Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Trasformazioni diffeomorfiche per l'incremento dei dati

Relatore:
Prof. Loris Nanni

Laureando:
Alessio Cocco
1219609

Anno Accademico 2021-2022
Data di laurea 19-09-2022

Abstract

L'incremento dei dati è una tecnica ampiamente utilizzata in molti compiti di apprendimento automatico, come la classificazione delle immagini, per ampliare virtualmente la dimensione di dati ed evitare l'overfitting.
Tra le varie tecniche di data augmentation che modificano le immagini originali con rotazioni, capovolgimenti, aggiunte di rumore e resize, questo elaborato propone una soluzione basata su trasformazioni diffeomorfiche su dataset di immagini che presentano shape molto variabili.
In seguito ai test effettuati siamo riusciti ad ottenere un Accuracy del 92.36% e, con ensemble di ResNet50 fino a 94.38%.

Indice

Introduzione **1**

1 Reti Neurali Convoluzionali **2**
1.1 Introduzione alle Reti Neurali Artificiali 2
1.2 Deep Learning e CNN 4
1.2.1 Convoluzione 5
1.2.2 Pooling 6
1.2.3 Funzione di Attivazione 6

2 Overfitting **7**
2.1 Overfitting 7
2.1.1 Accuracy 9
2.1.2 Precision e Recall 10
2.1.3 ROC Curve e AUC 11
2.1.4 Loss Curve 12
2.1.5 Overfitting, Perchè? 12

3 Data Augmentation **14**
3.1 Trasformazioni Geometriche 14
3.2 Aumento di Rumore 15
3.3 GAN 15

4 Ensemble **16**
4.1 Livello di Decisione 16
4.1.1 Majority Vote Rule 16
4.1.2 Borda Count 17
4.2 Confidenza 17
4.3 Boosting 18

5 Morphing **19**
5.1 Definizione 19
5.2 Metodo Proposto 20
5.2.1 Dataset, prima e dopo 21
5.2.2 Punti di contorno 21
5.2.3 Punti Medi 23

5.2.4 Triangolazione di Delaunay 23
5.2.5 Morphing 23

6 Fase Sperimentale 24
6.1 Interfaccia MATLAB 24
6.2 ResNet50 24
6.3 Test e Risultati Sperimentali 25

7 Conclusioni 29

Elenco delle figure

1.1 Modello generale del neurone M&P 2
1.2 Percettrone di Rosenblatt con pesi w e funzione di attivazione a gradino . . 3
1.3 Rete MLP a 3 Livelli 3
1.4 CNN per classificare numeri scritti a mano [8] 4
1.5 Convoluzione dell'input 3x3 con filtro 2x2, stride = 1 e padding = 0 5
1.6 Max Pooling dell'input 4x4 con filtro 2x2, stride = 2 e padding = 0 6
1.7 Funzioni di attivazione 6

2.1 Esempio di Matrice Binaria 8
2.2 Matrice di confusione per classificazione multi-classe di numeri scritti a mano. 9
2.3 Esempio di ROC Curve. 11
2.4 Esempio di AUC. 11
2.5 Esempio di andamento di una Loss curve per training e validation. 12
2.6 Esempio di grafici di Accuracy e Loss 12

3.1 Trasformazioni Geometriche 14
3.2 Aumento di rumore 15
3.3 Volti di persone non esistenti, creati tramite GAN 15

4.1 Borda Count con 3 classificatori 17
4.2 Boosting 18

5.1 Morphing di due immagini. 19
5.2 Flow Chart dell'algoritmo di morphing utilizzato 20
5.3 Rappresentazione di chain code 21
5.4 Proiezione dei punti della catena 22
5.5 Triangolazione di Delaunay 23

6.1 Architettura di ResNet50 24
6.2 Morphing di granuli di acronomia aculeta 25
6.3 Morphing di esemplari di Farfalla Monarca 25
6.4 Maschere di granuli di acronomia aculeta 26
6.5 Maschere di esemplari di Farfalla Monarca 26
6.6 Maschera creata con modifica del metodo 26
6.7 Morphing Originale e Modificato 27
6.8 Ensemble di ResNet50 28

Introduzione

Le reti neurali convoluzionali profonde sono la scelta principale per quanto riguarda la classificazione delle immagini. Non solo perchè hanno ottenuto risultati straordinari in molti compiti di Computer Vision, ma anche perchè queste reti imparano ad estrarre le caratteristiche direttamente dalle immagini, evitando così che sia un umano a fornire il metodo di estrazione più adatto. Le feature che vengono quindi imparate dalle CNNs superano di gran lunga l'efficacia delle feature fatte a mano, soprattutto grazie alla riduzione progressiva della risoluzione spaziale delle immagini, e dell'aumento della profondità delle feature maps.

Tuttavia, queste reti sono fortemente dipendenti dai big data per evitare l'overfitting. L'overfitting si riferisce al fenomeno per cui una rete neurale si adatta ai dati osservati, perchè ha un numero eccessivo di parametri rispetto al numero di osservazioni. Questo fenomeno si verifica soprattutto quando il tempo di addestramento è troppo elevato rispetto al numero di pattern di training. Ciò comporta la presenza di un momento del training dopo il quale si nota un andamento decrescente dell'accuracy del validation set. In generale le CNNs tendono a overfittare quando sono allenate su dataset molto piccoli. L'overfitting riduce la capacità di queste reti di generalizzare ciò che hanno imparato e di conseguenza classificare correttamente campioni mai visti. La difficoltà maggiore per i ricercatori per quanto riguarda l'addestramento di una CNN è la raccolta di dataset sufficientemente grandi per soddisfare i bisogni della rete e per evitare l'overfitting.

In molti ambiti, come la medicina, è difficile e molto costoso creare dei dataset adeguati per il training di una rete dove molte immagini richiederebbero spese e lavoro eccessivi. Tra le varie tecniche che si possono adottare per evitare l'overfitting, quali dropout [1], zero-shot [2], batch normalization [3]. Questa ricerca si concentra sull'incremento di dati, in inglese Data Augmentation(DA). I metodi di Data Augmentation mirano a incrementare il numero di dati disponibili per il training, a partire da un dataset iniziale, attraverso la modifica delle immagini originali grazie a varie trasformazioni. Le tradizionali tecniche di incremento dei dati per la classificazione delle immagini creano nuovi campioni dai dati di addestramento originali, ad esempio capovolgendo, distorcendo, aggiungendo una piccola quantità di rumore o ritagliando una parte dell'immagine originale.
In questo elaborato verrà preso in considerazione un metodo alternativo per il Data Augmentation, che si basa sul Morphing [4]. Infine verrà effettuato un confronto tra il metodo di morphing originale e una variante da me sviluppata [19].

Capitolo 1

Reti Neurali Convoluzionali

In questo primo capitolo viene introdotto il concetto di Rete Neurale Convoluzionale, in inglese Convolutional Neural Networks(CNN), a partire dai primi modelli di Reti Neurali percorrendo la storia fino ad arrivare alla rete più usata attualmente in ambito di classificazione.

1.1 Introduzione alle Reti Neurali Artificiali

Una rete neurale artificiale viene definita come un modello computazionale, composto da "neuroni" artificiali, ispirato alle reti neurali biologiche che costituiscono il cervello degli animali.
Il primo modello di neurone fu proposto dal neurofisiologo Warren McCulloch e dal matematico Walter Pitts nel 1943 [5]. Il neurone in questione svolgeva la funzione di una porta logica, capace di assumere due soli stati; una rete composta da questi neuroni permetteva di calcolare semplici funzioni booleane, come AND, OR e NOT. Il modello in Figura 1.1 rende visibile il concetto alla base di quest'idea, una serie di valori booleani vengono dati in input alla parte di sinistra g che a sua volta li aggrega e li rende disponibili a f che prenderà una decisione.

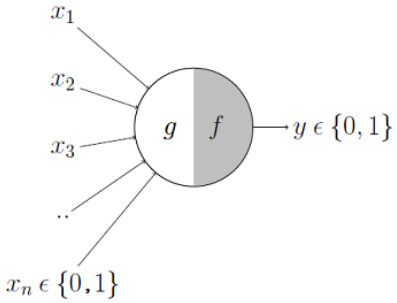


Figura 1.1: Modello generale del neurone M&P

Tuttavia questo modello ha importanti limitazioni: gli input possono essere solamente dei booleani e contribuiscono tutti con la stessa importanza nella decisione.

