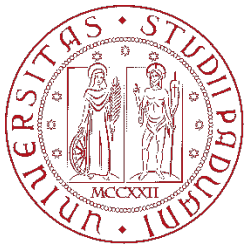


UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione - DEI

Tesi di Laurea in Ingegneria Informatica

DIFFEOMORFISMO APPLICATO AL DATA AUGMENTATION PER OGGETTI CON FORME E TEXTURE ALTAMENTE VARIABILI

Relatore:
Prof. Loris Nanni

Laureando:
Filippo Lazzarin

Anno accademico 2021-2022

Data di Laurea: 14/11/2022

Abstract

In questo documento viene analizzata una nuova tecnica utilizzata in ambito di Data Augmentation che pone le proprie basi sul morphing.
L'addestramento di reti neurali convoluzionali (CNN) richiede una grossa mole di dati, i quali possono essere estremamente difficili da ottenere specialmente in settori come quello medico. La tecnica proposta promette quindi di ottenere nuove immagini astratte, generate a partire da quelle esistenti, che siano il più verosimili possibile rispetto ai soggetti reali per ottenere un corretto addestramento delle reti neurali. Per la realizzazione delle nuove immagini vengono utilizzati in successione diversi metodi, tra i quali uno dei più importanti e innovativo è il Morphing. Il paper dunque si focalizza principalmente sui procedimenti utilizzati nel morfismo, analizzandone i punti di forza e le problematiche che ne derivano e che non permettono di ottenere immagini affidabili in ogni circostanza.

Sommario

Indice delle figure

Introduzione

Negli ultimi anni le reti neurali convoluzionali sono state tra gli strumenti più all'avanguardia nell'ambito dell'intelligenza artificiale, in particolare per quanto riguarda le applicazioni in Computer Vision come la classificazione dell'immagine e l'object detection. Questo è dovuto principalmente grazie al fatto di estrarre le features più importanti direttamente dall'immagine in maniera automatizzata ed efficiente, a differenza di quanto si faceva prima individuando le caratteristiche principali manualmente. Tuttavia questa tipologia di reti neurali, per essere efficienti ed essere in grado di generalizzare in modo da ottenere i risultati cercati, necessitano di una grande quantità di dati, i quali possono essere particolarmente difficili e costosi da ottenere specialmente in ambiti altamente specializzati come quello medico dove gli oggetti di interesse spesso sono rappresentati da particelle le cui dimensioni minuscole rendono l'ottenimento dell'immagine una procedura molto lunga, complessa e costosa. Per risolvere questo problema quindi spesso ci si affida a tecniche di Data Augmentation che permettono di generare nuove immagini artificiali, partendo da quelle esistenti, che cercano di essere il più verosimili a immagini reali. Dunque in questo elaborato viene discusso il metodo e l'implementazione di una tecnica di Data Augmentation chiamata morphing, che tramite l'utilizzo di funzioni come la triangolazione di Delaunay permette di ottenere nuove immagini estraendo le feature da immagini esistenti. Infine, vengono presentate le prestazioni di una rete allenata con questa tecnica e viene discusso il motivo per cui una mia implementazione alternativa, la Thin Plate Spline, non ha funzionato.

Capitolo 1
Reti Neurali Artificiali

In questo primo capitolo vengono inizialmente introdotte le reti neurali artificiali e la loro evoluzione nel tempo. In seguito sono poste a confronto reti neurali MLP e reti neurali convoluzionali (CNN). Per finire viene evidenziato il ruolo fondamentale delle funzioni di attivazione nelle reti e come esse influiscono sul loro funzionamento.

1.1 Cos'è una Rete Neurale Artificiale

Una rete neurale artificiale viene definita come un modello matematico e informatico di calcolo basato sulle reti neurali biologiche. Il primo modello di neurone fu introdotto per la prima volta da McCulloch e Pitts nel 1943 [13]. Tale modello può essere considerato come una porta logica, capace di assumere due soli stati: in questo modo una rete composta da questi neuroni permetteva di calcolare semplici funzioni booleane, come AND, OR e NOT. Nella Figura 1.1 è possibile notare una certa corrispondenza tra il modello biologico e quello matematico, dove la parte a sinistra raccoglie una serie di dati che elabora attraversa una funzione *f* e restituisce un output binario *y.*

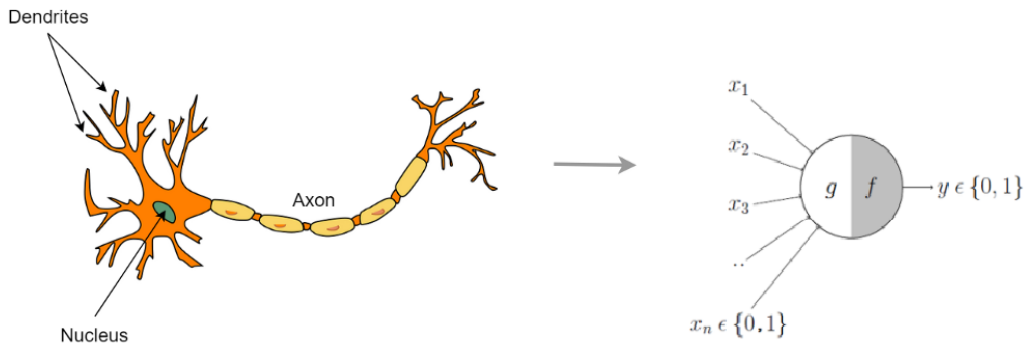


Figura 1.1: Neurone biologico e neurone M&P a confronto

Come è possibile intuire, il neurone artificiale proposto da M&P porta delle limitazioni importanti: i valori di input possono essere solamente booleani, la soglia di decisione θ deve essere prestabilita, tutti gli input contribuiscono con la stessa importanza nella decisione e non è possibile rappresentare funzioni non lineari come ad esempio la XOR.

Nel 1958, dopo che John von Neumann definisce il modello degli autori precedenti come scarsamente produttivo in quanto non aveva una struttura per poter svolgere operazioni complesse [11], Frank Rosenblatt propone un nuovo modello di neurone chiamato Percettrone [14]. Il modello di Rosenblatt mirava alle funzioni quali l'immagazzinamento delle informazioni e la loro influenza sul riconoscimento dei pattern. Il Percettrone costituisce un progresso decisivo rispetto al modello binario di McCulloch e Pitts in quanto è possibile avere pesi sinaptici variabili. Nella Figura 1.2 è possibile vedere come ad ogni input sia associato un certo peso, permettendo al Percettrone di pesare i valori in maniera distinta; inoltre si hanno input a valori reali e una funzione di attivazione a scalino.

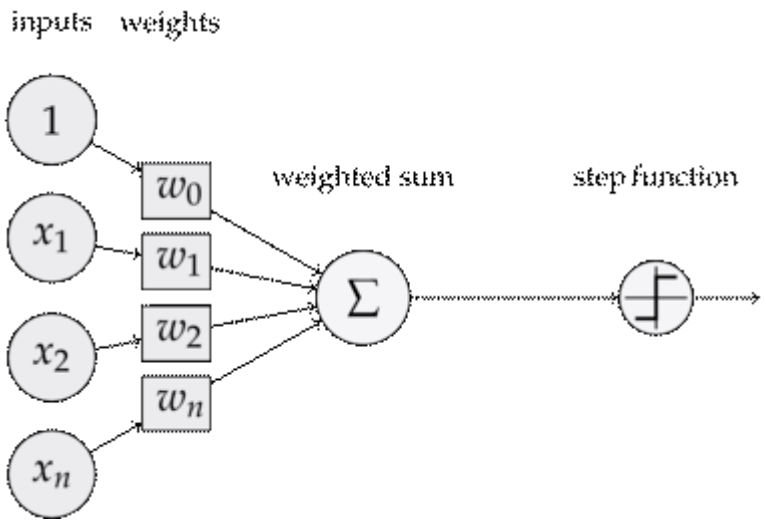


Figura 1.2: Percettrone

Le caratteristiche appena descritte rendono il nuovo modello di Rosenblatt ideale per la classificazione lineare in quanto i vari pesi sinaptici permettono di individuare un iperpiano di separazione delle classi tramite la seguente formula:

$$b + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = 0$$

Tramite un algoritmo di apprendimento è possibile modificare i valori dei pesi *w* al fine di minimizzare l'errore di classificazione.
Tuttavia anche il Percettrone aveva i propri limiti in quanto era in grado di apprendere solo le funzioni linearmente separabili, per questo la ricerca subì un notevole rallentamento in quello che viene definito "AI winter". La ricerca riprese solo negli anni '80 quando venne introdotto il modello Multilayer Perceptron (MLP), ovvero una rete *feedforward* con almeno tre strati di nodi composta da un input, un output e almeno un livello intermedio (chiamato *hidden layer*). Ogni strato è completamento connesso al successivo, e ogni nodo è un neurone con una funzione di attivazione non lineare, eccetto per i nodi di input.
Nonostante le grandi potenzialità delle reti MLP a tre strati, funzioni lineari molto complesse risultano difficilmente approssimabili per cui è necessario aumentare il numero di strati interni, accrescendo di conseguenza anche la complessità computazionale in maniera esponenziale fino a renderla impraticabile.

