

1. *rand30*, che restituisce un valore casuale tra 1 e 30.
2. *rand01*, che restituisce un valore casuale tra 0 ed 1.
3. *rand255*, che restituisce un valore casuale tra 0 ed 255.

3.2.2 Fitness

```
def evalImage(individual, points):
    func = toolbox.compile(expr=individual)

    sqerror=0

    for p in points:
        try:
            sqerror += numpy.power( ( func(
                input_sample[p[0]-1,p[1]-1],
                input_sample[p[0]-1,p[1]],
                input_sample[p[0]-1,p[1]+1],
                input_sample[p[0],p[1]-1],
                input_sample[p],
                input_sample[p[0],p[1]+1],
                input_sample[p[0]+1,p[1]-1],
                input_sample[p[0]+1,p[1]-1],
                input_sample[p[0]+1,p[1]+1],
            ) - result_sample[p]), 2)
        except:
            sqerror += 1000.0

    return sqerror / len(points),
```

Figura 3.3: Codice sorgente della funzione fitness per l’algoritmo generatore di filtri per il denoise

Il codice in figura 3.3 mostra la funzione utilizzata come fitness dell’algoritmo GP. Ogni individuo viene valutato in base all’*errore quadratrico medio* (MSE) tra i pixel dell’immagine, elaborata mediante il filtro ottenuto dalla GP, “*func(input_sample[p[0] - 1,p [1] - 1], ...)*”, e l’immagine senza rumore,

“*result_sample[p]*”; minore è il valore del MSE e migliore sarà l’individuo. Da un punto di vista matematico la fitness può essere scritta così:

$$\frac{\sum_{i=1}^n (x_i - \hat{x}_i)^2}{n} \quad (3.10)$$

Con n pari al numero di pixel dell’immagine di riferimento, x_i è il pixel i-esimo dell’immagine generata dal filtro e \hat{x}_i il pixel dell’immagine di riferimento.

La fitness appena definita viene registrata nell’algoritmo GP mediante il metodo della toolbox “*toolbox.register()*” a cui vengono passati i punti da valutare.

3.2.3 Generazione ed evoluzione della popolazione

La popolazione viene generata mediante il metodo *half and half* che genera metà popolazione con il metodo *grow* e l’altra metà mediante il metodo *full* (entrambi visti nella sezione 1.2.4); la profondità limite degli alberi dei primi individui è pari a 3.

La selezione avviene mediante un *torneo* a cui partecipano 5 individui, il *crossover* utilizza il metodo *one point* (sezione 1.2.5) mentre le mutazioni sono uniformi: viene selezionato casualmente un punto su un albero dell’individuo e il sotto-albero avente come radice quel punto viene sostituito da un nodo terminale. Gli individui modificati dagli operatori di crossover e mutazione hanno la loro fitness invalidata, questi individui devono quindi essere clonati prima di essere modificati in modo tale che la popolazione modificata sia indipendente dalla popolazione in input. Una popolazione P_0 viene clonata mediante la funzione *toolbox.clone()* e il risultato viene messo nella popolazione della prole P_1 . Viene eseguito un loop su P_1 per accoppiare gli individui consecutivi x_i e x_{i+1} mediante il crossover *one-point* con una probabilità passata dall’utente. I figli risultanti y_i e y_{i+1} sostituiscono i loro rispettivi genitori in P_1 . Successivamente un secondo loop su P_1 appena modificato muta ogni individuo della popolazione con una probabilità pari ad un parametro passato dall’utente. Quando un individuo è mutato viene sostituito direttamente nella popolazione P_1 .

Gli individui generati tramite crossover o una mutazione vengono nuovamente valutati, DEAP permette di utilizzare la *parallel computation* (multithreading) per accelerare i tempi d'esecuzione. Per ogni generazione vengono salvati gli individui migliori in una classifica (*Hall of fame*) con alcune statistiche a riguardo, è ordinata lessicograficamente in ogni momento in modo che il primo elemento sia sempre l'individuo migliore con il miglior valore della fitness. L'algoritmo restituisce come output gli n migliori individui (con n scelto dall'utente) ottenuti dall'esecuzione su un *logbook*, la dimensione della *hall of fame* scelta è 3. L'applicazione dei filtri convolutivi risultati ottenuti con il GP avviene mediante l'algoritmo illustrato nella sezione 4.1.

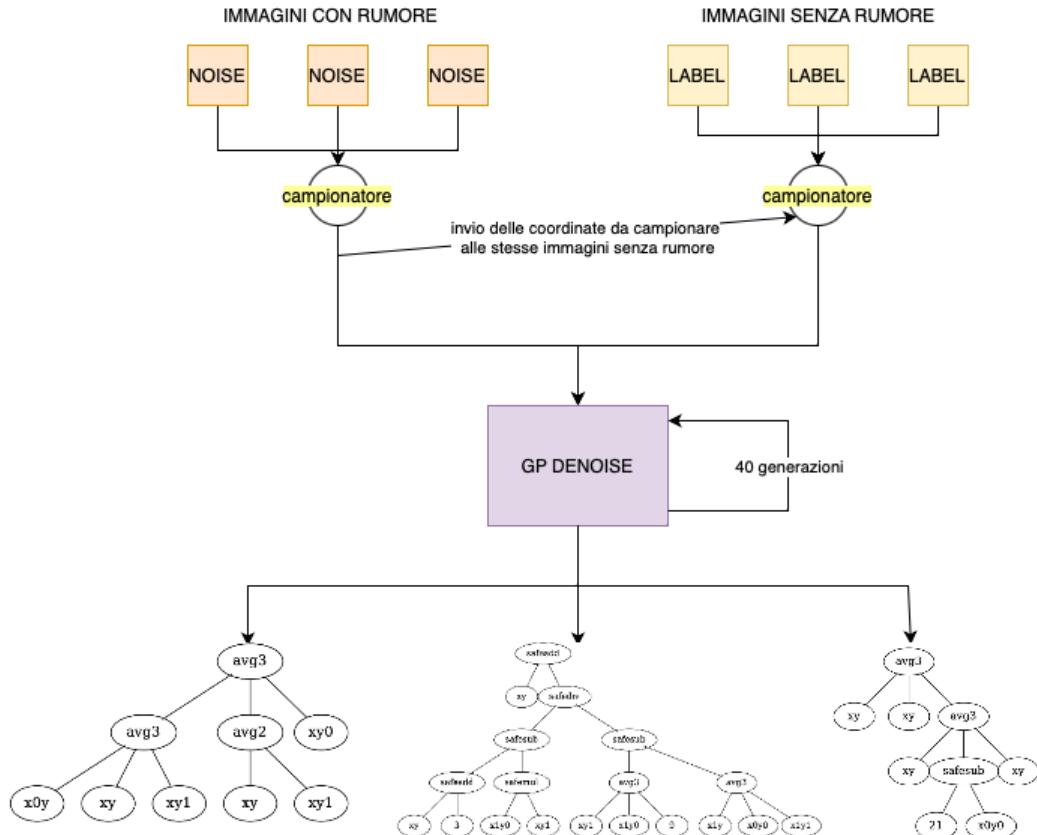


Figura 3.4: Schema sul funzionamento dell'algoritmo per la generazione di filtri per il denoise

3.2.4 Parametri utilizzati ed alcuni esempi

Tutti i valori dei parametri utilizzati si basano su valori comuni impiegati in letteratura e su dei tentativi di *grid search* su altri valori vicini ad essi.

- La profondità massima dell'albero è 8; numeri troppo grandi oltre ad aumentare notevolmente i tempi d'esecuzione potrebbero portare a soluzioni con operazioni ridondanti ottenibili con alberi più piccoli.
- La probabilità di crossover è pari a 0.3, la probabilità di mutazione invece è pari a 0.4.
- Il numero di generazioni è 40, la popolazione è composta da 400 individui (ovvero 400 alberi), un giusto compromesso tra performance delle soluzioni e tempo d'esecuzione.

Di seguito alcuni dei migliori individui ottenuti dall'esecuzione dell'algoritmo sul training set.

