



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea

**ANALISI E RICONOSCIMENTO DEI MORFOTIPI DELLA
SPALLA TRAMITE APPROCCI BASATI SUL DEEP LEARNING**

Relatore: Chiar.mo Prof. Alberto Signoroni

Correlatori: Ing. Mattia Savardi

Dott.sa Maristella Francesca Saccomanno

Laureando:

Michele Signori

Matricola n. 723243

Indice

Introduzione	III
1 Background e Lavori Correlati	1
1.1 Struttura Anatomica della Spalla	1
1.2 Approcci alla Modellazione di Forma per la Spalla	3
1.3 Approcci Basati sul Deep Learning	5
2 Lettura delle Immagini e Segmentazione	7
2.1 Descrizione delle Immagini	8
2.1.1 Considerazioni sulle TC	9
2.2 Segmentazione	10
2.2.1 Lettura e Conversione delle Immagini	11
2.2.2 Resampling	12
2.2.3 Segmentazione	13
2.2.4 Correzione delle Label	16
2.3 Visualizzazione delle Labelmap e Considerazioni	16
2.3.1 Visualizzazione	17
2.3.2 Considerazioni sulla Segmentazione	20
2.4 Upload su XNAT	21
2.4.1 Conversione da NIFTI a RTSTRUCT	22
2.4.2 Upload su XNAT	22
3 Preprocessing e Generazione dei Dataset	24
3.1 Ricerca della Bounding Box	24
3.1.1 Divisione di Omeri e Scapole	25
3.1.2 Considerazioni Preliminari	26
3.1.3 Algoritmo di Ricerca	27

3.2	Cropping e Memorizzazione dei Dataset	32
3.2.1	Cropping e Padding	32
3.2.2	Memorizzazione dei Dataset	34
4	Convolutional Autoencoder ed Estrazione delle Feature	35
4.1	Gli Autoencoder	36
4.1.1	La Struttura Generale	36
4.1.2	I Convolutional Autoencoder	38
4.2	Il Modello di Estrazione delle Feature	40
4.2.1	Downsampling	40
4.2.2	Architettura del Modello	41
4.2.3	Parametri dell'Addestramento	48
4.3	Analisi dei Risultati: Addestramento e Predizioni	51
4.3.1	Modello degli Omeri	52
4.3.2	Modello delle Scapole	58
4.4	Estrazione delle Features e Visualizzazione	61
4.4.1	Estrazione delle Features	62
4.4.2	Visualizzazione delle Features	63
5	Clustering e Classificazione	69
5.1	Il Clustering	70
5.1.1	Descrizione Generale	70
5.1.2	Clustering sul Dataset Originale	72
5.1.3	Suddivisione delle Spalle e Clustering sul Dataset Modificato	76
5.2	Tool di Visualizzazione	79
5.2.1	Grafico interattivo	79
5.2.2	Rendering 3D	82
5.3	Analisi dei Risultati	84
5.3.1	Metriche sui Cluster Finali	84
5.3.2	Considerazioni sui Rendering delle Scapole	90
6	Conclusioni e Sviluppi Futuri	95
Bibliografia		99

Introduzione

Il seguente lavoro di tesi tratta l'analisi e il riconoscimento dei morfotipi della spalla, cioè l'individuazione dell'esistenza di eventuali tipologie standard di spalle date da una particolare similarità di conformazione delle ossa che le compongono. Questa esigenza proviene direttamente dal settore di ricerca in ambito ortopedico, nel quale ci si pone la domanda dell'effettiva esistenza di questi morfotipi, in quanto una classificazione ufficiale ancora non esiste. Oltre al riconoscimento dei morfotipi, è importante anche l'eventuale collegamento che questi ultimi potrebbero avere con alcuni pattern degenerativi che portano a malformazioni della struttura ossea della spalla, con probabile necessità di un'operazione. L'esistenza dello stesso risulterebbe essere un profondo aiuto per gli ortopedici durante l'esecuzione delle operazioni in quanto essi sarebbero in grado di personalizzare e in parte automatizzare il processo dell'individuazione del corretto tipo di operazione da effettuare al paziente in base alla sua tipologia di spalla. Il seguente lavoro, però, non si occuperà dell'intera ricerca, cioè sia del riconoscimento dei morfotipi che dell'eventuale collegamento di questi ultimi con pattern degenerativi, ma tratterà solamente la prima parte della stessa. In particolare, l'obiettivo sarà quello di realizzare dei tool che permetteranno ai medici di consultare il risultato della classificazione ottenuta in modo rapido e intuitivo, permettendo loro di fornire delle opinioni sulla qualità della stessa ed eventualmente correggendo alcuni aspetti. Questo avverrà tramite l'analisi di circa 1200 TC (Tomografia Computerizzata) di spalle patologiche fornite dall'azienda svizzera Medacta [19]. La disponibilità di questo tipo di dati ha quindi permesso lo svolgimento della fase di ricerca dei morfotipi basati sulla conformazione di spalle patologiche. L'eventuale collegamento di questi ultimi con i rispettivi morfotipi in spalle sane sarà possibile solo tramite la disponibilità di dati sulle stesse.

La scelta dell'utilizzo di metodi basati sul Deep Learning, invece, è data dalle potenzialità che questo tipo di approcci attualmente possiede e che deriva dall'evoluzione che le tecniche di

questo tipo hanno subito nel corso degli ultimi anni. Con l'aumento esponenziale della capacità computazionale dei calcolatori, ora è possibile addestrare modelli anche molto complessi e computazionalmente onerosi in termini di risorse in tempi decisamente ristretti. Il progetto è stato possibile grazie all'utilizzo di una macchina messa a disposizione dall'Università avente una capacità computazionale molto elevata che ha permesso di gestire l'addestramento dei modelli necessari allo svolgimento del lavoro, così come l'utilizzo di tool per il processing di immagini 3D. L'intelligenza artificiale, in particolare il Deep Learning, è un argomento molto in voga negli ultimi anni, proprio per merito della grande evoluzione che esso ha subito. Grazie a questo, esistono tantissimi studi sull'applicazione di questi modelli in ambiti più disparati, compreso quello medicale di interesse per questo lavoro. Le tipologie di modelli complessi attualmente più utilizzate si dividono essenzialmente in due categorie chiamate reti ricorrenti e reti convoluzionali. La prima è utilizzata prevalentemente nell'ambito del Natural Language Processing (NLP), cioè l'elaborazione del linguaggio naturale, mentre la seconda è molto usata per l'elaborazione di audio, immagini e video. È proprio quest'ultima, infatti, quella che sarà utilizzata per l'elaborazione delle immagini 3D provenienti dalle TC nel corso del progetto.

Nei prossimi capitoli verranno mostrate ed analizzate nel dettaglio tutte le fasi che hanno portato alla realizzazione del progetto e al risultato finale. Iniziando da un primo capitolo introduttivo in cui verrà esposto il problema in modo più dettagliato, descrivendo anche i lavori correlati presenti in letteratura e i loro collegamenti con il problema in esame, i successivi capitoli mostreranno il progetto vero e proprio. Il secondo capitolo tratta infatti la prima fase di importazione delle immagini dalle TC in un ambiente Python e la segmentazione delle ossa necessaria per gli step seguenti, basata su un sistema di Deep Learning preaddestrato. Il terzo capitolo tratterà il lavoro compiuto sulle maschere di segmentazione, attuato con lo scopo di modificare i dati ottenuti e creare un dataset che sia adatto ad essere processato dal modello di Deep Learning utilizzato, descritto approfonditamente nel quarto capitolo. Nel quinto capitolo, verrà riportata la fase finale di clustering e classificazione dei risultati forniti dal modello, così come le considerazioni finali sull'output ottenuto. Nell'ultimo capitolo, saranno infine riportate le conclusioni e gli sviluppi futuri di questo lavoro.

Tutto il codice relativo al progetto è disponibile al seguente link: https://github.com/mik0699/shoulder_morphotypes_analysis/tree/main

Capitolo 1

Background e Lavori Correlati

Questo primo capitolo contiene tutta la parte preliminare che è stata svolta antecedentemente alla fase pratica e che riguarda lo studio dell'argomento specifico tramite l'analisi di articoli presenti in letteratura per definire la struttura del lavoro. Questa fase ha permesso infatti di analizzare problemi simili a quello posto e di ottenere in primo luogo una visione più generale del problema. Ha inoltre permesso di apprendere alcune nozioni più specifiche nell'ambito medico necessarie alla comprensione del problema assegnato. Inoltre, precedentemente alla fase del progetto che riguarda l'estrazione delle feature, è stata svolta un'altra sessione di ricerca per l'individuazione del metodo più opportuno da utilizzare in questo caso e che verrà anch'essa riportata di seguito.

1.1 Struttura Anatomica della Spalla

Prima di descrivere gli studi analizzati, è necessaria una breve introduzione riguardante le parti d'interesse dell'anatomia della spalla. L'articolazione della spalla è un complesso meccanismo costituito da numerosi elementi ossei, muscolari e legamentosi. Essa permette la più ampia gamma di movimento del corpo umano, ma è molto comune che subisca lesioni o condizioni patologiche di diverso tipo. Le ossa che compongono la spalla sono tre: la clavicola, l'omero e la scapola. In questo lavoro, così come in tutti quelli annessi, sono considerate però solamente due di esse: l'omero e la scapola. Esse interagiscono tra di loro durante il movimento e sono le parti che compongono l'articolazione scapolo-omerale, che rappresenta la parte della spalla che

origina più problemi. Per questo motivo, di seguito sarà in breve presentata la struttura di queste due ossa e le parti che le compongono.

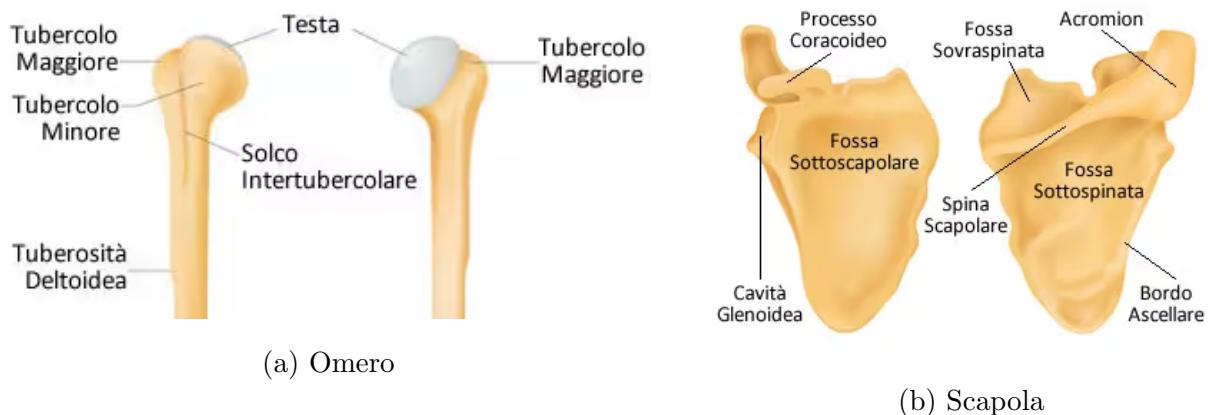


Figura 1.1: Anatomia di omero e scapola ¹

La figura 1.1 mostra le parti principali di omero e scapola. L'omero costituisce lo scheletro del braccio ed è protagonista di due articolazioni: l'articolazione scapolo-omerale e l'articolazione del gomito. È formato da parti differenti, che è possibile vedere in figura 1.1a: la più interessante per il progetto in questione è la testa dell'omero, in quanto essa si inserisce perfettamente all'interno della cavità glenoidea della scapola e permette di dare vita all'articolazione scapolo-omerale. La scapola, invece, rappresenta l'osso più importante della spalla, in quanto interagisce con entrambe le altre ossa della stessa. Inoltre, essa è sede di aggancio per la maggior parte dei muscoli della spalla. Ha una forma più complessa di quella dell'omero, ed è composta da numerose parti. Verranno di seguito analizzate solamente le principali e più importanti per il lavoro in questione:

- Cavità glenoidea (o glena): è un incavo a forma sferica destinato a ospitare la testa dell'omero, per formare la già citata articolazione scapolo-omerale
- Acromion: è una porzione ossea piuttosto evidente e facilmente riconoscibile, destinata ad interagire con la clavicola per formare l'articolazione acromioclavicolare. Ha inoltre il compito di agganciare un'estremità del legamento coraco-acromiale
- Processo coracoideo: proiezione ossea a forma di uncino che rappresenta l'altro aggancio del legamento sopracitato. Inoltre, è origine di numerosi muscoli e altri legamenti.
- Fossa sovraspinata (o fossa del sovraspinato): è l'area liscia che si estende sopra alla spina scapolare. Rappresenta la sede di aggancio del muscolo sovraspinato.

¹ Immagini tratte da: <https://www.my-personaltrainer.it/salute-benessere/ossa-spalla.html>

Le altre parti che compongono la spalla, che non sono state descritte, sono individuabili tramite la figura 1.1b.

1.2 Approcci alla Modellazione di Forma per la Spalla

Come già evidenziato, il problema posto è molteplice. In primo luogo vi è l'identificazione dei morfotipi delle spalle e successivamente la valutazione di un'eventuale associazione tra i morfotipi trovati e dei pattern degenerativi che porterebbero a un'alterazione del morfotipo stesso consistente per spalle all'interno della stessa classe di partenza. Come già affermato, però, nello specifico verrà qui trattata solamente la prima parte. Il problema sostanziale, infatti, consiste nell'inesistenza in letteratura di una suddivisione ufficiale e provata delle spalle in morfotipi. Esistono però altre classificazioni legate a parti specifiche della spalla ed in particolare della scapola.

A questo proposito, uno studio [22] ha evidenziato l'importanza della forma della glena e la possibilità di pianificare il posizionamento di un impianto per la spalla grazie alla sua analisi ancor prima dell'operazione. In questo lavoro viene utilizzato un SSM (Statistical Shape Model) per la ricostruzione di un difetto della glena e per la predizione di alcune informazioni rilevanti sulla glena nativa. Un SSM è una rappresentazione matematica di una forma anatomica o di un oggetto basata su dati statistici raccolti da un insieme di campioni. Questi modelli vengono utilizzati principalmente per analizzare e descrivere la variabilità delle forme in un insieme di dati. In questo specifico caso, un modello di questo tipo è stato costruito partendo da 66 scapole sane e, applicando delle deformazioni artificiali a queste scapole, è stato esaminato in che misura questo modello risultasse preciso nel ricostruire la corretta forma della glena originale. La comparazione tra alcune misure sulla glena originale e su quella ricostruita dopo l'inserimento del difetto ha permesso di ottenere dei buoni risultati di ricostruzione, ottenendo quindi un modello in grado di ricostruire con una buona precisione la corretta superficie della glena. Gli SSM sono molto utilizzati grazie alla loro semplicità e alla loro potenza, infatti molti altri lavori hanno sperimentato l'utilizzo di questi ultimi per studi simili al precedente. In un altro studio infatti, gli SSM sono stati nuovamente utilizzati per riuscire a comprendere il collegamento tra variazioni morfologiche di alcune parti della scapola e le patologie della spalla derivanti [2]. Le parti della scapola considerate in questo caso sono però differenti rispetto a

quelle dell'articolo precedente, in quanto in quest'ultimo vengono considerate l'inclinazione e la forma dell'acromion in associazione ad alcuni tipi di patologie della spalla. Entrambi questi studi analizzano quindi alcune parti della scapola e cercano di trovare delle correlazioni tra certe caratteristiche di queste parti e determinati tipi di patologie. Da notare che le parti considerate sono differenti, segno che quindi esistono opinioni disparate in merito a quale parte dell'osso sia più significativa per lo sviluppo di determinati tipi di patologie. L'obiettivo di questi studi non è quindi esattamente lo stesso di quello della presente tesi, ma è opportuno considerare questi ultimi come casi particolari che aiutano a capire la direzione da intraprendere. Ciò che è emerso consiste quindi nel fatto che probabilmente particolari caratteristiche morfologiche di alcune parti della scapola possano portare a patologie correlate. Per questo motivo nel seguente lavoro saranno considerate le scapole nella loro interezza, proprio per cercare di rendere il più generale possibile questo studio.

È opportuno notare che entrambi questi studi considerano però solamente la scapola, mentre uno studio più recente [28] ha invece considerato anche alcune caratteristiche della relazione tra l'omero e la scapola, in particolare la HHM (Humeral Head Migration), cioè la posizione della testa dell'omero rispetto a determinate parti della scapola in spalle patologiche con artrite. L'idea è anche in questo caso quella di cercare una possibile associazione tra questa misura di HHM nelle spalle patologiche e la morfologia scapolare. I pazienti con artrite sono stati suddivisi in gruppi in base al tipo di HHM che possedevano e, nuovamente tramite l'utilizzo di SSM, è stato possibile trovare delle correlazioni tra questa variabile e la conformazione della scapola. In particolare è stato trovato che una caratteristica in particolare, chiamata versione della glena, è risultata essere un buon metodo per predire lo sviluppo di HHM. Questo studio ha aiutato nella scelta di prendere in considerazione anche gli omeri in quanto alcune patologie, come l'artrite, derivano essenzialmente dall'interazione tra l'omero e la scapola ed è quindi importante analizzare entrambi.

Tutti questi studi utilizzano gli SSM come metodi per la costruzione di modelli, ma nessuno di essi ha come obiettivo finale una vera e propria classificazione, anche se comunque delle classificazioni specifiche riguardanti alcune parti della scapola scaturiscono. Per la scelta del metodo di classificazione utilizzato ha però aiutato uno studio corrispondente che tratta la presenza di morfotipi nel ginocchio [12]. Questo lavoro utilizza delle feature per la descrizione della forma delle ossa basate su misure morfologiche e altre sui pesi di un SSM. Queste feature vengono poi fornite in input a due algoritmi di clustering differenti che sono infine valutati

tramite alcune metriche. Seppur il risultato finale abbia mostrato che i dati non esibissero alcun morfotipo, questo studio è stato utile in quanto si è posto lo stesso problema trattato nel presente lavoro e ha quindi permesso di ottenere parecchi spunti. Per quanto riguarda l'estrazione delle feature, è stato deciso di provare un approccio più complesso e più moderno, basato sul Deep Learning, ma l'idea del clustering è invece stata mantenuta e utilizzata.

Riassumendo, questi paper hanno aiutato la fase preparatoria del progetto nella quale è stato necessario informarsi sull'argomento in generale e nello specifico su articoli che trattano argomenti simili, per analizzare i risultati da essi ottenuti e valutare l'approccio giusto da scegliere. Per quanto riguarda la scelta dell'Autoencoder come modello di estrazione delle feature, verranno ora esposte alcune considerazioni riguardanti articoli in merito.

1.3 Approcci Basati sul Deep Learning

Per quanto riguarda il metodo utilizzato per la parte di estrazione delle feature dalle immagini da dare poi in input a un algoritmo di clustering, è stato deciso di utilizzare degli approcci completamente non supervisionati, cioè approcci nei quali il modello non possiede alcuna informazione aggiuntiva ma lavora solamente utilizzando i dati stessi. Esistono molti studi in merito, che utilizzano modelli differenti per lo stesso scopo. Uno di essi, per esempio, utilizza una rete siamese convoluzionale per estrarre delle feature dalle immagini [26]. L'idea di questo approccio si basa sulla similarità tra immagini. La rete siamese infatti è in grado di ricevere in ingresso coppie di immagini e confrontarle attribuendo a ognuna di esse una misura di similarità e adattando i pesi del modello in base a questa misura. La rete in questo modo riesce a imparare e posizionare le immagini appartenenti alla stessa classe, cioè simili tra loro, vicine le une alle altre mentre invece quelle appartenenti a classi diverse lontane tra di loro. In questo modo si ottiene un risultato utile per suddividere in classi i dati. Questo approccio è sicuramente interessante, ma in questo progetto è stato preferito adottare un metodo più consolidato, basato sui Convolutional Autoencoder, così come è stato fatto in [3]. Questo articolo propone un modello di Convolutional Autoencoder per l'estrazione di feature da immagini di polmoni tramite l'utilizzo di dati senza label, con solamente un piccolo insieme di essi che possiede invece delle informazioni. La differenza tra il modello utilizzato in questo articolo e l'approccio adottato nel presente progetto sta nel fatto che nel primo sono stati utilizzati anche

alcuni dati con label e l'apprendimento è stato parzialmente supervisionato, seppur possedendo la maggior parte dei dati sprovvisti di label. Le conclusioni in [3] affermano che tra gli approcci provati, quello quasi totalmente non supervisionato, chiamato anche data-driven, ha ottenuto i risultati migliori rispetto ad altri che, per esempio, utilizzavano feature create ad hoc. Questa conclusione ha fornito un aiuto per quanto riguarda la scelta del corretto approccio da utilizzare per questa tesi, che però risulterà essere completamente non supervisionato, non avendo noi a disposizione alcuna label.

Infine, un ultimo approccio che è stato preso in considerazione è quello del Deep Clustering. Questi metodi utilizzano delle reti di tipo convoluzionale [1] o dei convolutional autoencoder [3] per l'apprendimento delle feature, mentre in contemporanea queste ultime vengono anche date in ingresso ad un algoritmo di clustering che produce delle label per questi dati. Le label così prodotte, insieme al modello convoluzionale adottato, vengono utilizzate per l'addestramento del modello completo. In questo modo l'estrazione delle feature e il clustering non rappresentano più due fasi separate, ma vengono unite in un unico modello, facendo sì che anche i risultati del clustering possano entrare a far parte dell'addestramento del modello stesso. Questi tipi di approcci sono molto interessanti e adatti al tipo di problema preso in considerazione nel presente studio, ma si basano su immagini bidimensionali. Avendo a disposizione delle immagini 3D, la complessità di adattamento di questi tipi di modelli al caso particolare tridimensionale non è stata ritenuta per il momento necessaria e si è preferito mantenere le parti di Autoencoder e di Clustering separate. Questi tipi di metodi sono però stati molto utili per apprendere il funzionamento delle due parti, seppur in modo separato e non congiunto come invece avviene nel Deep Clustering.

Dopo aver letto e analizzato tutti questi articoli che trattano problemi correlati a quello in esame, è stato deciso l'approccio da utilizzare per il seguente lavoro, considerando una serie di fattori tra cui il formato dei dati ed eventuali limitazioni tecniche. Nei prossimi capitoli saranno descritte nel dettaglio tutte le fasi del progetto, provviste di esempi e spiegazioni su ogni argomento trattato.

Capitolo 2

Lettura delle Immagini e Segmentazione

Questo capitolo riguarda la prima parte del progetto vero e proprio. Come primo step, è stato necessario apprendere le modalità di importazione delle TC, che sono memorizzate in DICOM [20], un formato standard specificatamente utilizzato per memorizzare dati medici. L'importazione è stata gestita utilizzando il linguaggio di programmazione Python. Una volta importate le immagini, è possibile passare alla prima fase vera e propria che consiste nella segmentazione. Prima però, è stata necessaria una parte di resampling delle immagini in quanto esse sono risultate avere dimensioni variabili dei voxel che le compongono. Una volta attuato il resampling, le TC verranno prese una alla volta e date in input ad un tool, chiamato TotalSegmentator [29], contenente una rete convoluzionale preaddestrata che permette di riconoscere le scapole e gli omeri partendo da immagini in un formato particolare chiamato NIFTI. Verranno in seguito riportati esempi e figure che mostrano alcune delle segmentazioni ottenute. Infine, verrà descritta una fase aggiuntiva che ha coinvolto l'upload delle maschere di segmentazione sulla piattaforma informatica di imaging chiamata XNAT [21] in modo da renderle disponibili agli ortopedici che hanno collaborato al progetto.

2.1 Descrizione delle Immagini

Come primo step, è necessario descrivere il formato DICOM (Digital Imaging and Communications in Medicine) delle immagini mediche che è stato utilizzato per la memorizzazione delle TC a cui sono stati sottoposti i pazienti. La tomografia computerizzata è una tecnica che consente di esaminare parti diverse del corpo. In questo caso specifico è stata utilizzata per le ossa che compongono la spalla. Tramite l'utilizzo di raggi X, che vengono trasmessi attraverso il corpo e registrati da un rilevatore, essa è in grado di produrre immagini che consentono di visualizzare tutti i dettagli anatomici. In base alla quantità di raggi che attraversa le varie parti del corpo, è possibile ricostruire diversi tipi di tessuti, tra cui le ossa. Le immagini ottenute tramite questa metodologia sono tipicamente in formato DICOM [20]. Questo formato è uno standard per la memorizzazione di immagini biomediche. I file DICOM contengono non solo l'immagine in sé, ma anche una serie di informazioni che riguardano la diagnosi e i dati del paziente, chiamati metadati. Questi metadati forniscono dettagli cruciali, come il nome del paziente, la data e l'ora dello studio, i parametri di acquisizione dell'immagine, la modalità di acquisizione (in questo caso TC), e molte altre informazioni cliniche. La parte riguardante l'immagine è invece suddivisa in slice, ognuna delle quali consiste in un'immagine bidimensionale che mostra i dettagli anatomici del paziente. Unendo le varie slice è possibile ottenere un rendering in 3D della parte interessata.

Per la visualizzazione di questo tipo di immagini esistono degli appositi tool. Per questo progetto è stato inizialmente utilizzato un visualizzatore chiamato OHIF Viewer [9], integrato in un portale chiamato XNAT [21], che consente la rapida condivisione e gestione delle immagini. Tramite questo tool è possibile visualizzare le TC in modalità differenti, compreso il rendering 3D della parte interessata.

L'immagine 2.1 rappresenta un esempio visto in OHIF di una slice di un file DICOM proveniente dalla TC ricavata dalla spalla di un paziente, nella quale si riescono a riconoscere abbastanza facilmente i profili dell'omero e della scapola, che sono stati indicati tramite una freccia. Questo tool consente però la sola visualizzazione di file di questo tipo. Per compiere elaborazioni sugli stessi è invece necessario importarli in un linguaggio di programmazione come Python.

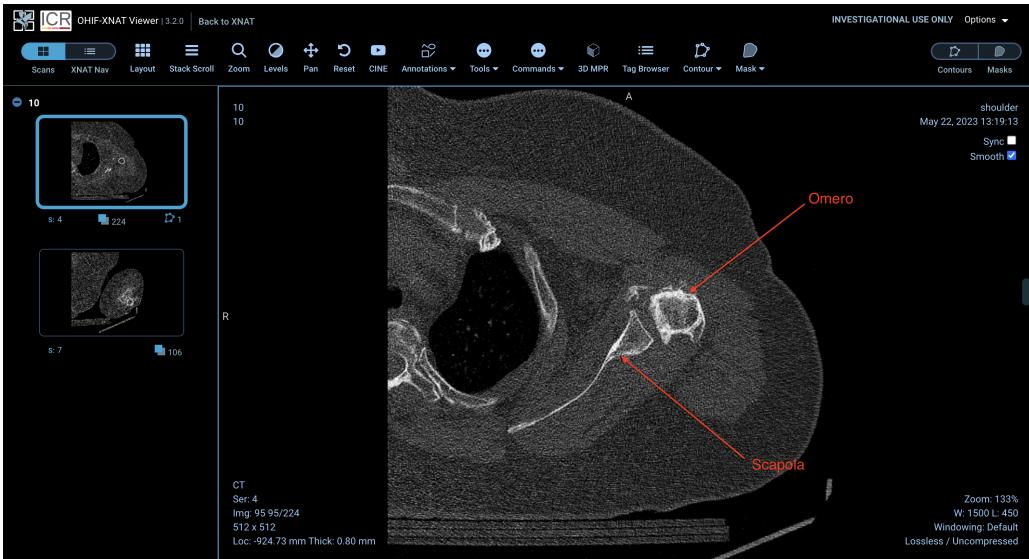


Figura 2.1: Una schermata di OHIF Viewer in XNAT rappresentante una slice dell’immagine DICOM di un paziente

2.1.1 Considerazioni sulle TC

Idealmente, le 1250 TC di spalle e gomiti dovrebbero essere tutte uguali in termini di formato e dimensione, ma in realtà non è così. Infatti, molte TC presentano caratteristiche differenti le une dalle altre che è opportuno evidenziare e che sono emerse proprio durante questa prima fase. La differenza principale tra le varie TC sta nel fatto che per molti pazienti esse risultano essere suddivise in due differenti blocchi, uno rappresentante la spalla e l’altro il gomito, mentre per altri le informazioni sul gomito non sono presenti, ma esistono solamente quelle riguardanti la spalla. Il target di queste TC è effettivamente la spalla, ma il gomito solitamente viene comunque registrato perché tramite esso è possibile risalire ad alcune informazioni utili riguardo l’orientamento e il posizionamento dell’omero.