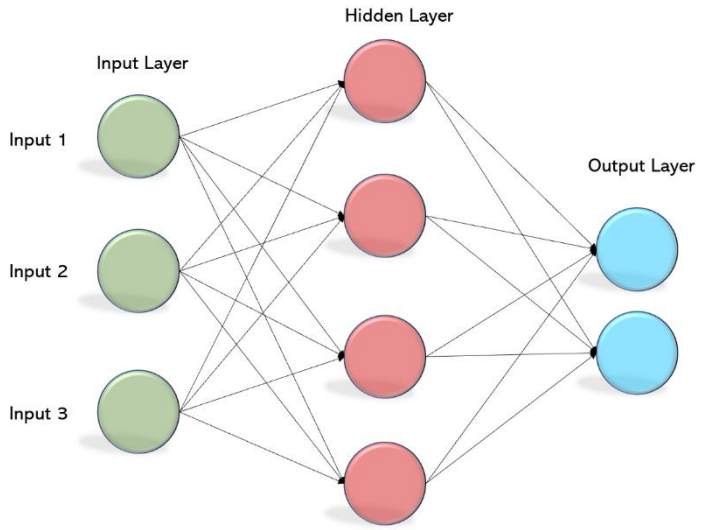


Figura 1.3: Schema di una rete MLP a tre strati

1.2 Deep Learning e CNN

Il Deep Learning è il campo di ricerca dell'apprendimento automatico che si basa su diversi livelli di rappresentazione strutturati in maniera gerarchica, dove i concetti di alto livelli sono definiti da quelli di livello più basso. Il Deep learning sfrutta le cosiddette reti neurali "profonde" (Deep Neural Network) composte da almeno due livelli nascosti (hidden) e ispirate nel funzionamento al sistema visivo degli organismi viventi. Il modello di applicazione più diffuso delle reti profonde consiste nelle Reti Neurali Convoluzionali (CNN), introdotte per la prima volta da Yann LeCun nel 1988 [9], e vengono utilizzate principalmente in ambiti di classificazione e di regressione. Le CNNs rappresentano una variante più articolata delle reti MLP, tuttavia riescono a ridurre notevolmente la complessità computazionale grazie alla condivisione dei pesi: neuroni dello stesso livello eseguono le stesse elaborazioni su porzioni diverse dell'input permettendo un processing locale e la diminuzione delle connessioni. Le reti convoluzionali si distinguono anche per la costruzione a blocchi durante la fase di "feature extraction", queste infatti sfruttano strati di convoluzione, di pooling e di attivazione che permettono di effettuare un pre-processing dei dati in input, come si può vedere nella Figura 1.4

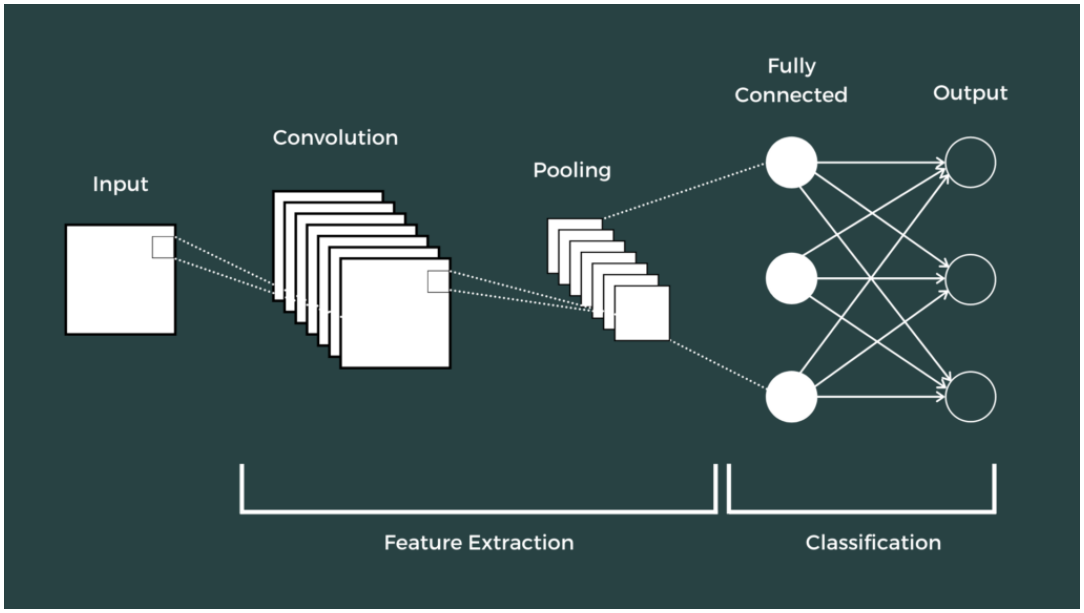


Figura 1.4: Funzionamento di una CNN

1.2.1 Convoluzione

Lo strato di convoluzione è l'elemento costitutivo di base di una CNN, attraverso il quale, tramite l'ausilio di filtri, è possibile estrarre delle features dalle immagini di cui si vuole analizzarne il contenuto. I filtri applicabili possono essere più di uno, e maggiore è il loro numero e più grande sarà la complessità delle features che si potranno individuare. In pratica, un filtro digitale (ovvero, piccola maschera) viene fatta scorrere sulle diverse posizioni dell'immagine in input, generando per ogni posizione di input un valore di output ottenuto tramite il prodotto scalare tra la maschera e la porzione dell'input coperta.

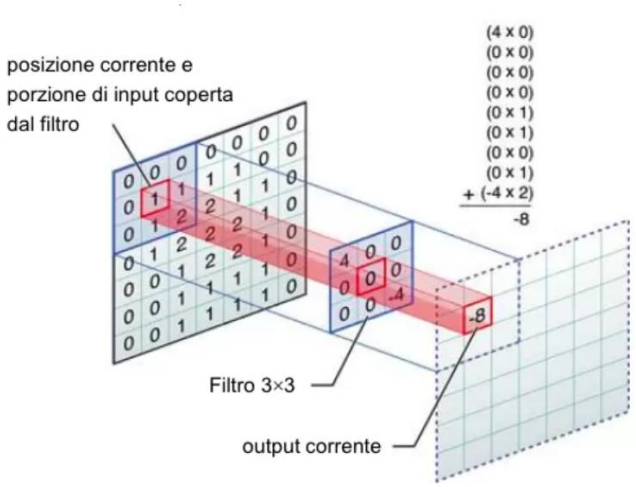


Figura 1.5: Esempio di convoluzione con un filitro 3x3

La dimensione spaziale di output è ottenuta tramite la seguente formula:

$$W_{\text{out},i} = \frac{W_{\text{in},i} - F_i + 2 * \text{Padding}}{\text{Stride}} + 1$$

$W_{\text{out},i}$: *dimensione spaziale della matrice in output*

$W_{\text{in},i}$: *dimensione spaziale della matrice in input*

F_i : *dimensione spaziale del filtro applicato*

Padding : strati di padding applicati lungo la dimensione i

Stride : numero di righe e colonne di cui si trasla ogni iterazione

1.2.2 Pooling

Un altro concetto fondamentale delle CNN è il pooling, che è una forma di "*downsampling*" dell'immagine. L'immagine in ingresso viene tagliata in una serie di quadrati con nxn pixel in modo da non avere sovrapposizioni. Il segnale all'uscita di ogni quadrato è definito in funzione dei valori assunti dai diversi pixel all'interno del rettangolo stesso. Il compito del pooling è quindi ridurre la dimensione spaziale di un'immagine, diminuendo così la quantità di parametri e calcoli all'interno della rete neurale. Dunque risulta pratica frequente inserire uno strato di pooling tra due strati convoluzionali successivi di un'architettura di rete neurale convoluzionale per ridurre l'apprendimento eccessivo che porta a un'elevata complessità computazionale. Esistono diversi metodi di Pooling, tra i più importanti si trovano l'Average Pooling e il Max Pooling, il cui funzionamento viene illustrato nella Figura 1.6

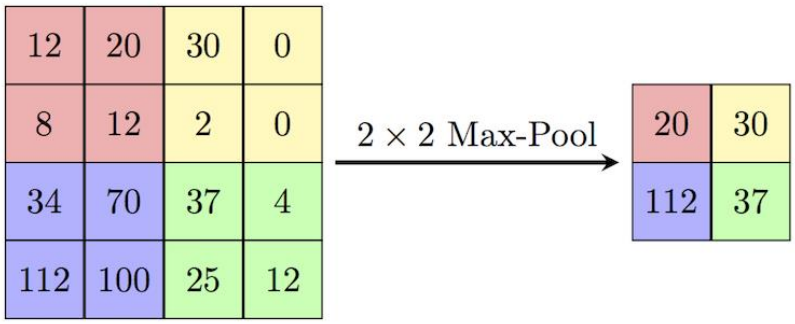


Figura 1.6: Pooling effettuato tramite il max-pooling con filtri 2x2

1.2.3 Funzione di Attivazione

Ciascun nodo della rete riceve in input dei valori che vengono immagazzinati in variabili, le quali connettendosi ad altri nodi della rete rielaborano questi valori per restituire un output che si basa sul peso *w* delle loro connessioni. L'operazione di combinazione di tali valori è data da una funzione di attivazione che deve fornire in output un intervallo di valori compreso tra {0, 1}; ovvero il valore fornito deve avere output vicino ad 1 quando vengono adeguatamente stimolati (effetto soglia), per poter propagare l'attività all'interno della rete, e 0 quando non devono essere attivati.