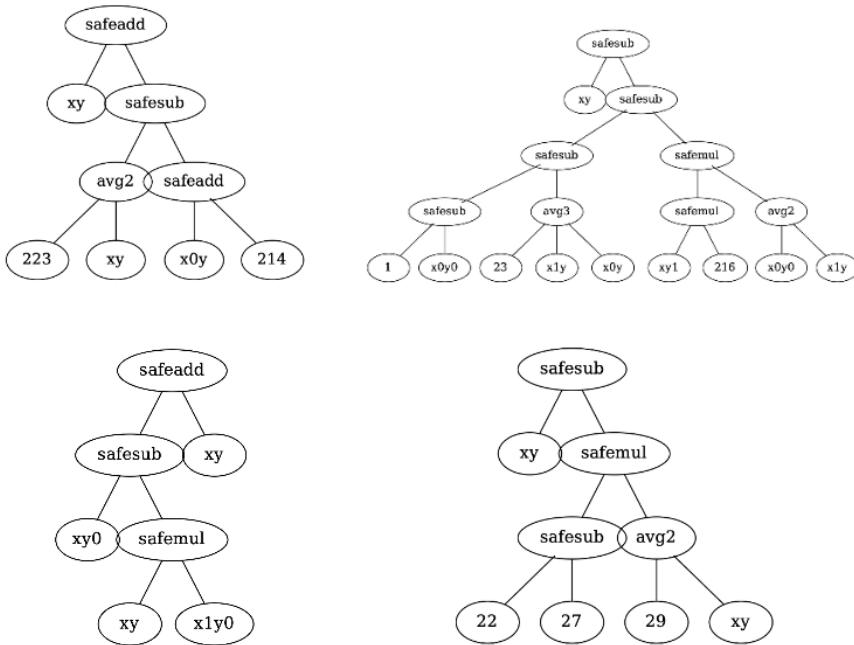


Figura 3.5: Individui ottenuti dall'algoritmo GP di denoise per un Kernel 3x3

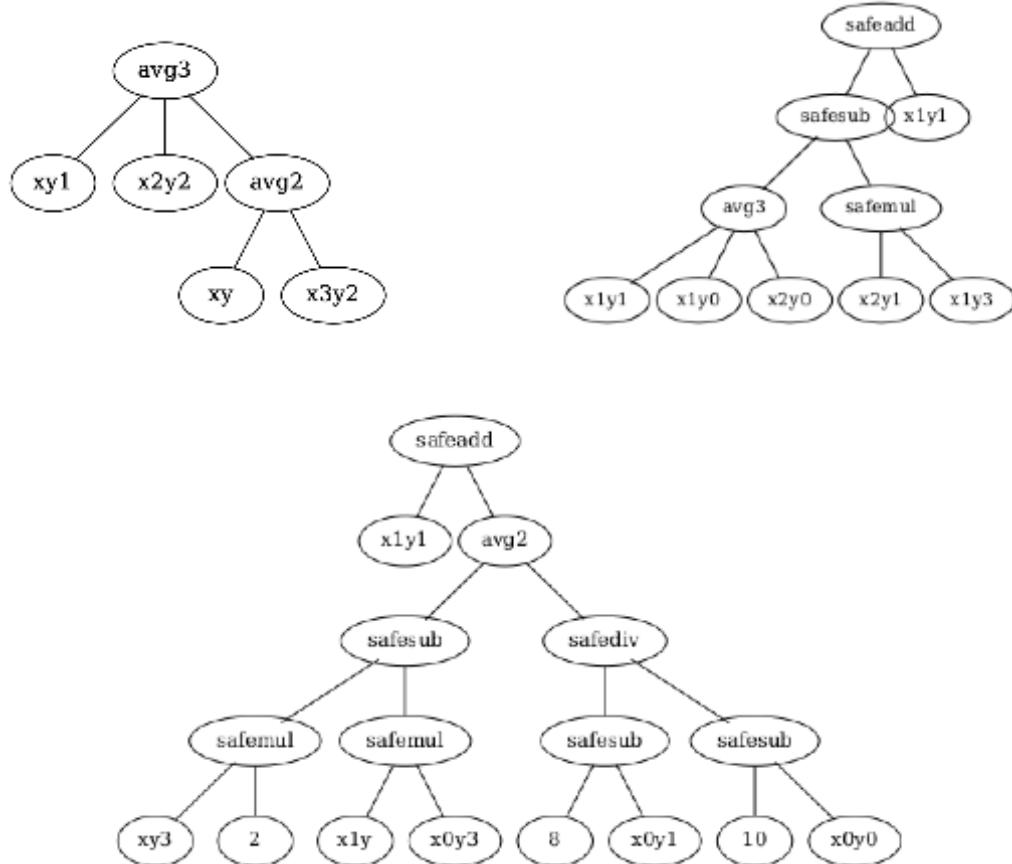


Figura 3.6: Individui ottenuti dall'algoritmo GP di denoise per un Kernel 5x5

Il tempo d'esecuzione dell'algoritmo di generazione di filtri per denoise è di circa 5 minuti. Le immagini rappresentate in figura 3.5 e 3.6 sono state generate mediante la libreria *Graphviz*. L'algoritmo restituisce anche un file txt in cui l'individuo viene rappresentato sotto forma di stringa:

```
safesub(xy, safesub(safeadd(x0y, 124), safemul(221, x0y0)))
```

3.3 Uso della GP per la creazione di filtri per l'edge detection

Per la creazione di filtri per il task di edge detection è stato utilizzato un approccio molto simile a quello visto per il task di denoise.

Anche in questo caso abbiamo una task di *supervised learning* con l'uscita desiderata rappresentata dall'immagine con i contorni *ground truth* forniti dal dataset e come input l'immagine originale. L'algoritmo ha in ingresso n campioni 30×30 casuali presi dalle immagini del training set, per l'attività di tesi n è pari a 10.

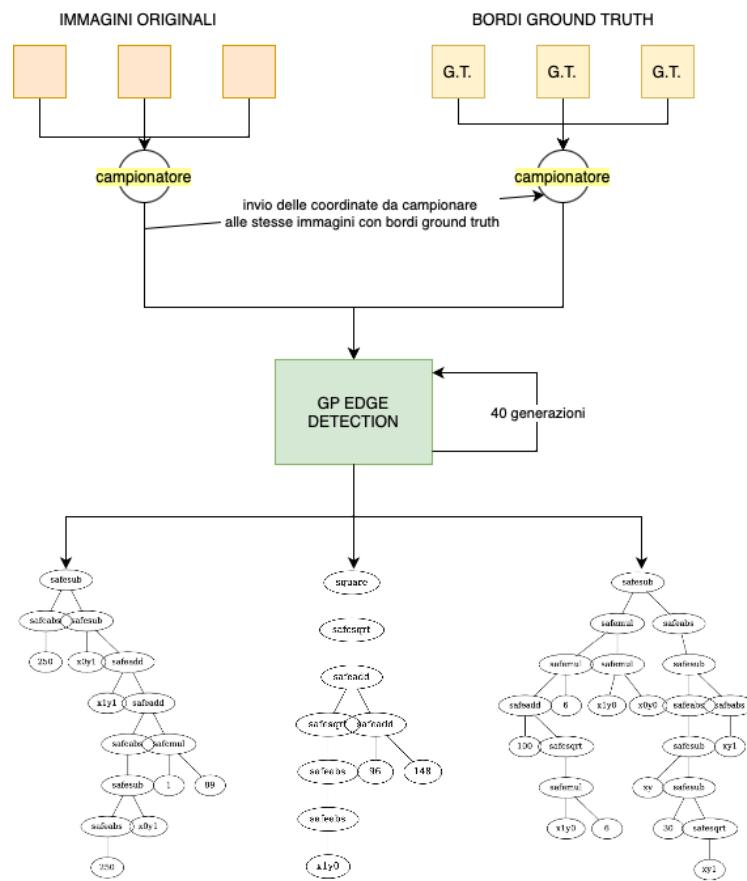


Figura 3.7: Schema dell'algoritmo per la generazione di filtri per l'edge detection

Oltre al funzionamento, anche da un punto di vista strutturale si possono

notare molte somiglianze con l'algoritmo per la generazione di filtri per il denoise, questo perchè i domini su cui l'algoritmo di programmazione genetica ricerca la soluzione ottimale sono molto simili. Uno dei vantaggi dell'utilizzo della GP risiede proprio nella semplicità per l'adattamento a task differenti.

3.3.1 Nodi funzione e nodi terminali

Il dominio su cui lavora l'algoritmo di edge detection è $[-255, 255]$ in quanto deve individuare sia i bordi positivi che quelli negativi, per questo motivo le funzioni matematiche di base viste nella sezione 3.2.1 (esclusa la media) sono state introdotte modificando il limite inferiore da 0 a -255. Inoltre, sono state aggiunte altre 3 operazioni:

- *square*, è l'elevamento al quadrato di un numero protetto da una soglia di 255. Ha una cardinalità pari ad 1, in linguaggio matematico :

$$\text{square}(a) = \begin{cases} a^2 & \text{se } (a^2) < 255 \\ 255 & \text{altrimenti} \end{cases} \quad (3.11)$$

- *safeabs* applica il valore assoluto di un numero, anche questa funzione ha una cardinalità pari ad 1. In linguaggio matematico:

$$\text{safeabs}(a) = \begin{cases} x = -a & \text{se } a < 0 \\ x = a & \text{se } a > 0 \\ 255 & \text{se } x > 255 \\ 0 & \text{in caso di errori} \end{cases} \quad (3.12)$$

- *safesqrt* è la radice quadrata protetta di un numero, se la radice del numero è maggiore di 255 ritorna 255, se ci sono errori durante l'operazione (es. radice quadrata di un numero negativo) ritorna 0.

Ha cardinalità 1, può essere scritta come:

$$\text{safesqrt}(a) = \begin{cases} \sqrt{a} & \text{se } a > 0 \\ 255 & \text{se } \sqrt{a} > 255 \\ 0 & \text{altrimenti} \end{cases} \quad (3.13)$$

3.3.2 La funzione fitness

La funzione *fitness* dell'algoritmo viene valutata mediante l'utilizzo del MSE (formula 3.10), in questo caso però il filtro genera sia valori positivi che valori negativi in quanto bisogna valutare tutti gli edges possibili della finestra che si sta analizzando. Per questo motivo durante la fase della valutazione viene applicata la seguente trasformazione utilizzando il valore assoluto:

$$F_{finestra} = \frac{\sum_{i=1}^n (|x_i| - \hat{x}_i)^2}{n} \quad (3.14)$$

Il termine al numeratore $|(x_i)|$ rappresenta il valore assoluto del pixel i-esimo della finestra in input prodotto dall'elaborazione mediante il filtro, \hat{x}_i il pixel i-esimo della label ed n il numero di pixel del campione. L'equazione 3.14 rappresenta il valore della fitness di una finestra su un campione, il valore utilizzato per scegliere l'individuo migliore è dato dalla sommatoria delle $F_{finestra}$ di tutti i campioni passati come input, ovvero:

$$F_{totale} = \sum_{i=0}^m F_{finestra_i} \quad (3.15)$$

Con m pari al numero di campioni 30×30 analizzati.

3.3.3 Parametri utilizzati ed alcuni esempi

I meccanismi di generazione, selezione, crossover e mutazione sono gli stessi illustrati nella sezione 3.2.3; stesso discorso vale per i parametri utilizzati per il tasso di mutazione e di crossover e la dimensione della popolazione. Il numero di individui invece, è stato aumentato a 300 per permettere all'algoritmo di trovare individui migliori.