Un’ulteriore differenza risiede nella dimensione dei voxel delle immagini registrate. Infatti, ogni TC possiede, memorizzata nei metadati, l’informazione sull’effettiva dimensione in millimetri di ogni voxel. Idealmente, quest’ultima dovrebbe essere uguale in tutte le TC, ma in questo caso non è così. Infatti, sia nelle spalle che nei gomiti, la dimensione dei voxel tra una TC e l’altra risulta essere differente. Ciò non è un problema nelle fasi iniziali, in quanto lavorando con file di tipo medico, essi possiedono l’informazione sulla dimensione e si adattano di conseguenza. Il problema risulterà essere nelle fasi successive nelle quali si considereranno solamente i puri dati numerici delle immagini di questi file, in quanto un modello di Deep Learning non supporta dati

in formato medico. Se non opportunamente gestita, questa differenza di dimensione porterà ad immagini deformate. Per la sua gestione è stata quindi necessaria una fase, preliminare alla segmentazione, di resampling delle immagini, che sarà descritta successivamente. Per quanto riguarda le differenze tra spalle e gomiti, ciò che cambia maggiormente è la dimensione di ogni slice, cioè la dimensione lungo l'asse z del voxel, che nei gomiti è in media piuttosto superiore. Essa è direttamente collegata alla qualità della TC: solitamente TC con slice più spesse avranno una qualità inferiore in quanto l'informazione memorizzata da ogni slice è più grossolana. Ciò non determina però un grande problema in quanto la spalla rappresenta la parte importante da monitorare, mentre il gomito fornisce un'informazione accessoria e non è quindi necessario avere una qualità come quell'altra parte del corpo. Diretta conseguenza della dimensione elevata delle slice è il loro basso numero. Infatti, il media le TC dei gomiti possiedono meno slice rispetto a quelle delle spalle.

Infine, un'ultima differenza che influisce più che altro sul piano della visualizzazione, sta nel fatto che in alcune TC le slice della spalla e del gomito non sono contigue e quindi allineandole risulta esserci un buco all'interno dell'omero. Altre, invece, in minoranza rispetto alle prime, possiedono delle slice contigue tra di loro che permettono una visualizzazione totale dell'osso. Ciò non è un vero problema in quanto, come sarà anche evidenziato successivamente, le parti di omero interessanti sono la testa ed eventualmente la parte finale, mentre il centro dell'osso non contiene spesso informazioni utili. Per quanto riguarda la parte contenente le spalle nelle varie TC, essa è invece molto più uniforme e la qualità delle immagini è sempre molto più alta.

Tutte queste considerazioni contribuiscono all'elevata eterogeneità dei dati, che sarà fondamentale gestire. Infatti, avendo una quantità così elevata di dati tridimensionali, l'esecuzione di un semplice programma può impiegare anche parecchie ore ed è quindi di fondamentale importanza evitare eccezioni che possano interrompere l'esecuzione dello stesso o portare a risultati incoerenti.

2.2 Segmentazione

Verrà ora descritta la parte centrale di questo capitolo che riguarda la segmentazione delle ossa della spalla e del gomito tramite il tool chiamato TotalSegmentator [29]. Per lavorare su queste immagini è stato scelto di utilizzare il linguaggio di programmazione Python, grazie

alla sua estrema versatilità e alla presenza di moltissime librerie differenti che gli permettono di adattarsi a qualunque compito. Inoltre, esso rappresenta la scelta predominante per lo sviluppo di modelli di Deep Learning, grazie al supporto della comunità e la documentazione disponibile per le principali librerie adatte a questo scopo, come Tensorflow. Il codice è stato interamente scritto grazie alla piattaforma Jupyterlab, che permette lo sviluppo di notebook Python su una macchina remota, in questo caso un server Nvidia DGX dotato di 8 GPU A100 da 80GB che è stato possibile utilizzare per questo lavoro. Prima di passare alla segmentazione vera e propria, è opportuno descrivere alcune fasi preliminari necessarie alla stessa.

2.2.1 Lettura e Conversione delle Immagini

Il primo step necessario consiste nella lettura delle immagini, cioè l'importazione dei file DICOM delle TC in un ambiente di programmazione come Python. Come appena affermato, esistono moltissime librerie di Python che lo rendono molto versatile, una delle quali è chiamata PyDicom. Come anche il nome suggerisce, essa permette la lettura, la scrittura e la manipolazione di immagini mediche nel formato DICOM. Esso rappresenta un formato piuttosto complesso, in quanto è composto da metadati collegati alle immagini vere e proprie. Per questo motivo la maggior parte dei tool, soprattutto quelli di elaborazione delle immagini o quelli basati su modelli di Deep Learning, come TotalSegmentator, preferisce immagini in formato NIFTI. Esso infatti è un formato molto più facilmente gestibile in quanto memorizza tutta l'informazione in unico file, dotato di un semplice header che permette di mantenere i metadati essenziali del formato DICOM. Ciò permette un processing e una lettura più rapidi dei dati contenuti all'interno. Il formato NIFTI (Neuroimaging Informatics Technology Initiative) è uno standard utilizzato originariamente per l'archiviazione e la condivisione di dati di neuroimaging, ma adesso adottato anche per immagini mediche in diversi ambiti. Un ulteriore vantaggio di questo tipo di formato è la facile integrazione con le principali librerie di Python usate per la gestione di dati multidimensionali, in particolare Numpy, che è una libreria fondamentale per il calcolo scientifico e che fornisce supporto per array multidimensionali e funzioni matematiche avanzate per operare su questi array in modo efficiente. Il primo passo è quindi consistito nell'effettuare la conversione delle immagini da DICOM a NIFTI in modo che possano essere date in input al tool di segmentazione. Per fare ciò, un'altra libreria Python chiamata dicom2nifti è stata necessaria. Essa contiene una funzione che, semplicemente ricevendo in ingresso la lista delle slice di un file DICOM, riesce a convertirle in un unico file NIFTI che conterrà i metadati

più importanti del file DICOM. Alcuni di essi infatti andranno persi, come le informazioni sul paziente, le informazioni sullo studio (data, ora) o altre informazioni poco rilevanti tipiche del formato DICOM. Verranno però mantenute tutte le informazioni importanti che riguardano l'immagine stessa, come ad esempio la dimensione reale dei voxel. Applicando questa funzione a tutte le TC di spalle e gomiti, si otterrà un file NIFTI per ognuna di esse, che sarà utilizzato per la successiva fase di resampling, prima di essere dato in input al segmentatore.

2.2.2 Resampling

Questa fase è risultata necessaria a seguito dell'analisi delle TC descritta nella sezione 2.1.1. L'obiettivo di questa fase è quello di ottenere dei volumi che contengano voxel tutti della stessa dimensione. In questo modo, quando si estrarranno le immagini dai file NIFTI contenenti le segmentazioni, ogni valore rappresenterà un voxel la cui dimensione sarà costante in tutto il volume, conservando così lo stesso rapporto d'aspetto nella rappresentazione delle strutture anatomiche in tutte le TC. Inoltre, è stato deciso di generare volumi isotropi, cioè volumi nei quali le dimensioni lungo x,y e z del voxel siano le medesime. Per far ciò, è stato necessario determinare le dimensioni desiderate di uscita dei voxel e poi applicare l'effettiva operazione di resampling a tutte le TC per ottenere il file NIFTI desiderato. Questa operazione è stata effettuata solamente sulle spalle, in quanto, come già affermato, i gomiti presentano spesso immagini grossolane dell'osso e sono utilizzati solamente per ottenere informazioni generali quali l'orientamento o la direzione dello stesso.

Per le spalle, invece, la situazione è differente, in quanto interessa ottenere una morfologia dell'osso che rispecchi fedelmente quella originale. Dopo una prima analisi delle dimensioni dei voxel nelle varie TC delle spalle, è emerso che lungo x e y la loro dimensione varia tra 0.2mm e 0.97mm mentre lungo z il range risulta superiore, con valori che oscillano tra 0.2mm e 2.5mm. Questo comportamento è però totalmente attendibile, in quanto il numero di slice è variabile da una TC e l'altra, mentre le dimensioni di ogni slice sono sempre piuttosto costanti, con solamente poche eccezioni. È stato deciso, vista l'ampia diversità, di escludere le TC corrispondenti ai casi limite, tenendo solamente quelle con valori intorno a 0.5mm nelle tre direzioni, che rappresenta il valore di dimensione scelto dopo il resampling. Per quanto riguarda x e y, sono state mantenute le TC con dimensione dei voxel compresa tra 0.35mm e 0.65mm, mentre lungo z quelle con dimensione tra 0.3mm e 1mm. In questo modo, vengono escluse 106

TC, che sono un numero accettabile. Questa esclusione è stata effettuata in quanto per eseguire il resampling è necessaria un'interpolazione delle immagini, che comporta una variazione di dimensione delle stesse. L'interpolazione è una tecnica che consente di utilizzare i valori dei voxel conosciuti per stimare i valori di altri voxel necessari al resampling dell'immagine. Infatti, il numero di voxel dopo questa fase varierà rispetto a quello originale, in quanto modificando la dimensione degli stessi cambierà di conseguenza la quantità di informazione portata da ognuno di essi. Mantenendo anche le immagini con valori estremi, si sarebbero ottenute immagini in uscita troppo grandi o troppo piccole, mentre estraendo solamente quelle con dimensione vicina alla dimensione scelta è stata ridotta questa variabilità.

Una volta determinata la dimensione di uscita dei voxel, è possibile eseguire il resampling su ognuna delle immagini non escluse ed ottenere in uscita immagini NIFTI dotate di volumi isotropi e voxel tutti della stessa dimensione. Il resampling effettuato utilizza un'interpolazione spline di grado 2, chiamata anche quadratica, che permette di approssimare i valori dei voxel sconosciuti tramite un polinomio di secondo grado. Queste immagini saranno infine date in input al tool di segmentazione.

2.2.3 Segmentazione

La fase centrale di questo capitolo si occupa dell'effettiva segmentazione delle ossa. L'obiettivo è quello di ottenere delle maschere di segmentazione ognuna delle quali contiene informazioni sulle ossa riconosciute e sulla loro posizione nella TC originale. L'output del segmentatore sarà una labelmap contenente, per ogni voxel in cui è stato riconosciuto una certa struttura anatomica, un valore intero corrispondente alla label di quella struttura. Tramite queste label è poi possibile identificare quale osso tra quelli di interesse è stato individuato e isolarlo nelle immagini originali, sovrapponendo queste maschere di segmentazione alle TC e selezionando una o più label d'interesse. Nello specifico, il segmentatore utilizzato è chiamato TotalSegmentator [29].

TotalSegmentator è un tool che permette la segmentazione di oltre 117 strutture anatomiche in immagini TC, alcune delle quali sono rappresentate in figura 2.2. Esso è stato addestrato su un grande varietà di TC diverse e quindi funziona molto bene anche in presenza di TC con caratteristiche molto differenti tra loro. Per il suo funzionamento, esso utilizza il metodo nnU-

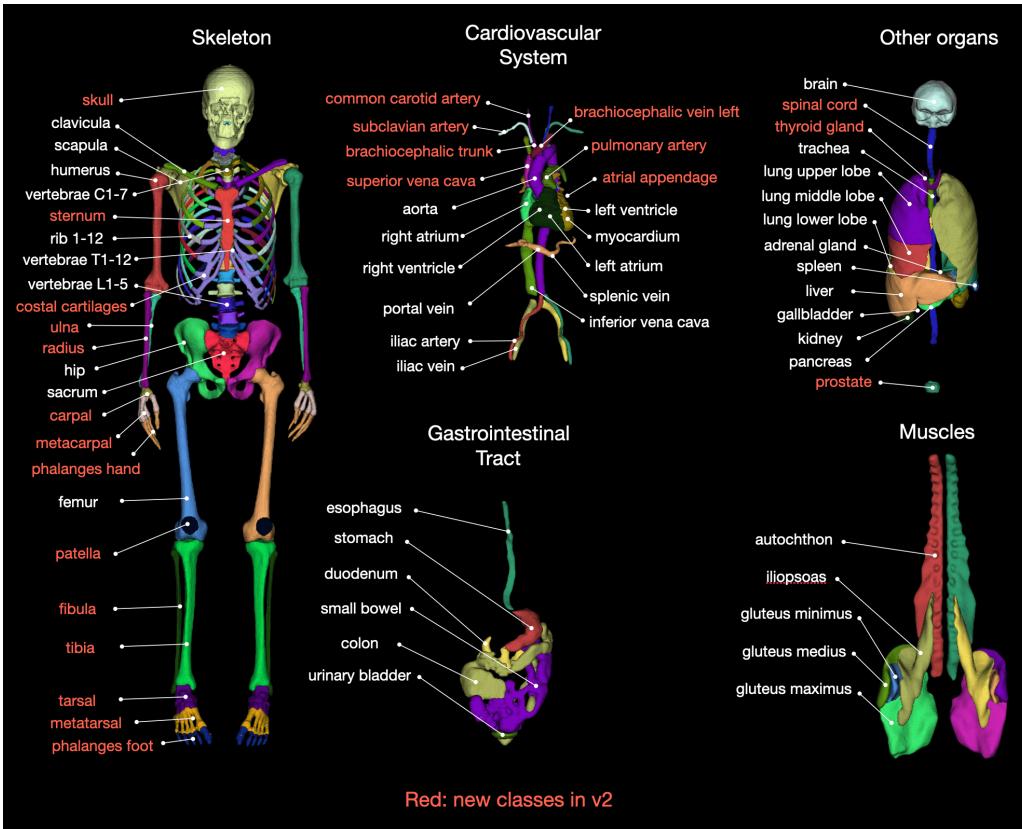


Figura 2.2: Rappresentazione delle strutture anatomiche che è possibile segmentare tramite il tool TotalSegmentator¹

Net [15]. Questo metodo permette di configurare una pipeline di segmentazione basata su una rete U-Net che si adatta ad ogni tipo di dataset in ambito medico. In base al dataset in ingresso, nnU-Net è in grado di analizzare i casi di training forniti e creare una fingerprint del dataset. Questa fingerprint servirà al metodo per generare varie configurazioni per la rete U-Net, in modo da trovare la configurazione migliore per lo specifico dataset. U-Net [24] è un’architettura di rete neurale utilizzata principalmente per compiti di segmentazione semantica. La rete è divisa in due parti: un encoder e un decoder. L’encoder permette di ridurre le dimensioni spaziali delle immagini ed estrarre alcune caratteristiche importanti delle stesse. Il decoder si occupa invece di costruire l’immagine segmentata e riportarla alle dimensioni originali, per ottenere la mappa di segmentazione. Un aspetto fondamentale è la presenza di skip connection tra l’encoder e il decoder, che permettono di preservare dettagli importanti durante il processo di segmentazione.

Nel caso specifico di questo lavoro di tesi, il tool TotalSegmentator è stato utilizzato per il riconoscimento di 4 classi specifiche nelle TC fornite, che corrispondono ad omero sinistro,

¹Immagine tratta da: <https://github.com/wasserth/TotalSegmentator>

omero destro, scapola sinistra e scapola destra. Ognuna di queste 4 classi possiede un valore intero, chiamato label, che ne permette l'identificazione nella mappa di segmentazione ottenuta in uscita dal segmentatore. L'omero sinistro corrisponde alla label 69, l'omero destro alla 70, la scapola sinistra alla 71 e quella destra alla 72. Riconoscendo questi valori nell'immagine finale è possibile visualizzare in modo differente le varie parti della labelmap e anche eventualmente isolare solo un certo osso dalla TC originale. L'aspetto importante sta però nel fatto che tramite questo tool è possibile isolare le parti di interesse da un'immagine che contiene moltissime altre informazioni, che in questo caso sono però considerate come del rumore, in quanto sono superflue per l'obiettivo posto. Nel caso specifico di questo lavoro, le parti di interesse riguardano essenzialmente la forma delle ossa principali che compongono la spalla, cioè scapola e omero. Segmentando solamente quelle due ossa tramite TotalSegmentator, sarà possibile fornire in ingresso al modello le sole informazioni che si vuole che esso consideri, eliminando le altre molteplici presenti nella TC originale.

I passi necessari alla segmentazione, che sono stati eseguiti per ognuna delle TC, sono i seguenti:

- Conversione della spalla dal formato DICOM a NIFTI
- Segmentazione di omero e scapola dalla spalla
- Memorizzazione del risultato della segmentazione in un file NIFTI in una cartella opportuna
- Conversione del gomito, se esiste, da DICOM a NIFTI
- Segmentazione dell'omero dal gomito
- Memorizzazione del risultato della segmentazione in un file NIFTI in una cartella opportuna
- Se necessario, correzione delle label del gomito come descritto nella sezione 2.2.4

L'esecuzione di questo programma, che permette la segmentazione di spalla e gomito di ognuna delle TC non escluse e la memorizzazione delle labelmap, ha richiesto circa 24 ore. La segmentazione, infatti, è un processo lungo che richiede parecchio tempo, soprattutto con una mole di dati importante come questa. Per l'esecuzione di questo programma, così per anche quelli futuri, è stata utilizzata una libreria di logging chiamata Loguru, che permette di produrre facilmente file di log che, se scritti opportunamente, permettono un'utile analisi del risultato della segmentazione. In questo specifico programma, il file di log contiene delle frasi che permettono, per ogni TC, di capire quali step siano stati eseguiti o meno per ognuna di esse, individuando quindi alcune informazioni utili sul processo di segmentazione come il numero effettivo di TC

che possiedono le slice sul gomito o il numero di quelle che hanno sollevato dei problemi durante l'esecuzione. Le considerazioni sulla qualità dell'output saranno descritte nella sezione 2.3. Prima di passare a questa sezione, però, verrà descritta brevemente la fase di correzione delle label riportata nell'ultimo punto della spiegazione della soluzione che è stata implementata per le TC dei gomiti.

2.2.4 Correzione delle Label

Eseguendo alcune prime prove di segmentazione, preliminari all'esecuzione finale su tutte le TC, è emerso che le label delle ossa dei gomiti risultavano spesso errate. Il tool di segmentazione, infatti, confondeva frequentemente l'omero destro con il sinistro e viceversa. Ciò è probabilmente dovuto, come già detto nella sezione 2.1.1, alla scarsa qualità di acquisizione dei gomiti. Le label degli omeri nelle spalle, invece, sono sempre risultate corrette. È stato quindi implementato un postprocessing alla segmentazione dei gomiti che, se necessario, permette di correggere eventuali errori nelle label degli omeri nei gomiti conformandole con quelle della rispettiva spalla. Il funzionamento dell'algoritmo è il seguente:

- Lettura delle segmentazioni di scapola e gomito della TC
- Controllo sull'omogeneità delle label: se la spalla contiene la label dell'omero sinistro e la labelmap dell'omero corrispondente nel gomito possiede alcuni voxel con label dell'omero destro, allora questi ultimi vengono tutti adattati e trasformati in label dell'omero sinistro. Comportamento analogo nel caso di spalla contenente l'omero destro
- Creazione del nuovo file NIFTI aggiornato e sostituzione di quello errato

Tutto ciò viene eseguito a seguito della segmentazione di ogni TC, considerando soltanto quelle che contengono informazioni sul gomito. In questo modo, al termine di questa fase, l'omero individuato nella spalla e quello nel gomito corrispondente non avranno più incongruenze di label.

2.3 Visualizzazione delle Labelmap e Considerazioni

Questa sezione riguarda la visualizzazione dell'output della segmentazione ed alcune considerazioni sulla qualità dello stesso emerse dal file di log e dalla visualizzazione grafica. Verranno

visualizzati i rendering 3D e le slice di alcune TC in formato NIFTI che sono state ritenute significative alle quali è stata sovrapposta la maschera di segmentazione ottenuta tramite il passo precedente. Ognuna di esse possiederà dei commenti legati alle sue particolarità. Infine verranno riportate alcune considerazioni generali sull'output della segmentazione.

2.3.1 Visualizzazione

Di seguito verranno riportate alcune immagini di TC che sono state considerate significative. Da alcune di esse è stata ottenuta una buona segmentazione mentre altre sono più problematiche. Queste figure sono state ottenute tramite l'utilizzo di uno strumento chiamato 3D Slicer [7][8]. Esso è un tool open-source per l'analisi e la visualizzazione di immagini mediche tridimensionali. È utilizzato principalmente in ambito biomedico e clinico per la segmentazione, la registrazione e l'analisi delle immagini. Inoltre, grazie alla sua flessibilità e alla molteplicità di estensioni disponibili, 3D Slicer è ampiamente adottato nella ricerca scientifica e nella pratica clinica per una vasta gamma di applicazioni, in quanto si adatta a molteplici ambiti medicali differenti. Nel caso specifico di questo lavoro, esso è stato adottato in quanto in grado di leggere il formato NIFTI, che OHIF non è invece in grado di gestire.

Per ogni paziente ritenuto significativo verranno quindi riportate:

- Visualizzazione frontale del rendering 3D della TC originale alla quale è stato sovrapposto il rendering 3D della maschera di segmentazione. In questa parte le TC di spalla e gomito sono state unite e allineate direttamente da 3D Slicer, in modo da ottenere un rendering che sembri unico, ma che in realtà deriva da due TC differenti. Il fatto che la spalla sia stata sottoposta a resampling mentre il gomito no, non influisce sulla visualizzazione in quanto l'informazione sulla dimensione dei voxel è presente nei file NIFTI e viene gestita dal visualizzatore. Infatti, neanche prima del resampling la dimensione dei voxel nelle spalle e i gomiti era la stessa.
- Una slice selezionabile per ogni piano di visualizzazione (trasverso, frontale e sagittale) della TC con la corrispettiva maschera di segmentazione sovrapposta

Questa parte è volta a mostrare l'output del modello così da evidenziare le diversità tra le varie TC e quindi di conseguenza tra le segmentazioni ottenute.

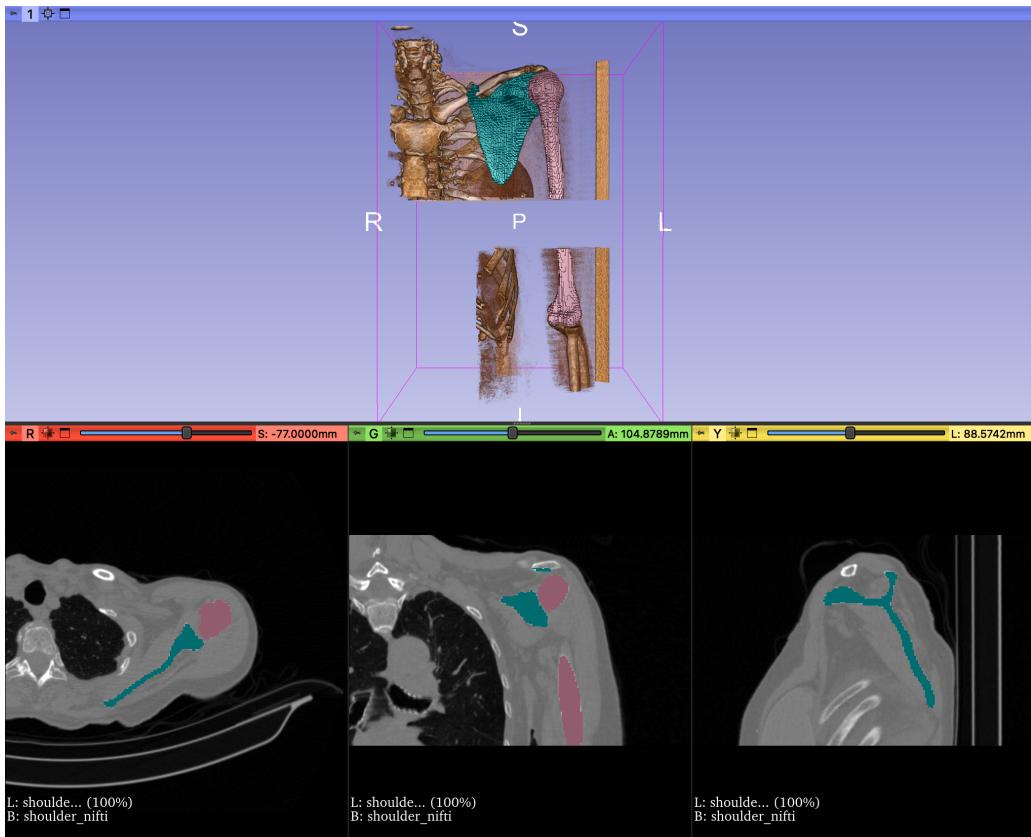


Figura 2.3: Visualizzazione TC e segmentazione del paziente 1

Paziente 1

La figura 2.3 mostra un esempio di TC segmentata in modo quasi perfetto. I colori nel rendering 3D e nelle slice significative indicano valori differenti delle label, ad indicare che le ossa sono state correttamente riconosciute. Da notare che il colore dell'omero nella spalla e nel gomito è lo stesso, segno che le label sono consistenti. Le slice sottostanti appartengono alla TC della spalla e mostrano la precisione della segmentazione in questo caso, aspetto che si può facilmente notare anche dal rendering.

Paziente 46

La figura 2.4 evidenzia due particolarità. La prima consiste nel fatto che le TC della spalla e del gomito sono state eseguite in modo consecutivo, come si può facilmente notare dal rendering 3D. Infatti, in questa immagine non vi è alcun buco di informazioni nell'omero, come invece avveniva in quella precedente. La seconda particolarità riguarda invece la segmentazione. Questo rendering infatti è stato ottenuto prima dell'implementazione della parte di correzione delle label dell'algoritmo di segmentazione, spiegata nella sezione 2.2.4. Ciò permette di mostrare il

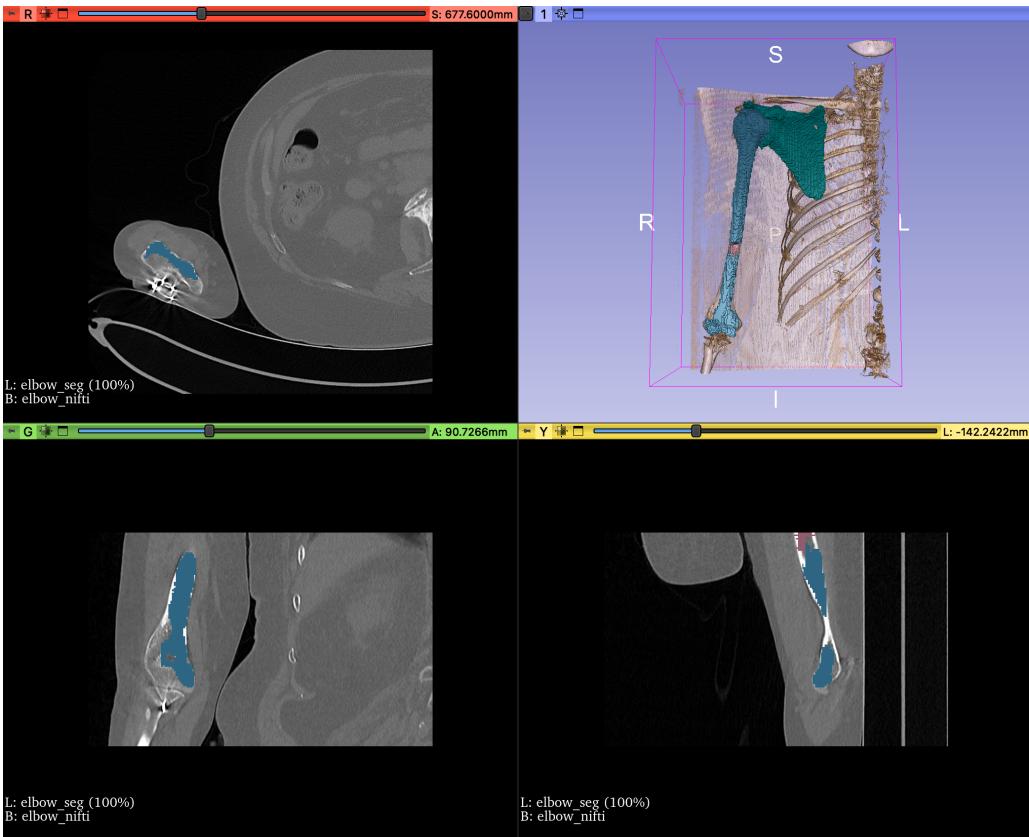


Figura 2.4: Visualizzazione TC e segmentazione del paziente 46

problema di inconsistenza delle label del gomito rispetto alla spalla. Si può infatti notare sia dal rendering che dal riquadro in basso a destra come una piccola parte iniziale di omero del gomito sia stata erroneamente classificata con una label differente (parte in marrone). In altre TC questa parte di errore risultava essere molto più grande e ciò comportava un'inconsistenza tra le label, che è stata risolta opportunamente. Inoltre, è necessario anche evidenziare come la parte dell'omero nel gomito segmentata con la label corretta sia comunque poco precisa rispetto alla precedente che invece seguiva perfettamente i contorni dell'osso. Questo è dovuto molto probabilmente ad una cattiva risoluzione della TC del gomito.