Tra le funzioni di attivazione più note si trovano la Rectified Linear Unit, o più semplicemente *ReLu*, definita da $\mathrm{f}(x) = x^+ = \max(0, x)$, che fornisce 0 per valori negativi mentre cresce linearmente per valori positivi come si può vedere in figura

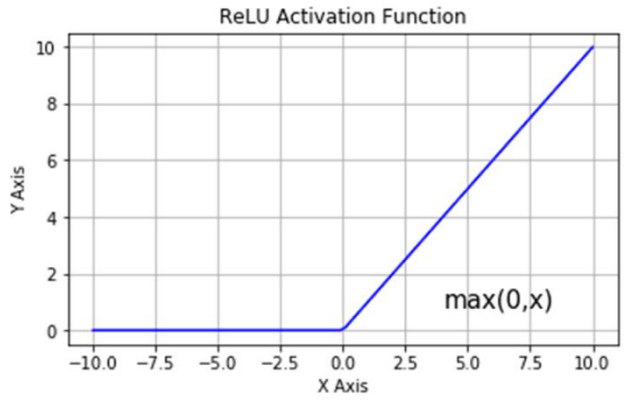


Figura 1.7: Funzione di Attivazione ReLu

Un altro tipo di funzione di attivazione largamente utilizzato è la Sigmoide, definita da

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

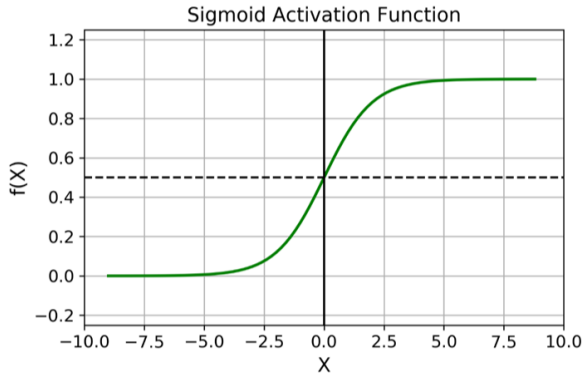


Figura 1.8: Funzione di attivazione Sigmoide

Capitolo 2
Data Augmentation

La tecnica del *Data Augmentation*, ovvero l'aumento di dati, consiste in un insieme di metodi per la manipolazione e la trasformazione dei dati con lo scopo di ampliare le dimensioni del dataset di partenza.

Infatti, questo tipo di tecnica viene utilizzata principalmente per affrontare il problema più frequente per quanto riguarda le reti neurali convoluzionali: la mancanza di un dataset sufficientemente popolato per addestrare la rete, il cui problema viene maggiormente accentuato quando bisogna dividere il dataset in training, validation e test.
I metodi di data augmentation si dividono in due categorie, tramite Image Manipulation e Deep Learning Approaches (Figura 2.1), tuttavia tra i più utilizzati si trovano le trasformazioni geometriche, i kernel tramite il Gaussian Noise Injection e GAN.

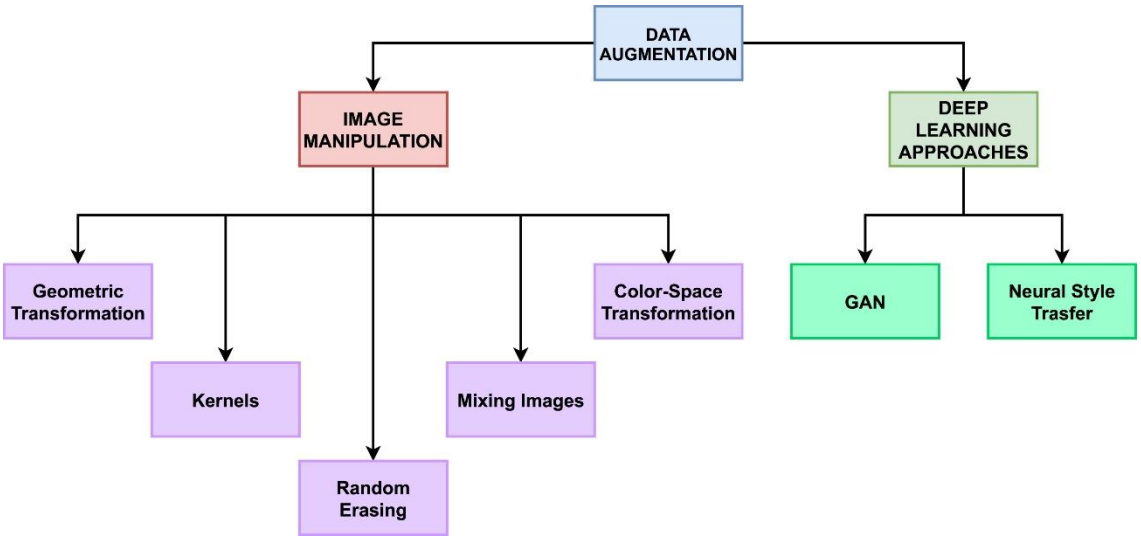


Figura 2.1: Tecniche di Data Augmentation

2.1 Trasformazioni Geometriche

Tra la diverse tecniche esistenti per aumentare il dataset, le trasformazioni geometriche rappresentano quella più facilmente implementabile e che richiede il minor sforzo computazionale. Tuttavia, affinché l'immagine generata possa essere utilizzata per l'addestramento deve essere il più verosimile all'immagine originale. Tra le trasformazioni geometriche più comuni si trovano la rotazione, il capovolgimento verticale e orizzontale, il Padding, la traslazione e il ridimensionamento.

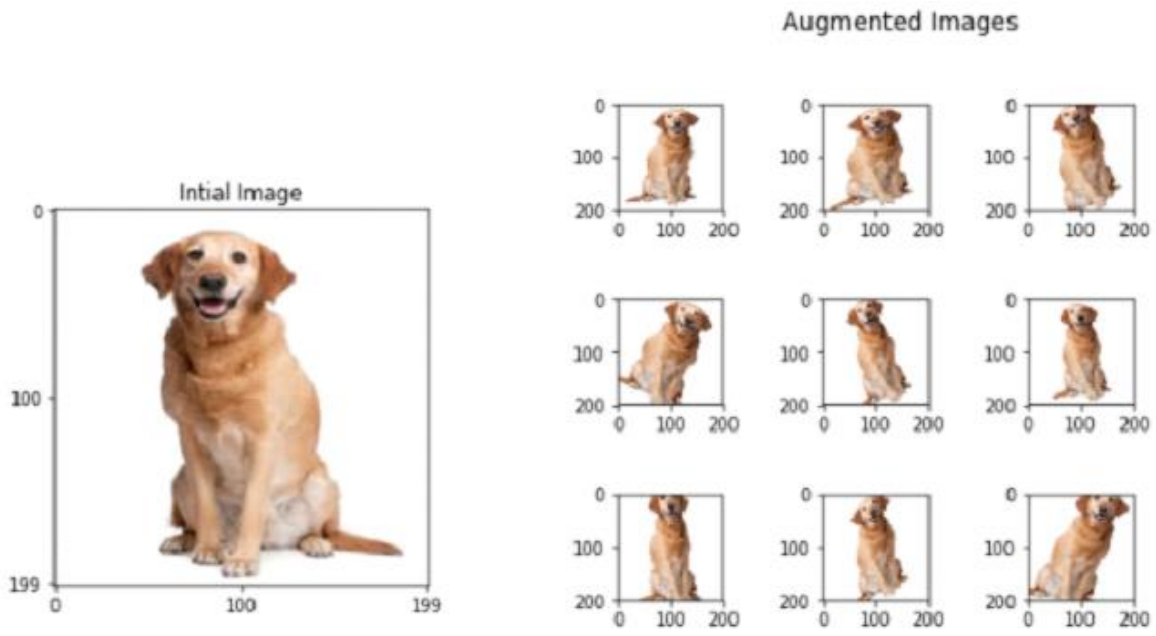


Figura 2.2: Esempio di trasformazioni geometriche

2.2 Noise Injection

L'utilizzo di filtri in fase di preprocessing dell'immagine permette di ottenere ottimi risultati in fase di addestramento delle reti. In particolare, aggiungere rumore gaussiano all'interno dell'immagine in fase di addestramento rende il riconoscimento da parte della CNN più difficile, riducendo l'overfitting e migliorando la generalizzazione.