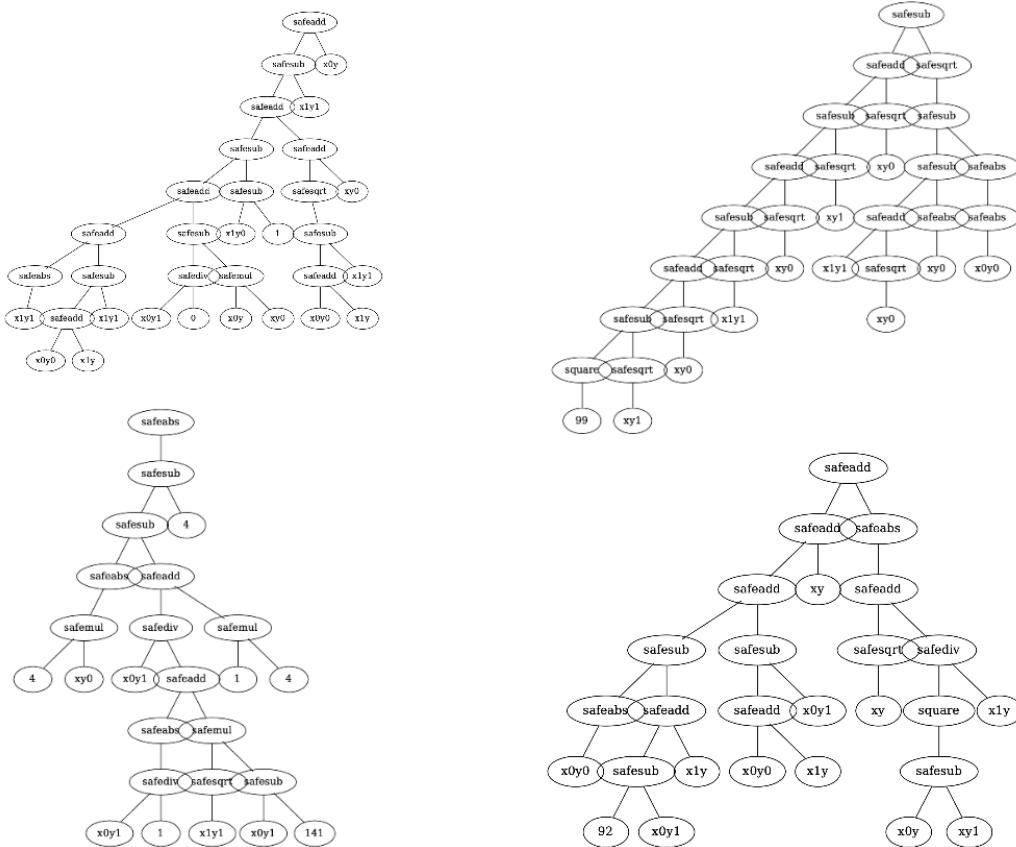


Figura 3.8: Individui ottenuti dall’algoritmo GP di edge detection per un Kernel 3x3

I *logbook* contengono dei *checkpoint* dell’esecuzione dell’algoritmo: ogni dieci generazioni viene salvato l’individuo migliore, permettendo di conservarlo anche se durante l’evoluzione si estinguesse e non comparisse nell’ultima generazione.

Si può notare come per la risoluzione della task di edge detection siano necessari alberi più “complessi”, questo è dovuto al fatto che il task di edge detection da un punto di vista computazionale richiede maggiore lavoro su ogni singolo pixel.

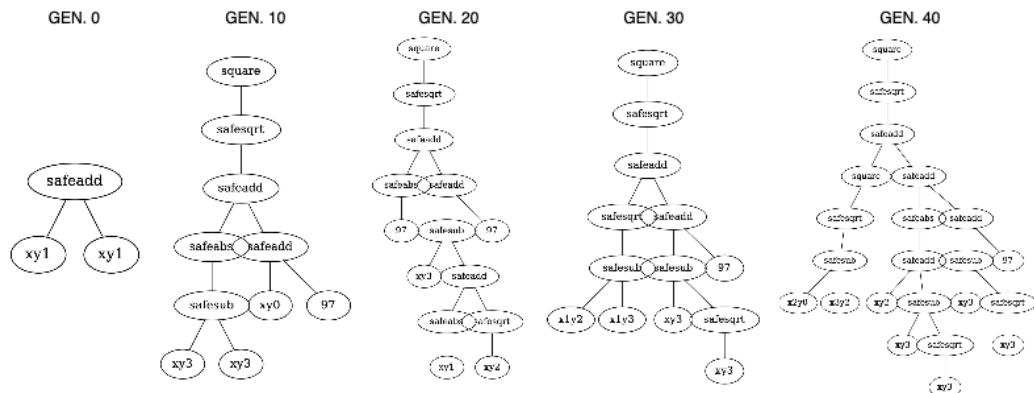


Figura 3.9: Esempio di evoluzione di un individuo con algoritmo GP per l'edge detection

A tutto questo corrisponde anche un aumento in termini di minuti sul tempo d'esecuzione dell'algoritmo: dai 6/7 minuti per il denoise ad oltre 20 per l'edge detection.

Capitolo 4

Risultati

Prima di passare in rassegna i vari risultati ottenuti dall'attività di tesi è necessario approfondire come si passa dai file contenenti gli individui migliori ottenuti dagli algoritmi, alle immagini elaborate mediante tali filtri.

4.1 Algoritmo per applicare i filtri ottenuti sul dataset

Il *logbook* è un dizionario esportato come file in formato PKL (Python Pickle file), creato da pickle, un modulo Python che consente di serializzare su file nel disco oggetti di qualsiasi tipo (dizionari, array, matrici etc.) e deserializzarli nel programma in fase di runtime.

Il dizionario contiene 4 keys: “*file-name*” che contiene il percorso in cui si trova l’immagine da elaborare.

La key “time” contiene la data d’esecuzione dell’algoritmo mentre la key “*BEST*” include l’individuo migliore di sempre (posizione 0 nella hall of fame) in formato stringa di testo.

La key “logbook” racchiude l’oggetto *logbook* di DEAP, contenente tutte le statistiche raccolte durante l’esecuzione dell’algoritmo in ordine cronologico (rispetto alle generazioni).

L’utente può decidere quale statistiche devono essere salvate nel logbook; nel

caso dell’attività di tesi vengono salvate la dimensione media degli individui, il valore minimo, il valore massimo e la deviazione standard della fitness di ogni generazione.

Mediante una funzione nativa di Python l’algoritmo estrae il contenuto del logbook e genera l’albero dell’individuo migliore utilizzando la funzione di DEAP “*PrimitiveTree.from_string()*”. La funzione, oltre alla stringa dell’albero, necessita un PrimitiveSet, ovvero l’insieme di nodi terminali e funzioni necessarie per interpretare l’albero del logbook.

Sono stati creati due algoritmi, uno per ognuna delle task affrontate per l’attività di tesi. L’insieme delle funzioni comprende sia le funzioni utilizzate per l’algoritmo di denoise che quelle per l’algoritmo di edge detection (sezioni 3.2.1 e 3.3.1); anche l’insieme dei nodi terminali è lo stesso visto per gli algoritmi di denoise ed edge detection (incluse le costanti effimere).

Una volta caricato l’individuo migliore del logbook, viene applicato all’immagine indicata dalla key “file_name” e il risultato salvato prima come array Numpy e poi convertito in immagine mediante la libreria PIL.

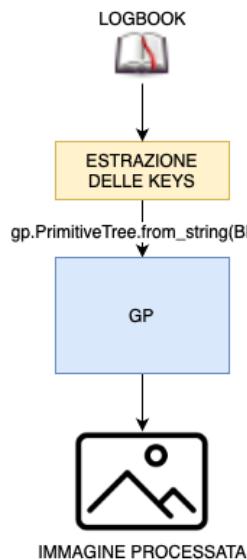


Figura 4.1: Schema sul funzionamento dell’algoritmo per l’image processing con i filtri ottenuti

4.2 Risultati ottenuti con i filtri per il denoise

Il training set è composto da 60 immagini, 30 per il denoise e 30 per l'edge detection. Durante la fase di addestramento vengono generate n finestre di dimensioni 30×30 da immagini del training set scelte a caso; il filtro risultante viene applicato su tutta l'immagine.

4.2.1 Risultati per Kernel di dimensioni 3x3

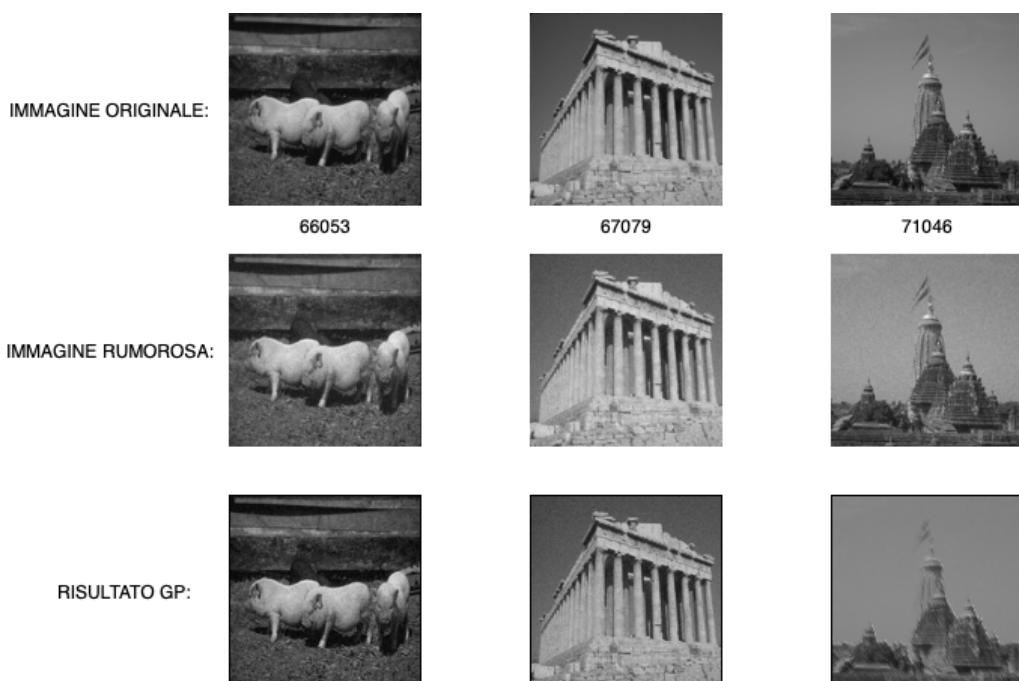


Figura 4.2: Esempio di filtri GP 3x3 di denoise applicati sul training set

La figura 4.2 mostra i risultati ottenuti applicando l'algoritmo della sezione 4.1 su alcune immagini del training set dopo aver ricavato dei filtri con Kernel 3x3 per il denoise mediante l'algoritmo di programmazione genetica (3.2). I risultati ottenuti (terza riga) presentano una riduzione del rumore evidente senza perdere troppo in termini di dettagli; in alcuni casi vengono accentuati maggiormente gli effetti dello smoothing, ad esempio

nell'immagine 71046 della figura 4.1 alcuni dettagli sono stati resi meno visibili rispetto all'immagine originale; questa tipologia di fenomeni è piuttosto comune quando si trattano filtri di questo tipo.

4.2.2 Risultati per Kernel di dimensioni 5x5

Le immagini ottenute dall'elaborazione mediante filtri 5x5 presentano maggiormente l'effetto *smoothing* rispetto a quelle in figura 4.2 a causa della dimensione del Kernel che è maggiore.