Paziente 103

Quest'ultimo esempio, contenuto nella figura 2.5, è stato riportato per evidenziare un ultimo caso limite, che è quello in cui l'omero nel gomito non viene affatto riconosciuto. Si può infatti notare come la spalla venga correttamente segmentata mentre nel gomito non sia presente alcuna maschera di segmentazione. Questo è nuovamente scaturito dalla scarsa qualità della TC del gomito, che si può più facilmente notare dalle slice in basso della visualizzazione.

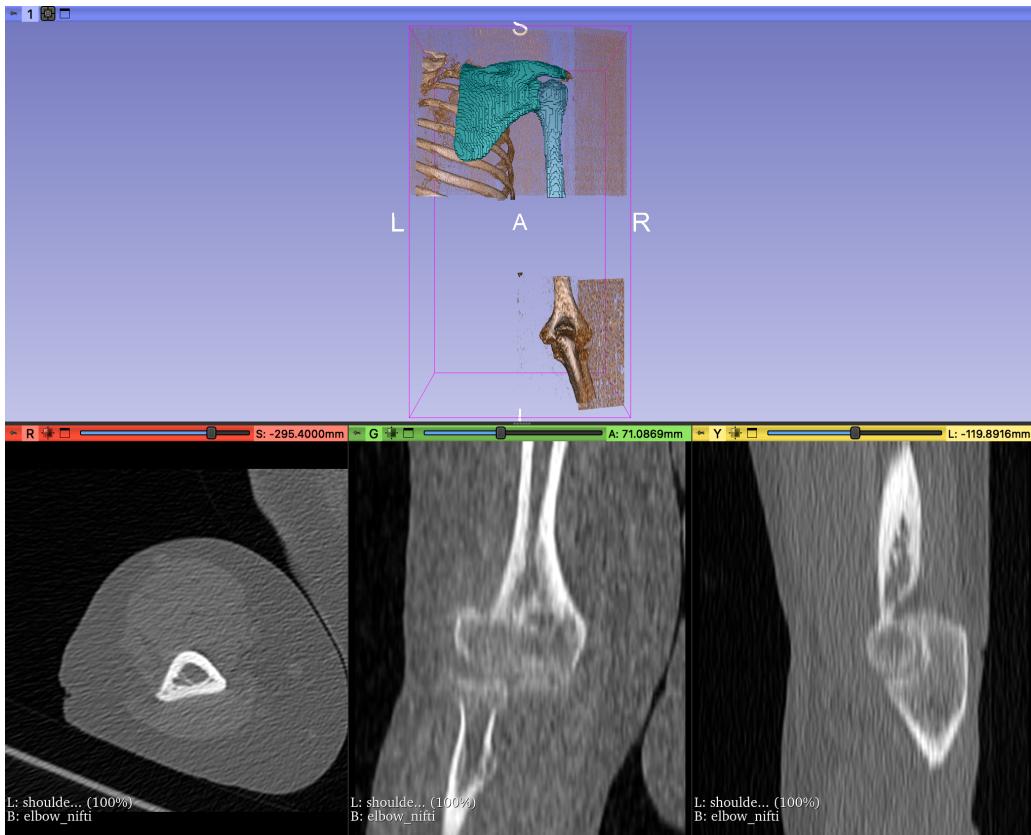


Figura 2.5: Visualizzazione TC e segmentazione del paziente 103

2.3.2 Considerazioni sulla Segmentazione

Gli esempi presentati non coprono tutti i diversi casi possibili, ma solamente alcuni di essi che sono stati individuati tramite un’ispezione manuale. I diversi problemi e le differenze che si presentano più spesso sono però essenzialmente quelli esemplificati, con qualche aggiunta già discussa nella sezione 2.1.1. In generale è possibile notare che i problemi si concentrano maggiormente nel gomito, con alcune segmentazioni che non riescono ad essere eseguite in modo ottimale o altre nelle quali addirittura le ossa non vengono riconosciute. Nella spalla la situazione è invece molto più stabile.

Dal file di log ottenuto dalla segmentazione, invece, sono state estratte altre informazioni più generali e numeriche che riguardano essenzialmente il numero di segmentazioni ottenute e quelle che invece hanno dato dei problemi. I dati estratti sono i seguenti:

- Delle 1250 TC fornite, 106 sono state escluse nella prima fase ed altre 37 in realtà non esistono, cioè entrambe le cartelle che dovrebbero contenere le slice di gomito e spalla sono vuote

- Delle 1107 che rimangono, 238 non contengono le slice del gomito, ma possiedono però le informazioni sulla spalla
- Per quanto riguarda la correzione delle label nel gomito, il 37% delle TC che possiedono informazioni sul gomito ha necessitato della correzione delle label, mentre il rimanente 63% è risultato avere label già corrette e consistenti con quelle della spalla

In sintesi, solamente 869 delle 1107 TC possiedono l'informazione completa di spalla e gomito. Questi risultati aiutano a comprendere la situazione generale che riguarda i dati forniti e di nuovo enfatizza la diversità delle TC e la prevedibile inconsistenza che è spesso presente quando si trattano dati reali. È anche emerso che le TC di alcuni pazienti effettivamente non esistevano, aspetto che non era stato in precedenza individuato.

A seguito di queste considerazioni è stato scelto, da questa fase in poi, di non considerare più i dati relativi alla parte omerale del gomito dei pazienti, seppur le segmentazioni delle stesse fossero state generate. Questa scelta è data soprattutto dalla scarsa qualità delle TC in questo caso e dalla conseguente scarsa qualità della segmentazione appena considerata. Inoltre, le poche informazioni aggiuntive che l'utilizzo di questi dati avrebbe portato non compensavano l'ulteriore complicazione che sarebbe scaturita nel considerarli. Visto che il progetto si basa interamente sulle spalle, è stato reputato sufficiente considerare i dati provenienti dalla segmentazione della scapola e dalla parte superiore dell'omero.

2.4 Upload su XNAT

In quest'ultima sezione verrà brevemente descritto il processo finale di caricamento delle maschere di segmentazione ottenute dalle spalle su XNAT [21], che consiste nel tool online per la visualizzazione di tutte le TC. Ciò è stato fatto per poter rendere disponibili ai medici che hanno collaborato al progetto l'output ottenuto dalla segmentazione in modo semplice e immediato. Il seguente processo comprende due fasi principali. La prima è quella di trasformazione delle segmentazioni dal formato NIFTI al formato RTSTRUCT, in quanto esso è uno dei pochi formati supportati dal visualizzatore di XNAT che permette di mostrare le mappe di segmentazione. Il formato RTSTRUCT permette di descrivere i contorni di strutture anatomiche di vario tipo. Ogni struttura è dotata di alcune caratteristiche come il suo nome, le forme dei contorni che la compongono e le informazioni sulla superficie delle stesse. La fase successiva

riguarda invece il caricamento vero e proprio delle RTSTRUCT su XNAT, che è stato possibile grazie ad un'opportuna libreria di Python.

2.4.1 Conversione da NIFTI a RTSTRUCT

La prima fase che coinvolge la conversione delle maschere da NIFTI a RTSTRUCT è piuttosto semplice, in quanto esiste una libreria opportuna che permette proprio la generazione di maschere in questo tipo di formato. Brevemente, i dati delle segmentazioni sono stati estratti dal corrispettivo file NIFTI generato dal processo descritto nella sezione 2.2.3. Questi dati, memorizzati sotto forma di un tensore a 3 dimensioni, sono stati elaborati in modo tale da aggiungere una ROI (Region of Interest) per ogni label presente nella segmentazione, controllando la presenza o meno di ognuna delle 4 label nei dati ed eventualmente isolando, per ogni ROI, solamente i dati dotati di una certa label. In questo modo si ottiene un file RTSTRUCT con un numero di ROI pari al numero delle diverse label presenti, in modo da isolare ogni osso e poterli distinguere nella fase di visualizzazione. Ad ogni ROI è stato anche assegnato un nome identificativo che indica l'osso corrispondente. Una volta generato questo tipo di file per ogni TC, è possibile passare alla fase successiva di caricamento.

2.4.2 Upload su XNAT

La fase di caricamento delle strutture su XNAT è invece un po' più complicata, in quanto esso possiede una struttura piuttosto rigida che è necessario rispettare. La struttura di XNAT è composta da uno o più progetti, all'interno dei quali esistono più soggetti. In questo specifico caso ogni soggetto corrisponde ad un paziente. È stato quindi necessario in primo luogo far coincidere l'identificatore del paziente alla RTSTRUCT corrispondente. Per ogni soggetto esistono degli esperimenti, che consistono in dati contenenti informazioni differenti sullo stesso paziente, come ad esempio TC di parti del corpo diverse. Per permettere la sovrapposizione del DICOM originale con la sua segmentazione è necessario quindi selezionare il medesimo esperimento che contiene la TC. Una volta impostati correttamente tutti i parametri è possibile, tramite un'apposita libreria Python, caricare una ad una tutte le segmentazioni.

Nel visualizzatore, esse appaiono come dei contorni attorno alle ossa segmentate, come mostra la figura 2.6. Ogni osso è distinguibile tramite un diverso colore e, sovrapponendo il puntatore ai

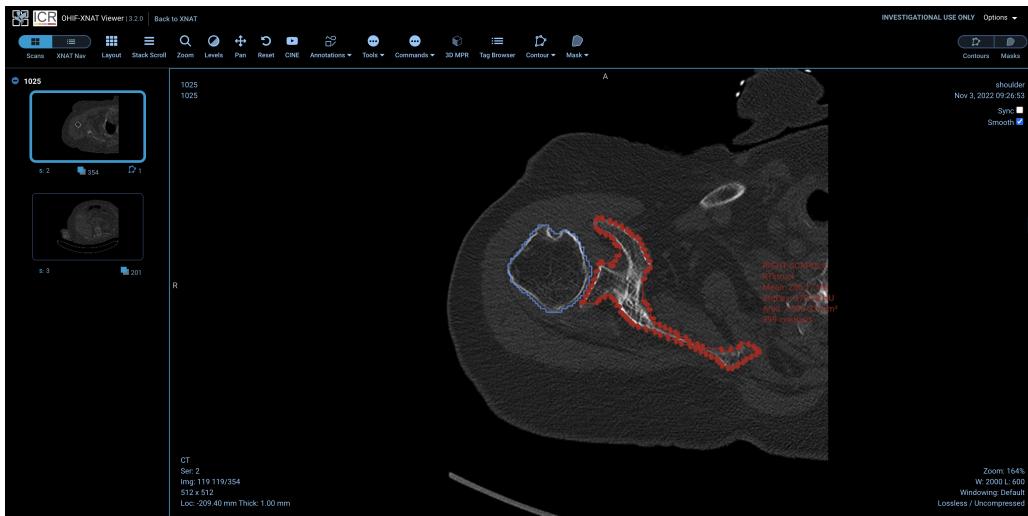


Figura 2.6: Schermata di XNAT OHIF Viewer con segmentazione

contorni, è possibile visualizzare il nome della ROI corrispondente e altre informazioni come, per esempio, l’area della regione. In questo modo è possibile per i medici esaminare le segmentazioni prodotte in modo semplice e veloce e fornire anche dei giudizi sulla loro qualità, evidenziando alcune parti importanti che il tool potrebbe aver erroneamente segmentato o addirittura non riconosciuto.

Dopo qualche incontro, infatti, è emerso che a molte segmentazioni delle scapole mancava una parte, chiamata fossa del sovraspinato, e anche che non sempre la segmentazione è risultata ottimale, con alcune scapole segmentate in modo molto impreciso. Gli omeri invece, avendo una forma più semplice da segmentare, possiedono una qualità maggiore e raramente ci sono parti segmentate male. Questo problema non è però facilmente risolvibile, in quanto il tool di segmentazione utilizzato è stato addestrato su certi dati e sarebbe necessario quindi un riaddestramento dello stesso tramite l’uso di altri dati aggiuntivi, che non è semplice ottenere. Per questo, il progetto è continuato con l’assunzione che le segmentazioni fossero sufficientemente soddisfacenti, visto che comunque le parti fondamentali delle ossa sono in grossa misura segmentate correttamente.

Al termine della parte relativa a questo capitolo, quindi, tutte le maschere di segmentazione sono state prodotte e memorizzate nelle opportune cartelle ed è possibile ora iniziare la fase di processing di queste maschere, necessaria per il futuro addestramento della rete Autoencoder che sarà implementata allo scopo di estrazione delle feature.

Capitolo 3

Preprocessing e Generazione dei Dataset

In questo capitolo verranno descritti i metodi e gli algoritmi utilizzati per uniformare i dati prodotti dalle segmentazioni in modo tale da renderli adatti a essere forniti in input ad un rete neurale. Come già detto nella sezione 2.1.1, da questo momento in poi saranno considerate solamente le TC dei pazienti riguardanti le spalle. Una delle limitazioni delle reti neurali riguarda proprio i dati di input, che devono essere necessariamente tutti della stessa dimensione. L'output della segmentazione contiene invece dati di dimensione differente, dovuti anche alla fase di resampling. L'argomento centrale sarà quindi la ricerca di una bounding box minima che riesca a contenere tutti i dati delle segmentazioni riducendo al minimo lo spazio occupato. Una volta trovata la bounding box, le immagini contenenti le segmentazioni andranno tagliate nel modo corretto per evitare perdite di dati e infine saranno creati i due dataset utilizzati per l'addestramento dei modelli. Durante lo sviluppo di questa parte sono stati scoperti altri dettagli sulle segmentazioni che verranno anch'essi riportati e discussi.

3.1 Ricerca della Bounding Box

Questa prima sezione è anche la più corposa, perché riguarda la parte più complicata di questa fase, che coinvolge la ricerca di una Bounding Box minima. L'idea è quella di riuscire a identificare le dimensioni minime nelle 3 direzioni spaziali in modo tale che le immagini in uscita

possiedano tutte la stessa dimensione in termini di numero di slice, numero di righe e numero di colonne. In questo modo, sarà possibile generare dei dataset contenenti immagini tutte della stessa dimensione senza alcuna perdita di dati e occupando il minor spazio possibile, il quale è un aspetto non irrilevante da considerare quando si parla di volumi di dati così significativi. La ricerca della Bounding Box è composta da più parti. Il primo step consiste nel dividere i dati delle segmentazioni per osso, in quanto l'obiettivo finale è quello di ottenere due dataset che contengano rispettivamente gli omeri e le scapole. Tutte le successive fasi saranno svolte differentemente sui dati riguardanti gli omeri e le scapole, ma gli algoritmi implementati saranno spesso molto simili tra di loro. Una volta separate le due ossa dai file contenenti le segmentazioni, prima di procedere all'esecuzione dell'algoritmo vero e proprio, sono state compiute diverse prove dalle quali sono emerse alcune importanti considerazioni che hanno permesso di escludere alcune TC da questa ricerca. Una volta identificate le segmentazioni problematiche, la parte successiva consiste nella ricerca vera e propria delle dimensioni della box. In realtà, come sarà spiegato più approfonditamente nella sezione opportuna, l'operazione non consiste solamente nella ricerca di queste dimensioni in quanto i dati, rappresentando ossa in posizioni diverse tra loro, necessitano anche di un cropping in posizioni differenti. La ricerca della bounding box quindi coinvolge anche la ricerca di questi punti di taglio personalizzati per ogni immagine. Una volta ricavate queste informazioni, è infine possibile procedere alla fase di cropping vera e propria, nella quale si taglano tutte le segmentazioni fino ad ottenere un dataset uniforme.

3.1.1 Divisione di Omeri e Scapole

Il primo semplice passo da eseguire riguarda la separazione dei dati riguardanti gli omeri e le scapole dai file contenenti le segmentazioni. Questo è necessario in quanto, come già detto, il desiderio è quello di ottenere due dataset differenti, uno contenente solamente gli omeri di tutti i pazienti e l'altro solamente le scapole. Il formato dei dati in uscita dal tool di segmentazione aiuta in questo senso in quanto, come già descritto nella sezione 2.2.3, i dati possiedono delle label che indicano il tipo specifico di osso riconosciuto. Quindi, per isolare gli omeri, ad esempio, basta sostituire ogni voxel che possiede come label il numero intero corrispondente all'omero destro e all'omero sinistro con un altro valore univoco, che in questo caso è stato scelto pari ad 1. In questo modo si elimina anche l'informazione sulla parte del corpo considerata e si uniformano alla stessa label sia omeri destri che sinistri. Un comportamento analogo avviene anche con le scapole. In uscita da questa prima fase si ottengono quindi, per ogni segmentazione,

delle immagini binarie, composte cioè solamente da valori pari a 0 e 1. È inoltre opportuno evidenziare come, durante questa conversione, venga anche modificato il formato di memorizzazione dei singoli voxel. Essendo necessari solamente due valori, i dati vengono memorizzati sotto forma di tensori Numpy in 3 dimensioni nei quali ogni voxel occupa il minimo spazio possibile, cioè 8 bit. Questo è molto importante perché il formato di memorizzazione di ogni voxel permetterà di risparmiare moltissimo spazio quando i dati verranno inseriti in un dataset e memorizzati.

3.1.2 Considerazioni Preliminari

Prima di sottoporre le immagini binarie così ottenute all'algoritmo di ricerca della bounding box è opportuno evidenziare alcuni aspetti emersi durante le prime prove di esecuzione di questo algoritmo. Durante queste, infatti, sono emersi altri aspetti legati alle TC che non erano precedentemente stati individuati.

Un primo problema trovato consiste nell'orientamento delle ossa. Dopo un'ulteriore analisi, infatti, è emerso che le TC sono state acquisite con il paziente in posizioni leggermente differenti. Certe volte, il braccio risulta inclinato e di conseguenza anche l'omero e la scapola. In questo modo, nel momento in cui viene definita una bounding box attorno ad esse, la sua dimensione cresce moltissimo in quanto è necessario ampliarla per tenere in considerazione tutto il dato dell'omero che risulta più ingombrante essendo inclinato. Ciò avviene in quanto la box risulta essere parallela agli assi e non può essere inclinata. Questo problema è stato gestito durante l'esecuzione dell'algoritmo implementato e presentato successivamente che si occuperà di ricavare le dimensioni minime della bounding box. Inoltre, un ulteriore aspetto considerato durante la scrittura dell'algoritmo appena citato riguarda il fatto che alcune TC contengono informazioni riguardo entrambe le spalle dei pazienti. Di conseguenza, per queste particolari TC sono state costruite due bounding box differenti e nel dataset finale le ossa della spalla destra e sinistra sono state considerate come due immagini separate, in quanto possono avere una struttura diversa seppur appartenenti alla stessa persona.

Altri errori di segmentazione sono invece emersi durante ulteriori esecuzioni dell'algoritmo soprattutto e sono state attuate modifiche allo stesso per cercare di ridurre il più possibile la presenza di questi ultimi nel dataset finale. Questa parte verrà però descritta nella successiva

sezione durante la spiegazione del funzionamento dell'algoritmo stesso.

3.1.3 Algoritmo di Ricerca

In questa sezione verrà descritto in dettaglio l'algoritmo che permette di ricavare le dimensioni della bounding box minima che racchiude tutti gli omeri e analogamente quella che racchiude tutte le scapole, così come la generazione dei file che permettono il cropping corretto delle stesse, i quali conterranno le posizioni in cui tagliare le immagini contenenti le segmentazioni per far sì che non avvenga alcuna perdita di dati. Verrà riportata solamente la parte dell'algoritmo completo riguardante l'estrazione delle dimensioni della bounding box e delle posizioni di taglio relative alle immagini dell'omero sinistro in quanto le altre, che riguardano le rimanenti ossa, sono quasi completamente analoghe. I punti di differenza saranno comunque evidenziati opportunamente. Di seguito l'algoritmo verrà diviso in parti, ognuna delle quali svolge una funzione ben precisa.

Inizialmente vengono definite tutte le variabili necessarie al funzionamento dell'algoritmo, come i valori delle label delle ossa e la lista delle TC escluse. Questa lista è stata in parte ottenuta tramite alcune iterazioni preliminari dell'algoritmo che hanno evidenziato problemi in alcune di esse, ma la maggior parte delle TC che la compongono erano già state ricavate durante l'analisi presentata in 2.3, nella quale sono state riportate solamente gli esempi più rilevanti. Quelle più problematiche, ad esempio quelle che contenevano una segmentazione totalmente errata, sono state direttamente escluse inserendole in questa lista. Altre invece verranno escluse durante l'esecuzione dell'algoritmo al presentarsi di determinate condizioni, in modo da rendere lo stesso il più generale possibile. Ad esempio, saranno escluse le segmentazioni con un numero di slice contenenti informazione inferiore ad un certo limite, che rappresentano spesso degli errori. Verranno inoltre eliminate dalle labelmap delle piccole parti di segmentazione isolate dall'osso che sono state erroneamente segmentate durante il processo.

Tutto il codice riportato di seguito è inserito all'interno di un ciclo e viene svolto per ogni TC. Verrà considerato il codice relativo alla bounding box attorno all'omero della spalla sinistra che è però molto simile a quello relativo alle altre ossa. La prima parte comprende una serie di controlli ed è la seguente:

```
1 shoulder_seg_data = np.asarray(nib.load(f"../processing/{i}/shoulder_seg_res.nii").dataobj)
# Leggo il file NIFTI contenente i dati sulla segmentazione della i-esima TC
```

```

2 if i not in excluded_cts: # Lista delle TC escluse a priori
3     unique_labels = np.unique(shoulder_seg_data) # Prendo i valori unici per vedere quali
        label sono presenti nell'immagine
4     if (left_humerus_label in unique_labels) and (left_scapula_label in unique_labels): #
        Spalla sinistra
5         shoulder_seg_humerus_data = np.where(shoulder_seg_data == left_humerus_label, 1, 0).astype
        (np.uint8) # Isolo i dati dell'omero sinistro
6         shoulder_seg_scapula_data = np.where(shoulder_seg_data == left_scapula_label, 1, 0).astype
        (np.uint8) # Isolo i dati della scapola sinistra

```

Listato 3.1: Lettura dei dati e primi controlli

In questa prima parte vengono letti tutti i dati necessari all'esecuzione dell'algoritmo, escludendo le TC appartenenti alla lista *excluded_ct*, inizializzata preventivamente e contenente un insieme di segmentazioni individuate come errate. Vengono poi isolate le informazioni della spalla sinistra necessarie al funzionamento del codice. L'algoritmo è analogo per l'omero nella spalla destra, considerando ovviamente le label opportune, mentre ci sono alcune piccole differenze per quanto riguarda le scapole, che verranno evidenziate in seguito. Al termine di questa parte, quindi, le due immagini contenenti i dati divisi di omero e scapola sono state ricavate.

```

1 max_prof_hum = np.max(shoulder_seg_humerus_data, axis=(0,1)) # Prendo il massimo all'interno
    di ogni slice
2 pos_array_prof_hum = np.diff(max_prof_hum, prepend=0).nonzero()[0] # Faccio la differenza tra
    l'array ottenuto e una sua versione traslata per vedere dove cambia e salvo le
    posizioni in cui la differenza non e' zero
3 # Stesso comportamento per le scapole
4 max_prof_scap = np.max(shoulder_seg_scapula_data, axis=(0,1))
5 pos_array_prof_scap = np.diff(max_prof_scap, prepend=0).nonzero()[0]
6
7 if pos_array_prof_scap.size != 0 and pos_array_prof_hum[-1]-pos_array_prof_scap[0] >=
    limit_slices:
8     # Entro nell'if solamente se la porzione di osso da salvare e' maggiore di una costante
        opportunamente scelta
9     logger.info(f"Processo ct {i}: spalla sinistra")
10    pos_array_prof_ct.append(pos_array_prof_scap[0])
11    pos_array_prof_ct.append(pos_array_prof_hum[-1])

```

Listato 3.2: Limiti bounding box sull'asse z

Questa parte dell'algoritmo si occupa del primo step del calcolo della bounding box e delle posizioni di taglio dell'immagine. Nella riga 1, viene ricavato il massimo per ogni slice e si ottiene così un array che contiene un valore per ogni slice che risulta essere pari a 0 se nella slice non è presente alcuna informazione mentre pari a 1 se invece è presente anche un solo voxel con valore unitario. Nella riga successiva viene poi prodotto un array che conterrà gli indici in

cui il valore del primo array cambia da 0 a 1 o viceversa. Questi indicano le posizioni in cui l'omero inizia e finisce verticalmente, in quanto si sta considerando il massimo per ogni slice. Questo array viene ottenuto eseguendo una differenza tra il vettore precedentemente definito e una sua versione traslata. Il risultato intermedio consisterà in un ulteriore array avente un valore pari a 0 nelle posizioni in cui il valore massimo non cambia ed un valore pari a 1 o -1 dove invece ci sono dei cambiamenti. Leggendo tramite la funzione *nonzero()* le posizioni in cui questi valori risultano essere diversi da 0, è possibile ottenere l'array desiderato. La medesima operazione viene fatta anche sulla scapola per ottenere le posizioni iniziale e finale della stessa. L'istruzione alla riga 7 permette di escludere alcune segmentazioni, cioè quelle che non presentano alcuna informazione sulla scapola oppure quelle che hanno una differenza tra la slice più in alto dell'omero e quella in più in basso della scapola inferiore ad una soglia scelta opportunamente. Questa porzione corrisponde alla parte di omero che infine verrà memorizzata. Si può notare come venga considerato il limite superiore dell'omero, per far sì che la testa dell'omero venga considerata, ma il limite inferiore della scapola. È stata effettuata questa scelta in quanto è constatato che la parte di omero d'interesse è quella posizionata vicino alla scapola ed è quindi stato deciso di rimuovere la parte di omero sottostante l'ultima slice della scapola in profondità. È stato infine deciso di escludere le segmentazioni formate da un numero di slice inferiore a un valore memorizzato nella variabile *limit_slices* in quanto considerate degli errori si segmentazione. Le istruzioni 10 e 11 permettono il salvataggio di queste posizioni in un array appropriato che conterrà, per ogni indice, le posizioni corrette in cui tagliare il corrispondente omero in profondità.

```
1 max_hor = np.max(shoulder_seg_humerus_data[:, :, pos_array_prof_scap[0]:], axis=1) # Asse orizzontale
2 pos_array_hor = np.diff(max_hor, prepend=0, axis=0).nonzero()[0]
3
4 index_split = np.where(np.diff(pos_array_hor, prepend=0)[1:] > limit)[0] # Faccio la differenza tra l'array contenente gli indici e la sua traslazione, prendendo solamente i valori maggiori di limit
5 lista_split = np.split(pos_array_hor, index_split+1) # Divido l'array nei punti indicati dall'array index_split
6 cur_width = 0
7 for el in lista_split:
8     if el[-1]-el[0] > cur_width: # Considero lo spazio verticale con la differenza maggiore
9         pos_array_hor = el
10        cur_width = el[-1]-el[0]
11
12 pos_array_hor_ct.append(pos_array_hor[0])
```

```
13 pos_array_hor_ct.append(pos_array_hor[-1])
```

Listato 3.3: Limiti bounding box sull'asse x

In questo listato è contenuto il codice analogo al precedente che calcola però i limiti massimi e minimi in direzione dell'asse x, cioè l'asse 1 dell'immagine. Le prime due istruzioni, infatti, sono analoghe alle precedenti ad esclusione del fatto che il massimo viene eseguito lungo l'asse 1. Inoltre, le slice non vengono interamente considerate per questo calcolo, ma viene considerata solamente la parte di omero selezionata al punto precedente. In questo modo è possibile trovare il limite superiore ed inferiore all'interno di ogni slice e ottenere così i valori di taglio che permetteranno di contenere tutto l'omero. In questa parte è stato però inserito un controllo aggiuntivo, che riguarda le righe dalle 4 alla 10. Questa porzione è stata aggiunta per permettere di ovviare ad un errore che veniva compiuto in alcune segmentazioni. Capitava infatti che alcune di esse avessero delle piccole parti di segmentazione isolate dal resto dell'osso, dovute ad un errore del tool. Queste parti venivano considerate come appartenenti ad un unico osso nel calcolo della bounding box in quanto possedevano la stessa label dell'osso stesso, ottenendo così delle misurazioni errate. La parte di algoritmo in esame permette proprio di escludere queste piccole parti errate dal calcolo delle posizioni di taglio delle immagini prendendo in considerazione solamente l'osso principale. Per fare ciò, nella riga 4 vengono ricavati gli indici di inizio e fine di ogni parte di segmentazione con una distanza orizzontale maggiore di *limit* dalle altre. Questo valore è stato scelto opportunamente in modo tale da separare le porzioni di segmentazione con la stessa label abbastanza distanti le une dalle altre, in modo da identificare delle eventuali porzioni errate. In questo modo, tramite l'istruzione 5, è possibile ricavare una lista di array ognuno dei quali conterrà una coppia di valori che indica l'indice di inizio e fine di ogni porzione individuata. Infine, tramite il ciclo, è possibile determinare e salvare definitivamente la porzione più grande, cioè quella che sicuramente rappresenterà l'osso stesso, escludendo quindi tutte le altre. Viene anche salvata nella variabile *cur_width* la larghezza di questa sezione, che verrà utilizzata alla fine per il calcolo delle dimensioni della bounding box. Una volta ricavate le corrette posizioni di taglio esse vengono salvate in modo analogo al precedente.