Nel 1958 lo psicologo Frank Rosenblatt propone nella sua pubblicazione [6] un modello di neurone chiamato Percettrone. Il modello di Rosenblatt mirava alle funzioni quali l'immagazzinamento delle informazioni e la loro influenza sul riconoscimento dei pattern. La chiave del progresso rispetto al modello precedente di M&P è la possibilità di avere pesi sinapsici variabili. In Figura 1.2 si può vedere come nel modello di Rosenblatt siano presenti i pesi w per ogni dato in input, questo permette al percettrone di pesare i dati ricevuti in modi differenti. Inoltre i valori in ingresso ora possono essere dei valori reali ed è presente una funzione di attivazione a scalino.

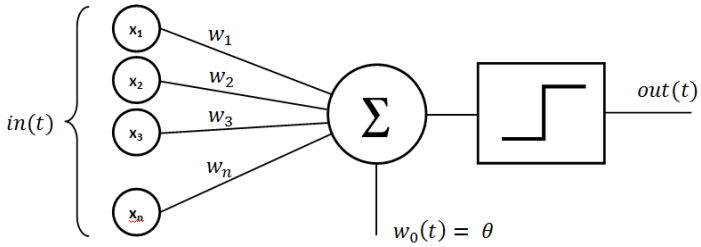


Figura 1.2: Percettrone di Rosenblatt con pesi w e funzione di attivazione a gradino

Queste novità introdotte da Rosenblatt rendono il percettrone ideale per compiti quali la classificazione lineare, grazie alla possibilità di individuare, utilizzando i vari pesi sinaptici, l'iperpiano di separazione usando la seguente formula:

$$b + w_1 x_1 + w_2 x_2 + ... + w_n x_n = 0 \tag{1.1}$$

La funzione dell'algoritmo di apprendimento è modificare i valori dei pesi w in modo da minimizzare l'errore di classificazione.

Anche il percettrone aveva i propri limiti, non era in grado di apprendere funzioni linearmente separabili. La ricerca riprese solo quando venne introdotto il modello Multilayer Perceptron(MLP). Il modello MLP consiste di almeno 3 nodi, compreso input e output. Il livello centrale viene detto Hidden Layer(livello nascosto). Questo modello è interamente connesso e ogni nodo è un neurone che utilizza funzioni di attivazioni non lineari. Nonostante le grandi potenzialità, una rete MLP con un solo hidden layer non riesce ad approssimare in modo efficiente funzioni lineari molto complesse; per ovviare al problema si può aumentare il numero di layer interni, aumentando tuttavia anche la complessità computazionale dell'addestramento fino a renderlo anche impraticabile.

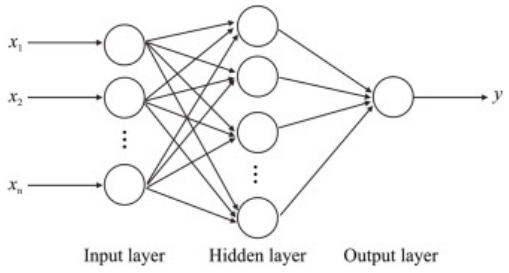


Figura 1.3: Rete MLP a 3 Livelli

1.2 Deep Learning e CNN

Il Deep Learning è una classe di algoritmi del machine learning che usa più layer per estrarre progressivamente feature di alto livello. Per esempio nell'ambito dell'elaborazione delle immagini, feature di basso livello identificano i contorni, mentre feature di alto livello identificano concetti rilevanti agli umani, come lettere, numeri o persino volti.

La maggior parte delle reti "profonde" usate sono basate su reti neurali artificiali, in particolare reti neurali convoluzionali(CNN), introdotte per la prima volta dall'informatico Yann LeCun [7]. Le CNNs sono una variante più complessa delle reti MLP e sono usate principalmente per problemi di classificazione e regressione. A differenza delle reti Multilayer Perceptron riescono a ridurre notevolmente la loro complessità attraverso il processing locale e i pesi condivisi. In particolare i neuroni di un livello sono connessi solo a un certo numero di neuroni del livello precedente e i pesi delle connessioni sono condivisi da gruppi di neuroni.
L'architettura di una rete convoluzionale è analoga al modo in cui i neuroni del cervello umano sono collegati e si ispira al modo in cui è organizzata la corteccia visiva. Ogni neurone risponde a uno stimolo solo in una regione ristretta del campo visivo, noto come campo recettivo, sarà poi un insieme di questi campi sovrapposti a coprire l'intera area visiva.

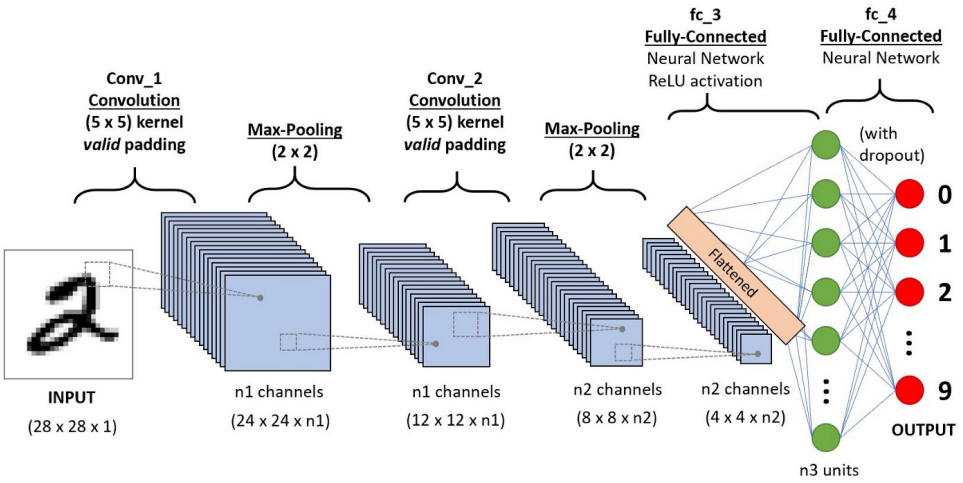


Figura 1.4: CNN per classificare numeri scritti a mano [8]

Il ruolo di una CNN è ridurre l'immagine di input in una forma più facile da processare, senza perdere le feature fondamentali per ottenere una buona predizione. I "blocchi" che costituiscono una rete convoluzionale sono generalmente livelli di pooling, convoluzione e attivazione; questa costruzione a "blocchi" permette di effettuare un "pre-processing" dei dati in input.

1.2.1 Convoluzione

Il livello convoluzionale è il livello chiave della rete, il suo compito è individuare con elevata precisione schemi come curve, angoli e circonferenze, più in generale tratti distintivi. Più filtri si applicano, maggiore sarà la complessità delle feature che si possono individuare. L'operazione di convoluzione consiste nell'applicare un filtro di dimensione $n*n$ alla matrice in input; con Padding si intende un layer di contorno facoltativo, a volte utilizzato per aumentare la dimensione dell'input in modo da evitare perdita dei dati che si trovavano nei bordi della matrice. Il risultato sarà una matrice di output ridotta le cui dimensioni si possono calcolare a priori usando la seguente formula:

$$W_{out,i} = \frac{W_{in,i} - F_i + 2 * Padding}{Stride} + 1 \tag{1.2}$$

$W_{in,i}$: dimensione spaziale della matrice in input
$W_{out,i}$: dimensione spaziale della matrice in output
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

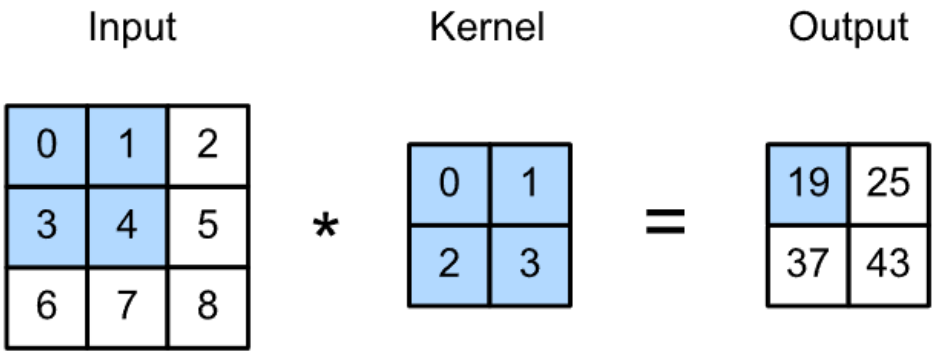


Figura 1.5: Convoluzione dell'input 3x3 con filtro 2x2, stride = 1 e padding = 0

1.2.2 Pooling

Un altro "blocco" importante nelle reti convoluzionali è il cosiddetto Pooling. Il funzionamento è molto simile alla convoluzione, ovvero si applica un filtro di dimensioni $n*n$ alla matrice in input e si ottiene una matrice di output di dimensione ridotta. Esistono vari metodi di Pooling, tra cui i più usati Max-Pooling [9] e Average-Pooling. A differenza della convoluzione, il pooling prende un area della stessa dimensione del filtro e sceglie solo un dato, il massimo nel caso del max-pooling, o la media nel caso dell'average-pooling.