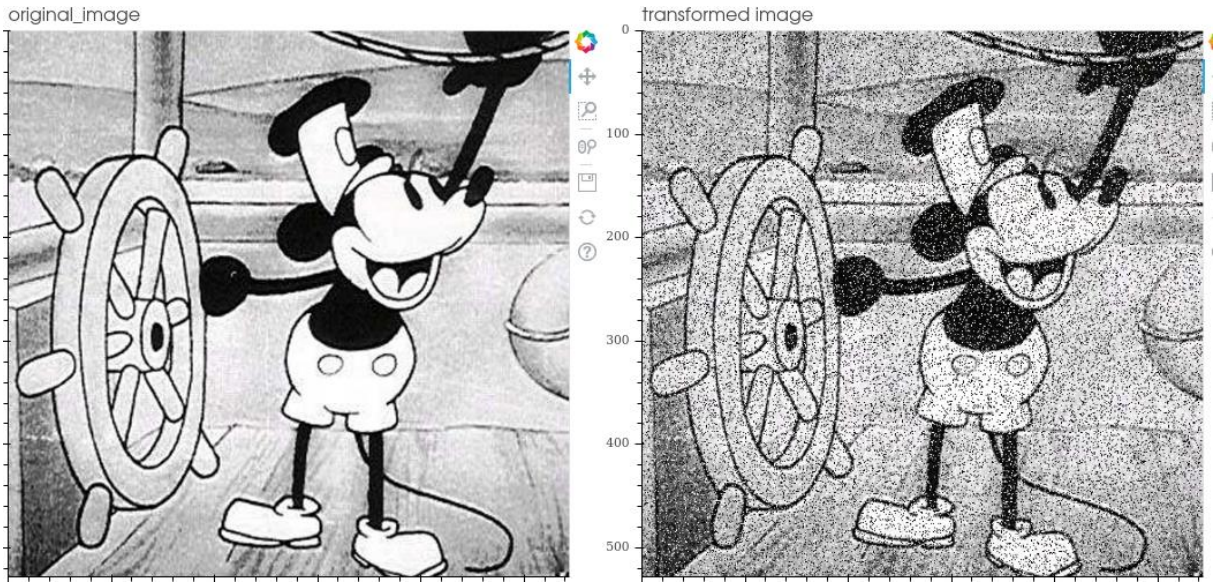


Figura 2.3: Applicazione del Noise Injection ad un'immagine

2.3 GAN

Il Generative Adversarial Network (GAN) viene proposto per la prima volta da Ian Goodfellow nel 2014 [5], e rappresenta una tecnica di data augmentation che si colloca su un approccio Deep Learning. Il suo funzionamento si basa su due agenti, un generatore e un discriminatore, dove il primo si occupa di generare immagini che siano il più simile possibile a quelle del dataset originale, mentre il secondo prende sia le immagini generate che quelle reali e prova a indovinare quali siano reali e quali no. Dunque, entrambi gli agenti vengono ottimizzati durante il processo di addestramento.

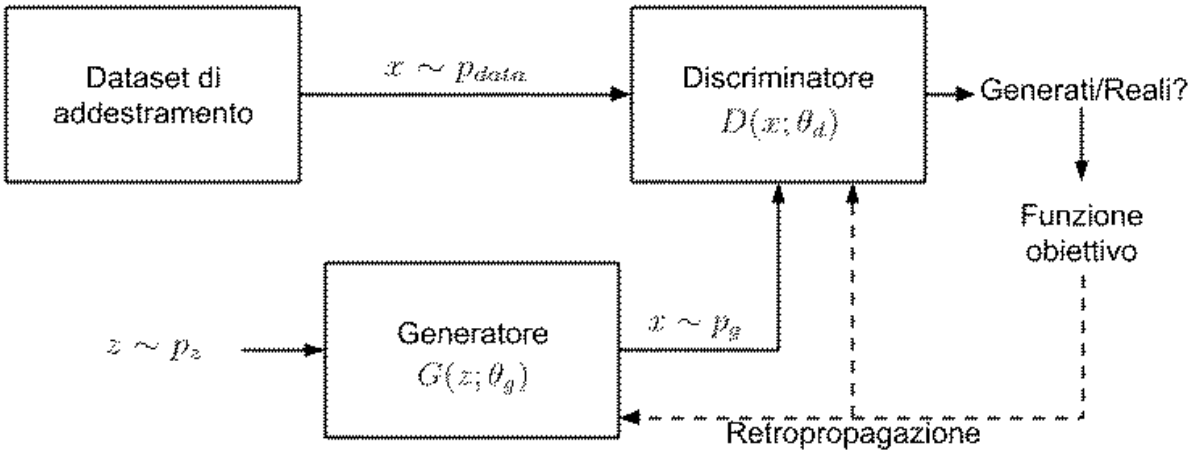


Figura 2.4: Schema di funzionamento di una GAN

Di conseguenza c'è un processo di training competitivo nel quale il generatore prova a imbrogliare il discriminatore producendo immagini che sembrano reali, mentre il discriminatore continua a migliorare la propria abilità di riconoscimento delle immagini. Dopo aver effettuato l'addestramento è possibile utilizzare il generatore per creare nuovi campioni.

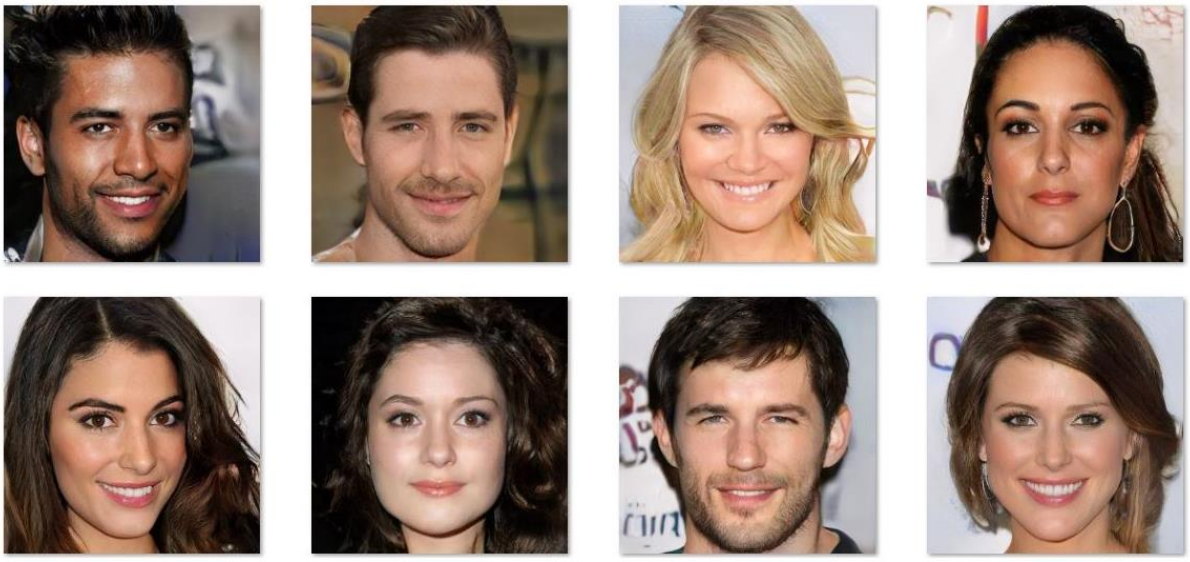


Figura 2.5: GAN faces

Nella Figura 2.5 si possono vedere dei volti di persone non esistenti generate tramite l'addestramento di una GAN su volti di celebrità.

Capitolo 3
Apprendimento Ensemble

L'apprendimento Ensemble raggruppa e utilizza diversi modelli durante la fase di addestramento per ottenere migliori prestazioni di predizione rispetto all'utilizzo di un solo modello dello stesso gruppo preso singolarmente. Un sistema ensemble prevede potenza di calcolo e risorse maggiori rispetto a un modello del sistema preso singolarmente, tuttavia, a parità di risorse di calcolo e archiviazione utilizzate, risulterà più accurato ed efficiente rispetto a quanto sarebbe preciso un singolo metodo addestrato con le stesse risorse.
Nonostante il numero di possibili ensemble che possono essere sviluppati sia illimitato, ci sono due metodi che attualmente dominano la scena: il *bagging* e il *boosting*

3.1 Bagging

Questa tecnica prevede di utilizzare una serie di classificatori deboli e omogenei, per poi addestrarli in parallelo e unirli in fase di decisione.
In una prima fase, chiamata *bootstrapping,* vengono creati degli insiemi della stessa dimensione prendendo in maniera casuale i pattern del dataset originale.

Figura 3.1: Creazione di sottoinsiemi tramite bootstrapping

L'idea alla base di questo procedimento è mantenere la caratteristica e l'indipendenza dei dati all'interno dei vari insiemi (di dimensione minore rispetto al dataset originale) con l'obbiettivo di addestrare i classificatori su dataset che siano il più diversi possibile tra loro.

Finita la fase di addestramento, i vari classificatori vengono aggregati a livello di decisione dove è possibile assegnare il pattern alla propria classe di appartenenza tramite metodi come la *Majority Vote Rule* e *Borda Count*.

3.1.1 Majority Vote Rule

Questo metodo assegna il pattern alla classe più votata dai vari classificatori, dove ogni classificatore assegna un pattern a una sola classe.
Siano $\mathrm{i} = \{\mathrm{C}_1, \mathrm{C}_2, \dots, \mathrm{C}_\mathrm{n}\}$ un insieme di classi e $k = \{K_1, C_2, \dots, K_m\}$ un insieme di classificatori. Quindi sia

$$\theta_{ij} = \begin{cases} 1 \text{ se } i \text{ è la classe votata da } K_j \\ 0 \text{ altrimenti} \end{cases} \quad (1 \leq i \leq n, 1 \leq j \leq m)$$

Il pattern viene assegnato alla classe z tale che:

$$z = argmax\left\{\sum_{j=1}^{m} \theta_{ij}\right\}$$

3.1.2 Borda Count

In questo caso ogni classificatore assegna il pattern alle varie classi tramite un ranking in base al grado di appartenenza. Dunque, viene creata una classifica a punteggi dove la classe con punteggio maggiore è quella a cui viene assegnato il pattern.