La parte successiva svolge esattamente lo stesso compito, ma considerando l'altra direzione, permettendo di ottenere i limiti destro e sinistro che consentono di tagliare esattamente l'omero senza perdite di dati e anche l'ampiezza della sezione attualmente calcolata. Essendo analoga

alla precedente, non è stata riportata.

L'ultima parte dell'algoritmo è quella che si occupa del calcolo delle dimensioni della bounding box comune a tutti gli omeri, che diventa molto semplice una volta effettuati i calcoli precedentemente descritti.

```
1 if cur_width > max_width:  
2     max_width = cur_width  
3  
4 if cur_height > max_height:  
5     max_height = cur_height  
6  
7 cur_slices = pos_array_prof_hum[-1] - pos_array_prof_scap[0]  
8 if cur_slices > max_slices:  
9     max_slices = cur_slices
```

Listato 3.4: Calcolo delle dimensioni della bounding box

Ad ogni ciclo, cioè per ognuna delle TC, viene aggiornata la dimensione massima in ognuna delle tre direzioni, chiamate in questo caso altezza, larghezza e profondità. I valori di *cur_width* e *cur_height* erano già stati ricavati nelle parti precedenti, mentre il valore di *cur_slices* viene calcolato al momento in base ai valori degli indici già memorizzati.

L'algoritmo completo è formato quindi dalla sequenza di tutte queste fasi analogamente ripetute anche per l'omero destro, il tutto racchiuso in un ciclo che itera su ogni segmentazione. Al termine dell'esecuzione del ciclo, in output verranno forniti tre array ognuno dei quali conterrà, per ogni elemento, un ulteriore array che indica le posizioni in cui tagliare la bounding box rispettivamente nelle tre direzioni in modo tale che essa sia in grado di contenere perfettamente l'osso, mentre conterrà un array vuoto nel caso in cui la TC non sia stata elaborata per uno qualsiasi dei motivi precedentemente riportati. Inoltre, verranno ritornati i valori di altezza, larghezza e profondità massimi, cioè le dimensioni della bounding box che permettono di contenere tutti gli omeri considerati. Questi array vengono infine salvati su file esterni e saranno utilizzati durante la fase di cropping vera e propria per ricavare le posizioni specifiche in cui tagliare ogni segmentazione.

L'algoritmo eseguito sulla scapola è del tutto analogo a quello qui riportato. L'unica differenza sta nel fatto che i limiti superiore ed inferiore della scapola non sono, come nell'omero, influenzati dall'altro osso, in quanto la scapola è necessaria nella sua interezza e quindi non ne viene tagliata nessuna parte.

3.2 Cropping e Memorizzazione dei Dataset

Una volta ricavati gli array contenenti le posizioni di taglio specifiche per ogni scapola e omero è infine necessario effettuare praticamente questi tagli e ottenere così un dataset formato da immagini tutte della stessa dimensione. Per far ciò è necessario inizialmente tagliare ogni osso in corrispondenza delle posizioni contenute nell'array e in seguito effettuare un padding per adattare l'immagine ritagliata alla dimensione della bounding box trovata. Dopo aver eseguito questo processo su tutte le immagini non escluse, i due dataset saranno memorizzati e potranno essere utilizzati per l'addestramento del modello. È stato scelto di generare i dataset utilizzando solamente le maschere di segmentazione e non l'applicazione delle stesse alle TC originali in quanto le informazioni fondamentali per il compito in questione risiedono quasi totalmente nella forma delle ossa e quindi nei contorni delle stesse, che sono molto ben evidenziati anche dalla sola maschera di segmentazione. Inoltre, in questo modo si riesce a risparmiare parecchio sulla memoria occupata da questi ultimi in quanto i voxel possiedono solamente valori binari e non i valori delle TC originali che spaziano in range molto più ampi.

3.2.1 Cropping e Padding

In questo paragrafo verrà descritto il metodo per ottenere, partendo dalla segmentazione, il volume con le dimensioni della bounding box comune precedentemente calcolate. Ciò coinvolge una prima fase di cropping e una successiva di padding. Per la parte di cropping, vengono lette una ad una tutte le TC e viene eseguito un controllo iniziale per comprendere se la TC considerata sia stata esclusa per uno qualsiasi dei motivi precedenti. Ciò è facilmente individuabile controllando la lunghezza dell'elemento in ogni posizione di uno dei tre array che contengono le posizioni di taglio: nel caso in cui l'elemento sia un array vuoto, cioè di lunghezza pari a 0, significa che quella spalla è stata esclusa e si passa direttamente alla successiva. Al contrario invece l'immagine contenente la segmentazione dell'osso preso in considerazione viene tagliata in ognuna delle tre dimensioni in corrispondenza delle posizioni adeguate ricavate dal corrispettivo elemento di ognuno degli array. Nel caso in cui un'immagine contenga informazioni complete su entrambe le spalle, questa operazione verrà eseguita due volte per la stessa TC, in modo da isolare le due ossa d'interesse e posizionarle in due elementi differenti del dataset.

Una volta eseguito il cropping, si otterrà un’immagine che contiene esattamente l’osso, senza più alcuno spazio superfluo nelle tre dimensioni. Le immagini però devono avere tutte la stessa dimensione, che è pari a quella della bounding box trovata tramite l’algoritmo descritto nella sezione 3.1.3. La dimensione di quest’ultima è pari alla dimensione della bounding box che permette di contenere l’osso più grande tra tutti quelli considerati. In questo modo si è certi che essa sia in grado di contenere anche tutti gli altri senza alcuna perdita di informazione. È quindi necessaria una fase di padding nella quale l’immagine viene contornata con voxel di background, che possiedono un valore pari a 0, fino a raggiungere la dimensione corretta nelle tre direzioni. Per far sì che l’osso rimanga al centro della bounding box è necessario calcolare accuratamente il numero di righe, colonne e slice da inserire prima e dopo il volume vero e proprio. Questa operazione può essere facilmente eseguita calcolando la differenza tra la dimensione della bounding box e la corrispettiva dimensione dell’immagine tagliata e dividendo infine per due il risultato ottenuto. Eseguendo la stessa operazione in tutte le direzioni è possibile ottenere l’immagine della dimensione corretta con l’osso centrato all’interno della stessa. L’immagine così ottenuta verrà aggiunta ad una lista che conterrà tutte le immagini della stessa dimensione processate in questo modo e che potrà infine essere trasformata in un dataset e memorizzata.

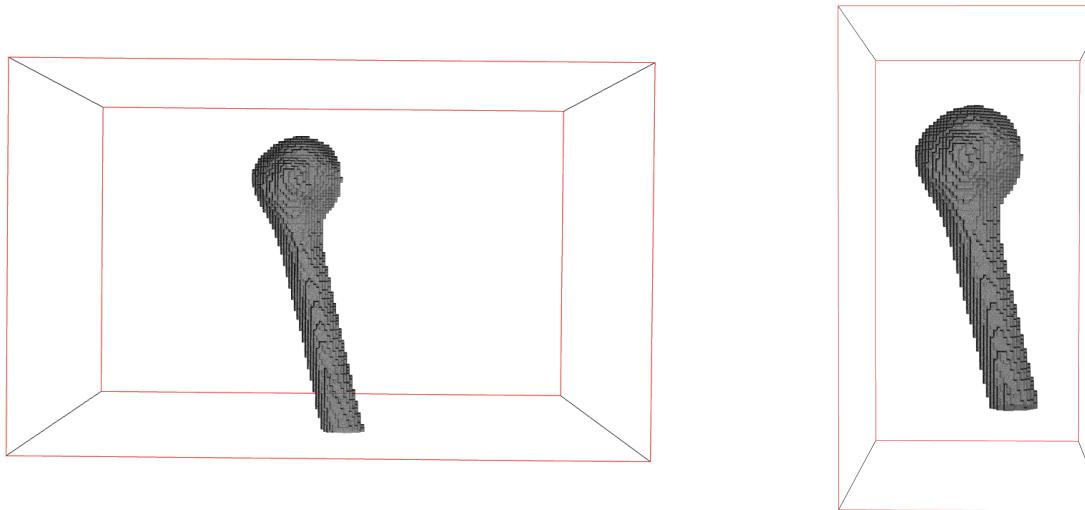


Figura 3.1: Bounding Box prima e dopo il preprocessing

La figura 3.1 mostra un esempio del cambiamento della bounding box intorno all’omero. Si può immediatamente notare come nel primo caso l’osso risulti leggermente decentrato e sia presente molto spazio inutilizzato attorno ad esso, mentre in altezza la box viene riempita

piuttosto bene. Nel secondo caso, in seguito al cropping, la bounding box ha una forma più alta e stretta, cioè più simile a quella di un omero, e lo spazio inutilizzato attorno all'osso è molto inferiore. Essa infatti è stata costruita tenendo in considerazione solo le misure di questi ultimi. Si può anche notare come nel secondo caso l'osso sia completamente centrato rispetto ai margini della bounding box, aspetto che invece non si manifesta nel primo in quanto la box in quel caso necessitava di comprendere anche la scapola e parte del corpo del paziente qui non utilizzata.

3.2.2 Memorizzazione dei Dataset

L'ultima fase consiste nella memorizzazione dei due dataset completi su disco. A questo scopo, è stata utilizzata la libreria Python h5py che consente la gestione di file in formato hdf5. La potenzialità di questo package risiede nel fatto che esso funziona in modo molto efficiente per la memorizzazione di grandi moli di dati e permette una facile manipolazione degli stessi interfacciandosi con la libreria Numpy. Sia la lettura che la scrittura dei dataset avvengono in tempi piuttosto rapidi anche con moli ingenti di dati. Nello specifico, il dataset in cui sono memorizzate le informazioni sugli omeri contiene 1095 immagini ognuna delle quali ha una dimensione di 288x210x393, per un totale di spazio occupato di 24.2 GB. Il dataset contenente le scapole è invece più grande, in quanto essa necessita di una maggior dimensione di memorizzazione essendo un osso più grande. Esso è composto da 1056 immagini ognuna delle quali ha una dimensione di 336x360x403, per un totale di spazio occupato di 47.9 GB. Si può notare come la terza dimensione, cioè quella delle slice, sia circa la stessa in quanto l'omero e la scapola iniziano più o meno nello stesso punto e l'omero è stato ritagliato in modo da terminare in corrispondenza dell'ultima slice della scapola. Ciò che fa la differenza è invece l'occupazione orizzontale e verticale dell'osso in ogni slice in quanto l'omero è un osso molto lungo ma stretto mentre la scapola ha una maggior superficie da memorizzare.

Concludendo, tutte le fasi descritte in questo capitolo hanno permesso di generare due dataset, che saranno utilizzati indipendentemente nella fase successiva, composti da omeri e scapole opportunamente tagliati in modo da occupare il minor spazio possibile rendendo più veloci tutte le operazioni di lettura e scrittura, così come l'addestramento del modello descritto nel successivo capitolo.

Capitolo 4

Convolutional Autoencoder ed Estrazione delle Feature

Il seguente capitolo tratta uno degli argomenti centrali di questo lavoro di tesi che riguarda la modalità di estrazione delle feature dalle immagini che compongono i dataset ottenuti come descritto nel precedente capitolo. L'obiettivo è quello di riuscire in seguito a suddividere le immagini in categorie tramite un algoritmo di clustering basato sulle feature così ottenute. È fondamentale eseguire questa fase nel migliore dei modi, in quanto la qualità delle feature è strettamente legata alla qualità del clustering e quindi di conseguenza a quella del risultato finale. Se il modello sarà in grado di apprendere le feature in modo adeguato, esse rispecchieranno le caratteristiche morfologiche dell'osso e quindi la suddivisione ottenuta sarà più funzionale rispetto all'obiettivo finale. Inoltre, questa fase risulta essere strettamente necessaria in quanto non sarebbe possibile effettuare un clustering sulla base di intere immagini ed è quindi obbligatorio crearne una rappresentazione più compatta che riesca a tenere conto delle loro principali caratteristiche.

Il modello scelto per questo scopo è chiamato Convolutional Autoencoder. Di seguito verrà inizialmente descritto il funzionamento generale di questi modelli e l'idea che sta alla loro base. Successivamente, verrà illustrato il modello specificatamente creato per questo lavoro, insieme alle fasi di preprocessing necessarie. Verrà infine riportata l'analisi dell'addestramento del modello tramite grafici che rappresentano l'evoluzione di alcune metriche durante lo stesso e valutata la capacità del modello mostrando le immagini di output che esso è in grado di

produrre, confrontandole con le immagini originali.

4.1 Gli Autoencoder

Gli autoencoder [16] sono modelli di Deep Learning che consentono di ottenere una rappresentazione dell'input in un numero inferiore di dimensioni e, tramite essa, ricostruire l'ingresso originale. Questa rappresentazione, chiamata encoding, permette di riassumere le feature più importanti dell'input senza alcun bisogno di label. Per questo motivo, essi sono chiamati modelli di tipo non supervisionato, oppure sono spesso riferiti anche con il termine di modelli auto-supervisionati, in quanto il loro ground truth è rappresentato dall'input stesso. Ne esistono di varie tipologie, ognuna adatta ad essere usata con tipi di input e per compiti differenti. I modelli di tipo convoluzionale utilizzano delle reti convoluzionali e sono stati appositamente creati per lavorare con immagini. Gli utilizzi principali di questo tipo di modelli coinvolgono l'estrazione di feature [3][26], così come la compressione di dati [4], la rimozione del rumore dalle immagini [10] e il rilevamento di anomalie [5].

4.1.1 La Struttura Generale

La struttura generale di un autoencoder è composta principalmente da tre parti: l'encoder, il bottleneck e il decoder, come mostrato nella figura 4.1.

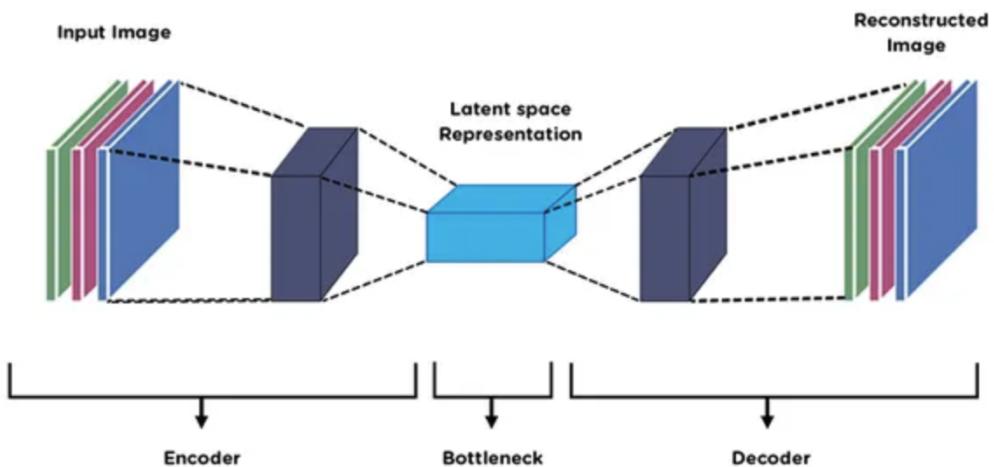


Figura 4.1: Struttura generale di un Autoencoder per il processing di immagini ¹

¹ Immagine tratta da: <https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f>

Ognuna di queste parti ha un compito ben preciso:

- L'encoder è la parte del modello che permette l'encoding dell'input, cioè la sua rappresentazione in uno spazio a dimensione inferiore. Questo avviene tramite una sequenza di livelli che progressivamente riducono le dimensioni dell'input fino ad ottenere la dimensione desiderata
- Il bottleneck contiene la rappresentazione più compressa dell'input, spesso chiamata anche latent space, e rappresenta la parte fondamentale quando si utilizza questo tipo di modelli come estrattori di feature. Esso rappresenta il collegamento tra l'encoder e il decoder. Uno degli obiettivi degli autoencoder consiste nel trovare il numero ideale di dimensioni del bottleneck che permettano una corretta ricostruzione dell'input.
- Il decoder è composto da livelli che progressivamente aumentano le dimensioni a partire dalla rappresentazione latente fino ad ottenere un output che abbia le stesse dimensioni dell'input. L'output da esso prodotto viene infine confrontato con il ground truth, che in questo caso è rappresentato dall'ingresso stesso, e viene calcolato l'errore di ricostruzione utilizzato per misurare l'efficacia del modello.

In base allo specifico utilizzo che si compie di questi modelli, il decoder possiede ruoli differenti. Ad esempio, nel caso esso venga usato come estrattore di feature, il decoder ha il solo ruolo di permettere l'addestramento della rete in modo che essa impari ad apprendere le giuste feature, ma una volta terminato l'addestramento esso viene scartato e la rete rimanente può essere usata per il compito assegnato. In altri ambiti, invece, il decoder è fondamentale anche dopo l'addestramento in quanto produce dati differenti da quelli di input, come ad esempio nell'applicazione di rimozione del rumore dalle immagini [10].

Il vantaggio principale che deriva dall'utilizzo di questo tipo di modelli rispetto ad altri metodi che permettono di ridurre le dimensioni dell'ingresso consiste nel fatto che essendo i primi basati su reti neurali, dotate quindi di funzioni di attivazione non lineari, riescono a catturare delle correlazioni complesse e non lineari tra i dati, compito che ad esempio un metodo di Principal Component Analysis (PCA) non riesce a realizzare.

Tipi diversi di autoencoder sono costituiti da differenti tipi di reti neurali in base allo specifico ambito di applicazione. I tipi principali di reti utilizzate sono: Feed Forward Neural Network, Recursive Neural Network (RNN) e Convolutional Neural Network (CNN). Per il task da portare a termine in questo progetto, che riguarda il processing di immagini, è stato necessario utilizzare una CNN. Questo tipo di Autoencoder prende il nome di Convolutional Autoencoder.

4.1.2 I Convolutional Autoencoder

I Convolutional Autoencoder (CAE) [17] sono modelli che possiedono la stessa struttura generale degli autoencoder descritta nella sezione 4.1.1, ma utilizzano come reti di encoder e decoder delle Convolutional Neural Networks (CNN). Questa tipologia di rete è stata appositamente creata per essere utilizzata con immagini o comunque con segnali che necessitano di preservare le informazioni spaziali all'interno degli stessi. Infatti, la loro caratteristica principale risiede nel fatto che, tramite l'utilizzo dell'operazione di convoluzione, essi sono in grado di preservare le relazioni spaziali che esistono fra gli input. Prendendo come esempio il caso delle immagini, ciò significa che questi metodi sono in grado di tenere in conto il fatto che due pixel siano posizionati uno accanto all'altro nell'immagine, sia verticalmente che orizzontalmente, aspetto che non sarebbe possibile considerare con reti semplici di tipo feed forward. Queste ultime, infatti, necessitano di vettori unidimensionali per funzionare, ottenibili solamente posizionando tutti i pixel dell'immagine su una sola riga e perdendo quindi le informazioni sulla loro posizione all'interno della stessa. Questi tipi di modelli sono principalmente usati per processare immagini, ma possono essere utilizzati anche per elaborare video o segnali audio.

È stato descritto nella sezione precedente che l'encoder ha il compito di ridurre le dimensioni dell'input fino ad ottenere una rappresentazione che riassume le sue feature principali. Nel caso specifico dei CAE, i livelli che compongono l'encoder sono di tipo convoluzionale. La figura 4.2 permette di comprendere meglio la struttura di questo tipo di modelli, mostrando un esempio di una rete di tipo CAE applicata ad immagini di ingresso di dimensioni pari a $28 \times 28 \times 1$. I primi due numeri rappresentano le dimensioni spaziali dell'immagine, mentre l'ultimo rappresenta il numero di canali.

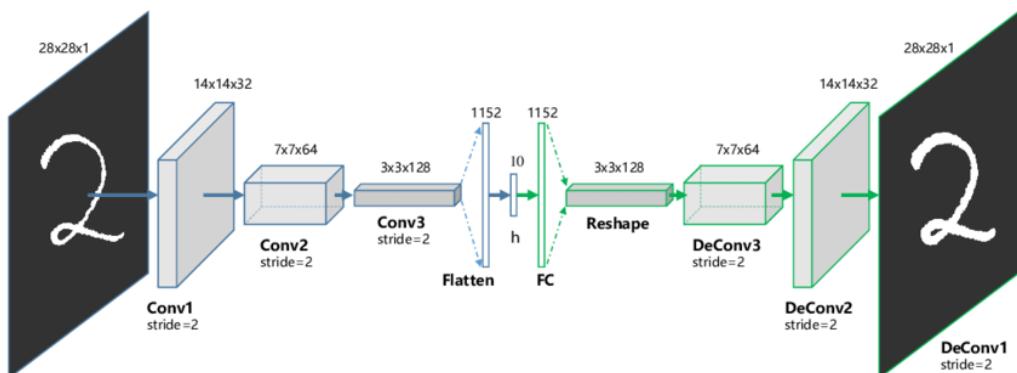


Figura 4.2: Esempio di architettura di un Convolutional Autoencoder con dimensioni di output dei livelli ²

Si può infatti notare come l’obiettivo dei livelli convoluzionali nell’encoder è generalmente quello di ridurre le dimensioni spaziali dell’immagine in input, incrementando allo stesso tempo il numero di canali tramite operazioni di convoluzione, chiamate anche filtraggio. Queste operazioni permettono alla rete di apprendere incrementalmente alcune caratteristiche specifiche dell’input tramite l’utilizzo di diversi filtri che portano a differenti canali di uscita. Ognuno di questi canali rappresenterà una certa caratteristica dell’input. Questo processo avviene iterativamente per alcuni livelli, fino a quando le informazioni spaziali dell’immagine non sono state ridotte a sufficienza e non è stato ottenuto un numero sufficiente di canali. L’output dell’encoder sarà quindi composto da molte piccole immagini a livello spaziale ognuna delle quali rappresenta un canale di uscita che il modello è stato in grado di ricavare tramite convoluzioni successive e che rappresenta per esso una determinata caratteristica dell’immagine originale.

Questo risultato viene poi passato al bottleneck, che si occupa dell’ultima parte della riduzione di dimensionalità. Esso è formato da una piccola rete feed forward, cioè completamente connessa, che non utilizza più livelli convoluzionali ma semplici livelli densi nei quali ogni neurone di un livello è collegato ad ogni neurone del successivo. Questo tipo di rete necessita però di dati unidimensionali e non è in grado di lavorare con dati multidimensionali come le immagini. L’output dell’encoding viene quindi ”appiattito” tramite l’utilizzo di un livello chiamato Flatten fino a raggiungere una sola dimensione e, tramite la sequenza di alcuni livelli di tipo denso, viene ottenuto l’encoding dell’immagine di ingresso della dimensione desiderata. È importante evidenziare che questa parte è la più onerosa in termini di tempo di addestramento proprio per la struttura dei livelli fully connected ed è quindi opportuno renderla il più contenuta possibile.

Una volta ottenuto l’encoding, è necessario ripristinare le dimensioni dell’input tramite la rete di decoder. Per prima cosa, è essenziale riottenere le dimensioni spaziali dell’ultimo livello dell’encoder. A questo scopo, vengono in primo luogo applicati ulteriori livelli feed forward all’encoding e infine eseguito un Reshape per trasformare il risultato nella dimensione desiderata. La rete di decoder è formata da livelli convoluzionali chiamati trasposti, in quanto effettuano l’operazione di convoluzione ”al contrario”. Essi infatti permettono di aumentare le dimensioni spaziali riducendo il numero di canali, all’opposto di ciò che avveniva nella rete dell’encoder. Tramite una sequenza di questi livelli è possibile ripristinare la dimensione iniziale dell’input e ottenere un’immagine che lo rappresenterà nel miglior modo possibile. Essa è stata ricostruita sulla base dell’encoding e quindi rappresenterà la conoscenza contenuta nello stesso. La dif-

²Immagine tratta da [11]

ferenza tra questa immagine e quella di input rappresenta l'errore di ricostruzione che verrà usato per l'addestramento della rete.

Quella appena descritta corrisponde quindi alla struttura generale di un CAE. Nelle prossime sezioni sarà invece analizzata la parte relativa all'implementazione di questo tipo di modello nel caso specifico in esame.

4.2 Il Modello di Estrazione delle Feature

In questa sezione verrà analizzato approfonditamente il modello adottato per l'estrazione delle feature dalle immagini che compongono i dataset prodotti come riportato in 3.2.2. Prima del modello, però, verrà descritta una parte di preprocessing che è stato necessario implementare per permettere l'addestramento dello stesso, che consiste in un downsampling delle immagini d'ingresso. In seguito a questa parte, sarà descritta la struttura del modello, insieme alle motivazioni che hanno portato al compimento di alcune scelte piuttosto di altre. Infine, verranno riportati i parametri adottati per l'addestramento, che sono comuni ai due modelli creati per il processing degli omeri e quello delle scapole.