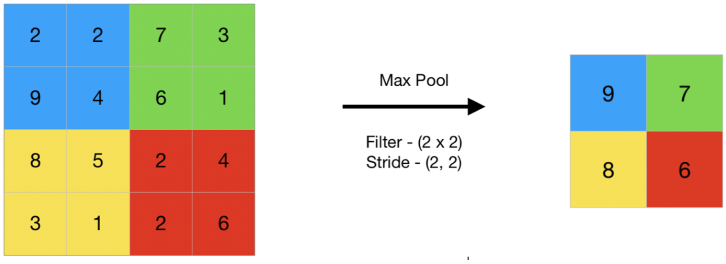
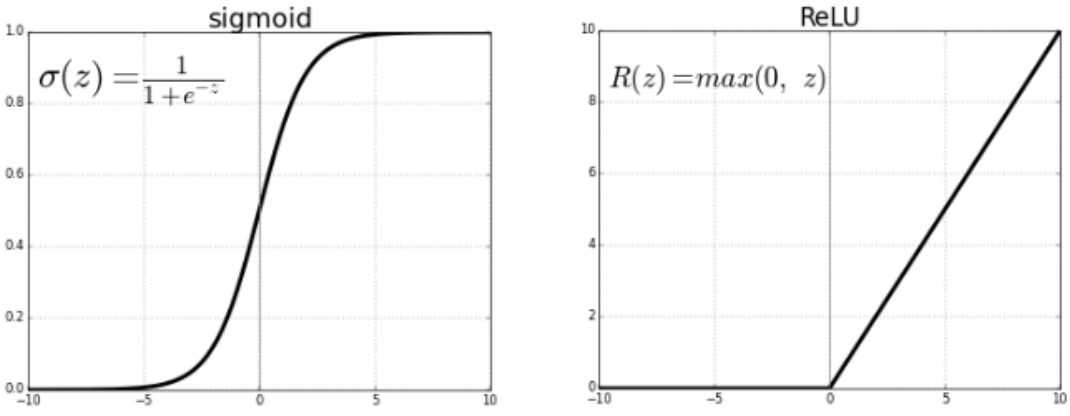


Figura 1.6: Max Pooling dell'input 4x4 con filtro 2x2, stride = 2 e padding = 0

1.2.3 Funzione di Attivazione

Le funzioni di attivazione si pongono l'obiettivo di annullare i valori non utili ottenuti nei livelli precedenti, e sono generalmente poste dopo i livelli convoluzionali. In una rete neurale biologica le funzioni di attivazione potrebbero corrispondere all'impulso nervoso generato dal neurone e trasmesso all'assone, e nel caso in cui il segnale non sia sufficientemente alto viene ignorato.
Alcune tra le funzioni di attivazione più utilizzate sono la standard logistic function e la ReLU [10].



(a) Sigmoide **(b)** ReLU

Figura 1.7: Funzioni di attivazione

Capitolo 2

Overfitting

Nel capitolo di Overfitting e Data Augmentation vedremo le motivazioni che hanno spinto le Reti Neurali Convoluzionali a necessitare di grosse moli di dati, e verranno illustrate alcune tra le tecniche più conosciute e usate per incrementare i dataset qualora non ci siano un numero sufficiente di pattern per addestrare la rete ed evitare Overfitting.

2.1 Overfitting

Con il termine Overfitting, ovvero sovraddattamento, si intende il momento in cui una Rete Neurale produce un analisi che corrisponde troppo o esattamente a un certo insieme di dati, mentre non riesce a generalizzare e quindi ad adattarsi con dati aggiuntivi e osservazioni future. Generalmente quando una CNN viene addestrata ha a disposizione quello che viene chiamato "training set"; se si verifica overfitting la rete ha "imparato a memoria" e si è adattata eccessivamente al dataset di training e commette troppi errori quando deve analizzare altri dati.

Per comprendere a fondo l'overfitting bisogna prima introdurre il concetto di Accuracy, Loss e ROC curve, tre importanti metriche utilizzate per valutare l'addestramento e la classificazione di una rete neurale. Tutte le predizioni effettuate da una CNN vengono valutate e incidono in modi diversi su Accuracy, Loss e ROC. Prima di soffermarci su questi concetti è bene capire come viene valutato il lavoro di classificazione di una Rete Neurale.

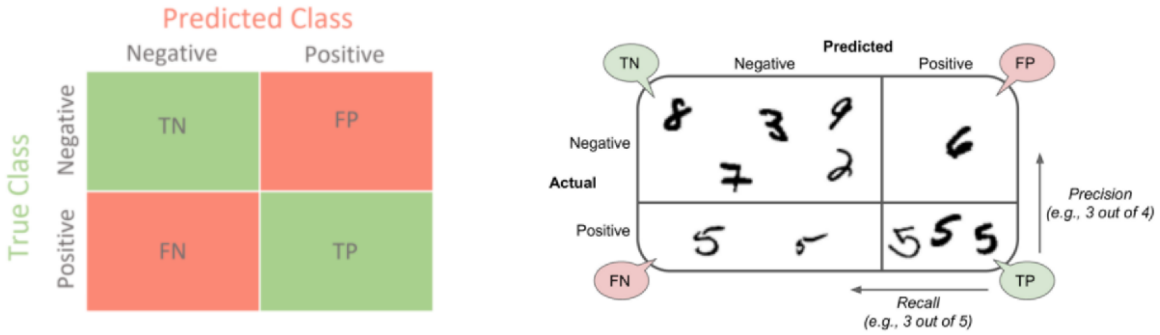
Consideriamo un problema di classificazione binario, avendo le due classi Positivi e Negativi, se si utilizza un sistema con soglia, un pattern viene classificato in base alla probabilità di appartenere a una determinata classe, se la probabilità calcolata è maggiore a una soglia prestabilita viene classificato Positivo, Negativo altrimenti. Se invece non si usa il sistema con soglia un pattern viene classificato Positivo se viene riconosciuto come la classe identificata come Positivo, Negativo in ogni altro caso. Per esempio nella classificazione di Cani, le porzioni di immagini in cui appare un cane viene contrassegnato come Positivo, Negativo in porzioni in cui non appare.

Chiaramente la rete può commettere degli errori di classificazione, perciò è necessario valutare quanti dei pattern sono stati classificati correttamente e di conseguenza quanti errori sono stati commessi.

Dato un numero N di pattern da classificare, il risultato di ogni tentativo di classificazione può essere:

- Vero Positivo(VP): un pattern Positivo è stato correttamente assegnato ai Positivi.
- Vero Negativo(VN): un pattern Negativo è stato correttamente assegnato ai Negativi.
- Falso Positivo(FP): un pattern Negativo è stato erroneamente assegnato ai Positivi.
- Falso Negativo(FN): un pattern Positivo è stato erroneamente assegnato ai Negativi.

Può essere più chiaro il concetto se si osserva la matrice di confusione in Figura 2.1, dove viene evidenziata la differenza tra la classe vera e la predizione effettuata.



(a) Matrice Binaria generale **(b)** Matrice Binaria con esempio

Figura 2.1: Esempio di Matrice Binaria

Nel caso invece di classificatore multi-classe si valutano gli errori tra i valori veri e i valori predetti, risulta molto utile la matrice di confusione per capire come sono distribuiti gli errori. Nella Figura 2.2 [11] si possono notare nelle righe le classi di appartenenza e sulle colonne le predizioni effettuate. Per esempio considerando la cifra "0", è stata classificata correttamente, circa il 46% dei casi, a seguire nella stessa riga si possono vedere le percentuali di errori commessi. Idealmente la matrice dovrebbe contenere i valori più alti nella diagonale, valori alti in altre celle indicano errori molto alti.