Classificatore 1

Classe	Rank
1	5
2	4
3	3
4	2
5	1

Classificatore 2

Classe	Rank
4	5
2	4
5	3
1	2
3	1

Classificatore 3

Classe	Rank
2	5
4	4
3	3
1	2
5	1

Risultato

Classe	Rank
2	13
4	11
1	9
3	7
5	5

Figura 3.2: Borda Count con 3 classificatori

3.2 Boosting

Come per il bagging, anche il boosting utilizza una serie di classificatori deboli per poi aggregarli e ottenerne uno forte con prestazioni migliori. Tuttavia, il principio di questa tecnica consiste nell'utilizzare i classificatori deboli sequenzialmente e in maniera adattiva: ogni classificatore della sequenza dà maggiore importanza alle osservazioni dei classificatori precedenti che avevano ottenuto scarsi risultati, dando maggiore importanza alle considerazioni più difficili. Per capire quali informazioni conservare dai classificatori precedenti vengono utilizzati degli algoritmi di boosting, come ad esempio Adaboost.

Figura 3.3: Esempio di Boosting

3.2.1 Adaboost

Adaboost, meglio noto come *Adaptive Boosting*, utilizza un processo iterativo di ottimizzazione dei pesi dei vari classificatori, dove viene aggiunto un classificatore debole ad ogni iterazione che cerca la migliore coppia (coefficiente, peso) da aggiungere all'ensemble ricorrente: tale soluzione permette di ridurre la complessità computazionale che si avrebbe nel caso in cui si cerchi il peso tramite un'unica iterazione. Quindi il classificatore forte può essere ottenuto tramite la seguente formula:

$$s_L = \sum_{l=1}^{L} c_l \times w_l$$

$$s_L: Classificatore\ forte$$
$$c_l: Classificatore\ debole$$
$$w_l: Peso\ del\ classificatore\ debole$$

In particolare, i classificatori vengono aggiunti in modo sequenziale:

$$s_l = s_{l-1} + c_l \times w_l$$

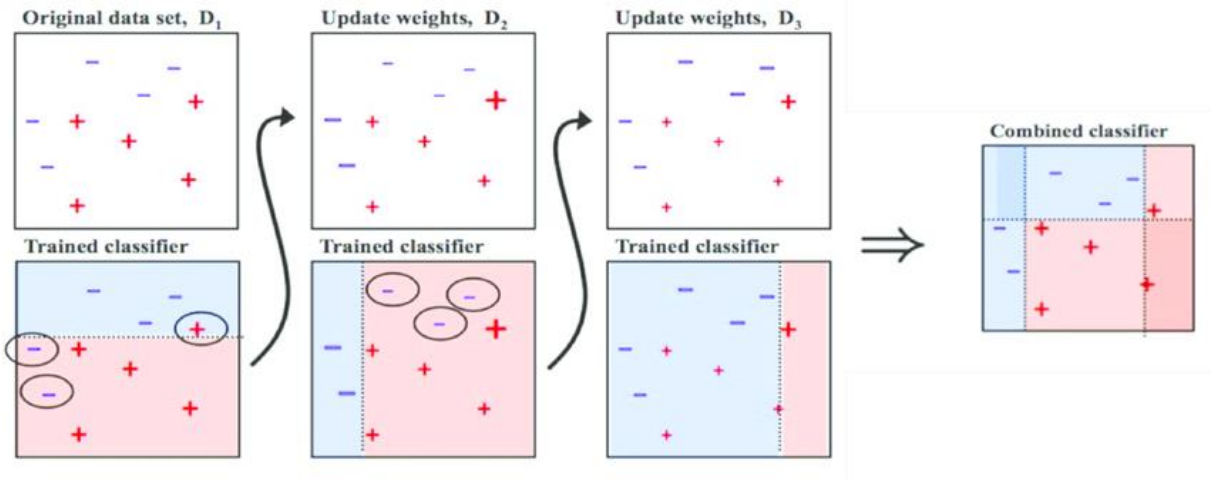


Figura 3.4: Schema di funzionamento di Adaboost

Capitolo 4
Morphing

In questo capitolo viene esposta nel dettaglio la tecnica del Morphing, partendo dalla sua definizione e analizzando rispettivamente due tecniche diverse, ovvero la triangolazione di Delaunay proposta nel paper “Diffeomorphic transforms for data augmentation of highly variable shape and texture object”[12] e la “Thin Plate Spline” di cui ho proposto un’implementazione discussa nei capitoli successivi.

4.1 Definizione

Il morfismo consiste in un’interpolazione geometrica che viene ampiamente utilizzata in ambito di computer vision per ottenere una transizione da una figura ad un’altra in modo fluido e che dia sensazione di continuità. La procedura per ottenere l’interpolazione delle due immagini si basa principalmente su una mappatura lineare definita tramite una corrispondenza tra le figure presenti nelle due immagini.

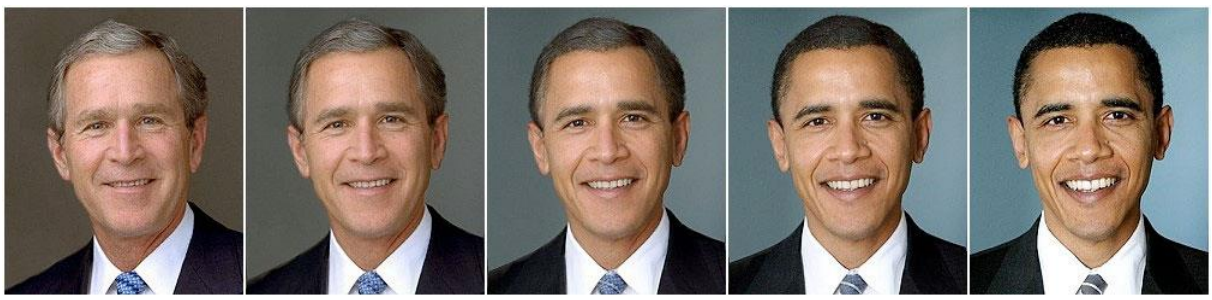


Figura 4.1: Esempio di Morphing tra i presidenti Bush e Obama

4.2 Corrispondenza tra punti – Metodo Proposto

Il primo passo fondamentale per ottenere l'interpolazione tra le due immagini consiste nel definire una corrispondenza nei punti che definiscono il contorno delle figure di interesse. L'algoritmo scelto per calcolare i contorni si basa su "Elliptical Fourier Descriptor (EFD)" in modo da ottenere i coefficienti di Fourier di un contorno "chain-encoded"[1]. Con "chain-code" si intende un metodo di segmentazione, il cui funzionamento consiste nel codificare separatamente ogni componente collegato del contorno. Per ogni regione di spazio viene selezionato un punto al confine e il codificatore, spostandosi lungo il confine, trasmette un simbolo che rappresenta la direzione del movimento, fino a tornare al punto iniziale. I movimenti possono appartenere a due schemi diversi, uno definito dall'insieme $\{i|i = 0,1,2,3\}$ dove il direzionamento è calcolato da $\{i * 90°\}$ e conosciuto come 4-directional chain code, e uno definito dall'insieme $\{i|i = 0,1, \dots ,7\}$ con direzionamento ottenuto da $\{i * 45°\}$ e conosciuto come 8-directional chain code.

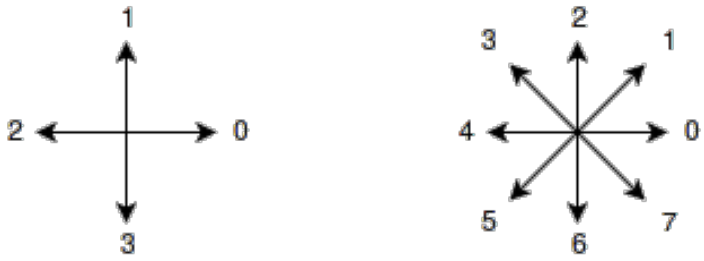


Figura 4.2: Chain Code con direzionamento di 90° e 45°

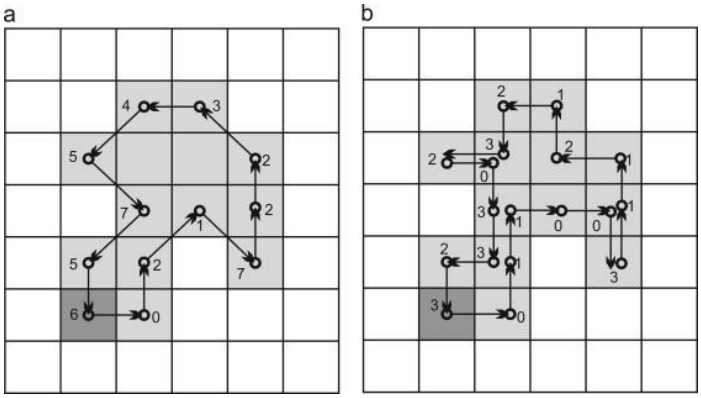


Figura 4.3: Esempio di segmentazione ottenuta utilizzando rispettivamente 45°(a) e 90°(b)

Una volta ottenuti i contorni delle figure di interesse si può procedere a calcolare la variazione delle proiezioni negli assi x e y, Δx_i *e* Δy_i, tramite le seguenti equazioni:

$$\Delta x_i = \mathrm{sgn}(6 - a_i)\mathrm{sgn}(2 - a_i)$$
$$\Delta y_i = \mathrm{sgn}(4 - a_i)\mathrm{sgn}(a_i)$$
$$\Delta t_i = 1 + \left(\frac{\sqrt{2} - 1}{2}\right)(1 - (-1)^{a_i})$$

Dove a_i rappresenta l'*i*-esimo elemento della chain code mentre Δt_i è il modulo del segmento tra due punti adiacenti.