L'implementazione del modello, così come il suo addestramento, è stata eseguita tramite la libreria Tensorflow. Essa è una famosa libreria open source sviluppata da Google utilizzata per la costruzione e l'addestramento di modelli di reti neurali. È dotata di API ad alto livello chiamate Keras che semplificano molto questo tipo di operazioni, ma che lasciano comunque una vasta possibilità di personalizzazione delle architetture create. Inoltre, essa possiede una community attiva e aggiornata che consente di trovare molto facilmente supporto, risorse e modelli preaddestrati.

4.2.1 Downsampling

Questa fase preliminare alla creazione e addestramento del modello è stata ritenuta necessaria in quanto le immagini memorizzate nei dataset risultavano essere molto grandi. Trattando immagini tridimensionali, il modello dovrà necessariamente essere più complesso e sarà dotato di molti più parametri per l'addestramento. Inoltre, la macchina su cui questo modello verrà addestrato richiederà di conseguenza molte più risorse. Per questo motivo è quindi risultato

necessario ridurre di dimensione queste immagini per permettere al calcolatore utilizzato la loro gestione.

Per effettuare questo tipo di operazione è opportuno che le immagini vengano ridotte di dimensione allo stesso modo in tutte le direzioni. Per far sì che ciò avvenga è necessario prima di tutto trovare un fattore di scala comune tramite il quale ridurre le dimensioni delle immagini e conseguentemente adattarle in modo tale che esse siano tutte divisibili per il fattore scelto. In questo caso specifico è stato scelto un fattore pari a 4. È quindi necessario effettuare un padding delle immagini del dataset prima di effettuare il downsampling in modo tale che ogni dimensione risulti essere divisibile per 4. Nel caso degli omeri, le immagini sono state trasformate da una dimensione pari a 288x210x393 a una dimensione di 288x216x400 e, dopo il downsampling, cioè la riduzione delle dimensioni delle immagini di un fattore pari a 4, la dimensione risulta essere pari a 72x54x100. Queste immagini ridotte saranno quelle che verranno date in input al primo livello del modello. Per quanto riguarda le scapole, in modo analogo, dopo il downsampling si otterranno immagini di dimensione pari a 84x90x102. Pur sapendo che tramite questa operazione si eliminano alcune informazioni sull'immagine stessa, in realtà il metodo utilizzato cerca di preservare il più possibile la struttura originale dell'immagine. Inoltre, essendo immagini binarie, questa operazione risulta più semplice e influisce di meno sulla qualità delle stesse, in quanto le informazioni generali sui contorni e quindi sulla forma delle ossa rimangono pressoché invariate.

4.2.2 Architettura del Modello

Una volta definita la dimensione delle immagini d'ingresso è possibile passare all'analisi dell'architettura del modello vero e proprio, così come è stato utilizzato durante la realizzazione del progetto. Di seguito verranno presentati tutti i livelli che compongono l'architettura con le rispettive dimensioni di output. Verranno poi spiegate una alla volta le scelte che sono state prese e i motivi dietro alle stesse.

L'architettura del modello utilizzato per le scapole è rappresentata nella tabella 4.1. Per brevità sarà riportata solo quest'ultima, in quanto molto simile a quella degli omeri. Le poche differenze saranno comunque evidenziate nel corso della spiegazione. La tabella è strutturata nel seguente modo: la prima colonna mostra il tipo di layer utilizzato con alcune delle sue in-

Livello	Dimensione di output	Parte
Input	(84, 90, 102, 1)	Encoder
Conv 3x3x3, padding, no stride	(84, 90, 102, 32)	
Conv 3x3x3, padding, stride 2	(42, 45, 51, 32)	
Conv 3x4x4, no padding, no stride	(40, 42, 48, 64)	
Conv 3x3x3, padding, stride 2	(20, 21, 24, 64)	
Conv 3x4x3, no padding, no stride	(18, 18, 22, 128)	
Conv 3x3x3, padding, stride 2	(9, 9, 11, 128)	
Conv 3x3x3, no padding, stride 2	(4, 4, 5, 128)	
Flatten	10240	Bottleneck
Dense	128	
Dense	10240	
Reshape	(4, 4, 5, 128)	
Transpose Conv 3x3x3, no padding, stride 2	(9, 9, 11, 128)	Decoder
Transpose Conv 3x3x3, padding, no stride	(9, 9, 11, 64)	
Transpose Conv 3x3x3, padding, stride 2	(18, 18, 22, 64)	
Transpose Conv 3x4x3, no padding, no stride	(20, 21, 24, 32)	
Transpose Conv 3x3x3, padding, stride 2	(40, 42, 48, 32)	
Transpose Conv 3x4x4, no padding, no stride	(42, 45, 51, 16)	
Transpose Conv 3x3x3, padding, stride 2	(84, 90, 102, 16)	
Transpose Conv 3x3x3, padding, no stride	(84, 90, 102, 1)	

Tabella 4.1: Architettura del Convolutional Autoencoder per il dataset contenente le scapole

formazioni, che verranno discusse di seguito. La seconda colonna mostra invece la dimensione dell'output di quel particolare livello. Tutti i livelli in questa architettura sono sequenziali, quindi l'output di un livello corrisponde esattamente all'input del livello successivo. Infine, nell'ultima colonna l'architettura è stata suddivisa nelle sue tre parti principali, che sono state già spiegate approfonditamente nella sezione 4.1.2.

Verranno ora analizzati i livelli uno alla volta, spiegando le scelte prese e alcuni dettagli tecnici. Il primo livello, chiamato Input, non è in realtà un livello vero e proprio, in quanto non esegue alcun processing ma si occupa solamente di copiare l'input, che in questo rappresenta una sola immagine del dataset, come si può anche notare dalla dimensione. Per quanto riguarda i

livelli convoluzionali, i primi tre numeri della dimensione di output rappresentano le dimensioni spaziali dell’immagine, mentre l’ultimo rappresenta il numero di canali.

Di seguito verrà riportata una descrizione generale dei parametri dei livelli convoluzionali del modello. I livelli dell’encoder e del decoder sono entrambi di tipo convoluzionale e possiedono le stesse caratteristiche, con qualche piccola differenza che verrà evidenziata nell’apposita sezione.

Per ogni livello vengono riportate 3 caratteristiche:

- La dimensione del kernel (rappresentata dai numeri successivi alla tipologia di layer): descrive la dimensione della parte addestrabile della rete, che è chiamata kernel. Esso è rappresentato da un tensore di dimensioni arbitrarie necessario per l’operazione di convoluzione e i suoi valori cambiano durante l’addestramento della rete. Per ogni livello esistono molti kernel diversi, pari al numero di canali di output. Ognuno di essi possiede una dimensione pari a quella specificata in tabella moltiplicata per il numero di canali d’ingresso, il tutto aumentato di uno. Ognuno dei valori che forma questi kernel rappresenta un parametro del modello che cambierà durante il suo addestramento. Si può notare che in questa specifica architettura il valore prevalente di dimensione del kernel è pari a $3 \times 3 \times 3$, in quanto esso rappresenta un valore consolidato che funziona molto bene nella maggior parte delle situazioni. Certi livelli hanno però un valore pari a 4 in alcune delle dimensioni. Ciò è stato fatto per un discorso legato alle dimensioni di output, in quanto la dimensione del kernel influisce anche su di esse, insieme agli altri fattori descritti di seguito. Infatti, si cerca sempre di ottenere delle dimensioni di valore pari, in quanto solitamente esse vengono divise per due tramite lo stride, descritto successivamente.
- Padding: per questo parametro sono stati presi in considerazione solo due valori: *padding* oppure *no padding*. Il padding è una pratica comune nei livelli convoluzionali ed è usato prevalentemente per ovviare al problema dell’operazione di convoluzione la quale modifica le dimensioni spaziali dell’immagine di output. Spesso infatti, nei livelli convoluzionali classici, si desidera che queste dimensioni vengano ridotte da livelli appositi, chiamati livelli di pooling, oppure tramite l’utilizzo dello stride nella convoluzione, in quanto tramite questo metodi si è in grado di gestire meglio questa riduzione e renderla indipendente dalla dimensione del kernel. In questo caso l’unico tipo di padding considerato, che è anche il più comunemente utilizzato, è chiamato padding *same*, in quanto effettua automaticamente un padding dell’immagine per far sì che l’operazione di convoluzione non influenzi le dimensioni di output della stessa. In alcuni casi il padding è stato omesso, per far sì che

la convoluzione producesse un output dalle dimensioni pari, mentre nella maggior parte dei casi è stato utilizzato.

- Stride: questo parametro è quello che permette l'effettiva riduzione delle dimensioni spaziali delle immagini nei livelli convoluzionali classici e l'aumento delle stesse nei livelli trasposti. Lo stride consiste nella modifica dell'operazione di convoluzione in modo da permettere di dividere o moltiplicare le dimensioni dell'immagine per un certo fattore di scala. In una normale convoluzione, infatti, il kernel viene fatto "scorrere" sull'immagine spostandolo di una posizione alla volta e quindi l'immagine in uscita risulta avere circa la stessa dimensione di quella in ingresso. Tramite lo stride, invece, è possibile specificare di quanto il kernel debba spostarsi sull'immagine. In questo modo, se ad esempio il valore dello stride è pari a 2, utilizzando un livello convoluzionale classico si ottengono immagini di dimensioni dimezzate mentre tramite un livello trasposto si ottiene un raddoppiamento delle stesse.

Per comprendere al meglio come le dimensioni si modificano in base a questi parametri, di seguito verranno riportate alcune delle formule che permettono di ottenere le dimensioni di uscita da un livello in relazione a una serie di valori di input, sia per il caso di livello standard che per quello trasposto. Per ognuna delle dimensioni dell'immagine, chiamando i il numero di pixel presenti in quella dimensione, k la corrispettiva dimensione del kernel, p il padding effettuato, cioè il numero di pixel di padding aggiunti al contorno, e s il valore dello stride, la corrispettiva dimensione dell'immagine di output per un livello convoluzionale standard si ottiene mediante la formula 4.1, mentre per un livello trasposto tramite 4.2.

$$o = \frac{i + 2p - k}{s} + 1 \quad (4.1)$$

$$o = (i - 1) \times s + k - 2p \quad (4.2)$$

Si può facilmente notare come lo stride e il padding influiscano in modo differente per il calcolo della dimensione di output. Lo stride, infatti, nella prima equazione risulta essere al denominatore mentre nella seconda viene moltiplicato per la dimensione dell'immagine. Inoltre, anche i segni del kernel e del padding sono invertiti, prova del fatto che la convoluzione trasposta si comporta in modo inverso a quella tradizionale e consente di eseguire un upsampling dell'immagine.

Dopo aver spiegato i vari componenti dei livelli convoluzionali, verrà ora analizzata la struttura generale dell'architettura suddividendola nelle tre parti, così come le scelte fatte durante la creazione della stessa.

Encoder

Partendo dall'encoder, è possibile notare come tutti i livelli, ad esclusione dell'input, abbiano una struttura piuttosto rigida, soprattutto per quanto riguarda il parametro stride. L'idea del modello è infatti quella di realizzare una struttura a coppie nella quale ognuna di esse lavora sullo stesso numero di canali di output. Il primo livello della coppia si occupa di aumentare il numero di canali di output rispetto al livello precedente, mentre il secondo possiede uno stride pari a 2 e permette quindi di dimezzare le dimensioni spaziali dell'immagine. Ognuna di queste coppie ha quindi il compito di raddoppiare le dimensioni spettrali e dimezzare quelle spaziali per ogni immagine, riuscendo a raggiungere all'ultimo livello un'immagine di dimensione 4x4x5 e 128 canali. Quest'ultimo livello è isolato e non fa parte di una coppia, in quanto ha il solo scopo di ridurre di un ulteriore fattore pari a 2 le dimensioni spaziali, senza modificarne il numero di canali. Per quanto riguarda il numero di livelli scelti e i valori dei canali di output, essi sono stati ottenuti in seguito a una serie di test. Sostanzialmente il numero di livelli deriva dalla necessità di ottenere in output dall'encoder un'immagine piccola in modo da non avere troppi parametri nel livello successivo di tipo fully connected. Per i canali di output è invece buona norma scegliere sempre delle potenze di due e raddoppiare il loro valore ad ogni livello o ad ogni gruppo di livelli, come nel caso in questione. Questi tipi di scelte non sono comunque mai assoluti e richiedono un certo numero di tentativi per l'identificazione dei corretti valori.

È importante infine evidenziare che ogni livello convoluzionale è dotato di un layer chiamato batch normalization, qui non riportato per questioni di brevità, che si occupa della normalizzazione dei dati. Questi livelli permettono di mantenere la media dell'output prossima a zero e la deviazione standard prossima ad uno. Essi sono molto importanti in quanto aiutano il modello a generalizzare meglio, riducendo il fenomeno dell'overfitting durante l'addestramento, di cui si parlerà nella sezione relativa, e anche a ridurre il numero di epoche necessarie alla convergenza del modello. Questo livello in particolare esegue una normalizzazione su tutto il batch, che è composto dall'insieme di immagini che vengono processate in parallelo dalla rete. Questo layer viene collocato dopo il livello convoluzionale ma prima di eseguire la funzione di attivazione, che è la funzione che calcola l'output di ogni livello. Anch'essa è stata omessa dalla tabella per

motivi di brevità, ma è opportuno nominarla per completezza. In tutti questi livelli la funzione di attivazione scelta è la funzione *relu*, che è diventata uno standard nei livelli intermedi di questo tipo di reti in quanto possiede una serie di vantaggi che la rendono preferita rispetto ad altre.

Bottleneck

Passando ora alla seconda parte, cioè il bottleneck, si nota immediatamente che l'output del primo livello possiede una sola dimensione. Questa tipologia di livello, chiamata flatten, si occupa infatti solamente di "appiattire" l'immagine in uscita dall'encoder, posizionando i pixel uno accanto all'altro fino ad ottenere un vettore monodimensionale. Ciò è necessario per il processing dei dati da parte dei livelli densi successivi, che come già detto non possono gestire dati multidimensionali. A seguito del flatten sono stati inseriti due livelli di tipo Dense. Essi, a differenza degli altri, sono livelli di tipo fully connected, nei quali ogni elemento di un livello è direttamente collegato a quello del successivo. Il numero di parametri del livello risulta essere quindi molto elevato, in quanto si ottiene moltiplicando la dimensione d'ingresso per quella di uscita. Per questo motivo si cerca di limitare il più possibile questi tipi di livelli e sfruttare quelli convoluzionali, che hanno la caratteristica di avere pesi condivisi e quindi un minor numero di parametri. Per lo scopo in questione, però, sono necessari almeno due livelli Dense: il primo si occupa di ridurre la dimensione fino a 128, che è stata scelta come dimensione dell'encoding, mentre il secondo si occupa di ripristinare la dimensione in uscita dal livello flatten. In seguito ai due livelli fully connected è presente un ultimo livello che svolge il lavoro inverso del flatten, cioè ripristina le dimensioni dell'immagine in uscita dall'encoder per permettere il decoding della stessa. Nel caso del bottleneck, l'unica scelta importante che è stata effettuata riguarda la dimensione dell'encoding, che è molto rilevante in quanto indica il numero di feature che rappresenteranno ogni immagine. Questo numero è stato scelto in primo luogo in modo tale da essere una potenza di due, convenzione che è sempre buona norma utilizzare per qualsiasi valore in queste reti, come ad esempio i canali di uscita dei layer convoluzionali, in quanto sono numeri molto più semplici da elaborare per il calcolatore. Inoltre, è stato ritenuto opportuno fornire in uscita un numero di feature superiore a 100 ed è quindi stata scelta la potenza di due più vicina a tal valore.

Anche in questa parte sono stati omessi dei livelli che si occupano della regolarizzazione dei dati, chiamati livelli di dropout. Essi eliminano ad ogni step dell'addestramento una certa

percentuale di neuroni dal livello in modo casuale. Ciò consente di ottenere una maggior generalizzazione da parte del modello riducendo la possibilità di overfitting. In breve, l'idea è quella di evitare che i neuroni vicini tra di loro dipendano l'uno dall'altro e quindi si eliminano in modo casuale ad ogni iterazione alcuni di essi in modo tale che le caratteristiche apprese dagli stessi siano il più possibile indipendenti tra di loro, diminuendo così l'errore di generalizzazione. In questo modello, il dropout è stato applicato prima di ogni livello denso, con una probabilità di dropout pari a 0.3, che rappresenta la probabilità che un neurone venga rimosso.

Decoder

L'ultima parte del modello è quella riguardante il decoder. Esso è composto interamente da livelli di tipo Transpose Convolution, che come già accennato effettuano la stessa operazione di convoluzione di quelli standard ma su un'immagine d'ingresso modificata. Quest'immagine viene modificata grazie ai valori di padding e stride, che si comportano in un modo differente, come evidenziato anche dalla formula 4.2. Questo tipo di livello permette quindi di effettuare un'operazione di upsampling dell'ingresso, necessaria al decoder per permettere al modello di ottenere in output un'immagine della stessa dimensione di quella di input partendo da un'immagine piccola a livello spaziale. Ciò viene attuato aumentando iterativamente le dimensioni spaziali e diminuendo di conseguenza il numero di canali in uscita ad ogni livello. Anche in questo caso, come nell'encoder, il modello è suddiviso in coppie di livelli, il secondo dei quali possiede uno stride pari a 2 che è necessario per l'upsampling. Il primo e l'ultimo livello fanno un'eccezione nella struttura. Il primo svolge un ruolo analogo all'ultimo dell'encoder ed è stato inserito per simmetria con lo stesso. L'ultimo livello, invece, è predisposto per la sola riduzione dei canali di uscita fino ad un valore pari ad uno, in quanto la dimensione di uscita del modello deve coincidere con quella d'ingresso. Inoltre, quest'ultimo possiede una funzione di attivazione differente rispetto agli altri, in quanto è necessario ottenere dei valori di output per ogni voxel compresi tra 0 e 1. Per questo scopo è stata utilizzata la funzione sigmoidale che consente, dato un qualsiasi valore di input, di ottenere un valore di output compreso tra 0 e 1. Idealmente, l'obiettivo sarebbe quello di ottenere in output immagini binarie, ma ciò non è possibile in quanto il modello produce come valore un numero floating point che indica la probabilità che quel determinato voxel sia pari ad uno.

Anche nel decoder, analogamente all'encoder, sono stati introdotti livelli di batch normalization per la normalizzazione dei dati.

Il modello dell’omero risulta essere molto simile a quello della scapola ad eccezione di qualcuno dei valori di padding e di stride dovuti alla diversa dimensione dell’immagine di input. La struttura generale però, così come le scelte che sono state effettuate, è analoga alla precedente e non necessita di ulteriori commenti. Nella prossima sezione verranno analizzati e spiegati i principali parametri dell’addestramento comuni a entrambi i modelli.

4.2.3 Parametri dell’Addestramento

Questa sezione si occupa di elencare e spiegare i principali parametri riguardanti l’addestramento dei modelli come la scelta dell’optimizer, della loss function e delle metriche utilizzate per la valutazione.

Per quanto riguarda l’optimizer, è stato utilizzato Adam con learning rate pari a 10^{-4} . L’optimizer rappresenta l’algoritmo che permette l’aggiornamento dei pesi della rete durante l’addestramento. Adam è diventato uno standard per l’ottimizzazione di reti neurali in quanto funziona in modo ottimale nella maggior parte delle applicazioni. In breve, esso consente di assegnare un learning rate adattivo ad ogni parametro sulla base della storia dei gradienti. Ciò permette al modello di convergere più velocemente e in modo più accurato. Il learning rate è un parametro fondamentale di un optimizer in quanto indica il fattore di moltiplicazione dei pesi ad ogni iterazione. Più esso è alto, più l’optimizer convergerà velocemente con il rischio però di cambiamenti più drastici nei pesi. Un learning rate più basso comporta invece solitamente una maggiore stabilità, ma aumenta il tempo di addestramento del modello e può portare a situazioni in cui lo stesso finisce in una condizione di stallo. Il valore di 10^{-4} è stato selezionato e seguito di una serie di prove. I valori tipici si aggirano spesso comunque tra valori compresi tra 10^{-3} e 10^{-5} .

Per quanto riguarda la funzione di loss, è stata utilizzata una Binary Crossentropy. La funzione di loss ha il compito di calcolare l’errore commesso dal modello e il suo valore determina l’aggiornamento dei pesi. È quindi fondamentale scegliere la funzione corretta in base al tipo di output che il modello produce. Se l’output è compreso nel range [0,1], come in questo caso, una delle scelte migliori è quella di utilizzare la funzione Binary Crossentropy. L’obiettivo del modello durante il suo addestramento è quello di minimizzare il valore di questa funzione di errore. Per conseguire questo obiettivo esso cercherà, tramite l’optimizer, di minimizzare il

gradiante della loss modificando i suoi pesi. La formulazione di questa funzione è riportata nell'equazione 4.3, nella quale I rappresenta l'immagine, N indica il numero totale di voxel dell'immagine, y_i rappresenta il valore del voxel i nell'immagine del ground truth, che nel caso di un Autoencoder è l'immagine d'ingresso, e \hat{y}_i è il corrispondente voxel dell'immagine di uscita, cioè la predizione del modello.

$$BCE = -\frac{1}{N} \sum_{i \in I} (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)) \quad (4.3)$$

Questa funzione effettua quindi una media dei valori ottenuti dalla formula all'interno della sommatoria per ogni voxel dell'immagine. Quest'ultima produce un valore tanto più alto quanto il valore predetto risulta essere vicino a quello reale. Da notare che il modello produce dei valori compresi tra 0 e 1 e non esattamente binari, come già evidenziato anche nella sezione 4.2.2. Il valore finale della BCE sarà quindi tanto più basso quanto più il modello produce delle predizioni accurate, in quanto è presente un segno meno che anticipa la sommatoria. Questo comportamento è corretto in quanto l'obiettivo del modello è appunto quello di minimizzare il valore di questa funzione.

Per quanto riguarda le metriche per la valutazione del modello durante il training, la più comunemente utilizzata è l'accuracy. In questo specifico caso, però, producendo il modello delle immagini in uscita che hanno il compito di riprodurre quelle in ingresso, è stato considerato più opportuno utilizzare un'altra metrica chiamata Dice coefficient. Questo coefficiente è utilizzato in generale per misurare la similarità tra due insiemi, ma il discorso è analogo quando si parla di sovrapposizione di immagini. È necessario ricordare che le immagini trattate sono binarie, in quanto consistono in maschere di segmentazione, e ciò rende possibile la semplice applicazione di questo tipo di metrica. Infatti, esse sono facilmente considerabili come degli insiemi in quanto possiedono solamente due valori. Il valore del Dice coefficient è compreso tra 0 ed 1, dove 0 indica che le due immagini non hanno alcuna sovrapposizione mentre 1 indica che le due immagini sono completamente sovrapposte. La formulazione è riportata nell'equazione 4.4, nella quale il numeratore indica il doppio della cardinalità dell'intersezione dei due insiemi mentre il denominatore indica la cardinalità dell'unione dei due.

$$Dice = \frac{2 \times |A \cap B|}{|A \cup B|} \quad (4.4)$$

Per quanto riguarda le immagini, è più intuitivo pensare al numeratore come il doppio dell'area di overlap tra le due e al denominatore come l'area totale occupata dalle stesse. Da questa interpretazione risulta immediato come questa metrica sia perfettamente adatta allo scopo

in questione. Avendo a disposizione maschere binarie di segmentazione, questo coefficiente permette un’immediata misura della capacità del modello di generare un’immagine il più simile possibile a quella in ingresso. Ciò permette di considerare anche piccoli dettagli dei contorni delle ossa che, come sarà in seguito mostrato, consistono nelle parti più importanti, ma anche più complicate, da apprendere per il modello.

Infine, gli ultimi parametri considerati sono il numero di epoch, il batch size e il validation split. Per quanto riguarda le epoch, esse rappresentano il numero di iterazioni compiute dall’addestramento. Ad ogni iterazione, vengono fornite in ingresso al modello tutte le immagini del training set che vengono processate e permettono l’aggiornamento dei pesi. Questi ultimi non vengono aggiornati per ognuna delle immagini di input, ma esse vengono raggruppate in batch che sono gruppi di immagini processate in parallelo dal modello. La dimensione del batch è rappresentata dal parametro batch size. Più il batch size è grande, più immagini vengono processate contemporaneamente. Inoltre, l’impatto di ogni immagine sull’aggiornamento dei pesi del modello risulta essere inferiore, in quanto essi vengono aggiornati solamente dopo che i valori di loss ottenuti sottponendo al modello tutte le immagini del batch sono stati calcolati. La macchina necessita anche di più risorse per la gestione delle immagini in parallelo in quanto esse utilizzano più memoria della GPU. Il calcolatore qui usato per l’addestramento, infatti, faticava a gestire batch size superiori a 16, che è quindi il valore che è stato adottato. Una delle limitazioni di avere un batch size piccolo consiste però nella possibilità di ottenere un training molto rumoroso in quanto ogni singola immagine influenza l’aggiornamento di tutti i pesi del modello. Con un batch size più grande, invece, l’impatto di un’immagine è mediato dalle altre e i pesi vengono aggiornati soltanto una volta che esse sono state tutte processate.

L’ultimo parametro è chiamato validation split. Il suo valore indica la percentuale di training set che si vuole utilizzare come validation set. Il training set rappresenta l’insieme delle immagini sulle quali il modello viene addestrato. L’obiettivo, solitamente, è quello di permettere al modello di essere in grado, dopo il training, di generalizzare e funzionare in modo ottimale anche con immagini su cui non è stato addestrato. Per ottenere una stima di questa capacità durante l’addestramento, alcune immagini del training set non vengono fornite al modello, ma vengono utilizzate per misurare la sua capacità di generalizzazione durante il training. Queste immagini formano il validation set. La percentuale di immagini del training set dedicate al validation qui scelta è pari al 20%. La limitazione principale sta nel fatto che queste immagini non possono essere usate per l’addestramento e quindi, soprattutto in presenza di pochi dati, il mo-

dello potrebbe non riuscire ad apprendere abbastanza. Al contrario, però, è molto importante considerare questo insieme di immagini perché consente di rilevare il fenomeno dell’overfitting. Esso accade quando il modello non è più in grado di generalizzare perché impara a classificare le immagini del training set troppo accuratamente e quindi si specializza troppo sulle stesse. Questo fenomeno è facilmente individuabile tramite l’utilizzo di un validation set. Quando durante l’addestramento le performance su questo insieme diminuiscono drasticamente, ciò è segno che il modello sta iniziando ad andare in overfitting ed è opportuno interromperne il training.

È necessario però evidenziare che gli Autoencoder sono dei tipi di modelli particolari, soprattutto quando usati a scopo di estrazione delle feature. Infatti, l’obiettivo di questi ultimi è quello di estrarre delle feature dalle immagini presenti nel training set e non tanto la possibilità di estrarre da immagini esterne. In questo caso, quindi, un leggero overfitting è concesso. È opportuno però che quest’ultimo non diventi troppo preponderante. In tal caso infatti il modello non sarebbe più in grado di estrarre feature che riassumano le caratteristiche generali dell’immagine perché avrebbe imparato esattamente a ricostruire ogni immagine del training set producendo feature prive di un significato generale. Questo è il motivo per cui, seppur di dimensione contenuta, è stato deciso di utilizzare un validation set anche per questo compito.

Una volta descritti tutti i parametri generali, comuni ad entrambi i modelli, nella prossima sezione sarà riportata l’analisi dei risultati ottenuti dagli addestramenti degli stessi, analizzando l’andamento della funzione di loss e del Dice coefficient ad ogni epoca.