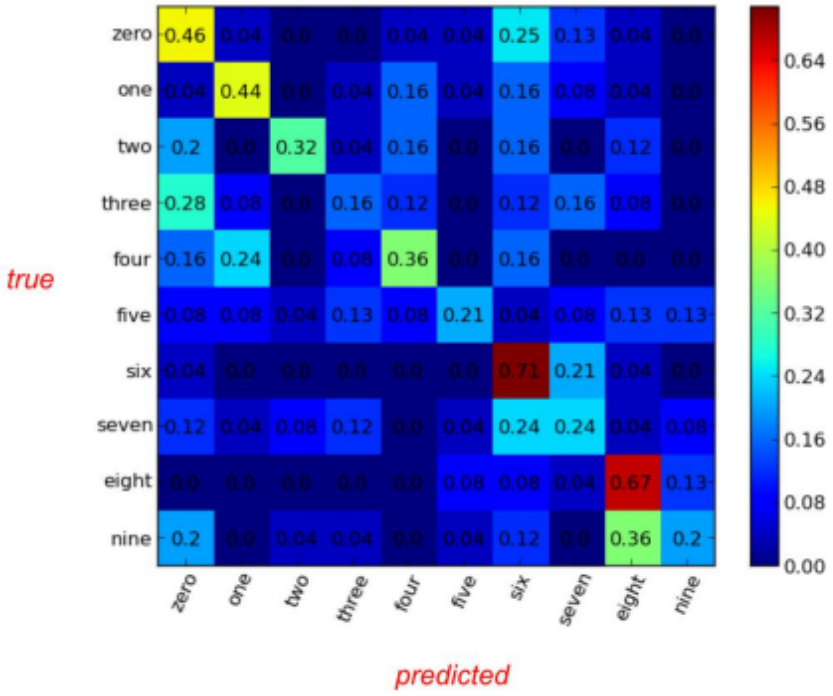


Figura 2.2: Matrice di confusione per classificazione multi-classe di numeri scritti a mano.

2.1.1 Accuracy

L'accuracy o accuratezza è una metrica per valutare la classificazione di un modello. L'accuracy si esprime con la seguente formula:

$$Accuracy = \frac{Numero\ di\ predizioni\ corrette}{Numero\ totali\ di\ predizioni} \tag{2.1}$$

Nel caso invece di classificazione binaria può essere calcolata anche in termini di Positivi e Negativi:

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.2}$$

Tuttavia un accuracy molto alta si può raggiungere anche nel caso in cui una delle due classi sia più rara dell'altra. A primo impatto è difficile capire perchè, con il seguente esempio risulterà piu chiaro:
Si immagini di avere a disposizione 100 esempi di tumori, in fase di classificazione abbiamo ottenuto i seguenti punteggi considerando Positiva la classe di Tumori Maligni e Negativa la classe di Tumori Benigni:

Vero Positivo: 1	Falso Positivo: 1
Falso Negativo: 8	Vero Negativo: 90

In questo caso l'accuracy risulta:

$$Accuracy = \frac{1+90}{1+90+1+8} = 0.91 = 91\% \tag{2.3}$$

Dei 91 tumori benigni sono stati classificati correttamente 90 tumori. Se invece guardiamo i tumori maligni, solo 1 tumore e' stato classificato correttamente come maligno, quindi 8 tumori maligni non sono stati trovati. Anche se abbiamo ottenuto un'accuracy del 91%, questa rete non si comporta meglio di una con lo 0% di accuracy.

2.1.2 Precision e Recall

Esiste un ulteriore indicatore chiamato Precision - Recall che risolve i problemi di accuracy evidenziati in precedenza qualora sia necessario, aggiungendo due metriche che valutano la bontà del dato di accuracy trovato; con Precision si intende quanto accurato è il sistema di classificazione, mentre Recall indica quanto risulta selettivo. Precision e Recall vengono calcolati come segue:

$$Precision = \frac{VP}{VP+FP} \tag{2.4}$$

$$Recall = \frac{VP}{VP+FN} \tag{2.5}$$

Riprendendo l'esempio precedente dei tumori, proviamo a calcolare i livelli di Precision e Recall e cerchiamo di capire cosa rappresentano nel nostro caso:

$$Precision = \frac{1}{1+1} = 0.5 \tag{2.6}$$

Il nostro modello ha una precisione dello 0.5, cioè quando predice un tumore maligno è vero al 50% dei casi.

$$Recall = \frac{1}{1+8} = 0.11 \tag{2.7}$$

Invece per quanto riguarda il Recall abbiamo trovato un valore di 0.11, questo indica che il nostro modello riesce a identificare un tumore maligno solo nell'11% dei casi, i restanti vengono identificati come tumori benigni.

2.1.3 ROC Curve e AUC

Con il termine ROC Curve(Receiver Operating Characteristic Curve) [12] si intende il grafico che mostra le prestazioni di un modello di classificazione. La curva è costruita sui due parametri:

$$Tasso\ di\ Veri\ Positivi = \frac{VP}{VP + FN} \tag{2.8}$$

$$Tasso\ di\ Falsi\ Negativi = \frac{FP}{FP + VN} \tag{2.9}$$

Nella Figura 2.3 sono rappresentate varie ROC curve, si vede chiaramente che un risultato migliore si ottiene quando la curvatura è più accentuata.

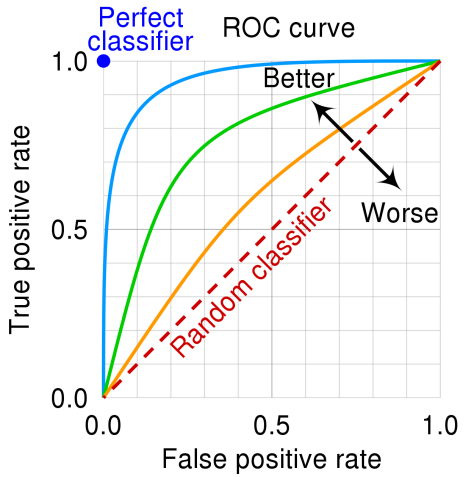


Figura 2.3: Esempio di ROC Curve.

Con il termine AUC(Area Under the ROC Curve) si intente l'area sottostante alla curva ROC, un valore alto di AUC indica una migliore performance del modello.

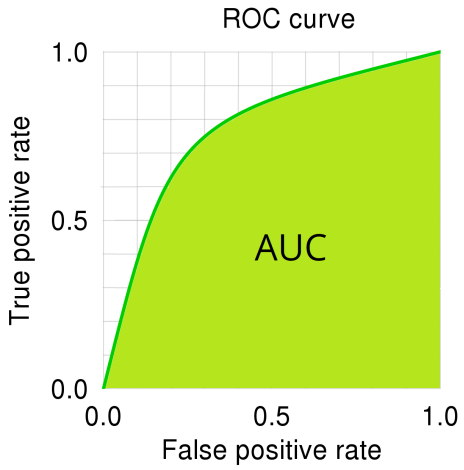


Figura 2.4: Esempio di AUC.

2.1.4 Loss Curve

Il grafico Loss mostra l'andamento della curva Loss sia per il "training set" che per il "validation set". La curva indica il grado di adattamento ai dati ricevuti nel training e nel validation. In Figura 2.5 un esempio di Loss Curve, in blu la curva del training e in rosso la curva del validation, si nota un punto in cui la loss del validation inizia a salire, da quel momento il modello sta iniziando a "overfittare".

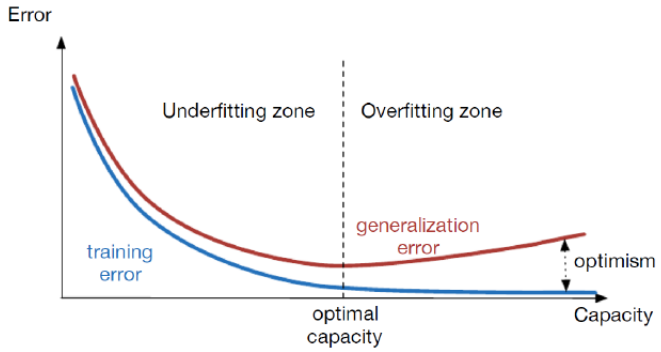


Figura 2.5: Esempio di andamento di una Loss curve per training e validation.

2.1.5 Overfitting, Perchè?

Ora che abbiamo analizzato a fondo diversi fattori che interessano l'addestramento, vediamo il classico comportamento di una Rete Neurale in fase di training:

Figura 2.6: Esempio di grafici di Accuracy e Loss

Si vede chiaramente che esiste un momento nel training in cui l'Accuracy inizia a decrescere e la Loss a crescere; in quel momento il modello sta iniziando a overfittare, si sta adattando ai pattern di training e non riesce a generalizzare per nuove immagini.

In seguito sono elencate alcune tra le tecniche più utilizzate per evitare che il modello overfitti:

- Early Stopping: consiste nell'interruzione dell'addestramento appena prima che inizi ad overfittare.
- Data Augmentation: viene aumentata la dimensione del dataset originale attraverso vari metodi.
- Feature Selection: si tratta di selezionare le feature più utili da estrarre attraverso il modello, per renderlo meno complesso.
- Regularization: prevede l'aggiunta di una penalità per i dati in input con coefficienti troppo grandi, così da limitare la varianza del modello.
- Ensemble: si utilizzano più modelli per l'addestramento le cui previsioni vengono aggregate per identificare il risultato finale. I metodi più noti sono Bagging e Boosting. Nel Bagging ogni classificatore ha lo stesso peso nella votazione finale della predizione, invece nel Boosting si aggiunge il peso in base all'errore di accuratezza che ogni modello commette in fase di apprendimento.