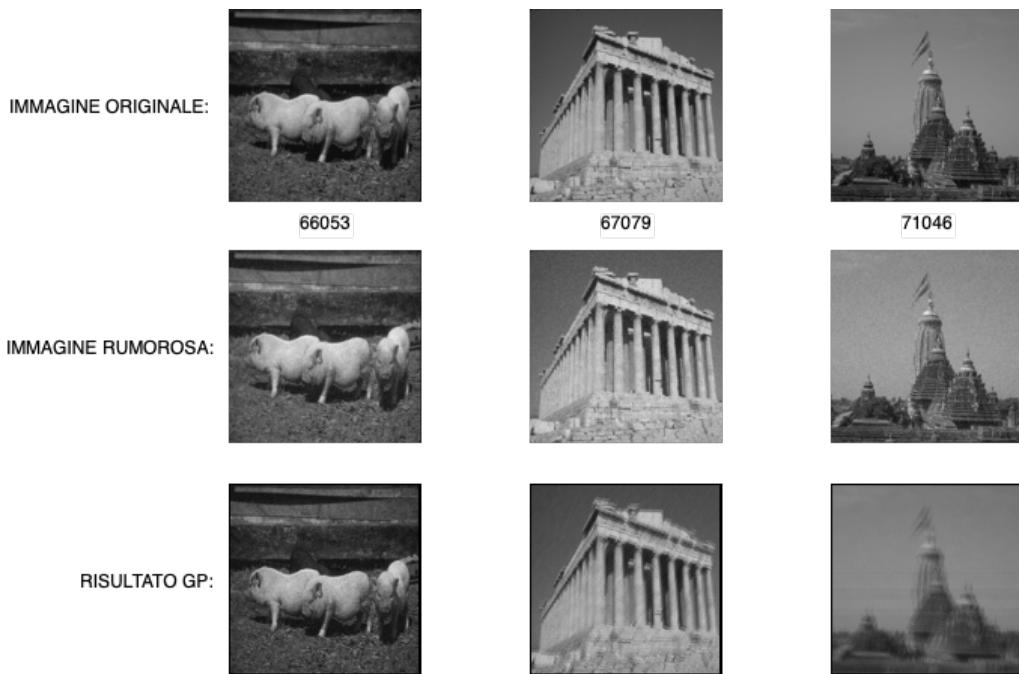


Figura 4.3: Esempio di filtri GP 5x5 di denoise applicati sul training set

Anche in questo caso si possono definire come buoni i risultati ottenuti: le immagini non hanno perso molto in termini di dettagli e presentano una buona riduzione del rumore.

In termini di tempo d'esecuzione entrambi gli algoritmi terminano la generazione dei filtri intorno ai 5/6 minuti; per avere immagini con meno *smoothing* è quindi consigliabile utilizzare il Kernel 3x3.

4.2.3 Statistiche riguardo l'evoluzione

Durante l'evoluzione dell'algoritmo GP vengono raccolte alcune statistiche sulle popolazioni di ogni generazione mediante la classe *Statistics* fornita da DEAP.

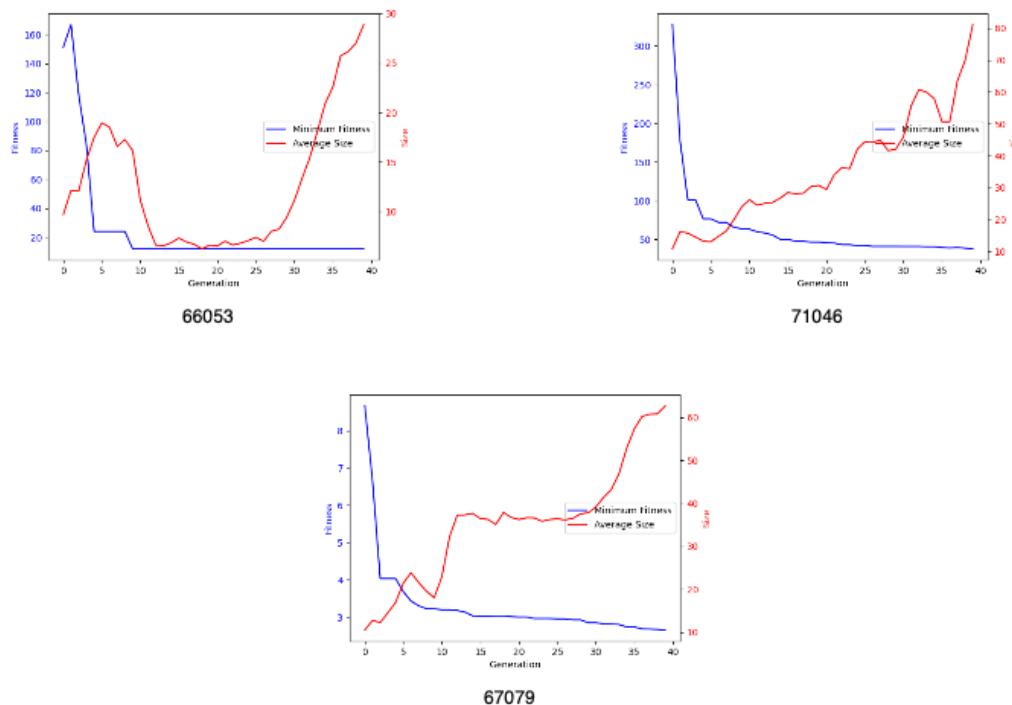


Figura 4.4: Grafici sulle statistiche raccolte durante l'evoluzione dei filtri 3x3 di denoise

I grafici in figura 4.4 sono stati creati con *Matplotlib*: in blu è evidenziato l'andamento del valore minimo della fitness per ogni generazione, in rosso è rappresentata la dimensione¹ media degli individui. Il valore minimo diminuisce all'aumentare delle generazioni, questo perché l'obiettivo per il problema d'ottimizzazione proposto è di minimizzare il valore della fitness. La dimensione media, invece, aumenta e decresce in base alle operazioni di crossover e alle mutazioni effettuate in quella generazione; il motivo è la na-

¹Numero di nodi dell'albero

tura di GP in cui il risultato di crossover e mutazione può avere profondità maggiore degli individui originali. Sia l'ordine di grandezza dei valori minimi della fitness che della media delle dimensioni degli individui variano molto da un'esecuzione ad un'altra.

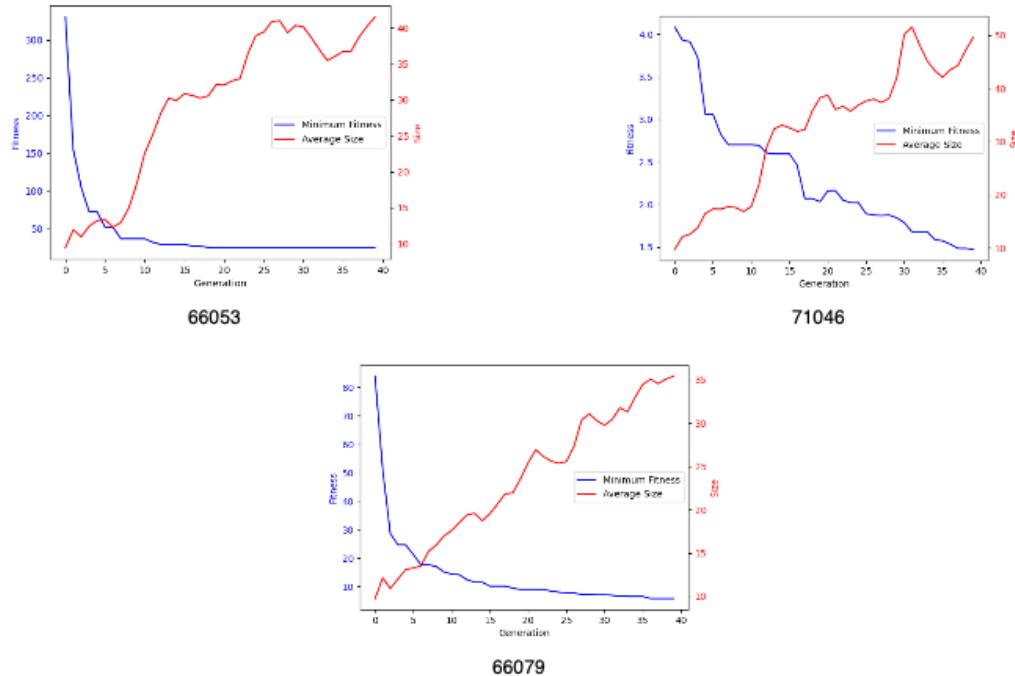


Figura 4.5: Grafici sulle statistiche raccolte durante l'evoluzione dei filtri 5x5 di denoise

È interessante notare come sia per la creazione di filtri con Kernel 3x3 che per quelli con Kernel 5x5, dopo le prime 5 generazioni, avviene una decrescita rapida del valore minimo della fitness, inoltre la dimensione media degli individui dei filtri con Kernel 5x5 tende ad essere minore rispetto a quella dei filtri con Kernel 3x3.

4.2.4 Confronto con metodi esistenti di denoise

Per fare una valutazione comparativa dei risultati ottenuti sono stati applicati due filtri di denoise alle immagini in figura 4.2 mediante uno script creato con la libreria OpenCV. I filtri applicati sono:

- *Median Smoothing*: è un filtraggio convolutivo il cui scopo è quello di rimpiazzare ogni pixel di un’immagine con la mediana dei suoi vicini per risentire meno rispetto alla media, di pixel con valori anomali rispetto ai suoi vicini.
- *Gaussian Smoothing*: anch’esso è un filtraggio convolutivo, la differenza risiede nella composizione del Kernel: in questo caso i suoi valori variano come una Gaussiana in funzione della distanza dal pixel centrale della finestra.