4.3 Analisi dei Risultati: Addestramento e Predizioni

Dopo aver analizzato approfonditamente l’architettura dei modelli, verranno di seguito discussi i risultati derivanti dall’addestramento degli stessi. La ricerca dei parametri migliori descritti nella sezione 4.2.2 ha richiesto un certo numero di addestramenti preliminari, ma di seguito verrà riportato solamente quello con i parametri finali. Durante l’analisi dell’addestramento verranno anche mostrate le predizioni effettuate dal modello su alcune immagini, in modo da fornire un’interpretazione visiva dell’evoluzione subita dallo stesso. Nelle prossime sezioni saranno analizzati approfonditamente gli addestramenti e le predizioni di entrambi i modelli, commentando contemporaneamente i risultati ottenuti dagli stessi.

4.3.1 Modello degli Omeri

Entrambi i modelli sono stati addestrati per un totale di 250 epoch. Nelle prime 200 è stato utilizzato un validation set, mentre per le ultime 50 il modello è stato addestrato sul dataset completo. Ciò è stato fatto per permettere al modello di apprendere la modalità di estrazione delle feature da ognuna delle immagini fornite, senza eliminare la presenza del validation set necessaria al rilevamento dell'overfitting. Il numero di epoch è stato scelto a seguito di una serie di prove, ma non è facile determinarne il numero ideale per gli Autoencoder in quanto essi, come già affermato, sono modelli particolari che non rispettano esattamente gli accorgimenti dei modelli standard per quanto riguarda la terminazione dell'addestramento. Solitamente infatti, nei modelli normali, l'addestramento viene interrotto poco dopo che le performance sul validation set iniziano a decrescere, cioè al presentarsi del fenomeno dell'overfitting. In questi tipi di modelli, invece, un lieve overfitting è accettato grazie al particolare utilizzo che si fa degli stessi. Per questo non è sempre semplice determinare il numero ideale di epoch.

Ogni 50 epoch, i pesi del modello sono stati salvati in modo da poter apprezzare i cambiamenti che esso subisce nel tempo fornendo figure rappresentanti esempi di predizioni da esso generate. Verranno inoltre riportati i grafici che rappresentano l'evoluzione della funzione di loss e del Dice coefficient del training set e del validation set tra le varie epoch. Ciò permetterà di analizzare l'andamento dell'addestramento nel tempo.

Prime 50 epoch

Le prime epoch sono solitamente quelle in cui il modello impara maggiormente, in quanto inizialmente i pesi sono impostati in modo casuale ed è proprio durante queste prime epoch che essi iniziano ad ottenere un significato per lo stesso.

Dalla figura 4.3 si può infatti notare come la curva della loss sia molto ripida inizialmente per poi attenuarsi intorno a 10 epoch. Inoltre, la riduzione maggiore della loss avviene proprio nelle prime 3 o 4 epoch, in cui il modello apprende le feature principali delle immagini in ingresso. Dalla epoca 10 alla 50 si può notare come la loss inizi ad avere un valore più stabile che decresce molto più lentamente, segno che il modello sta cercando di apprendere i dettagli delle immagini. Per quanto riguarda il rapporto con la validation loss, il comportamento è assolutamente accettabile in quanto essa rimane in prossimità della training loss e decresce lentamente seguendo l'andamento della stessa. Delle informazioni più pratiche sulla qualità

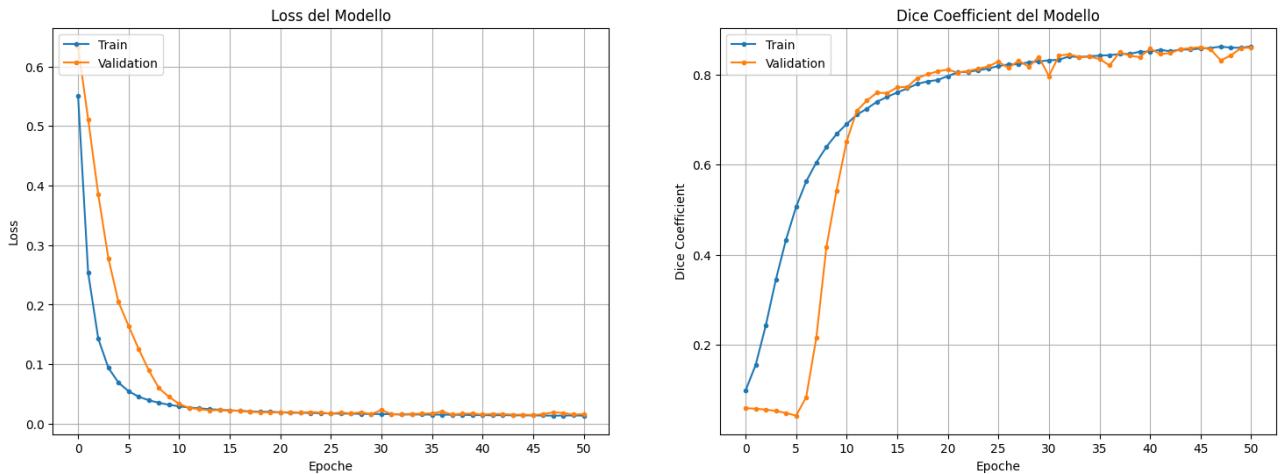


Figura 4.3: Loss e Dice Coefficient nelle prime 50 epoche del modello dell’omero

delle predizioni del modello possono essere fornite dal Dice coefficient, che come già detto indica la misura di sovrapposizione tra le immagini di input e le corrispettive in output. Anche in questo caso la forma è a ramo di iperbole, ma al contrario rispetto alla loss in quanto il modello proseguendo con l’addestramento riesce ad apprendere sempre meglio come ricostruire le immagini di input e di conseguenza l’area di overlap tra l’input e l’output aumenta. Da questo grafico è possibile notare altri dettagli che da quello della loss non risultano così chiari. Sul validation set, questo coefficiente rimane pressoché costante fino alla quinta epoca, dalla quale inizia a crescere molto rapidamente. Ciò è probabilmente segno del fatto che durante le prime 5 epoche il modello stia apprendendo delle informazioni, in quanto la loss diminuisce, ma non è ancora in grado di produrre feature abbastanza generali da permettere una buona ricostruzione delle immagini non appartenenti al training set. Dalla quinta epoca in poi, invece, il gap tra training e validation inizia a ridursi fino a diventare molto piccolo intorno alla decima epoca, punto in cui anche la differenza tra le due loss non è più così ampia. Dal grafico del Dice coefficient si intuisce anche meglio il rapporto tra loss e validation nelle epoche successive alla decima: il loro valore è molto simile e in certe epoche capita addirittura che il coefficiente sul validation set superi leggermente quello sul training, comportamento accettabile nelle prime epoche di addestramento. Ciò che permette di capire che il modello sta apprendendo e anche generalizzando è il costante aumento della metrica sia sul training che sul validation.

Per ottenere anche un’interpretazione visiva dell’evoluzione del modello e di ciò che esso praticamente produce, verranno riportate come esempio alcune slice prese da una particolare immagine appartenente al training set, insieme all’output che il modello produce per quelle slice e lo stesso output binarizzato. Nella figura 4.4 sono riportate 16 slice ottenute dall’immagine numero 150

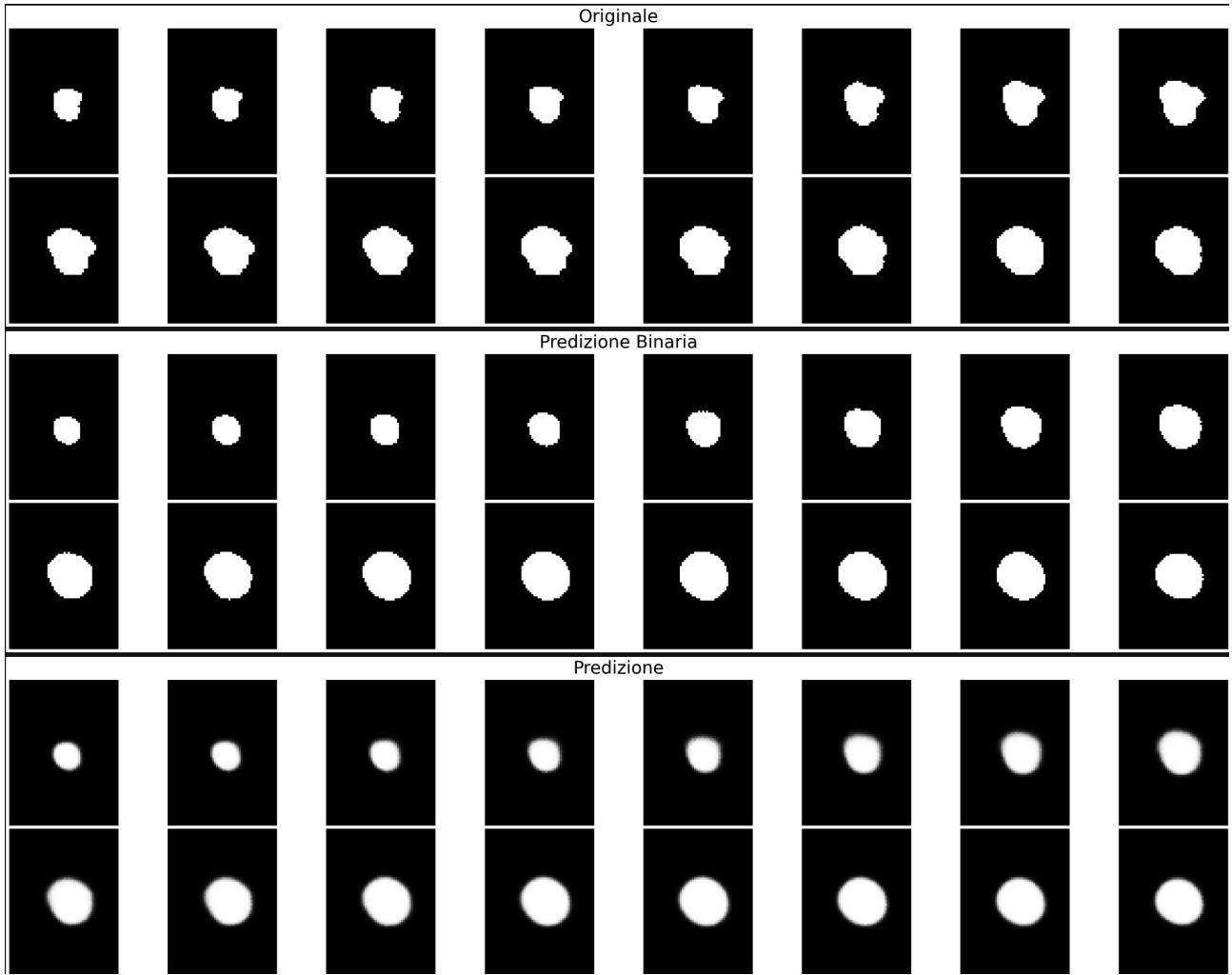


Figura 4.4: Confronto di alcune slice dell’omero dopo 50 epochhe

che rappresentano sezioni della testa dell’omero, che è la parte fondamentale dell’osso per il compito in questione. Si può infatti notare come nell’immagine originale certe slice abbiano una forma molto particolare che è importante che il modello impari a ricostruire nei dettagli. Le slice binarizzate sono state ottenute impostando ad 1 i voxel con valore superiore a 0.5 e a 0 quelli con valore inferiore, in modo da produrre un’immagine più definita e facilitare il confronto con l’immagine di input. La vera predizione del modello è riportata invece nell’ultima parte della figura. I diversi toni di grigio in questo caso indicano che i valori non sono binari ma sono compresi tra 0 e 1. Si può notare come sui contorni il modello sia più incerto, in quanto essi sono meno definiti e i colori più scuri, mentre nel centro dell’osso il colore è molto vicino al bianco. La forma dell’immagine prodotta dal modello è ancora molto arrotondata rispetto a quella originale, segno che esso necessita del proseguimento dell’addestramento per essere in grado di ricostruire anche i dettagli sulla forma dei contorni. È importante ricordare infatti che lo scopo finale di questo lavoro consiste nel riconoscimento dei morfotipi, che riguardano

la particolare conformazione delle ossa, ed è quindi molto importante che il modello sia in grado di ricostruirne anche i dettagli. Una volta che il modello avrà ottenuto la conoscenza necessaria, significherà che nelle feature saranno presenti delle informazioni che permettono la ricostruzione degli stessi. Il Dice coefficient su questa immagine dopo 50 epoche risulta essere già pari a 0.88 in quanto l'area di overlap è per la maggior parte quella centrale dell'osso. Da questo momento il miglioramento del valore di questo coefficiente sarà molto più limitato ma sarà interamente dovuto al perfezionamento della forma dei contorni e quindi alla parte più importante, ma anche più complicata, da imparare per il modello.

Epoche Successive

Le rimanenti 150 epoche sono state inserite tutte nello stesso grafico in quanto l'evoluzione del modello è piuttosto costante e i valori di loss e Dice coefficient variano molto poco. Questa piccola variazione ha però un grande impatto in quanto, come sarà meglio mostrato nelle prediction, il modello sarà in grado di perfezionare i contorni delle immagini predette.

La figura 4.5 mostra quindi il grafico relativo all'evoluzione del modello tra le epoche 50 e 200. In generale, si può notare come la loss sul training continui lentamente a diminuire mentre per quanto riguarda il validation è opportuno evidenziare diversi aspetti. Innanzitutto, dal grafico della loss, si nota chiaramente che l'andamento è più incostante, con valori che oscillano e una differenza massima tra un valore e il successivo di 0.01 nella situazione peggiore. Le oscillazioni sembrano molto elevate, ma è necessario notare che la scala sull'asse delle y è diversa da quella del precedente grafico in quanto adesso esso è concentrato su una porzione di addestramento in cui i valori subiscono variazioni minime. In realtà queste oscillazioni sono piuttosto limitate e quindi considerate accettabili. Si può inoltre notare come questo valore stia iniziando lentamente ad aumentare ad ogni epoca. Questo è segno di overfitting, in quanto il modello continua ad apprendere sempre meglio sulle immagini di training mentre inizia ad avere difficoltà a generalizzare sulle immagini del validation. Come già affermato più volte, però, nel caso degli Autoencoder è possibile accettare un leggero overfitting in quanto l'obiettivo primario è quello di classificare le immagini presenti nel training set. In questo caso, effettivamente, l'overfitting non è eccessivo in quanto la loss cresce molto lentamente. Si può notare come, in circa 150 epoche, essa sia aumentata solamente di una quantità pari a 0.015 nei casi peggiori. Ciò avviene a favore dell'apprendimento sul training set sul quale la loss, seppur lentamente, continua a diminuire in modo pressoché costante. Il comportamento del Dice coefficient è

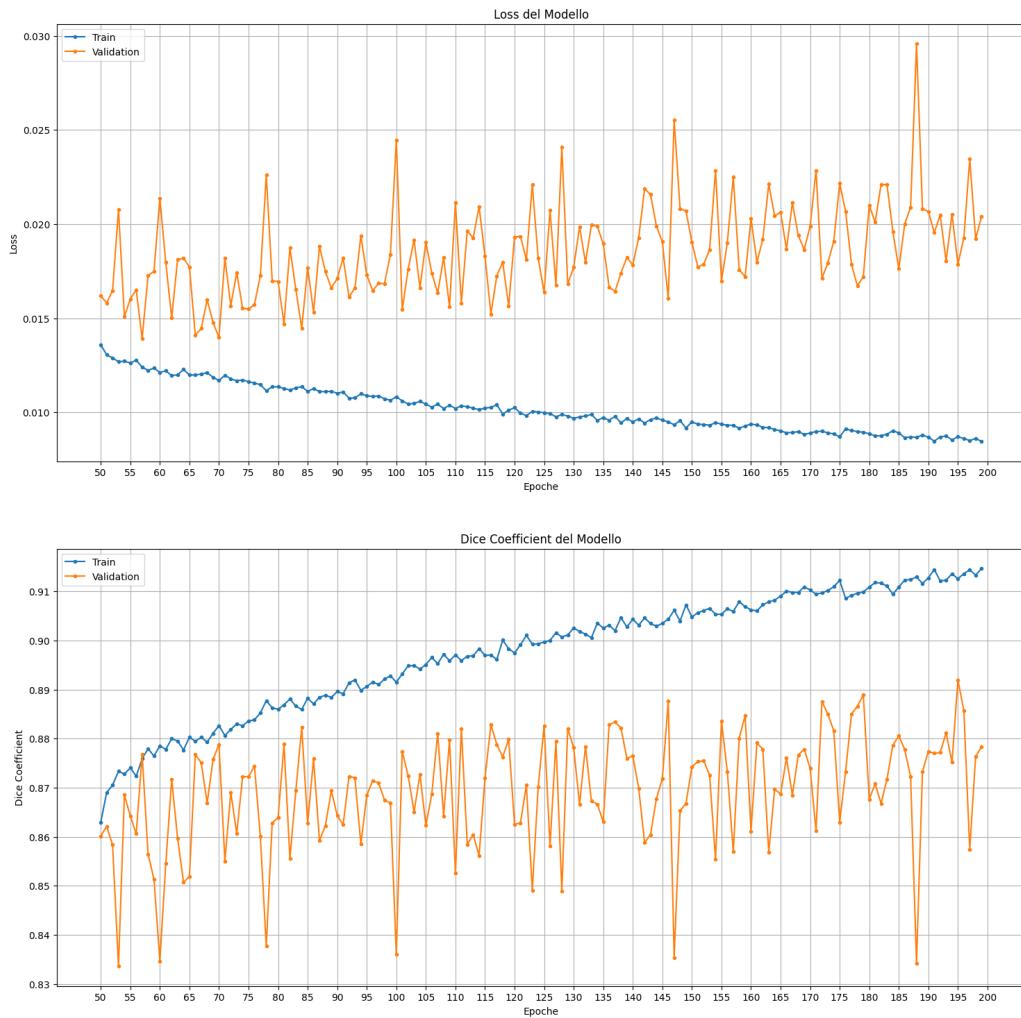


Figura 4.5: Loss e Dice Coefficient nelle epoche dalla 50 alla 200 del modello dell’omero

correlato a quello della loss, anche se si può notare come in questo caso, pur essendo molto oscillante, il Dice coefficient sul validation non decresca all’aumentare delle epoche, ma il suo comportamento rimanga piuttosto costante, con picchi massimi che possiedono un valore di circa 0.88. Questo è segno che il modello non sta sbagliando a predire le immagini del validation set, ma solamente non riesce ad andare più nel dettaglio sui contorni delle stesse. Questo fatto, per la specifica applicazione di estrazione delle feature, non risulta essere critico. L’aspetto positivo è invece quello del continuo miglioramento di questo coefficiente sul training, che raggiunge un valore pari a 0.915 ed è in continua crescita.

La parte più significativa è quella che riguarda l’evoluzione delle prediction del modello durante il suo addestramento. Saranno di seguito riportate alcune delle slice mostrate al termine delle prime 50 epoche con le corrispondenti predizioni del modello dopo 100, 150 e 200 epoche. L’immagine considerata è la stessa mostrata nella figura 4.4 e rappresenta, come già

detto, un’immagine del training set. Questo permetterà di mostrare l’evoluzione dell’output del modello durante il suo addestramento.

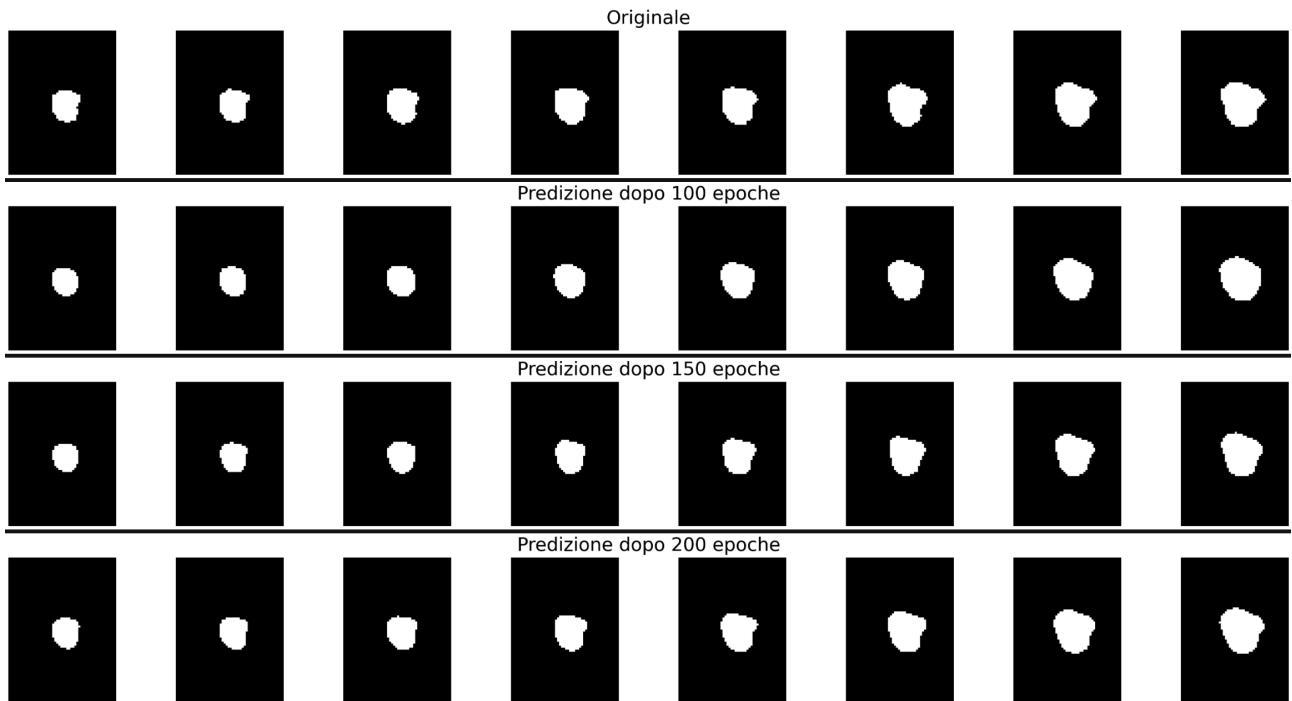


Figura 4.6: Confronto di alcune slice dell’omero dopo 100,150 e 200 epoche

Dalla figura 4.6 risulta essere chiara l’evoluzione del modello al proseguire dell’addestramento. Si può infatti notare come dopo 100 epoche le predizioni siano ancora piuttosto arrotondate, ma già dopo 150 epoche si riescano ad intravedere i primi dettagli che sono poi enfatizzati all’epoca 200. Anche dopo 200 epoche, però, la forma non è ancora esattamente la stessa di quella originale. L’obiettivo dell’addestramento non è infatti quello di permettere al modello di riprodurre esattamente l’immagine di input, altrimenti si avrebbe un fenomeno di overfitting troppo marcato. Dal momento in cui il modello inizia a dare forma ai contorni, significa che esso è in grado di individuare delle caratteristiche che in uscita portano l’immagine ad avere la giusta forma e a inserirle quindi nelle feature prodotte. Lasciando proseguire ulteriormente l’addestramento ci sarebbe il rischio che il modello continui ad apprendere sempre più dettagliatamente i contorni delle immagini di training, mentre quelle di validation peggiorerebbero ulteriormente. Se ciò accadesse il rischio sarebbe quello di estrarre delle feature poco significative, inadatte cioè a descrivere nel miglior modo possibile le immagini fornite.

Una volta concluso l’addestramento, il prossimo e ultimo passo consiste nell’addestrare il modello per ulteriori 50 epoche su tutto il dataset, compreso anche il validation set. Questo

permette ad esso di apprendere la corretta modalità di estrazione delle feature anche da queste ultime, in quanto anch'esse fanno parte delle immagini che dovranno infine essere classificate. Per i motivi sopra riportati, questo addestramento non deve continuare per troppe epoche, altrimenti si riproporrebbe il problema dell'overfitting, che senza validation set è molto difficile da individuare.

4.3.2 Modello delle Scapole

In questa sezione verrà invece mostrato l'addestramento del modello utilizzato per le scapole. Alcuni aspetti sono comuni all'addestramento dell'altro modello e verranno pertanto tralasciati. Saranno però evidenziate le differenze più importanti che saranno dovute specialmente alla diversità tra la conformazione degli omeri e quelle delle scapole.

Per quanto riguarda i grafici della loss e del Dice coefficient, nel caso delle scapole sarà riportato un unico grafico riguardante tutte le 200 epoche. In questo caso è infatti più importante il comportamento generale del modello, in quanto nelle prime 50 epoche esso è simile a quello sugli omeri. Dalla figura 4.7 l'andamento dell'addestramento risulta piuttosto simile a quello dell'omero, segno che anche in questo caso il modello è stato in grado di apprendere delle caratteristiche dell'input ad esso fornito. Il grafico della loss in questo caso è molto meno significativo in quanto essa decresce molto velocemente all'inizio per poi attenuarsi dopo già circa 15 epoche ed è quindi difficilmente distinguibile dal grafico la sua variabilità. Si può però notare come essa, seppur lentamente, continui a decrescere fino all'ultima epoca sul training mentre sul validation l'andamento iniziale è decrescente ma poi si discosta intorno a 80 epoche e si stabilizza, senza però mai peggiorare evidentemente.