Tra i vari motivi che possono causare Overfitting, il più comune è l'addestramento su dataset di dimensioni insufficienti. In questo elaborato approfondiamo la tecnica del Data Augmentation e in particolare un metodo specifico: il Morphing.

Capitolo 3

Data Augmentation

Vista l'enorme importanza di avere un dataset adeguato per l'addestramento di una Rete Neurale Convoluzionale, una delle maggiori difficoltà ad oggi nell'ambito del Deep Learning è la ricerca e la creazione di dati sufficienti per ottenere buoni risultati durante il training supervisionato di CNNs. Sopratutto in ambiti quali la medicina e biologia la creazione di immagini è un lavoro molto dispendioso sia dal lato economico che dal lato pratico, servirebbero molte ore di laboratorio per estrarre tutte le immagini necessarie. Per questo motivo la tecnica dell'incremento dei dati è molto utilizzata e in costante ricerca di nuovi metodi sempre più efficaci di creazione di dati verosimili alla realtà. I metodi di Data Augmentation più comuni utilizzano principalmente trasformazioni geometriche, aumento di rumore e trasformazioni basate su GAN.

3.1 Trasformazioni Geometriche

I metodi più semplici si basano sulle trasformazioni geometriche, tuttavia le immagini create devono essere il più verosimili possibile. Alcune tra le trasformazioni maggiormente utilizzate sono mostrate in Figura 3.1 da Sinistra a Destra: Originale, Capovolgimento Verticale, Capovolgimento Orizzontale, Padding, Crop. E altre non presenti nell'immagine come Rotazioni e Scaling.

Figura 3.1: Trasformazioni Geometriche

3.2 Aumento di Rumore

Questo metodo prevede l'aggiunta di rumore ai dati in ingresso utilizzati per l'addestramento, così da rendere più difficile per la CNN in fase di classificazione trovare una soluzione che corrisponda all'insieme di immagini di training, riducendo di conseguenza l'overfitting e migliorando la generalizzazione.

(a) Immagine Originale **(b)** Immagine con aumento di rumore

Figura 3.2: Aumento di rumore

3.3 GAN

Il metodo GAN [13] (Generative Adversarial Network) utilizza un generatore e un discriminatore, il primo si occupa di creare immagini il più possibile simili a quelle esistenti, mentre il secondo cerca di valutare quali siano le immagini originali e quali quelle create.

Figura 3.3: Volti di persone non esistenti, creati tramite GAN

Capitolo 4

Ensemble

L'apprendimento Ensemble prevede l'utilizzo di più modelli in fase di addestramento in modo da ottenere una predizione migliore rispetto ai singoli modelli di cui è costituito. Gli algoritmi che effettuano la predizione si occupano di cercare in uno spazio di ipotesi quella più adatta a rappresentare il problema dato, tuttavia anche in uno spazio sufficientemente grande è molto difficile trovare la previsione corretta. Gli ensemble combinano i risultati di più modelli per ottenere una predizione migliore. Un sistema ensemble prevede potenza di calcolo e risorse maggiori rispetto a un singolo modello; ciò nonostante risulterà più accurato ed efficiente, a parità di risorse di calcolo e archiviazione utilizzate, rispetto a quanto sarebbe accurato un singolo metodo addestrato con le stesse risorse.

4.1 Livello di Decisione

Ogni classificatore effettua la propria predizione e fornisce in output la propria decisione che consiste nella classe di appartenenza del pattern. Esistono vari modi per combinare le decisioni di tutti i classificatori.

4.1.1 Majority Vote Rule

Questo metodo consiste nell'assegnare il pattern alla classe più votata, tenendo in considerazione che ogni classificatore vota per una sola classe.
Siano $i = \{Cs_1, Cs_2, ..., Cs_n\}$ un insieme di classi e $j = \{C_1, C_2, ..., C_m\}$ un insieme di classificatori, e

$$\theta_{ij} = \begin{cases} 1 & \textit{se } i \textit{ è la classe votata da } C_j \\ 0 & \textit{altrimenti} \end{cases} \quad (1 \leq i \leq n,\ 1 \leq j \leq m) \tag{4.1}$$

Il pattern viene assegnato alla classe t tale che:

$$t = \underset{i=1...n}{\operatorname{argmax}} \left\{ \sum_{j=1}^{m} \theta_{ij} \right\} \tag{4.2}$$

4.1.2 Borda Count

In questo caso si chiede a ogni classificatore di assegnare una classifica delle probabilità anzichè una singola classe. La classifica viene convertita in punteggi che vengono sommati, la classe con il 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 4.1: Borda Count con 3 classificatori

4.2 Confidenza

Ogni classificatore fornisce la confidenza di classificazione del pattern rispetto a ogni classe. Si ottiene un vettore di confidenze $conf_j = [conf_j 1, conf_j 2, ..., conf_{jn}]$. Esistono poi vari metodi per unire le confidenze ottenute:

$$sum = \sum_{j=1}^{m} conf_j \quad t = \underset{i=1...n}{\text{argmax}}\{sum_i\} \tag{4.3}$$

$$prod = \prod_{j=1}^{m} conf_j \quad t = \underset{i=1...n}{\text{argmax}}\{prod_i\} \tag{4.4}$$

$$max_i = \max_{j=1...m} conf_{ji} \quad t = \underset{i=1...n}{\text{argmax}}\{max_i\} \tag{4.5}$$

$$min_i = \min_{j=1...m} conf_{ji} \quad t = \underset{i=1...n}{\text{argmax}}\{min_i\} \tag{4.6}$$

Un altra variante possibile consiste nella somma pesata, ovvero la somma dei vettori confidenza viene eseguita pesando i classificatori in base al grado di abilità g_j, che può essere calcolato in base alle prestazioni del singolo classificatore.

$$avgsum = \sum_{j=1}^{m} g_j \cdot conf_j \quad t = \underset{i=1...n}{\text{argmax}}\{avgsum_i\} \tag{4.7}$$

4.3 Boosting

La tecnica del Boosting [14] prevede la combinazione di più classificatori "deboli" per la creazione di un classificatore "forte". L'apprendimento è incrementale e prevede l'aggiunta ad ogni iterazione di un classificatore efficace sui pattern critici. In questo modo ad ogni iterazione vengono pesate maggiormente le valutazioni sui pattern classificati erroneamente il ciclo precedente.

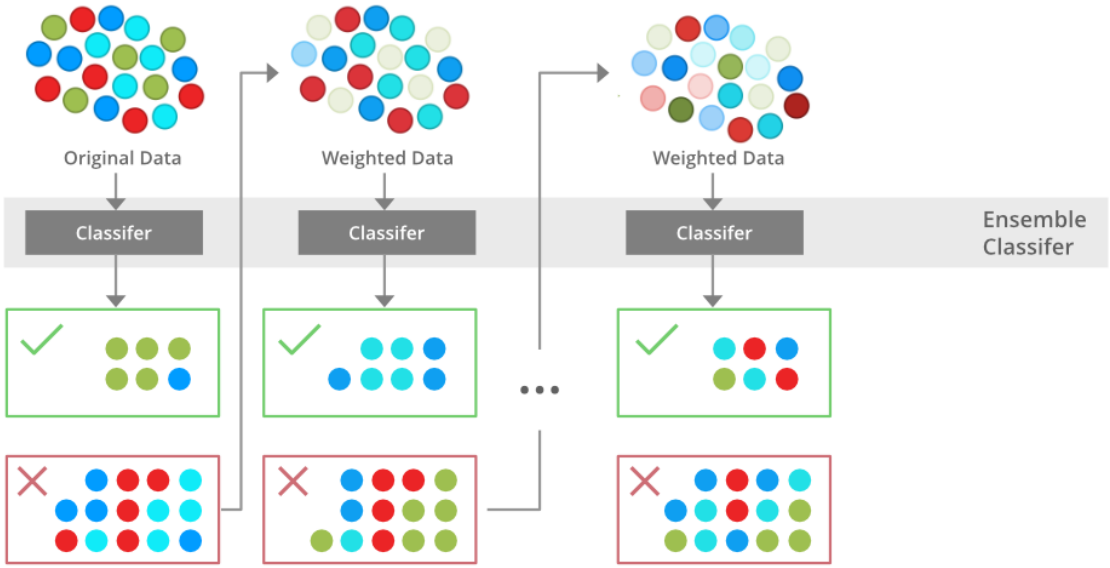


Figura 4.2: Boosting

Come si può vedere in Figura 4.2 alla seconda iterazione i pattern "blu", che erano stati classificati erroneamente nella prima iterazione, sono pesati maggiormente e di conseguenza vengono classificati correttamente.