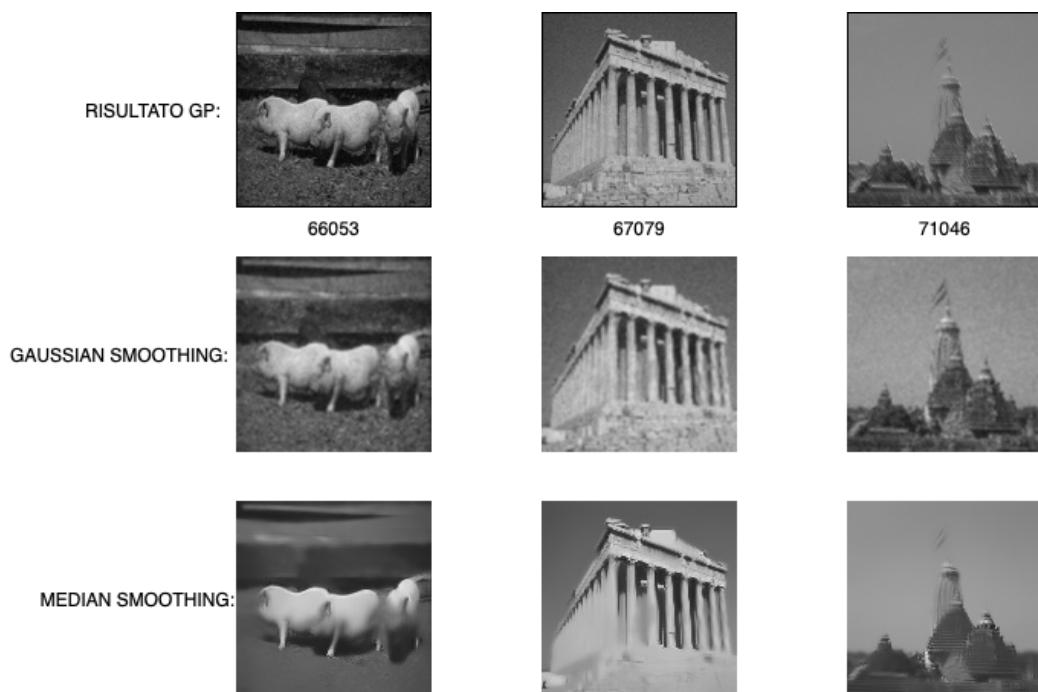


Figura 4.6: Confronto dei risultati ottenuti con un filtro avente Kernel 3x3 con metodi di denoise esistenti

I risultati dei filtri sviluppati per il denoise sono sicuramente superiori a quelli del *median smoothing* che, oltre ad aver eliminato quasi completamente il rumore, hanno anche perso molti dettagli che potrebbero servire per le possibili elaborazioni future da eseguire (ad esempio una task di *segmentation*). Si potrebbe definire uno scontro alla pari con il *Gaussian smoothing*: i risultati dell'algoritmo GP sono leggermente migliori in termini di qualità.

Lo stesso confronto è stato fatto anche con gli esempi dei filtri di denoise con Kernel 5x5.

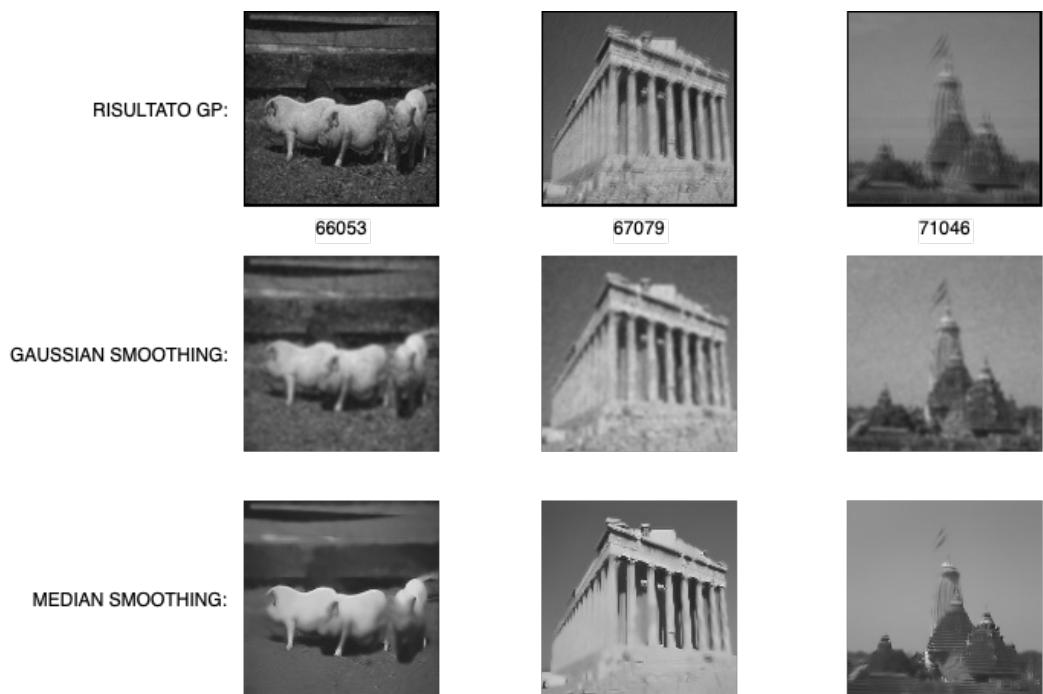


Figura 4.7: Confronto dei risultati ottenuti con un filtro avente Kernel 5x5 con metodi di denoise esistenti

Nonostante i risultati ottenuti con l'utilizzo di filtri GP siano leggermente migliori rispetto agli altri metodi proposti (mantenendo le stesse differenze viste con i filtri 3x3), bisogna fare delle considerazioni importanti per quanto riguarda questo tipo di valutazioni. Gli algoritmi creati per generare i filtri di denoise hanno bisogno di un'immagine di riferimento senza rumore per

ottenere dei risultati, mentre quelli utilizzati per il confronto non ne hanno bisogno.

4.3 Risultati ottenuti con i filtri per l'edge detection

Sia alla label che ai risultati mostrati in questa sezione è stata applicata una funzione di *threshold* per rendere le immagini binarie: sotto una soglia, che per l'attività di tesi è pari a 20, il pixel ha valore zero mentre i pixel con valori maggiori della soglia avranno valore 255. Un pixel bianco indica che in quel punto è presente un bordo.

4.3.1 Risultati con Kernel 3x3

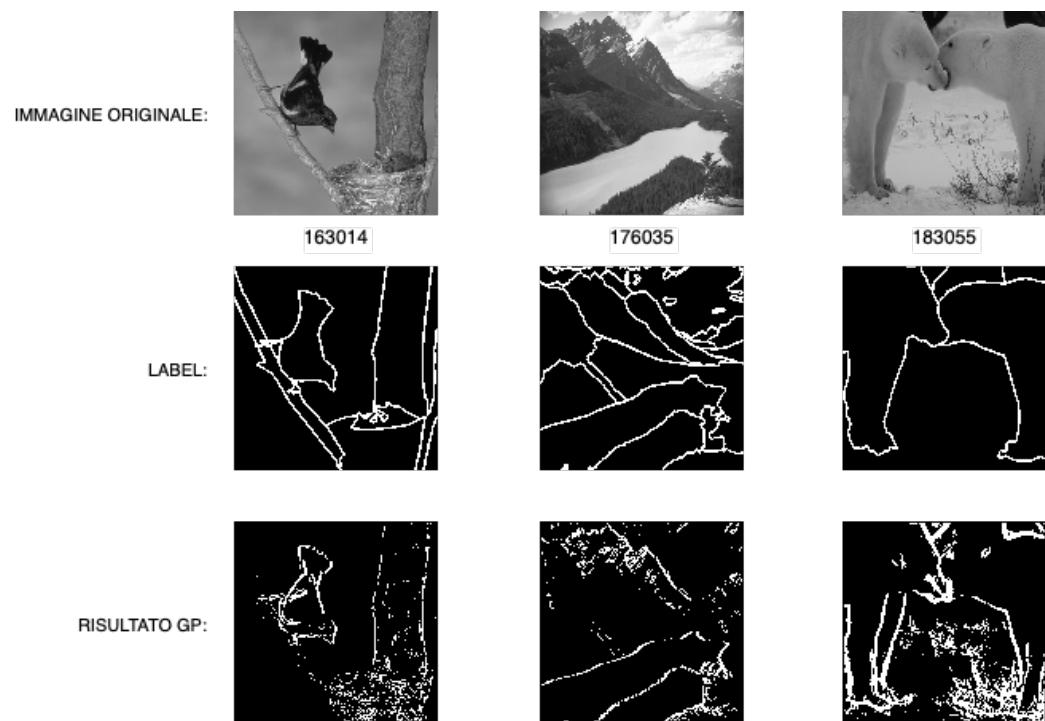


Figura 4.8: Esempio di filtri GP 3x3 di edge detection applicati sul training set

Le immagini scelte per la figura 4.8 rappresentano soggetti diversi su cui effettuare l'edge detection. In tutte le immagini i soggetti da cui ricavare i bordi (indicati dalla label) presentano edge con tonalità diverse su sfondi non uniformi (es. 176035). Nonostante le differenze tra le varie immagini, i risultati ottenuti dai 3 filtri riescono ad avere dei buoni esiti rendendo ben visibili i bordi desiderati, permettendo l'identificazione dei soggetti senza grossi problemi.

4.3.2 Risultati con Kernel 5x5

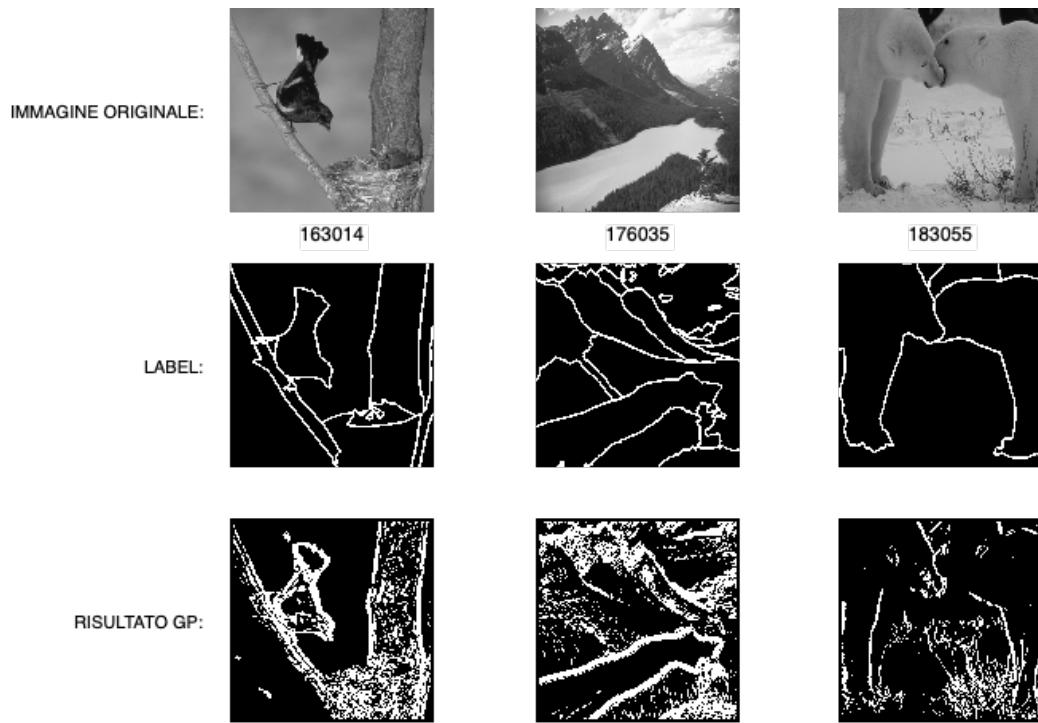


Figura 4.9: Esempio di filtri GP 5x5 di edge detection applicati sul training set

L'aumento della dimensione del Kernel incide significativamente sulle prestazioni ottenute: i bordi risultano più spessi (e come si vedrà anche nella sezione 4.3.4 il fenomeno è comune per i detectors 5x5), risaltano però punti o tratti non voluti, ad esempio nello sfondo dell'immagine 183055, rendendo il risultato meno preciso.