Il grafico che fornisce più informazioni è però quello che riguarda il Dice coefficient, in quanto si nota maggiormente l'apprendimento del modello e il gap tra il training e il validation è più evidente. L'inizio è analogo a quello dell'omero, ma in questo modello sul validation esso impiega più tempo per iniziare a salire di valore: infatti, esso rimane costante per circa 9 epoche contro le 5 del modello dell'omero. Il suo valore rimane per molte epoche in prossimità di quello di training, talvolta superandolo anche nelle prime 50 epoche. Dall'epoca 80 in poi, invece, il Dice coefficient sul validation si stabilizza mentre quello sul training continua ad aumentare e quindi il gap si evidenzia molto di più, analogamente alla situazione del modello

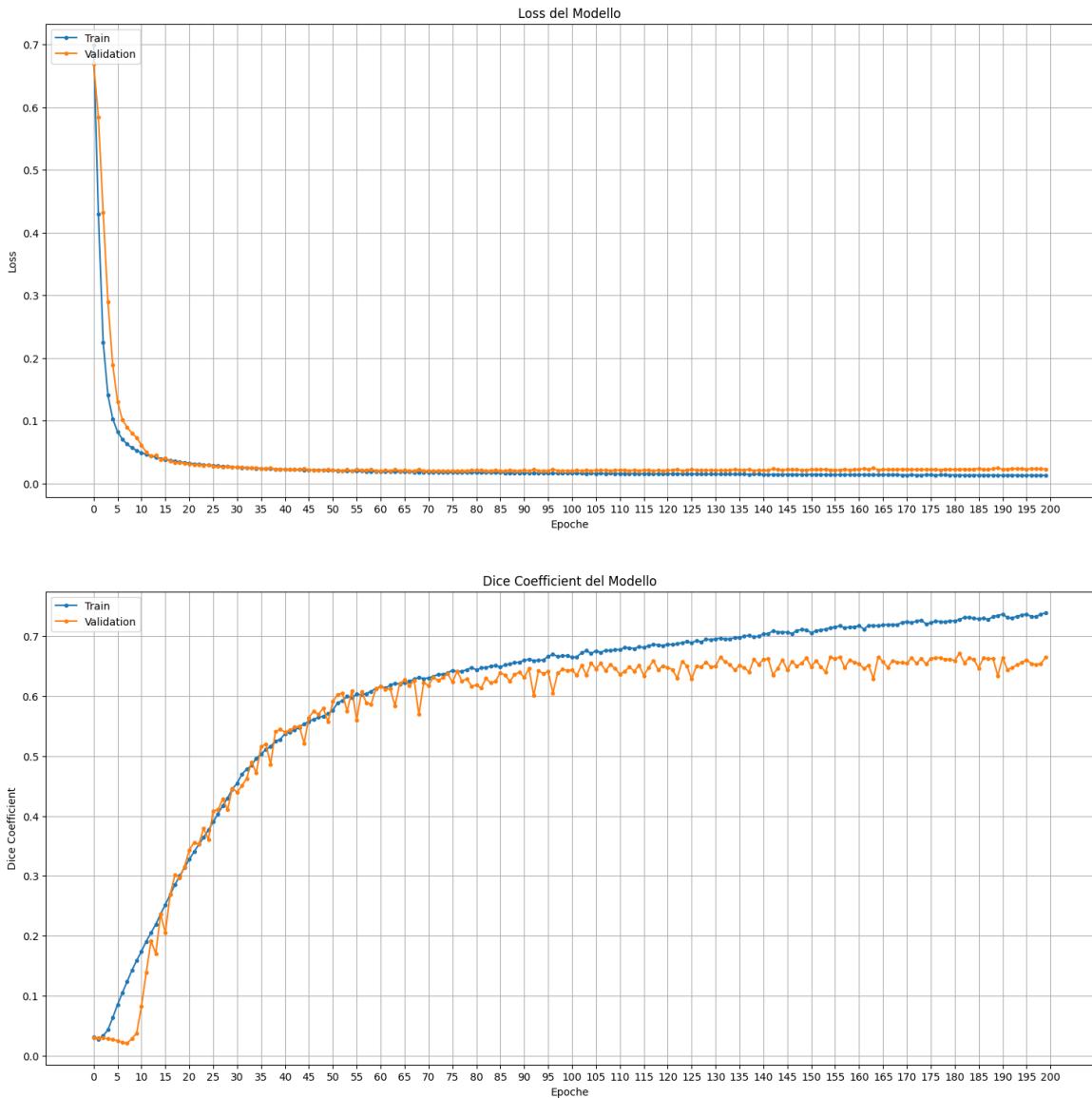


Figura 4.7: Loss e Dice Coefficient in tutte le 200 epoche del modello della scapola

degli omeri. Si può notare che il valore massimo raggiunto per le scapole è molto inferiore a quello dell'omero, in quanto dopo 200 epoche sul training si ottiene un Dice coefficient di circa 0.75, in contrapposizione allo 0.915 del modello degli omeri. Queste differenze sono dovute essenzialmente alla diversa forma delle due ossa. L'omero, infatti, possiede una forma meno elaborata della scapola, che invece è più complicata da ricostruire in quanto provvista di molti dettagli importanti e forme più complesse. Questo aspetto è facilmente intuibile osservando le predizioni del modello di seguito riportate.

Nella figura 4.8 sono mostrate le diverse predizioni del modello ad epoche differenti di un'immagine della scapola appartenente al training set. Gli output in questo caso sono stati tutti binarizzati per rendere la visualizzazione più semplice e definita. Innanzitutto si può notare co-

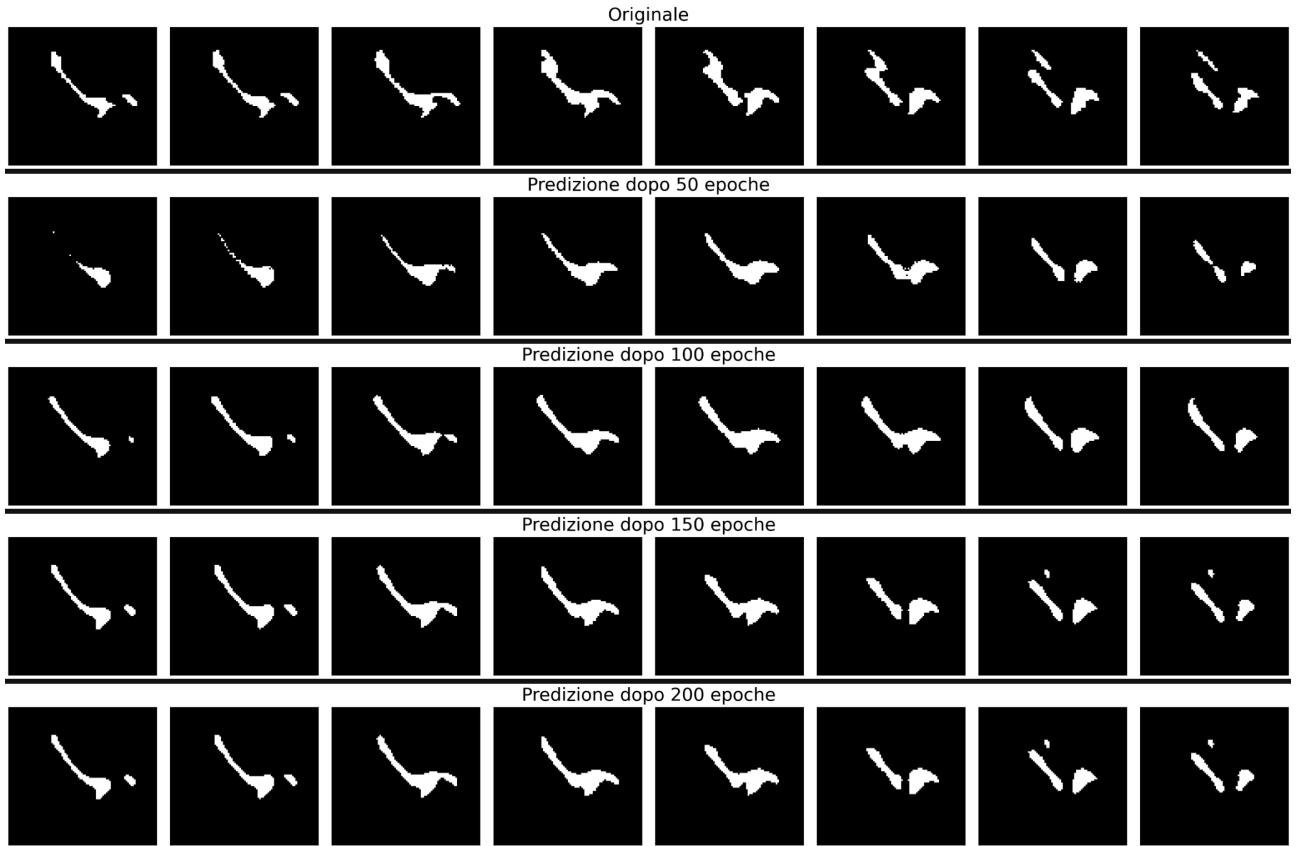
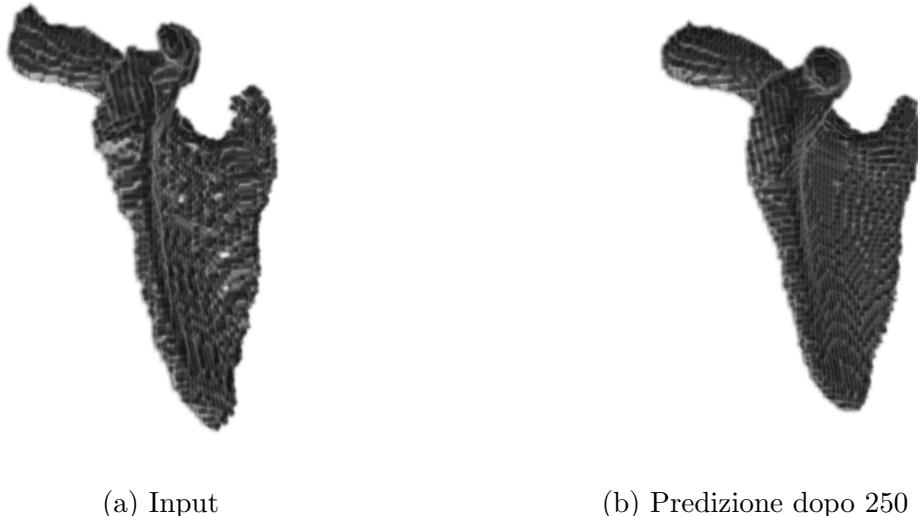


Figura 4.8: Confronto di alcune slice della scapola dopo 50, 100, 150 e 200 epoche

me l'area della scapola in ogni slice sia molto più limitata di quella dell'omero ed essa possiede forme più complesse da apprendere per il modello. Questo fatto permette di intuire facilmente il motivo per cui il valore del Dice coefficient per le scapole risulti essere molto più ridotto rispetto al caso degli omeri. Infatti, possedendo l'osso una forma più allungata in ogni slice, l'area di overlap sarà minore in quanto i contorni, che sono la parte più complicata da predire, sono preponderanti. Dopo 50 epoche, il Dice coefficient è pari a 0.52 e le predizioni sono infatti piuttosto grossolane, con forme arrotondate che però iniziano a possedere la forma generale dell'osso, seppur mancanti di molti dettagli. Dopo 100 epoche, il Dice coefficient è cresciuto fino a un valore di 0.65 con buone migliorie, seppur alcuni dettagli, soprattutto nelle parti piccole e isolate delle varie slice, non vengano ancora correttamente predetti. Le predizioni dopo 150 e 200 epoche sono visualmente molto simili ed anche il miglioramento del Dice coefficient, il cui valore passa da 0.69 a 0.72, è più ridotto. Questo comportamento è assolutamente normale e dimostra il fatto che il modello è riuscito a identificare la maggior parte dei dettagli e delle forme e si sta focalizzando solamente su ultimi particolari, non necessariamente fondamentali per l'obiettivo finale. Come per gli omeri, infatti, non è necessario ottenere in output l'esatta forma dell'input, in quanto questo significherebbe che il modello è andato in overfitting.

Anche nel caso delle scapole è necessaria la fase finale in cui si allena ulteriormente il modello per altre 50 epoch su tutto il dataset, compreso anche di validation set, per lo stesso motivo citato alla fine della sezione 4.3.1.



(a) Input

(b) Predizione dopo 250 epoche

Figura 4.9: Rendering 3D della scapola numero 200

Per fornire una rappresentazione più immediata della capacità del modello al termine dell'addestramento, nella figura 4.9 vengono riportate due immagini che mostrano il rendering tridimensionale di una scapola del training set e la corrispettiva predizione del modello dopo 250 epoch. Innanzitutto è necessario evidenziare che la scapola di input rappresenta la versione ridotta a seguito del downsampling della stessa, come descritto nella sezione 4.2.1. Da questa visualizzazione si può notare immediatamente come la forma della scapola predetta sia più arrotondata di quella originale, ma i contorni e le forme generali sono stati completamente ricostruiti, a segno del fatto che il modello è riuscito ad apprendere il modo molto preciso le caratteristiche morfologiche della stessa.

I modelli così addestrati possono quindi essere utilizzati per il loro scopo finale, cioè l'estrazione delle feature dalle immagini fornite loro in ingresso. Questa operazione sarà mostrata nella successiva sezione insieme alle due modalità di visualizzazione delle stesse.

4.4 Estrazione delle Features e Visualizzazione

In quest'ultima sezione del capitolo saranno mostrate le fasi terminali riguardanti il modello creato, cioè l'effettiva estrazione delle feature dalle immagini tramite l'utilizzo del modello

addestrato e alcune modalità di visualizzazione grafica delle stesse, che saranno fondamentali per la successiva parte relativa al clustering. Una volta effettuate queste due fasi, le feature estratte potranno essere sottoposte ad un algoritmo di clustering per la classificazione.

4.4.1 Estrazione delle Features

Questa prima parte è molto semplice in quanto, una volta addestrato opportunamente il modello, è possibile estrarre le feature dalle immagini semplicemente rimuovendo la parte di decoder e una parte del bottleneck per far sì che l'output del modello risulti essere l'encoding dell'input. Nel caso in esame per entrambi modelli è stata scelta una dimensione di encoding pari a 128, come mostrato nell'architettura nella sezione 4.2.2. Il primo dei due livelli densi del bottleneck diventa quindi il nuovo livello di output e tutto ciò che viene dopo dev'essere rimosso. Quest'operazione è stata implementata attribuendo un nome univoco ad ogni livello del modello, tramite il quale è possibile specificare il nuovo output creando un'architettura nuova mantenendo però i pesi del modello addestrato. Una volta ricavata l'architettura per l'estrazione delle feature è necessario dare in input al modello tutte le immagini (ricordando che nel nostro caso le immagini sono volumi 3D) del dataset considerato. Esso produrrà un vettore di encoding per ogni immagine, di dimensione pari a 128. Ognuno di questi vettori rappresenta le feature che il modello è stato in grado di estrarre da ogni immagine in ingresso e che la rappresenteranno nelle successive fasi. La qualità di queste feature dipende dalla qualità dell'addestramento del modello, operazione che è stata ampiamente discussa e approfondita nella sezione 4.3.

Una delle limitazioni principali di questo tipo di modelli, specialmente se utilizzati come estrattori di feature, è quella della limitata explainability, cioè la possibilità di comprendere i motivi dietro alle scelte compiute dal modello. Questo problema è tipico di tutti gli approcci basati su modelli di Deep Learning in quanto è spesso complicato capire praticamente come il modello riesca a ricavare le informazioni necessarie per effettuare le giuste predizioni. In questo specifico caso in cui è necessario estrarre delle feature, l'ulteriore limitazione consiste nel fatto che esse non hanno un significato particolare se prese singolarmente, ma sono solamente dei numeri che il modello ha imparato e tramite i quali riesce a riprodurre l'immagine in ingresso. Tramite questo vettore, preso nella sua interezza, il modello è in grado di ricavare le caratteristiche dell'immagine in ingresso che permettono la sua ricostruzione. Esso però, non è in grado di fornire una spiegazione sul significato di ognuna di queste feature, ad esempio il fatto che la prima ri-

guardi il volume, la seconda l'inclinazione dell'osso e via dicendo, anche perché solitamente non esiste un significato preciso. Esistono dei metodi, chiamati metodi di explainability, che cercano di ottenere alcune informazioni riguardo alle parti dell'immagine a cui il modello ha dato più o meno importanza durante l'addestramento, che non sono però al momento state implementate in quanto spesso piuttosto complicate, soprattutto nei modelli come gli Autoencoder.

Un possibile modo per comprendere che cosa il modello abbia prodotto a livello globale è quello di trovare un modo di visualizzare in un unico grafico tutte le immagini del dataset, ognuna rappresentata tramite le sue feature, in modo tale da comprendere le relazioni tra di esse. Questa operazione è possibile tramite approcci di dimensionality reduction, mostrati nella prossima sezione.

4.4.2 Visualizzazione delle Features

Questa parte è preliminare al clustering e permette la visualizzazione di feature multidimensionali in uno spazio a due dimensioni. Le feature prodotte dal modello, infatti, si possono considerare come appartenenti ad uno spazio di 128 dimensioni, una per ognuna di esse. Questo spazio è impossibile da visualizzare così com'è, in quanto i dispositivi che utilizziamo permettono visualizzazioni in due dimensioni o al massimo in tre dimensioni se si effettuano dei rendering. È pratica comune, quindi, a scopo di visualizzazione, ridurre il numero di feature delle immagini fino a due. Gli approcci che lo permettono sono chiamati tecniche di dimensionality reduction. Ne esistono diverse adatte a scopi differenti. Esse però hanno tutte come obiettivo quello di cercare di conservare le relazioni tra i dati, in modo tale che la riduzione porti ad una visualizzazione che abbia una struttura il più possibile simile a quella dello spazio di partenza. Le due metodologie qui considerate sono chiamate t-SNE e UMAP e verranno spiegate nelle successive sezioni.

t-SNE

t-SNE (t-distributed Stochastic Neighbor Embedding) [27] è un metodo di riduzione della dimensionalità molto usato per task di visualizzazione. Il metodo calcola inizialmente la similarità tra coppie di punti nello spazio originale usando una funzione Gaussiana, tramite la quale i punti più vicini hanno più probabilità di essere considerati simili. In seguito, vengono costruite due differenti distribuzioni di probabilità: una per le similarità nello spazio originale e un'altra

per le similarità nello spazio ridotto. Questo viene fatto con lo scopo di trovare un mapping tra i due spazi, cercando di minimizzare le differenze tra queste due distribuzioni di probabilità. Ciò viene eseguito aggiustando iterativamente le posizioni dei dati nello spazio di arrivo. Questo processo di ottimizzazione avviene minimizzando una quantità chiamata divergenza di Kullback-Leibler. Questa metrica è spesso utilizzata quando si vogliono confrontare due distribuzioni di probabilità e fornisce una misura dell'informazione persa quando una di esse viene usata come approssimazione dell'altra. Durante il processo di ottimizzazione, questo metodo tende a focalizzarsi sul preservamento delle informazioni locali dei dati rispetto a quelle globali, rendendolo particolarmente adatto per la visualizzazione di cluster.

La visualizzazione finale dello spazio prodotto da questo metodo dipende da alcuni parametri. Il più importante è chiamato perplexity e permette di bilanciare l'attenzione che il modello deve dare alla struttura locale piuttosto che a quella globale dei dati. Un valore più alto consente di preservare maggiormente la struttura globale, ma un valore troppo alto potrebbe distorcere le strutture locali dei dati. Esistono poi altri parametri che è opportuno citare come il numero di iterazioni, che indica il numero di volte che il processo di ottimizzazione viene eseguito. Aumentare questo numero può portare a risultati migliori, ma il tempo di esecuzione si allunga parecchio. Uno dei problemi di questo modello sta nel fatto che esso è molto sensibile alla variazione dei parametri, soprattutto per quanto riguarda i valori della perplexity. Valori diversi possono portare a risultati completamente differenti. In pratica però, nell'implementazione di seguito riportata, sono stati utilizzati i parametri di default che dopo una serie di prove sono risultati essere il miglior compromesso tra i valori in genere adottati.

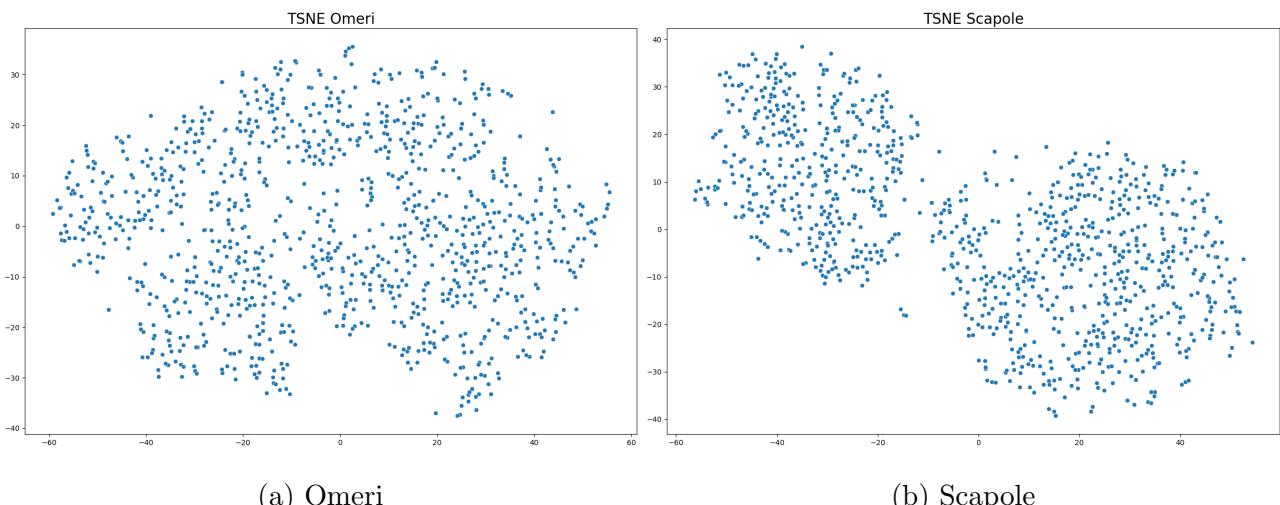


Figura 4.10: t-SNE applicato alle feature di omeri e scapole

La figura 4.10 mostra l'applicazione di questa tecnica ai due dataset in esame. Ogni punto sui due grafici rappresenta le feature estratte da un'immagine del dataset proiettate in due dimensioni. A colpo d'occhio si può subito notare come la nuvola di punti degli omeri sia più uniforme, mentre quella delle scapole sembri già formare due grossi cluster, che probabilmente differenziano le scapole destre da quelle sinistre. Questo aspetto negli omeri è molto più complicato da individuare, mentre le scapole avendo una forma più complessa sono più facili da distinguere anche a livello visivo.

UMAP

In questa sezione verrà mostrato un ulteriore metodo di dimensionality reduction, più moderno e dal funzionamento lievemente differente rispetto al precedente, chiamato UMAP (Uniform Manifold Approximation and Projection) [18]. È sempre buona norma provare approcci differenti che raggiungono lo stesso obiettivo, in questo caso la riduzione di dimensionalità, in quanto il loro funzionamento potrebbe essere più adatto a un dataset piuttosto che a un altro. Inoltre visualizzare proiezioni differenti degli stessi dati in due dimensioni permette un confronto tra gli stessi in modo da comprendere quale rappresentazione sia più adatta per lo specifico caso in esame.

UMAP ha un funzionamento molto simile a quello di t-SNE, ma in media è più veloce e riesce a preservare in modo più fedele sia le relazioni locali tra i dati che quelle globali, obiettivo che t-SNE faticava a raggiungere. La differenza principale tra i due metodi sta nella parte iniziale nella quale vengono calcolate le distanze tra i punti nello spazio di partenza. Una volta ricavate le stesse, la parte di ottimizzazione e mappatura in una distribuzione analoga nello spazio a dimensioni ridotte è piuttosto simile, seppure UMAP adotti alcuni stratagemmi per velocizzare il processo. Per la determinazione della similarità, invece, non viene utilizzato un kernel Gaussiano, ma un approccio nel quale ad ogni punto viene assegnato un raggio di una certa dimensione. Due punti sono considerati simili se i loro raggi si sovrappongono, cioè se il cerchio costruito attorno ad uno interseca quello costruito attorno all'altro. La scelta del valore di questo raggio è molto importante: se esso è troppo piccolo si avranno tanti piccoli gruppi di punti isolati, mentre se esso è troppo grande si rischia di avere un'unica grande nuvola di dati di poco significato. Per ovviare a questo problema UMAP determina il valore di questo raggio localmente, sulla base del suo n -esimo nearest neighbor, in modo da permettere un tuning più semplice del valore che sia relativo alla struttura dei dati.

I parametri fondamentali in questo metodo sono essenzialmente due: *n_neighbors* e *min_dist*. Anche nel caso di UMAP, essi sono usati per controllare il bilanciamento tra la struttura locale e globale dei dati nella proiezione finale. Il più importante è il primo, che imposta la dimensione del raggio per la determinazione della similarità tra due punti. Questo parametro è quello che effettivamente controlla il bilanciamento: un valore più basso permette al modello di focalizzarsi maggiormente sulla struttura locale limitando il numero di punti simili, mentre valori alti permettono di ottenere una rappresentazione della struttura globale perdendo però alcuni dettagli. Il parametro *min_dist* invece riguarda la distanza minima tra i punti nello spazio di arrivo a ridotta dimensionalità. Esso controlla quanto strettamente UMAP debba raggruppare insieme i punti, con valori bassi che conducono a raggruppamenti più stretti. Valori più alti invece fanno sì che i punti vengano raggruppati meno strettamente, focalizzandosi sul preservamento della struttura generale.

La differenza principale tra t-SNE e UMAP risiede quindi nel bilanciamento tra struttura locale e globale. UMAP è spesso migliore nel preservare la struttura globale nella proiezione finale. Inoltre, esso è meno sensibile al tuning dei suoi parametri, anche se comunque l'output risulta essere molto differente al cambiare degli stessi. Il vantaggio sostanziale sta nel fatto che t-SNE, con valori di perplexity molto bassi, non riesce a preservare quasi niente della struttura globale e necessita di valori molto alti dello stesso per riuscire a rispecchiare la stessa, con conseguente aumento del tempo di elaborazione. Con UMAP invece questo effetto è più contenuto in quanto già con valori piuttosto piccoli dei parametri la struttura globale dei dati si riesce a visualizzare. Inoltre, esso è in media più veloce di t-SNE. Pur possedendo una serie di vantaggi rispetto a t-SNE, non è sempre detto che esso debba essere l'approccio da preferire in qualsiasi situazione. In alcuni casi specifici potrebbe risultare più adatto l'utilizzo di t-SNE, motivo per cui è sempre buona abitudine provare entrambi gli approcci.

Dalla figura 4.11 si può notare come, anche a livello visivo, la struttura generale assomigli meno a una nuvola casuale di punti, anche nel caso dell'omero. Inoltre, i due gruppi delle scapole già notati nell'approccio tramite t-SNE qui emergono ancora meglio, a segno del fatto che quasi sicuramente la struttura globale nello spazio originale è suddivisa in modo abbastanza evidente in questi due grandi gruppi. Nel caso specifico della visualizzazione tramite UMAP non sono stati mantenuti i valori di default dei parametri ma il parametro *n_neighbors* è stato alzato fino ad un valore pari a 30, per enfatizzare la struttura globale, che è ciò che interessa maggiormente in questo caso, senza però compromettere troppo la struttura locale come succederebbe con

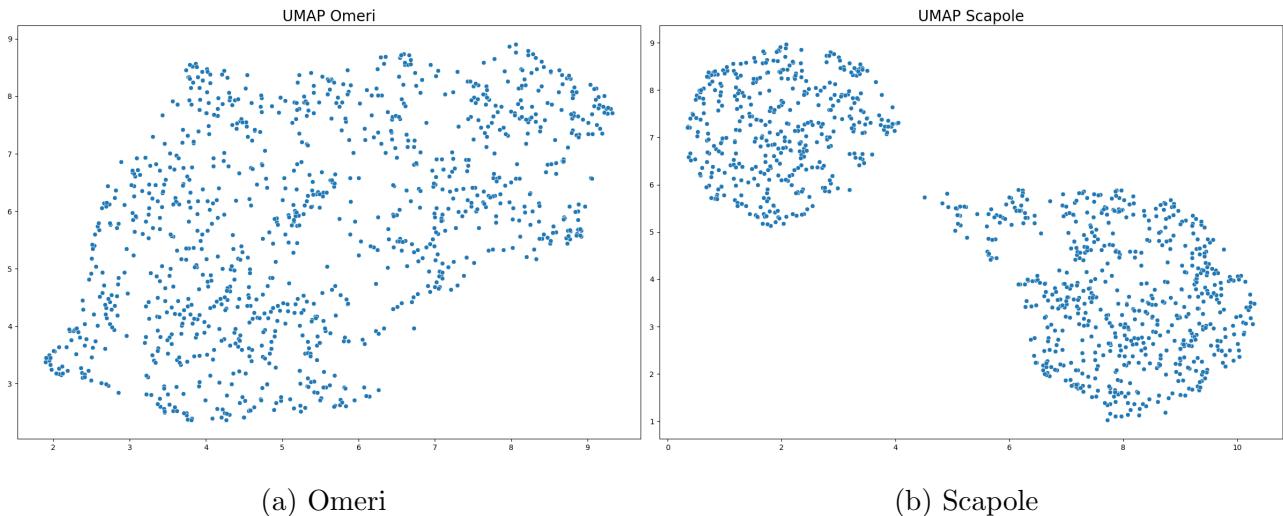


Figura 4.11: UMAP applicato alle feature di omeri e scapole

t-SNE.