Capitolo 5

Morphing

In questo capitolo verrà illustrata nel dettaglio la tecnica del morphing, a partire dalla definizione fino ad analizzare il codice proposto nel paper "Diffeomorphic transforms for data augmentation of highly variable shape and texture object" [4].

5.1 Definizione

Il morphing è un effetto digitale utilizzato per la transizione tra immagini senza soluzione di continuità, più semplicemente si tratta di una tecnica utilizzata per effettuare la transizione tra due immagini; è una soluzione alternativa e più nuova della dissolvenza. Si può osservare un esempio di morphing applicato a due immagini di farfalle in Figura 5.1.

Figura 5.1: Morphing di due immagini.

5.2 Metodo Proposto

L'idea alla base del metodo trattato in questo elaborato è di applicare la tecnica del morphing al dataset originale considerando tutte le combinazioni possibili di due immagini, così da aumentare notevolmente il volume del dataset originale. Come evidenziato nel Flow Chart in Figura 5.2, per la creazione di un'immagine si parte da due immagini del dataset che, insieme alle rispettive maschere, vengono processate dall'algoritmo di Data Augmentation.

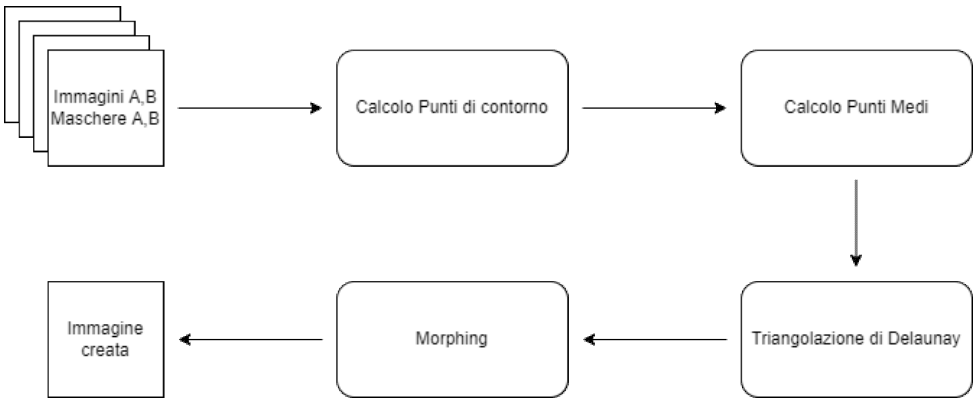


Figura 5.2: Flow Chart dell'algoritmo di morphing utilizzato

Prima di analizzare più a fondo i vari passi dell'algoritmo vediamo in generale il concetto chiave su cui si basa. Data una coppia di immagini A,B il primo step è stabilire la corrispondenza tra ogni pixel(x_a,y_a) di A e i pixel(x_b,y_b) di B, e di conseguenza il pixel(x_c,y_c) dell'immagine C creata sarà:

$$x_c = (1 - \alpha)x_a + \alpha x_b \tag{5.1}$$

$$y_c = (1 - \alpha)y_a + \alpha y_b \tag{5.2}$$

e l'intensità del pixel:

$$C(x_c, y_c) = (1 - \alpha)A(x_a, y_a) + \alpha B(x_b, y_b) \tag{5.3}$$

$\alpha \in [0, 1]$: parametro che controlla la somiglianza alle immagini A,B.

Con l'incremento progressivo del parametro α il metodo riesce a creare 5 immagini; inizialmente simili ad A e mano a mano sempre più simili a B.

5.2.1 Dataset, prima e dopo

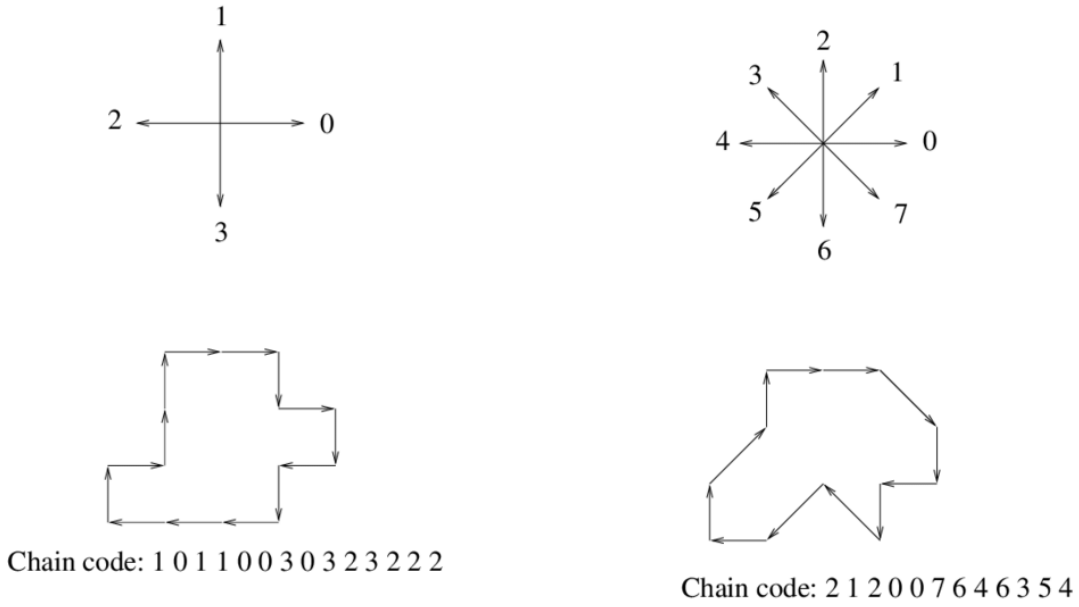
Innanzitutto vediamo l'impatto di questo metodo sul dataset originale. L'algoritmo prevede la creazione di 5 immagini per ogni coppia del dataset, a partire dalla prima crea immagini sempre più simili alla seconda, come in figura 5.1. Supponiamo quindi di avere un dataset di dimensione n, tutte le possibile coppie possiamo calcolarle come segue:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n * (n-1)}{2} \tag{5.4}$$

Moltiplicando di un fattore 5 possiamo ricavare la dimensione finale del dataset. Supponendo di avere un dataset originale di 100 immagini, riusciamo a ottenere un dataset di 24 750 immagini.

5.2.2 Punti di contorno

Il primo passo che compie l'algoritmo dopo aver scelto le 2 immagini su cui applicare il morphing è calcolarne i contorni attraverso la "Elliptical Fourier Descriptor(EFD)" [15]. In questo modo si riesce ad ottenere un contorno dell'immagine iniziale dove viene calcolato il "chain-code". Con "chain-code" si intende un metodo di segmentazione, il cui funzionamento consiste nel codificare separatamente ogni componente collegato. 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. Vedi esempio in Figura 5.3.



(a) 4-chain code **(b)** 8-chain code

Figura 5.3: Rappresentazione di chain code

Una volta ottenuti i punti della catena, possiamo calcolare le variazioni tra i vari punti lungo gli assi x e y come segue:

$$\Delta x_i = sgn(6 - a_i)sgn(2 - a_i) \tag{5.5}$$

$$\Delta y_i = sgn(4 - a_i)sgn(a_i) \tag{5.6}$$

$$\Delta t_i = 1 + \left(\frac{\sqrt{2} - 1}{2}\right)(1 - (-1)^{a_i}) \tag{5.7}$$

a_i : i-esimo elemento della catena

Δt_i : modulo del segmento tra due punti vicini

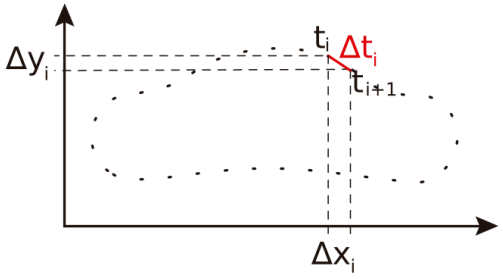


Figura 5.4: Proiezione dei punti della catena

Consideriamo ora la serie di Fourier:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{N}[a_n cos(nt) + b_n sin(nt)] \tag{5.8}$$