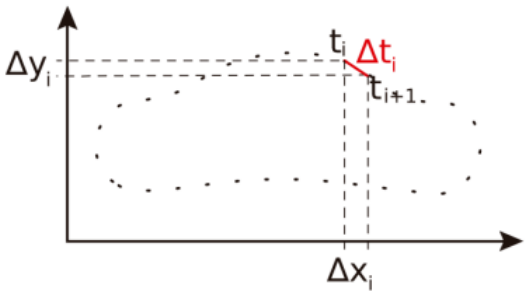


Figura 4.4: Riferimento dei punti della catena

Partendo dall'equazione della serie di Fourier:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{N} [a_n \cos(nt) + b_n \sin(nt)]$$

Possiamo ottenere i coefficienti di Fourier a_n *e* b_n corrispondenti alle proiezioni di x e di y tramite le seguenti equazioni:

$$x(t) \longrightarrow \begin{cases} a_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p} [cos\frac{2n\pi t_p}{T} - cos\frac{2n\pi t_{p-1}}{T}] \\ b_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p} [sin\frac{2n\pi t_p}{T} - sin\frac{2n\pi t_{p-1}}{T}] \end{cases}$$

$$y(t) \longrightarrow \begin{cases} a_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta y_p}{\Delta t_p} [cos\frac{2n\pi t_p}{T} - cos\frac{2n\pi t_{p-1}}{T}] \\ b_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta y_p}{\Delta t_p} [sin\frac{2n\pi t_p}{T} - sin\frac{2n\pi t_{p-1}}{T}] \end{cases}$$

Dove *K* è il numero dell'armonica utilizzata, *n* l'ordine dei coefficienti armonici e *T* il perimetro.
Infine l'ampiezza dell'*n*-esima armonica si ottiene tramite:

$$\mathrm{amp_n} = \frac{1}{2}\sqrt{\mathrm{a(x)_n^2 + b(x)_n^2 + a(y)_n^2 + b(y)_n^2}}$$

Una volta che il contorno delle figure è stato approssimato, i punti corrispettivi tra le due immagini sono stati selezionati. Quindi, a questo punto si può procedere per ottenere una corrispondenza tra i due punti (x_i, y_i) in I e (x_j, y_j) in J come intermezzo tramite le seguenti equazioni:

$$\mathrm{x_m} = (1-\alpha)\mathrm{x_i} + \alpha\mathrm{x_j}$$
$$\mathrm{y_m} = (1-\alpha)\mathrm{y_i} + \alpha\mathrm{y_j}$$

Mentre l'intensità dei pixel si ottiene tramite:

$$\mathrm{M(x_m, y_m)} = (1-\alpha)\mathrm{I(x_i, y_i)} + \alpha\mathrm{J\left(x_j, y_j\right)}$$

4.3 Triangolazione di Delaunay

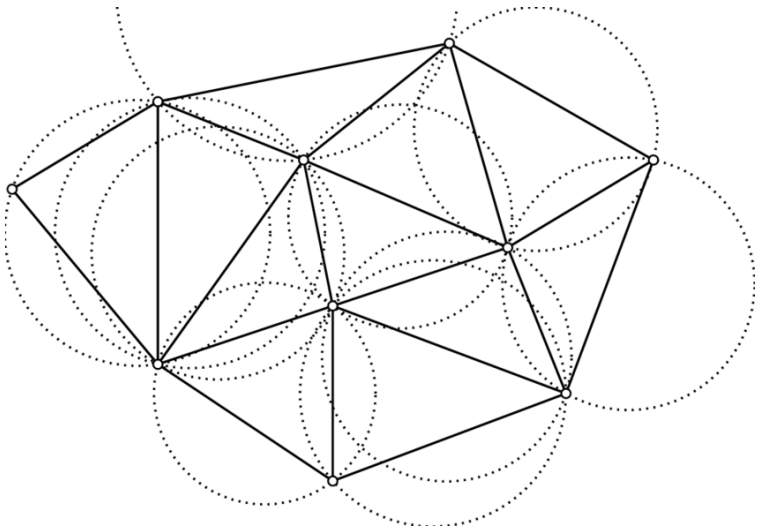


Figura 4.5: Triangolazione di Delaunay con i circumcerchi

La triangolazione di Delaunay [15] è formata da una rete di triangoli che rispettano una proprietà fondamentale, ovvero il circumcerchio di ciascun triangolo contiene solo i vertici del triangolo stesso, senza contenere punti di altri triangoli al proprio interno: questa proprietà rende questo tipo di triangolazione unica. Grazie a questa particolarità, la triangolazione di Delaunay trova spazio in diversi problemi di modellazione, in particolare in contesti di geometria computazionale. Il processo preliminare per ottenere la triangolazione consiste nell'estrazione dei punti di correlazione dell'immagine, ad esempio utilizzando la tecnica descritta precedentemente, per poi connettere i punti più vicini in modo da formare delle triangolazioni e verificare iterativamente che venga rispettata la proprietà fondamentale del circumcerchio. Il risultato finale sarà una "maglia" che permette di rappresentare correttamente la superficie dell'oggetto di interesse. Nella Figura 4.6 seguente è possibile vedere la sequenza delle operazioni.

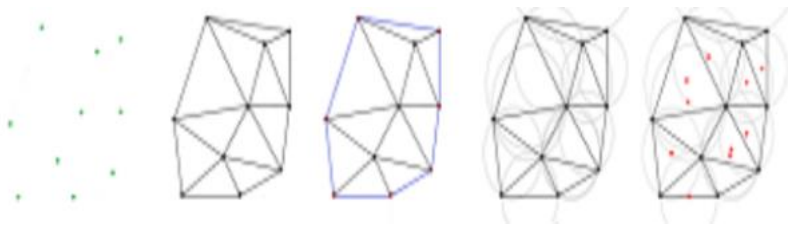


Figura 4.6: Passaggi della triangolazione di Delaunay in cui si parte da dei punti per poi ottenere dei triangoli che rispettano la proprietà fondamentale del circumcerchio

4.4 Thin Plate Spline

Le Thin Plate Spline sono una tipologia di funzioni spline poliarmoniche (ovvero, una funzione costituita da polinomi raccordati tra loro con lo scopo di interpolare in un intervallo un insieme di punti) utilizzate in ambito di geometria computazionale. Per prima cosa il metodo consiste nell'ottenere tutti i punti di controllo della prima immagine e i punti di interpolazione della seconda, per poi utilizzare una funzione radiale di base il cui kernel sarà maggiore nei punti in cui la distanza tra il punto di controllo e quello di interpolazione è bassa, come si vede in Figura 4.7. Kernel di valore alto hanno maggiore influenza rispetto agli altri.

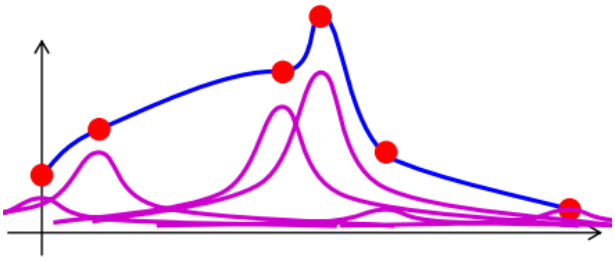


Figura 4.7: Funzione radiale di base 1-D dove sugli assi si trovano rispettivamente i punti di corrispondenza e quelli di interpolazione

Nel nostro caso il kernel della funzione radiale di base sarà calcolato come:

$$\mathrm{U(r)} = \mathrm{r^2 \log(r)}$$

Dove r rappresenta la distanza tra il punto di corrispondenza e quello di interpolazione: $\mathrm{r} = |\left((\mathrm{x_i}, \mathrm{y_i}) - (\mathrm{x}, \mathrm{y})\right)|$

A questo punto è possibile ottenere la funzione di interpolazione:

$$\mathrm{f(x)} = \sum \alpha_\mathrm{i} \mathrm{U(x, x_i)}$$

Dove α_i rappresenta il peso della funzione radiale di base attorno al punto x_i. Il valore di α può essere ottenuto risolvendo il sistema:

$$\vec{\alpha} = \mathrm{U^{-1}} \times \vec{\mathrm{x}}$$

Capitolo 5
Metriche di Prestazione

I modelli di Machine Learning forniscono sempre delle previsioni che hanno un carattere probabilistico, anche nel caso in cui venga fornito un risultato assoluto. Per valutare la performance degli algoritmi implementati possono essere utilizzate diverse metriche di prestazione che permettono di tenere traccia della qualità del modello e prendere decisioni in base ai dati forniti.