4.3.3 Statistiche riguardo l'evoluzione

Così come visto per gli algoritmi di denoise nella sezione 4.2.3, anche durante l'evoluzione dei filtri di edge detection vengono salvate nel logbook le statistiche riguardanti il valore minimo della fitness per ogni generazione e la dimensione media degli individui.

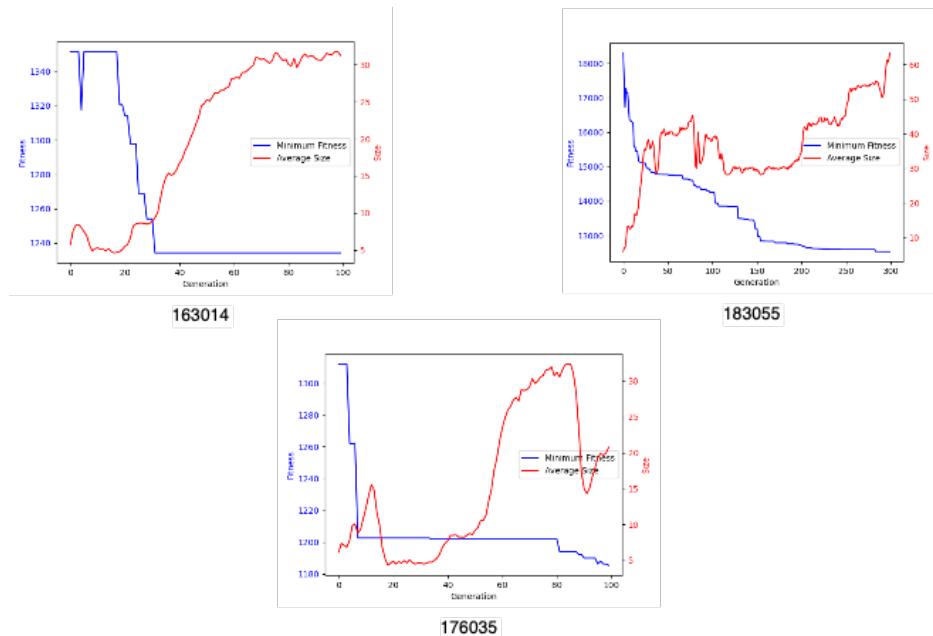


Figura 4.10: Grafici delle statistiche raccolte durante l'evoluzione dei filtri 3x3 di edge detection

Rispetto ai grafici della sezione 4.2.3 i valori minimi della fitness decrescono meno rapidamente da una generazione ad un'altra; questo è dovuto alla complessità maggiore della task. Anche in questo caso la media delle dimensioni degli individui ha un andamento variabile, con valori massimi che oscillano tra i 30 ed i 70 nodi: le dimensioni tendono ad essere mediamente più grandi in quanto il numero di generazioni necessarie per arrivare alla convergenza è di gran lunga superiore a quello utilizzato per il denoise. Questi valori sono ovviamente indicativi e non indicano una caratteristica generale di tutti gli individui generati durante l'attività di tesi.

Osservando i grafici delle figure 4.10 e 4.11 è evidente come per le immagini in

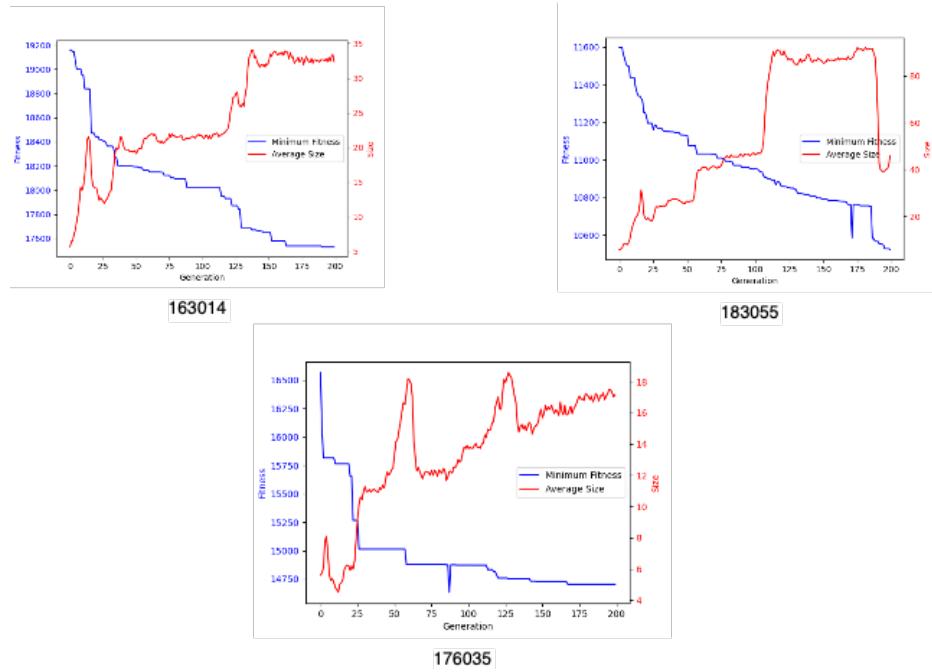


Figura 4.11: Grafici sulle statistiche raccolte durante l’evoluzione dei filtri 5x5 di edge detection

cui il valore minimo della fitness è più basso corrispondono risultati migliori nel rilevamento dei bordi. I picchi della dimensione media degli individui, invece, non sono correlabili con i valori minimi della fitness. Definendo come *true positive* il caso in cui un pixel nell’immagine di riferimento e nell’immagine elaborata dal filtro siano entrambi pari a 255, *true negative* il caso in cui entrambi siano 0 e così via...

Sono stati ricavati i seguenti valori di *F1 score*²:

immagine	3x3	5x5
163014	0,895	0,789
176035	0,880	0,749
183055	0,854	0,879

²metrica usata su dataset sbilanciati, data dalla formula $F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

4.3.4 Confronti con metodi esistenti di edge detection

I metodi utilizzati per confrontare i risultati ottenuti dall'attività di tesi per l'edge detection sono due, entrambi implementati con la libreria *OpenCV*:

- *Sobel edge detector*, calcola le derivate di primo ordine dei pixel seguendo gli assi delle x e delle y utilizzando i Kernel:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$
 e ne effettua la composizione $G = \sqrt{G_x^2 + G_y^2}$
- *Laplacian edge detector*, utilizza un solo Kernel per approssimare le derivate di secondo ordine:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

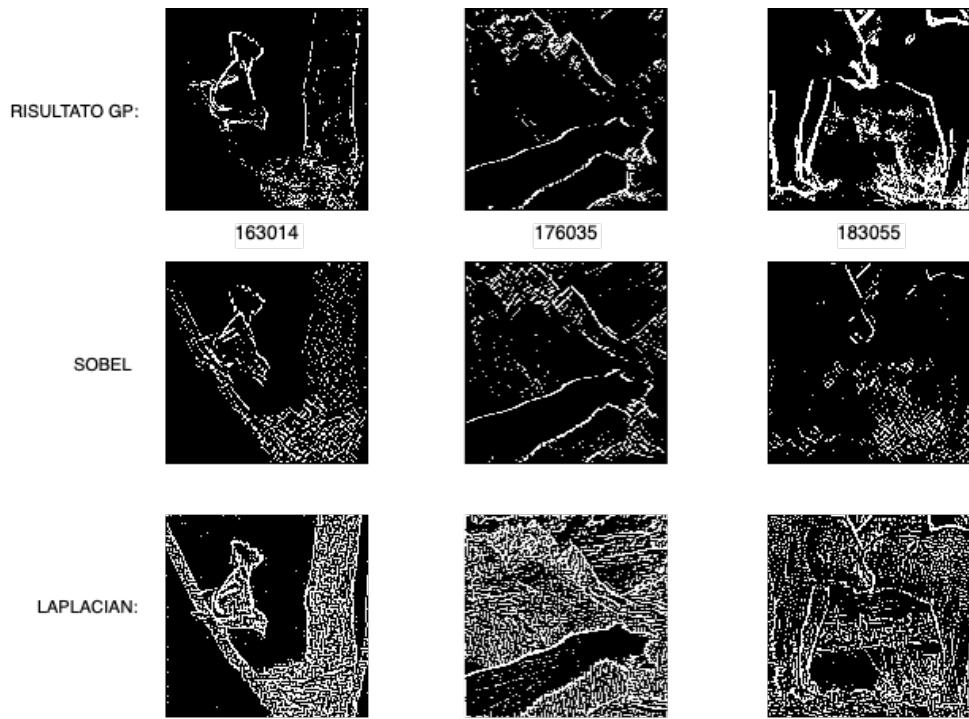


Figura 4.12: Confronto dei risultati ottenuti con un filtro per l'edge detection 3x3 con metodi esistenti

La prima differenza che è possibile notare tra i risultati ottenuti con la programmazione genetica ed i filtri proposti per il confronto è la presenza di meno edges superflui e irrilevanti ai fini di possibili elaborazioni future (come la segmentazione) in quelli prodotti dal GP rispetto al filtro di Laplacian; un esempio evidente è rappresentato dalla figura 183055 dove l'elaborazione con il filtro Laplacian presenta molti tratti discontinui.