Possiamo semplificare le proiezioni in x e y con due serie di Fourier, i cui rispettivi coefficienti sono:

$$x(t) \longrightarrow \begin{cases} a_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p}[cos\frac{2n\pi t_p}{T} - cos\frac{2n\pi t_{p-1}}{T}] \\ b_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta x_p}{\Delta t_p}[sin\frac{2n\pi t_p}{T} - sin\frac{2n\pi t_{p-1}}{T}] \end{cases} \tag{5.9}$$

$$y(t) \longrightarrow \begin{cases} a_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta y_p}{\Delta t_p}[cos\frac{2n\pi t_p}{T} - cos\frac{2n\pi t_{p-1}}{T}] \\ b_n = \frac{T}{2n^2\pi^2} \sum_{p=1}^{K} \frac{\Delta y_p}{\Delta t_p}[sin\frac{2n\pi t_p}{T} - sin\frac{2n\pi t_{p-1}}{T}] \end{cases} \tag{5.10}$$

K : numero della armonica utilizzata

n : ordine dei coefficienti armonici

T : perimetro

E infine l'ampiezza dell'n-esima armonica si calcola come:

$$amp_n = \frac{1}{2}\sqrt{a(x)_n^2 + b(x)_n^2 + a(y)_n^2 + b(y)_n^2} \tag{5.11}$$

5.2.3 Punti Medi

Una volta che il contorno è stato approssimato, vengono selezionati i punti chiave in entrambe le immagini. La posizione corrispondente dei punti nell'immagine che verrà creata sono calcolati come punti intermedi tra i corrispettivi nelle 2 immagini, come già evidenziato precedentemente nelle Formule (5.1),(5.2).

5.2.4 Triangolazione di Delaunay

La triangolazione di Delaunay [16] si applica a un insieme di punti P appartenenti allo stesso piano ed effettua una triangolazione in modo tale che nessun punto appartenente a P sia all'interno della circonferenza circoscritta di ogni triangolo. Questa triangolazione massimizza il minor angolo di tutti gli angoli dei triangoli, più semplicemente cerca di evitare i triangoli più stretti.

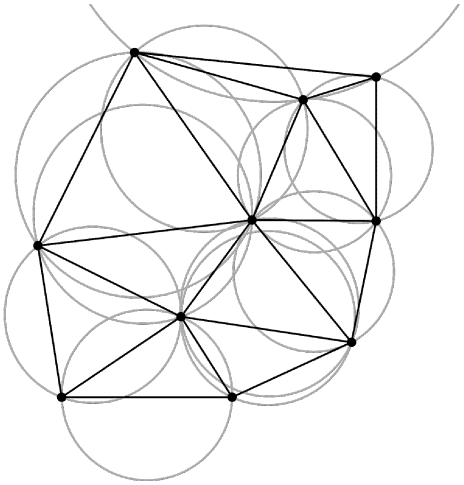


Figura 5.5: Triangolazione di Delaunay

Una volta calcolati i punti chiavi per le 3 immagini viene effettuata la triangolazione di Delaunay per ogni immagine

5.2.5 Morphing

Infine l'algoritmo iterando su tutti i pixel di C, determina il triangolo a cui appartiene il pixel e, applicando le trasformazioni affini per trovare il pixel corrispondente in A e B, determina il colore del pixel come mostrato nell'Equazione (5.3)

Capitolo 6

Fase Sperimentale

In quest'ultimo capitolo vediamo le modifiche apportate al codice originale del metodo proposto e i test che abbiamo effettuato per capirne l'efficacia. È stato modificato leggermente il codice python originale ed è stata implementata un'interfaccia MATLAB per rendere più user-friendly il metodo, il codice è disponibile su GitHub [19].

6.1 Interfaccia MATLAB

Sin dal primo momento abbiamo notato che il numero di immagini create era esageratamente grande, come si può notare nell'Equazione (5.4). La prima modifica apportata riguarda infatti la possibilità di scegliere il numero di immagini che si vogliono creare, più precisamente di quanto deve aumentare la dimensione del dataset, utilizzando il parametro *factor_increment*.

Un altro vincolo imposto originariamente è la creazione di 5 immagini per ogni coppia, ora è possibile scegliere implicitamente il numero di immagini che verranno create per ogni coppia modificando il parametro *alpha*.

6.2 ResNet50

La rete che abbiamo allenato è la ResNet50, una rete neurale convoluzionale a 50 livelli di profondità, pre-allenata con milioni di immagini dal database di ImageNet [18].

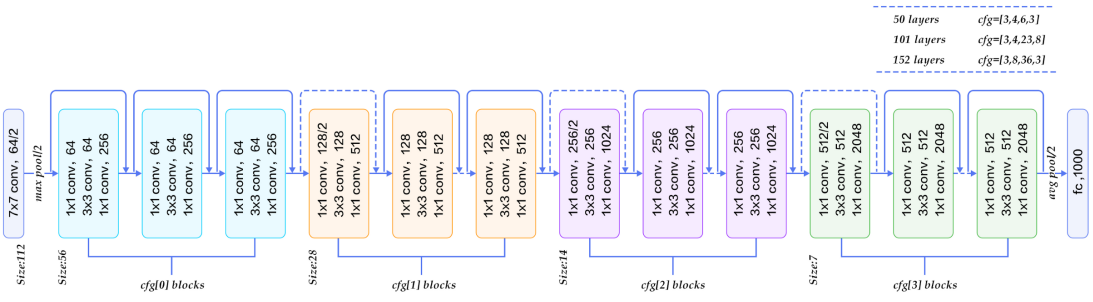
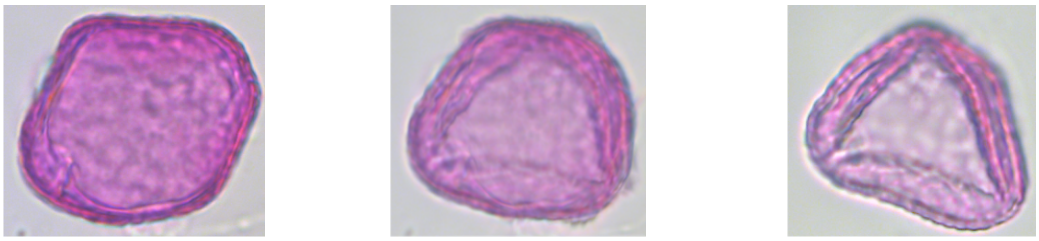


Figura 6.1: Architettura di ResNet50

6.3 Test e Risultati Sperimentali

Ogni test effettuato ha prodotto dei risultati in Accuracy e AUC, ci basiamo su queste metriche per valutare le performance ottenute ad ogni addestramento effettuato. Nel nostro caso valutiamo 1-AUC anzichè AUC, quindi a valore minore corrisponde prestazione migliore. Ogni test effettuato prevede l'aumento di immagini del dataset originale di un fattore 10x, inoltre tutte le immagini vengono create con il morphing delle due immagini originali con parametro $alpha = 0.5$, vedi Equazione (5.3).
Il primo dataset con il quale abbiamo testato il metodo conteneva immagini di granuli di pollini di varie specie al microscopio.



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

Figura 6.2: Morphing di granuli di acronomia aculeta

I risultati di questo primo training sono riportati nella tabella seguente:

Accuracy	1-AUC
0.9236	0.014298

Ci siamo successivamente spostati su un dataset contenente immagini di molte specie di farfalle, tuttavia qui abbiamo riscontrato i primi problemi. Se prima le immagini create sembravano veritiere, ora la maggior parte delle immagini create risultano tutt'altro che verosimili a esemplari di farfalle.

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

Figura 6.3: Morphing di esemplari di Farfalla Monarca

In questo caso l'addestramento non ha portato i risultati sperati, con questo dataset le performance crollano. Analizzando a fondo il metodo abbiamo riscontrato un comportamento insolito quando le immagini originali avevano contorni molto diversi tra loro; un esempio della differenza di comportamento si può notare nelle Figure seguenti, dove viene evidenziata la differenza dei contorni su cui lavora l'algoritmo nel caso dei Pollini e nel caso delle Farfalle.

(a) Maschera originale A **(b)** Maschera creata C **(c)** Maschera originale B

Figura 6.4: Maschere di granuli di acronomia aculeta

(a) Maschera originale A **(b)** Maschera creata C **(c)** Maschera originale B

Figura 6.5: Maschere di esemplari di Farfalla Monarca

Si può notare subito come il contorno su cui lavora(tutta la parte non nera dell'immagine) sia molto differente nei 2 casi. Per ovviare a questo problema la soluzione adottata consiste nel considerare il contorno della sola immagine completamente bianca, in modo tale che la maschera risultante sia la seguente:

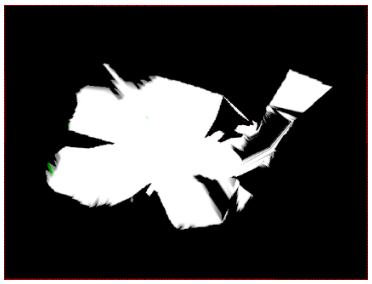


Figura 6.6: Maschera creata con modifica del metodo

Utilizzando questa nuova implementazione del metodo riusciamo a ottenere la seguente immagine(a destra):

(a) Morphing originale **(b)** Morphing dopo modifica

Figura 6.7: Morphing Originale e Modificato

Anche adottando questa soluzione le prestazioni con il dataset delle farfalle continuavano a crollare. Abbiamo quindi ripetuto il test con il dataset dei Pollini ma con la nuova implementazione del metodo. In questo caso abbiamo ottenuto un miglioramento per quanto riguarda l'AUC(1-AUC), a discapito dell'accuracy che è calata.