5.1 Confusion Matrix

In ambito di classificazione binaria, ciascun input può essere assegnato a una delle due classi, che solitamente sono etichettate come 1 (positiva) o 0 (negativa). Tuttavia, quando si passa un pattern al modello quello che restituisce non è un'etichetta ma una percentuale di assegnazione (e.g. 0.6) che, tramite l'iperparametro *threshold*, viene successivamente assegnato a una delle due classi. A seguito della classificazione possono avvenire quatto distinti casi:

TP (True Positive): La predizione era True e il pattern è stato assegnato a True
FN (False Negative): La predizione era True e il pattern è stato assegnato a False
FP (False Positive): La predizione era False e il pattern è stato assegnato a True
TN (True Negative): La predizione era False e il pattern è stato assegnato a False

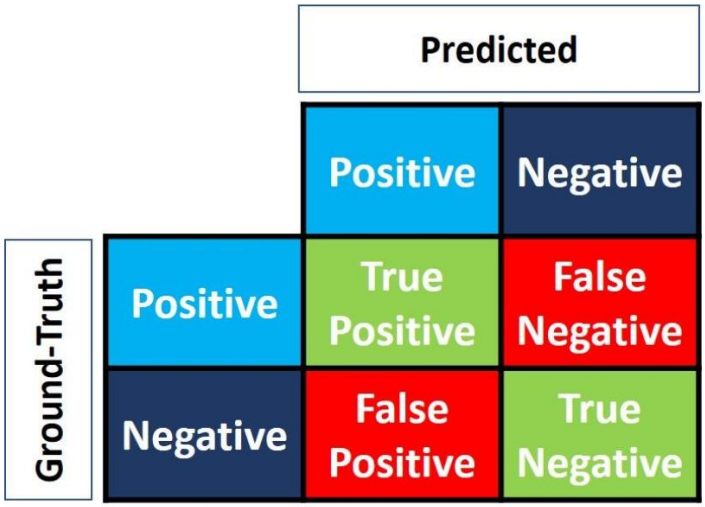


Figura 5.1: Confusion Matrix

L'obbiettivo del modello è massimizzare il numero di TP e TN tale per cui i pattern vengono classificati correttamente, per cui idealmente una Confusion Matrix deve avere valori alti sulla diagonale, caso che si applica anche in classificazioni non binarie ma multiclasse.

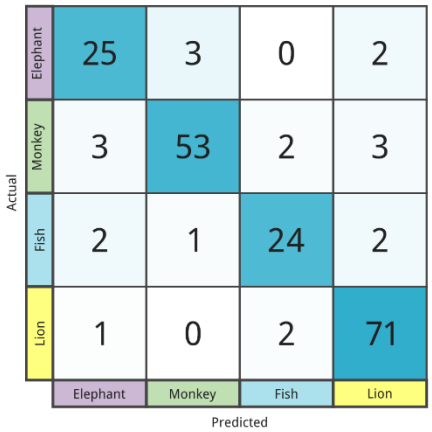


Figura 5.2: Confusion Matrix multiclasse

5.2 Accuracy e Precision

L'accuracy è una metrica che permette di quantificare l'errore e indica il rapporto tra il numero delle previsioni corrette di un evento sul totale delle volte che il modello lo prevede. In particolare, nel nostro caso misura la percentuale di volte in cui classifica correttamente l'immagine rispetto al numero totale di classificazioni. Il suo valore può essere calcolato tramite la seguente formula:

$$\text{Accuracy} = \frac{\text{True}_{\text{positive}} + \text{True}_{\text{negative}}}{\text{True}_{\text{positive}} + \text{True}_{\text{negative}} + \text{False}_{\text{positive}} + \text{False}_{\text{negative}}}$$

La Precision invece è una misura che permette di valutare l'accuratezza di un modello nella classificazione dei pattern come positivi. Il suo valore si ottiene tramite la seguente formula:

$$\text{Precision} = \frac{\text{True}_{\text{positive}}}{\text{True}_{\text{positive}} + \text{False}_{\text{positive}}}$$

Quando un modello effettua molti falsi positivi il denominatore cresce, diminuendo il valore di Precision. Per questo la Precision indica quanto il modello è affidabile nel classificare i pattern positivi.

5.3 ROC e AUC

Una curva ROC (Receiver Operating Characteristic curve) è un grafico che mostra le performance di un modello di classificazione tramite due parametri:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$FPR = \frac{FP}{FP + TN}$$

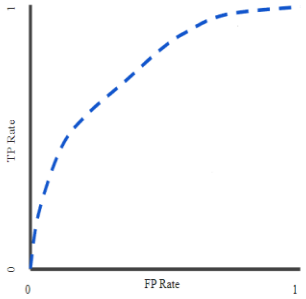


Figura 5.3: Curva ROC

La curva ROC quindi ci permette di vedere i valori di TPR vs FPR a seconda del livello di threshold. Ogni punto della curva ROC rappresenta il TPR vs FPR ad un certo livello di threshold.

La metrica AUC (Area Under the ROC Curve) misura l'area contenuta al di sotto della curva ROC compresa tra i punti (0,0) e (1,1). Tramite l'AUC possiamo ottenere una misura delle performance per tutti i possibili livelli di threshold.

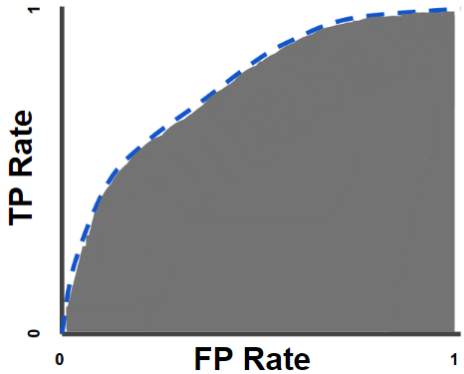


Figura 5.4: Area AUC

Capitolo 6
Fase Sperimentale

In quest'ultimo capitolo vengono riportati e discussi i risultati ottenuti utilizzando il metodo precedentemente descritto.

Il codice originale del paper a cui è ispirata questa tesi è stato leggermente modificato in modo da renderne più facile la lettura e l'utilizzo, con il codice disponibile sulla piattaforma GitHub[2].

6.1 ResNet-50

La rete scelta per l'addestramento è la ResNet-50, ovvero una rete neurale convoluzionale formata da 50 strati di profondità. In generale, ResNet (Residual Network)[7] è una rete neurale addestrata da milioni di immagini che viene utilizzata in molteplici campi di computer vision.

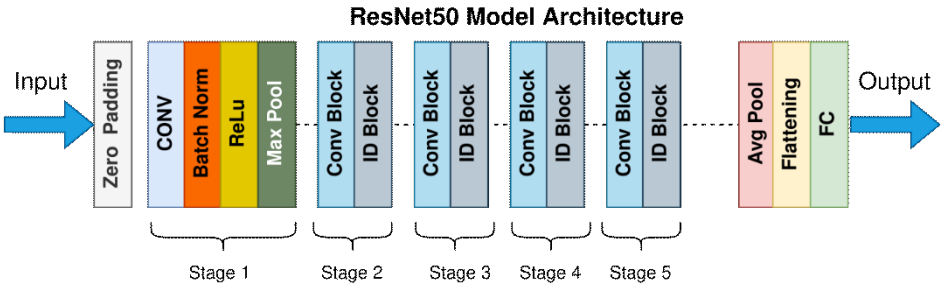
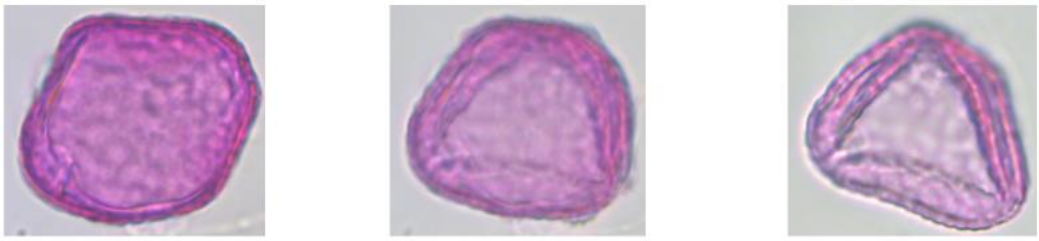


Figura 6.1: Architettura delle rete ResNet-50

6.2 Test e Risultati Sperimentali

La prima problematica riscontrata durante la fase sperimentale è stata l'enorme quantità di immagini generate dall'algoritmo già implementato. Quindi, la prima modifica apportata è la possibilità di personalizzare il numero di immagini generate tramite l'utilizzo di un parametro *factor_increment.* Inoltre è stato introdotto anche un parametro *alpha* che permette di selezionare il numero di immagini create ad ogni iterazione, ovvero per ogni coppia di immagini.

Successivamente, tramite la rete ResNet-50 è stato calcolato l'Accuracy e AUC per ogni test, metriche che ci permettono di valutare le prestazioni dell'addestramento come descritto nel capitolo precedente. Di fatto è stata utilizzata la 1-AUC invece che AUC, quindi a valore minore corrisponde una performance migliore. Il primo dataset su cui sono stati effettuati dei test conteneva immagini di granuli di pollini di varie specie al microscopio



(a)Immagine originale 1 (b)Immagine creata (c)Immagine originale 2

Figura 6.2: Morphing di granuli di acronomia aculeta

I risultati ottenuti tramite il primo training sono i seguenti:

Accuracy	1-AUC
0.9236	0.014298

Utilizzando un ensemble della rete ResNet-50 la quale utilizza metodi di Data Augmentation classici che consistono nel creare 2 immagini con riflessione (verticale e laterale), un'immagine scalando l'originale lungo entrambi gli assi, una ruotando l'immagine, una traslandola lungo entrambi gli assi e infine una effettuando un taglio, si ottengono prestazioni peggiori per quanto riguarda l'accuracy ma valori più alti per quanto riguarda l'1-AUC:

Accuracy	1-AUC
0.9393	0.0145

Tuttavia, utilizzando un dataset contenente le immagini di molte specie di farfalle, sono stati riscontrate diverse problematiche. Se prima le immagini create sembravano realistiche, col nuovo dataset le nuove immagini non risultano veritiere.