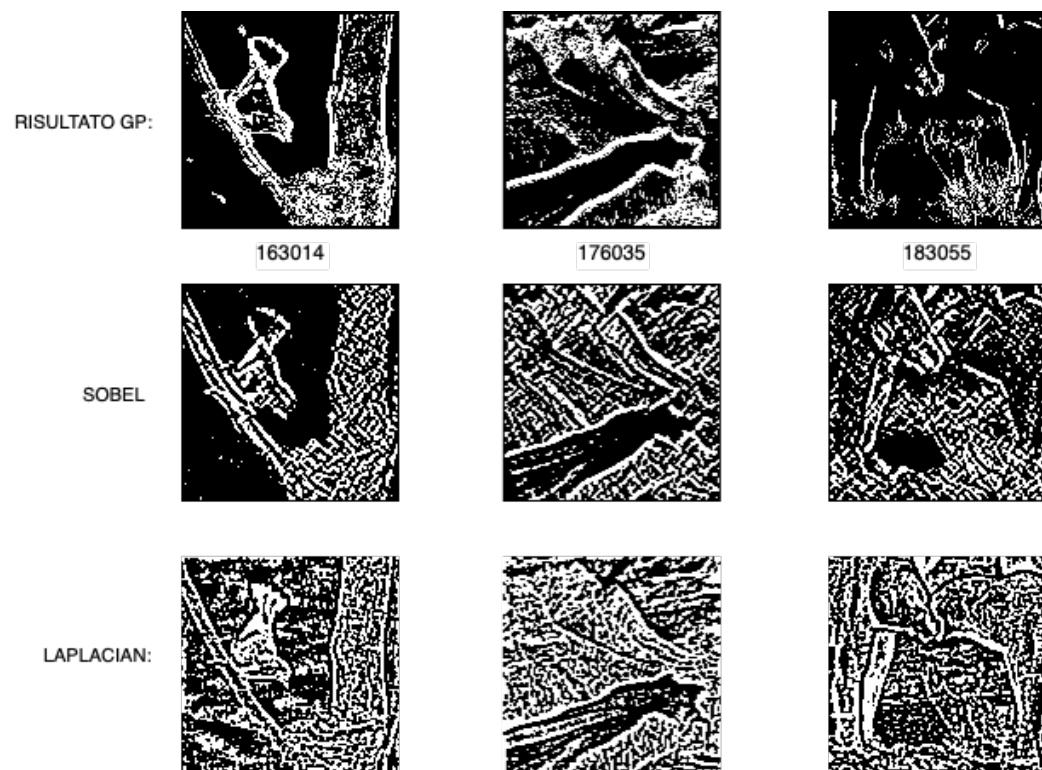


Figura 4.13: Confronto dei risultati ottenuti con un filtro per l'edge detection 5x5 con metodi esistenti

L'utilizzo di un Kernel più grande mette in risalto maggiormente le differenze dei filtri ottenuto mediante GP e quelli proposti, rendendo preferibile l'utilizzo dei primi per ottenere dei risultati più chiari e con meno bordi superflui.

4.4 Risultati ottenuti sul test set

La parte finale dell’attività di tesi consiste nell’applicare in cascata al *test set* dei filtri provenienti dall’addestramento visti nella sezione 4.2 e 4.3. Per simulare una situazione reale è stato aggiunto del rumore alle immagini del test set, prima di trovare i bordi viene quindi applicato un filtro di denoise ed il risultato viene dato in pasto al filtro di edge detection. L’algoritmo realizzato per l’applicazione dei filtri sulle immagini è simile a quello visto nella sezione 4.1. In input si hanno dei logbook scelti dall’utente e vengono estratti i filtri migliori mediante la key *BEST*, la funzione *PrimitiveTree.from_string()* è utilizzata per applicare i filtri alle immagini del test set. È importante sottolineare come nella fase di *testing* non sia stata impiegata nessuna delle immagini appartenente al training set.

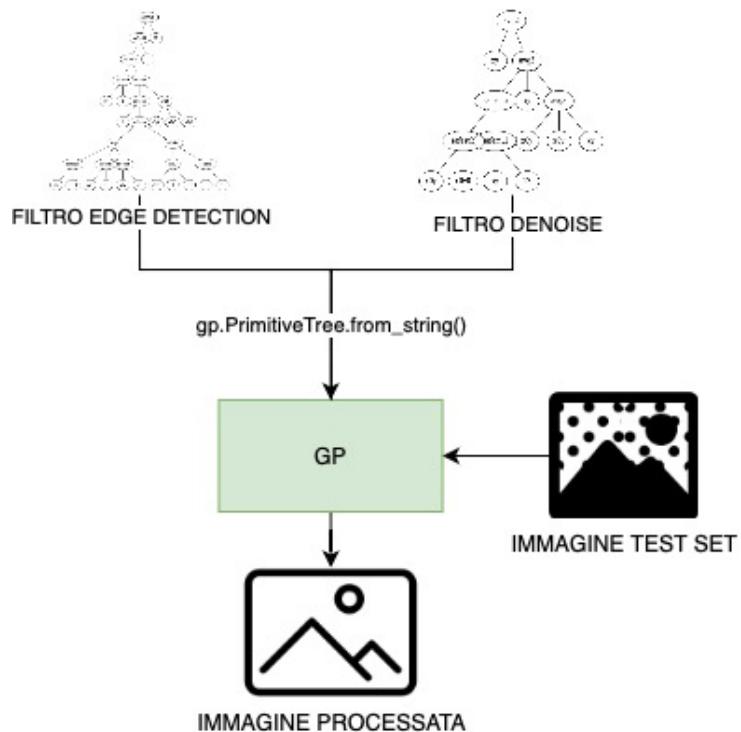


Figura 4.14: Schema sul funzionamento del test dei filtri generati

Per il confronto sono stati applicati in cascata prima il *Gaussian Smoothing* (sezione 4.2.4) e poi i due detectors visti nella sezione 4.3.4

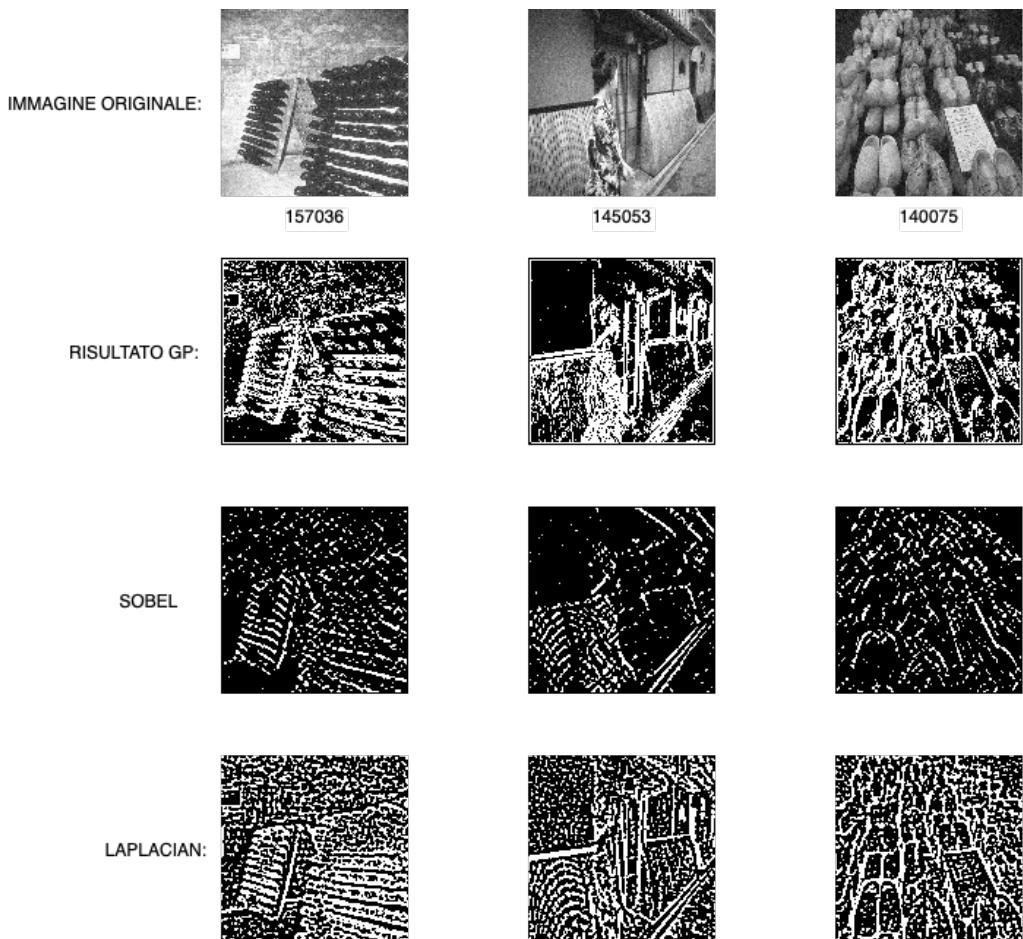


Figura 4.15: Confronto dei risultati sul test set con filtri 3x3

Anche in presenza di rumore i filtri generati mediante la programmazione genetica riescono ad ottenere dei buoni risultati, rilevando i contorni in maniera chiara senza troppe distorsioni dovute al rumore non rimosso dal filtro di denoise applicato prima dell'esecuzione dell'edge detection.