Accuracy	1-AUC
0.9191	0.0095571

Abbiamo quindi confrontato i risultati ottenuti con ensemble di ResNet50 che utilizza i metodi di Data Augmentation classici: crea 2 immagini con riflessione(verticale e laterale), un'immagine ottenuta scalando l'originale lungo entrambi gli assi, una ottenuta ruotando l'immagine, una traslandola lungo entrambi gli assi e infine una effettuando un taglio.

Accuracy	1-AUC
0.9393	0.0145

Si può notare subito come con ensemble l'accuracy sia migliore, ma i due morphing si comportano meglio per quanto riguarda l'AUC.

Infine come ultimo test abbiamo preso in considerazione l'ensemble descritto nel paper [17], il quale utilizza 14 metodi di Data Augmentation su 14 ResNet50 pre-addestrate con ImageNet, vedi Figura 6.8.

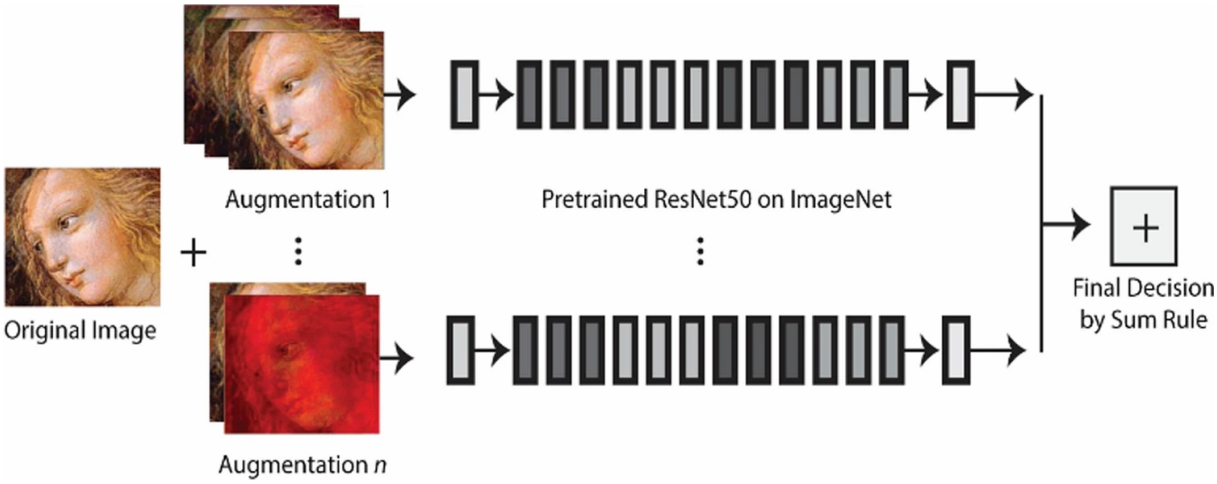


Figura 6.8: Ensemble di ResNet50

In seguito vediamo il confronto tra i risultati ottenuti da ensemble di partenza e dopo aver aggiunto il metodo di morphing tra i metodi di Augmentation utilizzati:

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

A differenza di tutti i casi precedenti, utilizzando il Morphing come metodo aggiunto siamo riusciti a migliorare le prestazioni dell'ensemble sia in Accuracy che in AUC.

Capitolo 7

Conclusioni

Riassumiamo in una tabella i risultati che abbiamo ottenuto con i nostri test:

	Accuracy	1-AUC
Morphing Originale	0.9236	0.014298
Morphing Modificato	0.9191	0.0095571
ResNet50 DA Classic	0.9393	0.0145
ResNet50 senza Morphing	0.9416	0.0122
ResNet50 con Morphing	0.9438	0.0096

Possiamo notare come il metodo se utilizzato come parte di un ensemble riesce a contribuire e migliorare le prestazioni della rete, se invece utilizzato autonomamente comporta la perdita di alcuni punti di accuracy. Dobbiamo tuttavia ricordare che il metodo non ha avuto risultati accettabili con il dataset di farfalle; se da un lato il metodo si è mostrato utile, dall'altro potrebbe far crollare le prestazioni se usato su dataset non adeguati. In conclusione possiamo affermare che nonostante le modifiche apportate, né il metodo originale né la modifica proposta sembrano produrre i risultati sperati con dataset le cui immagini hanno contorni molto diversi tra loro.

Il metodo si comporta abbastanza bene con pattern le cui shape sono molto simili e il modo in cui i pixel sono definiti è promettente, potrebbe essere interessante in futuro sperimentare nuove tecniche per la creazione di una figura che non stravolga la verosimilità del soggetto rappresentato, così facendo si potrebbe ottenere un risultato accettabile con qualsiasi tipo di dataset, indipendentemente dalla diversità o meno dei contorni.

Bibliografia

[1] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov (2014) *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, JMLR, vol. 15, no. 56

[2] Farhad Pourpanah, Moloud Abdar, Yuxuan Luo, Xinlei Zhou, Ran Wang, Chee Peng Lim, Xi-Zhao Wang, Q. M. Jonathan Wu (2022) *A Review of Generalized Zero-Shot Learning Methods*, IEEE in "Transactions on Pattern Analysis and Machine Intelligence", Early-Access

[3] Sergey Ioffe, Christian Szegedy *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, `https://static.googleusercontent.com/media/research.google.com/it//pubs/archive/43442.pdf`

[4] Noelia Vallez, Gloria Buenoa, Oscar Deniz, Saul Blanco (2022) *Diffeomorphic transforms for data augmentation of highly variable shape and texture objects*, ScienceDirect, vol. 219, no. 106775

[5] W. H. Pitts W. S. McCulloch (1943) *A Logical Calculus Of The Ideas Immanent In Nervous Activity*

[6] F. Rosenblatt (1957) *Perceptron–a perceiving and recognizing automaton*, Report 85-460-1, Cornell Aeronautical Laboratory

[7] Y. LeCun (1988) *A Theoretical Framework for Back-Propagation*, Proceeding sof the 1988 Connectionist Models Summer School, pages 21-28

[8] Sumit Saha (2018) *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, Towards Data Science

[9] Yamaguchi Kouichi, Sakamoto Kenji, Akabane Toshio, Fujimoto Yoshiji (1990) *A Neural Network for Speaker-Independent Isolated Word Recognition*, First International Conference on Spoken Language Processing (ICSLP 90). Kobe, Japan

[10] Brownlee Jason (2019) *A Gentle Introduction to the Rectified Linear Unit (ReLU)*, Machine Learning Mastery

[11] Cristiano Casadei (2018) *IA: Un po' di nozioni prima della pratica*, `https://www.developersmaggioli.it/blog/ia-un-po-di-nozioni-prima-della-pratica/`

[12] Fawcett Tom (2006) *An Introduction to ROC Analysis*, Pattern Recognition Letters. 27 (8): 861–874

[13] Fabio Henrique Kiyoiti dos Santos Tanaka, Claus Aranha (2019) *Data Augmentation Using GANs*, `https://arxiv.org/pdf/1904.09135.pdf`

[14] Emer Eric (2018) *Boosting (AdaBoost algorithm)*, MIT. Retrieved 2018-10-10

[15] F.P. Kuhl, C.R. Giardina (1982) *Elliptic Fourier features of a closed contour*, Comput.Graph. Image Process. 18, 236–258

[16] Boris Delaunay (1934) *Sur la sphère vide*, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk, 7:793-800

[17] Nanni L., Paci M., Brahnam S. (2022) *Feature transforms for image data augmentation*, Neural Comput & Applic, `https://doi.org/10.1007/s00521-022-07645-z`

[18] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei (2009) *ImageNet: A large-scale hierarchical image database*, IEEE Conference on Computer Vision and Pattern Recognition, pp. 248-255, doi: 10.1109/CVPR.2009.5206848

[19] Cocco A. (2022), *Morphing_DataAugmentation*, (Version 1.0) [Computer software], `https://github.com/Coccoexe/Morphing_DataAugmentation`