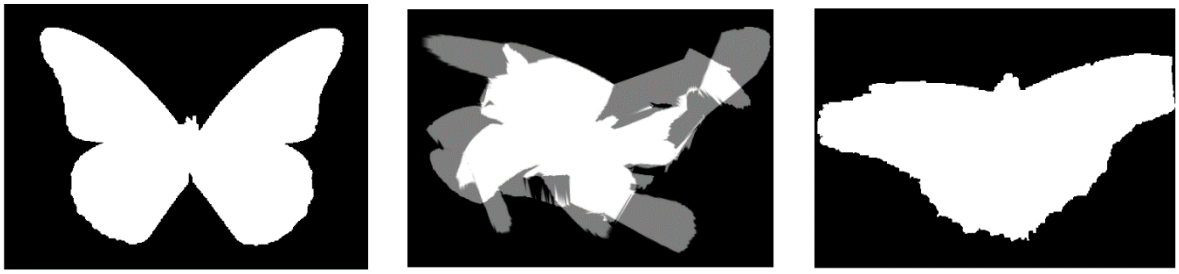
(a)Immagine originale A (b)Immagine creata (c)Immagine originale C

Figura 6.3: Morphing applicato al dataset delle farfalle

L'addestramento col nuovo dataset non ha portato a risultati sperati, infatti le performance sono crollate. Analizzando il problema si è giunti alla conclusione che il morfismo non funziona in modo ottimale quando si utilizzano immagini con forme molto diverse tra loro, dando risultati lontani dalla realtà. Infatti, come si può vedere nelle seguenti immagini, utilizzando il metodo su soggetti di forme molto simili si ottengono delle maschere che risultano veritiere (Figura 6.4), mentre utilizzando forme molto diverse tra loro, come nel caso delle farfalle, si ottengono delle maschere lontane dagli effetti sperati (Figura 6.5).

(a)Maschera polline 1 (b)Maschera generata (c)Maschera polline 2

Figura 6.4: Maschera ottenuta dal morphing del dataset dei pollini



(a)Maschera farfalla 1 (b)Maschera generata (c)Maschera fafalla 2

Figura 6.5: Maschera ottenuta dal morphing del dataset delle farfalle

Successivamente si è preso in considerazione l'ensemble descritto nel paper [10] il quale utilizza 14 metodi di Data Augmentation su 14 ResNet50 pre-addestrate con ImageNet, vedi Figura 6.6.

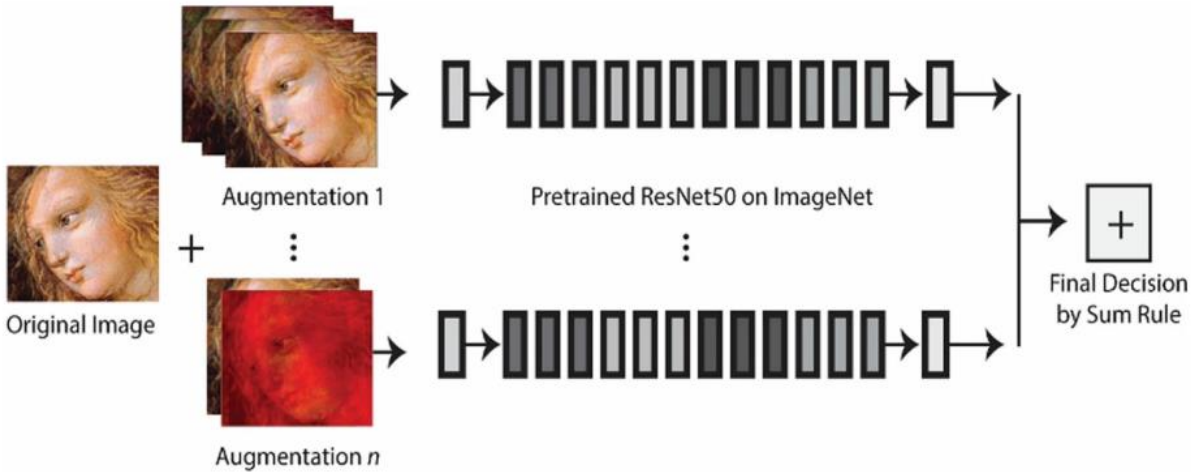


Figura 6.6: Ensemble di ResNet-50

In questo caso aggiungendo il morphing tra i metodi di Augmentation della rete si ottengono prestazioni migliori, sia in termini di Accuracy che di 1-AUC.

	Accuracy	1-AUC
Senza Morphing	0.9416	0.0122
Con Morphing	0.9438	0.0096

Per provare a ottenere una soluzione diversa rispetto al metodo proposto col fine di ottenere dei risultati più veritieri nel caso di applicazione del morfismo con figure molto diverse tra loro, ho provato a implementare la tecnica descritta nella sezione 4.4 e nel paper [6]. Tuttavia l'implementazione ha riscontrato diverse problematiche e il metodo finale dunque risulta incompleto e non utilizzabile per generare nuove immagini tramite morphing.

(a)Immagine originale 1 (b)Immagine creata (c)Immagine originale 2

Figura 6.7: Immagine ottenuta tramite implementazione TPS

Dopo aver calcolato correttamente i punti di corrispondenza, i punti di interpolazione e i parametri della matrice di interpolazione, nel momento in cui viene applicata la funzione si ha un problema di scalabilità delle dimensioni per cui si ottiene una deformazione troppo grande rispetto a quella desiderata. Come si può intuire, un eventuale addestramento porterebbe a dei risultati molto negativi.

Capitolo 7
Conclusioni

Riassumendo i risultati ottenuti nella seguente tabella

	Accuracy	1-AUC
Morphing originale	0.9236	0.014298
ResNet50 DA Classic	0.9393	0.0145
ResNet50 senza Morphing	0.9416	0.0122
ResNet50 con Morphing	0.9438	0.0096

Si può notare che il morphing proposto ha delle prestazioni leggermente peggiori rispetto all'ensenble della rete ResNet classica, tuttavia se applicato alla rete ResNet-50 descritta precedentemente si riescono ad ottenere delle prestazioni leggermente migliori.

Quindi in conclusione si può affermare che la tecnica del morphing si comporta bene quando viene applicato a dataset con immagini contenenti figure molto simili tra loro, come ad esempio nel dataset di pollini; tuttavia, quando si prova ad applicare il morphing ad immagini con figure molto diverse, come ad esempio quelle del dataset di farfalle, si ottengono dei risultati pessimi dove il metodo genera immagini lontane da essere veritiere, risultando inutilizzabile in ambito Data Augmentation.

Infine, il metodo Thin Plate Spline di cui ho provato a scriverne un'implementazione non è riuscito a ottenere risultati accettabili da poter essere utilizzato come alternativa alla triangolazione di Delaunay su cui si basa il morphing utilizzato.

Bibliografia

[1] Borut Žalik, N. L. (2013). *Chain code lossless compression using move-to-front transform and adaptive run-length encoding.* Faculty of Electrical Engineering and Computer Science, University of Maribor.

[2] Cocco, A. (2022). Morphing Data Augmentation. Tratto da https://github.com/Coccoexe/Morphing_DataAugmentation

[3] Connor Shorten, T. M. (2019). *A survey on Image Data Augmentation for Deep Learning.* Journal of Big Data 6, Article number: 60.

[4] Francisco López de la Rosa, J. L.-S. (2022). *Geometric transformation-based data augmentation on defect classification of segmented images of semiconductor materials using a ResNet50 convolutional neural network.* Elsevier.

[5] Goodfellow, I. J.-A.-F. (2014). *Generative Adversarial Networks.* arXiv.

[6] Huy Khanh, H. (2019). *Thin Plate Splines Warping*. Tratto da https://khanhha.github.io/posts/Thin-Plate-Splines-Warping/

[7] Jia Deng, W. D.-J. (2009). *ImageNet: A large-scale hierarchical image database.* IEEE. doi: 10.1109/CVPR.2009.5206848

[8] Ju Young Kang, B. S. (2012). *Application of morphing technique with mesh-merging in rapid hull form generation.* Elsevier.

[9] LeCun, Y. (1988). *A Theoretical Framework for Back-Propagation.*

[10] Loris Nanni, M. P. (2022). *Feature transforms for image data augmentation.* Neural Comput & Applic. Tratto da https://link.springer.com/article/10.1007/s00521-022-07645-z

[11] Neumann, J. v. (1958). *The Computer and the Brain.* New Haven:: Yale University Press.

[12] Noelia Vallez, G. B. (2021). *Diffeomorphic transforms for data augmentation of highly variable.* Elsevier.

[13] Pitts, W. S. (1943). *A logical calculus of the ideas immanent in nervous activity.*

[14] Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain.* Psychological Review.

[15] Simena Dinas, J. M. (2014). *A Review On Delaunay Triangulation With Application On Computer Vision.* IASET.