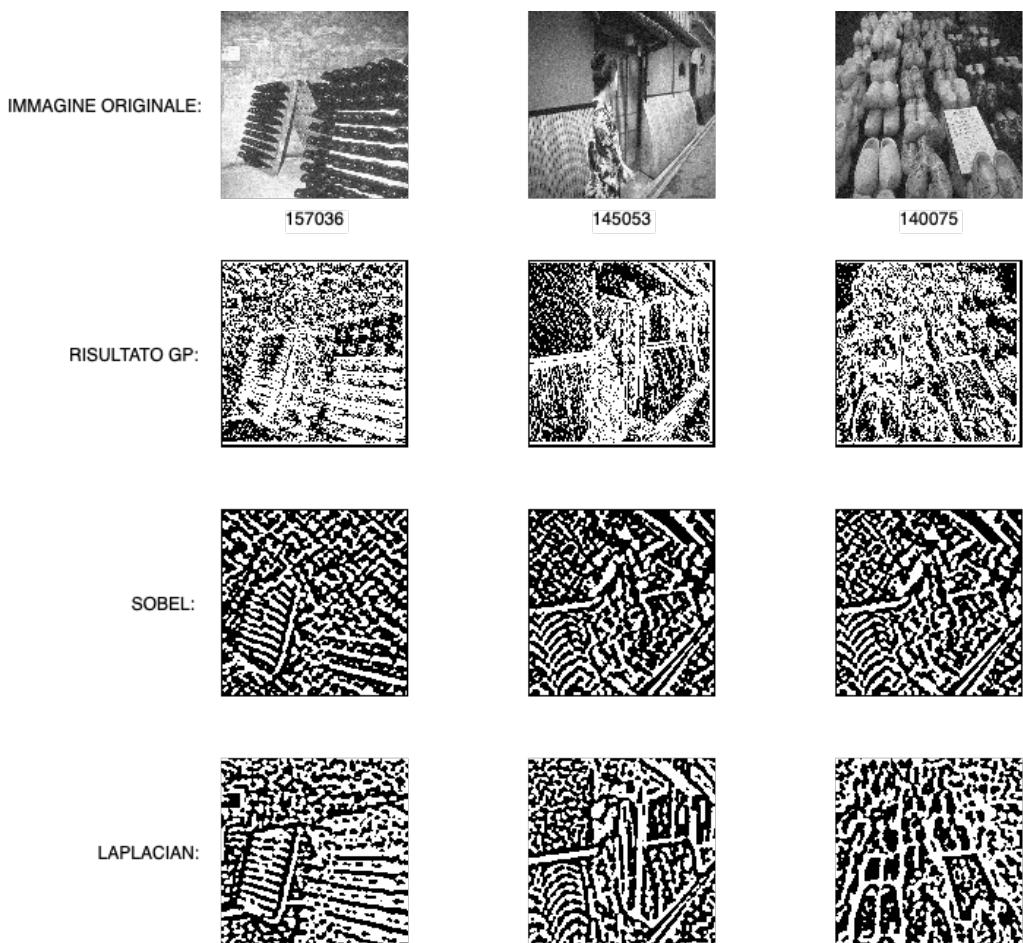


Figura 4.16: Confronto dei risultati sul test con filtri 5x5

L'aumento della dimensione del Kernel incide molto sui risultati dell'operatore di Sobel in presenza di rumore, i bordi rivelati perdono molti dettagli rispetto a quelli trovati dalla versione con Kernel 3x3; per i filtri ottenuti mediante il GP i risultati cambiano di poco.

In conclusione si può affermare che, i filtri ottenuti producono quanto richiesto dall'attività di tesi portando dei buoni risultati anche su immagini su cui non sono stati addestrati.

Conclusioni

Dopo aver dato un'introduzione sui concetti di elaborazione digitale delle immagini, Computer Vision e computazione evolutiva, si è passati all'analisi del dataset BSDS500 e delle operazioni preliminari eseguite per adattarlo agli obiettivi proposti. Il punto cardine dell'attività svolta è l'applicazione della programmazione genetica per generare dei filtri per risolvere task di denoise ed edge detection: in fase di realizzazione è emerso come attraverso la GP sia possibile svolgere task diverse sullo stesso dominio cambiando di poco gli algoritmi proposti ottenendo dei metodi *user-friendly* e dinamici. Le immagini elaborate dagli algoritmi sviluppati sul training set mostrano come i filtri ottenuti, sia nel caso di Kernel 3x3 che di Kernel 5x5, diano dei risultati positivi e competitivi rispetto alle altre soluzioni proposte per la risoluzione delle task affrontate. Nella fase di *testing* si è evidenziato come i filtri evoluti sul training set siano utilizzabili anche per altre immagini su cui non è stato effettuato nessun tipo di addestramento, e quindi generalizzabili.

Sviluppi futuri

La struttura del progetto realizzato per affrontare l'attività di tesi lascia spazio a possibili sviluppi futuri.

- Gli algoritmi proposti lavorano solo su immagini in bianco e nero. Si potrebbe estendere il dominio su cui lavorare anche su immagini a colore, effettuando le varie operazioni discusse in precedenza sui diversi canali di colore.
- Si potrebbe provare ad applicare i metodi proposti su dataset differenti e valutarne il comportamento, ad esempio su dataset di carattere medico contenenti immagini di radiografie, o di altri esami, con annotazioni su zone d'interesse da evidenziare realizzate da professionisti.
- Modificare la fitness degli algoritmi di programmazione genetica proposti per renderli utilizzabili su task di *unsupervised learning* per eliminare il vincolo imposto dal bisogno di una label per la fase di training.
- Provare ad affrontare task di medio o alto livello della Computer vision (paragrafo 1.3) partendo dagli algoritmi realizzati.

Bibliografia

- [1] John Uri. 55 years ago: Ranger 7 photographs the moon. *NASA History*, 2019.
- [2] Antonio Neves, Bruno Barbosa, Sandra Soares, and Isabel Dimas. Analysis of emotions from body postures based on digital imaging. 05 2018.
- [3] Tutorials Point. Digital image processing.
- [4] V.G.Wimalasena. Radiologic technology. Principal Sri Lanka school of Radiography.
- [5] Voratas Kachitvichyanukul. Comparison of three evolutionary algorithms: Ga, pso, and de. *Industrial Engineering and Management Systems*, 12:215–223, 09 2012.
- [6] Genetic algorithms - parent selection.
- [7] Stefan Sette and Luc Boullart. Genetic programming: Principles and applications. *Engineering Applications of Artificial Intelligence*, 14:727–736, 12 2001.
- [8] Riccardo Poli, William Langdon, and Nicholas Mcphee. *A Field Guide to Genetic Programming*. 01 2008.
- [9] Nicos Pavlidis, E.G. Pavlidis, and Michael Vrahatis. Optimizing trading strategies through genetic programming. 10 2006.

- [10] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley Publishing, 2nd edition, 2007.
- [11] Asmatullah Chaudhry, Asifullah Khan, Asad Ali, and Anwar M. Mirza. A hybrid image restoration approach: Using fuzzy punctual kriging and genetic programming. *International Journal of Imaging Systems and Technology*, 17(4):224–231, 2007.
- [12] Ruomei Yan, Ling Shao, Li Liu, and Yan Liu. Natural image denoising using evolved local adaptive filters. *Signal Processing*, 103:36–44, 2014. Image Restoration and Enhancement: Recent Advances and Applications.
- [13] Nemanja I. Petrovic and Vladimir Crnojevic. Universal impulse noise filter based on genetic programming. *IEEE Transactions on Image Processing*, 17(7):1109–1120, 2008.
- [14] Simon Harding. Evolution of image filters on graphics processor units using cartesian genetic programming. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1921–1928, 2008.
- [15] Ying Bi, Bing Xue, Pablo Mesejo, Stefano Cagnoni, and Mengjie Zhang. A survey on evolutionary computation for computer vision and image analysis: Past, present, and future trends. 2022.
- [16] Akshi Kumar and Sahil Raheja. Edge detection using guided image filtering and enhanced ant colony optimization. *Procedia Computer Science*, 173:8–17, 2020. International Conference on Smart Sustainable Intelligent Computing and Applications under ICITETM2020.
- [17] Wenlong Fu, Mark Johnston, and Mengjie Zhang. Genetic programming for edge detection: A global approach. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 254–261, 2011.

- [18] Basturk and Enis Gunay. Efficient edge detection in digital images using a cellular neural network optimized by differential evolution algorithm. *Expert Systems with Applications*, 36:2645–2650, 03 2009.
- [19] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.
- [20] Jamie Ludwig. Image convolution.
- [21] Azriel Rosenfeld. Picture processing by computer. *ACM Comput. Surv.*, 1969.
- [22] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2008.
- [23] Pragnan Chakravorty. What is a signal? [lecture notes]. *IEEE Signal Processing Magazine*, 35:175–177, 2018.
- [24] Rashi Bist, Ritu Vijay, and Shweta Singh. *Comparative Analysis of Fixed Valued Impulse Noise Removal Techniques for Image Enhancement: Second International Conference, ICACDS 2018, Dehradun, India, April 20-21, 2018, Revised Selected Papers, Part I*, pages 175–184. 04 2018.
- [25] T Huang. Computer Vision: Evolution And Promise. 1996.
- [26] Douglas J. Futuyma and Mark Kirkpatrick. *Evolution*. Sinauer Associates, 2018.
- [27] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.
- [28] John R. Koza and Riccardo Poli. A genetic programming tutorial. 2003.

- [29] John R. Koza. *Genetic programming: On the programming of computers by means of natural selection*. Mit press, 2000.
- [30] Guy L. Steele and Scott E. Fahlman. *Common lisp*. Digital Press, 1984.
- [31] J. R. Koza. Survey of genetic algorithms and genetic programming. *Proceedings of WESCON'95*.
- [32] Stefano Cagnoni, Evelyne Lutton, and Gustavo Olague. *Genetic and evolutionary computation for image processing and analysis*, volume 8. 01 2007.
- [33] GUSTAVO OLAGUE. *Evolutionary Computer Vision: The first footprints*. SPRINGER, 2018.
- [34] Mohd Awais Farooque1 and Jayant S. Rohankar. Survey on various noises and techniques for denoising the color image. *International Journal of Application or Innovation in Engineering*, 2(11):217–221, Nov 2013.
- [35] Asifullah Khan, Aqsa Saeed Qureshi, Noorul Wahab, Mutawara Hussain, and Muhammad Yousaf Hamza. A recent survey on the applications of genetic programming in image processing, 2019.
- [36] Abdul Majid, Choong-Hwan Lee, Muhammad Mahmood, and Tae-Sun Choi. Impulse noise filtering based on noise-free pixels using genetic programming. *Knowledge and Information Systems*, 32, 09 2012.
- [37] Wenlong Fu, Mark Johnston, and Mengjie Zhang. Genetic programming for edge detection: A global approach. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 254–261, 2011.
- [38] Improved skin lesion edge detection method using ant colony optimization. *Skin Research and Technology*, 25(6):846–856, 2019.
- [39] Wenlong Fu, Mengjie Zhang, and Mark Johnston. Bayesian genetic programming for edge detection. *Soft Comput.*, 23(12):4097–4112, 2019.

- [40] Wenbo Zheng, Chao Gou, Lan Yan, and Fei-Yue Wang. Differential-evolution-based generative adversarial networks for edge detection. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 2999–3008, 2019.
- [41] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.
- [42] Philippe Catine. Images restoration: introduction to signal and image processing, 2013.
- [43] Sumant Sekhar Mohanty and Sushreeta Tripathy. Application of different filtering techniques in digital image processing. *Journal of Physics: Conference Series*, 2062(1):012007, nov 2021.
- [44] Rafal Drezewski, Grzegorz Erland, and Karol Pajak. The bio-inspired optimization of trading strategies and its impact on the efficient market hypothesis and sustainable development strategies. *Sustainability*, 10:1460, 05 2018.
- [45] Mitra Basu. Gaussian-based edge-detection methods - a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 32:252 – 260, 09 2002.