Il fatto che si notino questi due grandi cluster senza l'applicazione di alcun algoritmo è un aspetto positivo in quanto segna che il modello è stato in grado di individuare due macro categorie in cui suddividere le scapole. Queste rappresentano molto probabilmente la distinzione tra scapola destra e sinistra, ma tramite l'applicazione di un algoritmo di clustering vero e proprio è possibile provarlo in modo immediato. Infatti, il controllo manuale di tutte le immagini di ogni gruppo rappresenterebbe un metodo troppo oneroso da attuare in termini di tempo. Tramite l'utilizzo di un algoritmo di clustering, invece, è possibile dividere lo spazio in un certo numero di gruppi e visualizzare solamente alcune immagini rappresentative di ognuno di essi per provare se i cluster prodotti siano effettivamente stati suddivisi tra scapole destre e sinistre. È opportuno notare però che l'obiettivo di questo lavoro è quello dell'individuazione dei morfotipi, che non dovrebbero dipendere dal fatto che la spalla considerata sia la destra o la sinistra. È quindi necessario eliminare questa informazione dal dataset in modo tale da permettere che l'apprendimento delle feature non venga condizionato da questa informazione. Il clustering risulta essere utile anche in questo caso, in quanto permette di ricavare rapidamente tutti gli indici delle immagini nel dataset appartenenti a uno dei due gruppi eventualmente ricavati e ribaltarle in modo da eliminare quell'informazione dai dati. Tutto ciò verrà spiegato più approfonditamente nel prossimo capitolo.

In conclusione, in questo capitolo è stata descritta la creazione del modello che ha permesso l'estrazione delle feature dalle immagini contenute nei dataset di scapole e omeri e la loro vi-

sualizzazione tramite approcci di riduzione di dimensionalità. Questi approcci hanno permesso di ottenere due rappresentazioni differenti degli stessi dati, in modo da comprendere a livello visivo la struttura generale degli stessi. È importante notare che la riduzione di dimensionalità è stata eseguita solamente a scopo di visualizzazione. Infatti, come sarà descritto più approfonditamente nel prossimo capitolo, il clustering verrà eseguito nello spazio originale e ne verrà solamente visualizzato il risultato tramite questi metodi applicando una colorazione diversa dei punti in base al cluster a cui essi appartengono. Nel prossimo capitolo si discuterà proprio di questi aspetti.

Capitolo 5

Clustering e Classificazione

In questo capitolo saranno presentate tutte le fasi che utilizzano le feature estratte tramite i modelli descritti nel capitolo precedente. La prima fase, quella di clustering, consente il loro raggruppamento in base ad alcuni criteri in modo da ottenere una suddivisione in gruppi, chiamati appunto cluster, il cui obiettivo è quello di raggruppare ossa con valori delle feature simili tra di loro, che dovrebbero rappresentare la morfologia dell'osso stesso. Dal momento che la prima estrazione delle feature, come già evidenziato, è molto probabilmente stata condizionata dalla posizione delle scapole nelle spalle, il clustering è stato applicato due volte su due dataset differenti. La prima ha un duplice obiettivo: fornire un primo raggruppamento in cluster dello spazio delle feature e, tramite l'analisi dei gruppi così formati, capire se essi consentano una suddivisione in spalle destre e sinistre, come i grafici nella sezione precedente suggeriscono. Una volta provata l'esistenza di questa suddivisione, è possibile separare le spalle della parte destra da quelle della parte sinistra e, tramite il ribaltamento delle immagini appartenenti a una delle due, ricavare un dataset uniforme nel quale questa informazione risulta essere assente. Il modello verrà infine riaddestrato su questo nuovo dataset e le feature così estratte sottoposte ad un ulteriore clustering, questa volta con lo scopo di valutare se sia direttamente possibile identificare i veri e propri morfotipi senza che la classificazione venga condizionata da questo macro fattore.

Una volta ottenuto il clustering finale, non più dipendente dall'orientamento della scapola, la fase successiva consiste nella creazione dei tool per permettere ai medici di visualizzare interattivamente i cluster ottenuti e consultare dei rendering tridimensionali di alcune ossa significative che vi appartengono, così da poter valutare la bontà di questi cluster e decidere se la suddi-

visione ottenuta possa essere opportuna oppure se è necessario modificare qualche aspetto del processo. Infatti, come sarà mostrato anche successivamente, esistono molti algoritmi differenti di clustering ognuno dei quali possiede parecchi parametri non semplici da settare correttamente in quanto spesso richiedono della conoscenza a priori. Infine, saranno analizzati e commentati i risultati ottenuti tramite l'utilizzo degli strumenti realizzati.

5.1 Il Clustering

In questa prima sezione verrà analizzato il processo di clustering svolto sulle feature estratte dai dataset tramite il modello autoencoder. Verrà innanzitutto riportata un'introduzione sugli algoritmi di clustering in generale, per poi analizzare nello specifico l'algoritmo applicato al caso in esame. In seguito, sarà analizzato il primo clustering effettuato sul dataset originale, mostrando anche grafici che ne riportano i risultati ottenuti tramite le tecniche di riduzione della dimensionalità illustrate nella sezione 4.4.2. Successivamente, verrà presentata la fase di divisione tra spalle destre e sinistre sulla base del clustering precedente, così come il riaddestramento dei modelli sui nuovi dataset ottenuti e l'esecuzione di un ulteriore clustering finale sulle nuove feature prodotte.

5.1.1 Descrizione Generale

Il clustering è una tecnica molto utilizzata al fine di raggruppare oggetti simili in gruppi chiamati cluster. Essi fanno parte della categoria dei metodi non supervisionati, in quanto non necessitano di alcuna label sui dati ma utilizzano solamente le informazioni contenute nei dati stessi. L'obiettivo di questi metodi è quello di ottenere una suddivisione in gruppi nei quali i punti appartenenti ad ogni cluster siano più simili agli altri punti appartenenti allo stesso cluster rispetto a quelli appartenenti a cluster differenti. Il raggruppamento dei dati si basa quindi strettamente su una misura di similarità tra gli stessi, che solitamente coinvolge delle misure di distanza. Esistono moltissime tipologie di algoritmi di clustering che utilizzano diverse misure di similarità per il raggruppamento dei dati e approcci differenti per l'identificazione dei cluster [23]. Le principali tipologie sono le seguenti:

- **Metodi Basati sui Centroidi:** questi metodi sono i più semplici in quanto si basano sulla distanza dei punti da raggruppare rispetto ad altri punti chiamati centroidi. Essi

corrispondono al valore centrale di ogni cluster e sono in numero fisso, deciso preventivamente all'utilizzo del metodo. Uno dei problemi principali di questo tipo di metodi è proprio la scelta a priori del numero di cluster e di conseguenza anche del numero di centroidi, che non varia durante l'esecuzione dell'algoritmo. Uno degli algoritmi più diffusi appartenenti a questa categoria è chiamato K-Means e sarà descritto successivamente in quanto metodo adottato per questo lavoro.

- **Clustering Gerarchico:** questi algoritmi si basano su misure di distanza tra i punti dello spazio e non richiedono la definizione di ulteriori punti come nel caso precedente. Ne esistono due differenti tipologie: metodi agglomerativi e divisivi. Nel primo caso, ogni punto è inizialmente considerato come appartenente ad un cluster differente ed essi vengono agglomerati in cluster sempre più grandi in base alla loro distanza. Nel secondo caso, invece, la situazione iniziale è composta da un unico grande cluster che viene diviso ricorsivamente fino ad ottenere una rappresentazione in cui ogni punto appartiene a un diverso cluster. In entrambi i casi vengono utilizzati dei grafici chiamati dendogrammi che rappresentano la sequenza di unioni o divisioni dei cluster. Il vantaggio di questi approcci è principalmente quello che non necessitano del numero di cluster a priori, ma tagliando orizzontalmente un dendogramma ad altezze diverse si ottengono suddivisioni differenti. Il problema principale è quello di essere computazionalmente molto onerosi e quindi non adatti a dati di grosse dimensioni.
- **Clustering Density-Based:** questa tipologia di algoritmi funziona in modo differente dai precedenti in quanto considera la densità dei dati e non la loro distanza per determinare l'appartenenza o meno ad un cluster. I dati sono divisi in regioni ad alta densità contornate da aree a più bassa densità. Il vantaggio di questi approcci consiste nell'essere in grado di identificare cluster di forma e dimensioni qualsiasi, al contrario dei precedenti che prediligono invece forme circolari o ellittiche. Inoltre, anche in questi metodi non è necessario specificare a priori il numero di cluster. Uno degli svantaggi risulta essere la difficoltà di identificare cluster con una densità variabile. Uno degli algoritmi più utilizzati facenti parte di questa categoria è l'algoritmo DBSCAN [6].

Esistono molti altri tipi di algoritmi di clustering, ma quelli appena elencati sono tra i più consolidati e utilizzati in letteratura.

Un ulteriore aspetto importante da considerare per questi tipi di algoritmi è rappresentato dalla valutazione delle loro performance. Infatti, essa non è sempre semplice in quanto solitamente

non sono disponibili le label per la valutazione della qualità delle suddivisione ottenuta. Esistono delle metriche diverse nel caso in cui si possiedano le label del ground truth oppure no. Nel secondo caso, che corrisponde al caso in esame, è più complicato valutare le performance, ma esistono delle metriche apposite che valutano certi aspetti del modello, come ad esempio la variabilità dei dati all'interno dei vari cluster o le distanze dei punti dal centroide. Esse permettono di ottenere una misura sulla qualità del clustering ottenuto, in modo da fornire anche un'indicazione sul numero ideale di cluster da utilizzare per algoritmi come K-Means che richiedono questo parametro come input. Alcune delle metriche più utilizzate sono: la Silhouette, l'indice Calinski-Harabasz e l'indice Davies-Bouldin. Esse forniscono informazioni differenti sulla qualità del clustering ottenuto dall'algoritmo. La Silhouette, per esempio, valuta per ogni punto le distanze intra-cluster e inter-cluster, cioè le distanze del punto da tutti gli altri dello stesso cluster e le distanze del punto dai punti del cluster più vicino. Questa metrica fornisce un valore tra -1 e +1 che risulta essere tanto più alto quanto più i cluster sono densi e ben separati.

Identificare il corretto metodo di clustering da applicare ed i suoi parametri ideali non è quindi un compito semplice, in quanto richiede molte prove differenti. In generale, però, esistono degli algoritmi, tra cui ad esempio K-Means e DBSCAN, che funzionano molto bene nella maggior parte dei casi e quindi si preferisce utilizzare quelli come primi approcci. Nella prossima sezione verrà presentato lo specifico algoritmo utilizzato per entrambi i compiti successivi, cioè l'algoritmo K-Means, ed i risultati ottenuti tramite l'uso dello stesso.

5.1.2 Clustering sul Dataset Originale

La prima parte consiste nell'applicazione di uno degli algoritmi di clustering alle feature prodotte dal modello addestrato sul dataset originale, cioè quello analizzato nel capitolo 4. Il clustering in questo caso è stato utilizzato allo scopo di fornire una prima suddivisione dello spazio delle feature e di valutare se la separazione nei due grandi gruppi delle scapole che si nota a livello visivo venga individuata anche da un algoritmo di clustering, che lavora nello spazio originale. Infatti, pur avendo utilizzato tool per la visualizzazione che preservano la struttura globale dei dati come UMAP e t-SNE, è necessario ricordare che lo spazio visualizzato è solamente una proiezione in due dimensioni di uno spazio molto più grande che quindi potrebbe non rappresentarlo in modo adeguato. Per questa fase, così come per la successiva, è stato

utilizzato l'algoritmo di clustering K-Means. Questa scelta è dovuta alla notorietà di questo metodo e alla sua versatilità, in quanto ne esistono molte versioni differenti adatte a diverse situazioni.

K-Means

Il metodo K-Means è un algoritmo iterativo basato sui centroidi. L'obiettivo di questo algoritmo è la separazione dei dati in K gruppi non sovrapposti, chiamati cluster, nei quali ogni punto può appartenere ad uno solo dei gruppi trovati. Esso ha come obiettivo quello di creare dei cluster i cui punti siano i più simili possibile, cercando allo stesso tempo di mantenere distanti i cluster tra di loro. Un punto viene assegnato ad un cluster se la sua distanza dal centroide di quel cluster è minima. Il centroide rappresenta la media di tutti i dati appartenenti ad un cluster. Gli step necessari al funzionamento dell'algoritmo sono i seguenti:

1. Specificare il numero K di cluster come parametro dell'algoritmo. Come già evidenziato, la scelta di questo parametro è critica in quanto il numero di cluster non cambia mai durante l'esecuzione
2. Inizializzare i primi centroidi tramite metodi opportuni selezionando K punti casuali dal dataset
3. Ripetere i seguenti passi fino a quando la posizione dei centroidi si stabilizza:
 - Calcolare la somma delle distanze al quadrato tra ogni punto e ogni centroide
 - Assegnare ogni punto al cluster più vicino, cioè quello che corrisponde al centroide con cui ha la distanza minima
 - Ricalcolare i centroidi di ogni cluster eseguendo la media dei valori dei dati appartenenti ad ognuno di essi

Il metodo produce quindi una label per ognuno dei dati di input che ne indica il cluster di appartenenza e che potrà essere visualizzata sui grafici ottenuti tramite i metodi descritti nella sezione 4.4.2.

Applicazione del Metodo

Per l'applicazione pratica del metodo è stata utilizzata la libreria chiamata Scikit-learn, che rappresenta la scelta più popolare per l'utilizzo di algoritmi di Machine Learning. Essa comprende, insieme a molti altri metodi, un vasto insieme di algoritmi di clustering, ognuno dei quali è personalizzabile tramite una serie di parametri. Il più importante dell'algoritmo K-Means, come già evidenziato più volte, risulta essere il parametro $n_clusters$, che indica il numero di

cluster che l'algoritmo dovrà identificare. Solitamente, per aiutare in questa scelta, vengono calcolate alcune delle metriche già citate nella sezione 5.1.1 per numeri variabili di cluster. In questo modo è possibile ottenere delle informazioni sull'evoluzione di queste metriche al variare del numero di cluster. È opportuno evidenziare, però, che non essendo disponibili delle label, queste metriche si possono basare solamente su dati geometrici e statistici come distanze, medie o varianze tra i dati. In uno spazio a 128 dimensioni come quello delle feature qui considerate, queste metriche non sempre forniscono dati utili e rilevanti, ma permettono comunque una valutazione generale delle performance.

Per quanto riguarda l'identificazione del numero opportuno di cluster, il metodo più utilizzato è chiamato "metodo del gomito". Esso si basa sul calcolo dell'inerzia del clustering, che rappresenta il valore della funzione da ottimizzare, cioè la somma delle distanze al quadrato di ogni punto dal centroide del cluster a cui appartiene. Il valore di questa funzione in generale decresce all'aumentare del numero di cluster. L'obiettivo del metodo è quello di individuare il "gomito" nel grafico, cioè il punto in cui la velocità di diminuzione di questa metrica inizia a diminuire. Solitamente quel punto rappresenta il numero ideale di cluster da considerare.

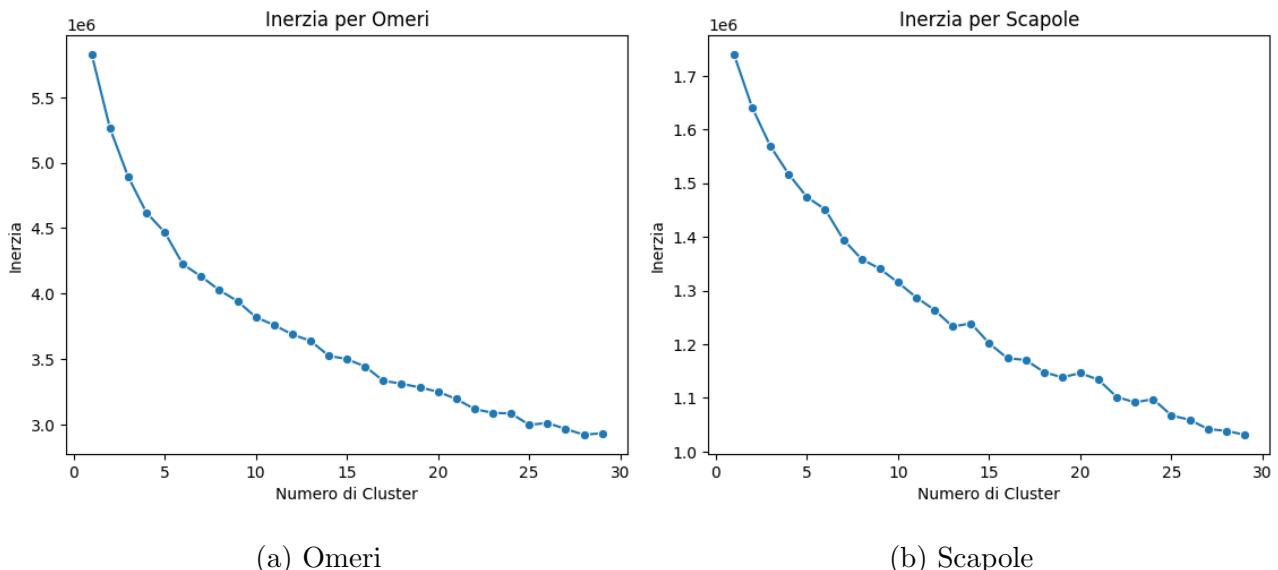


Figura 5.1: Valori di inerzia per un numero di cluster da 2 a 30

Dalla figura 5.1 è piuttosto evidente che, nel caso in esame, il gomito non sia facilmente individuabile, seppur sia stato considerato un numero di cluster massimo piuttosto elevato. Si può però notare che l'inerzia possiede un valore più basso nelle scapole rispetto agli omeri. Ciò è probabilmente dovuto all'aspetto evidenziato nella sezione 4.4.2 che riguardava i due grandi cluster che già si potevano individuare visivamente nelle scapole. Essendo lo spazio di partenza

già meglio suddiviso, il valore di inerzia sarà necessariamente più basso in quanto i dati sono più propensi ad essere raggruppati. Anche l'utilizzo di ulteriori metriche non ha fornito risultati molto utili per quanto riguarda la selezione del corretto numero di cluster.

È stato quindi deciso di considerare un numero di cluster pari a 15. Questo valore è stato scelto tenendo conto del fatto che, dalla letteratura, ci si aspetta di ottenere un numero di morfotipi non superiore a 10. È sempre opportuno scegliere un numero di cluster più elevato del numero di classi che effettivamente ci si aspetta di ottenere, in quanto è più semplice eventualmente agglomerare cluster piuttosto che dividerne. Inoltre, avendo applicato l'algoritmo al dataset originale, che presenta i due grandi gruppi, ci si aspetta di ottenere una netta separazione tra cluster di spalle destre e cluster di spalle sinistre. Mediante la separazione dei cluster di un tipo rispetto all'altro, sarà possibile creare un dataset più uniforme non più condizionato da questa caratteristica capovolgendo le immagini appartenenti a uno di questi due gruppi. Questa parte verrà però discussa nella successiva sezione.

Di seguito, verranno riportate le figure ottenute dalla sovrapposizione tra i grafici delle feature presentati nella sezione 4.4.2 e le label prodotte dall'algoritmo di clustering. Come tecnica di dimensionality reduction per la visualizzazione è stata scelta UMAP.

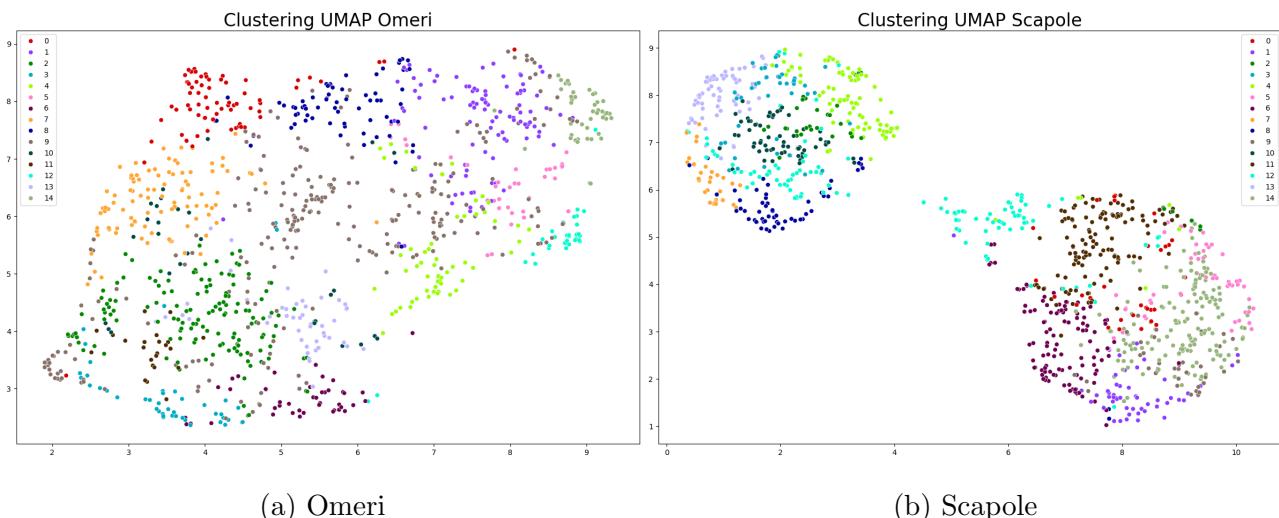


Figura 5.2: Visualizzazione dei 15 cluster trovati per omeri e scapole

Dalla figura 5.2, si può immediatamente notare come i cluster, seppur in genere abbastanza ben confinati in regioni dello spazio, appaiano abbastanza mischiati tra di loro e non ci sia una netta separazione tra di essi. Ciò accade in quanto il clustering viene eseguito nello spazio a 128 dimensioni e il suo risultato viene riportato sul grafico che rappresenta una proiezione di queste

feature in due dimensioni. I cluster individuati, quindi, saranno sicuramente ben separati nello spazio originale, ma quando vengono proiettati in due dimensioni perdono a livello visivo questa informazione.

Per quanto riguarda la suddivisione tra parte destra e sinistra, nel grafico delle scapole, che risulta essere quello con la separazione più evidente, sembra che la maggior parte dei cluster appartenga effettivamente solamente a una delle due parti. Solo uno di essi, cioè il cluster numero 12, possiede dei punti appartenenti a entrambe le parti identificate. Esso rappresenta però anche il cluster più dubbio in quanto, pur avendo anche alcuni punti appartenenti alla parte di destra, essi risultano lievemente discostati dalla forma generale e più propensi verso la parte di sinistra. Ciò potrebbe essere dovuto alla proiezione in due dimensioni dei dati, oppure al fatto che l'algoritmo di clustering non sia stato in grado di collocare questo cluster esattamente in una delle due parti. Possedendo le label dei dati fornite in output dal clustering, risulta però possibile visualizzare alcune immagini di ogni cluster e valutare quale ipotesi sia quella reale. Questa riflessione, insieme alle successive fasi, sarà discussa nella prossima sezione.

5.1.3 Suddivisione delle Spalle e Clustering sul Dataset Modificato

In questa sezione verrà riportata la fase che coinvolge la separazione dei cluster contenenti immagini di spalle destre e di spalle sinistre. Una volta individuata questa suddivisione, è possibile ribaltare i volumi di uno dei due gruppi in modo tale da eliminare questa informazione dal dataset e permettere al modello di concentrarsi sulla forma dell'osso per una migliore suddivisione finale in classi a scopo di individuazione dei morfotipi. Lo stesso processo sarà applicato anche agli omeri per motivi di consistenza con le rispettive scapole. Una volta ottenuti i dataset modificati, i modelli verranno riaddestrati sugli stessi e sarà eseguito un ulteriore clustering sulle nuove feature ottenute, modificando il numero di cluster di conseguenza per ricavare la classificazione definitiva.

Identificazione dei Cluster e Creazione dei Nuovi Dataset

Per questa prima fase è stata necessaria un'ispezione manuale di alcune delle scapole appartenenti ad ognuno dei 15 cluster ottenuti nella parte precedente. Per la scelta delle scapole da visualizzare è stato utilizzato l'algoritmo chiamato k-nearest neighbors (KNN). Questo metodo consente di ricavare i k punti più vicini a un dato fornito in input in uno spazio assegnato.

In questo caso specifico, esso è stato utilizzato per ricavare i tre punti più vicini al centroide di ogni cluster nello spazio a 128 dimensioni. Questi punti coincidono con le tre scapole più significative per ognuno dei cluster considerati, in quanto possiedono delle feature tali per cui esse sono state posizionate molto vicine al centro del cluster e quindi allo stesso tempo il più lontano possibile da tutti gli altri centroidi. Tramite l’ispezione visiva di queste scapole, è stato facilmente possibile provare l’esistenza di una divisione tra i cluster prodotti. Alcuni di essi infatti risultano composti solamente da scapole della parte destra, mentre altri contengono solamente scapole della parte di sinistra. È opportuno evidenziare che in realtà, per essere sicuri che il cluster sia formato effettivamente da tutte scapole di un solo tipo, sarebbe necessaria un’ispezione manuale di ogni immagine del dataset. Tramite questo metodo che utilizza il KNN, però, è possibile ricavare le scapole significative di ogni cluster, che dovrebbero rappresentare quasi totalmente le altre appartenenti allo stesso. Inoltre, la scelta di suddividere lo spazio in molti cluster è anche legata a questo motivo: se fossero stati utilizzati solamente pochi grandi cluster, sarebbe stato molto più complicato esaminarne il contenuto in quanto si sarebbero ottenuti meno centroidi su cui basare la ricerca.

Da questa analisi è infine emerso che dei 15 cluster considerati, 8 contenevano prevalentemente scapole della parte sinistra, mentre 7 contenevano scapole della parte destra. Conoscendo le label dei cluster di ognuna delle due parti, è risultato immediato ricavare tutti gli indici delle scapole nel dataset appartenenti ad ognuna delle due. Ne è risultato che nel dataset originale erano presenti 633 scapole della parte destra e 423 della parte sinistra. È possibile notare che, seppur il numero di cluster della parte destra sia inferiore, il dataset è formato da una maggioranza di scapole destre. Questo è segno che i cluster contenenti queste ultime sono risultati essere più grandi degli altri.

È stato quindi deciso di ribaltare le scapole della parte sinistra, in quanto meno numerose, ottenendo così un nuovo dataset composto da scapole dotate tutte dello stesso orientamento. La medesima operazione è stata eseguita con gli omeri, ricavando gli indici del dataset originale corrispondenti a quelli delle scapole e ottenendo così un nuovo dataset modificato anche per essi. Prima di passare alla vera e propria estrazione finale delle feature, è però opportuno evidenziare che è stato necessario trovare un modo per controllare preventivamente all’addestramento finale che effettivamente tutte le scapole di sinistra fossero state specchiate a destra, in quanto ciò non è assicurato dalla sola ispezione delle scapole centrali di ogni cluster. Per far ciò, è stato compiuto un piccolo addestramento preliminare di poche epoche sul dataset

modificato, in quanto l'informazione tra destra e sinistra è ricavabile già dalle prime epoche dell'addestramento. Visualizzando il risultato dello stesso, è stato possibile provare che effettivamente la nuvola di punti che si era formata non conteneva più alcuna separazione evidente ed è quindi di conseguenza stato possibile iniziare l'addestramento finale del modello.

Estrazione delle Nuove Feature e Clustering

A questo punto, è necessario addestrare nuovamente i modelli descritti nella sezione 4.2 fornendo loro in input i due dataset modificati, con l'obiettivo di estrarre delle nuove feature dagli stessi che non tengano però conto dell'informazione sulla posizione dell'osso nel corpo. I modelli sono stati addestrati allo stesso modo e utilizzando gli stessi parametri descritti in 4.3. Anche le fasi successive, quali l'estrazione delle feature e il clustering, sono avvenute con le stesse modalità, con l'unica differenza che riguarda il numero di cluster estratti. Infatti, è stato deciso di ridurre questo numero a 10 in quanto con le nuove feature la divisione tra spalla destra e sinistra non esiste più e ci si aspetta di conseguenza che i cluster non dipendano più da questa caratteristica. Come già affermato, il numero di morfotipi dovrebbe essere difficilmente superiore a 10 e quindi, per una consultazione successiva più rapida, è stato scelto questo numero come riferimento.

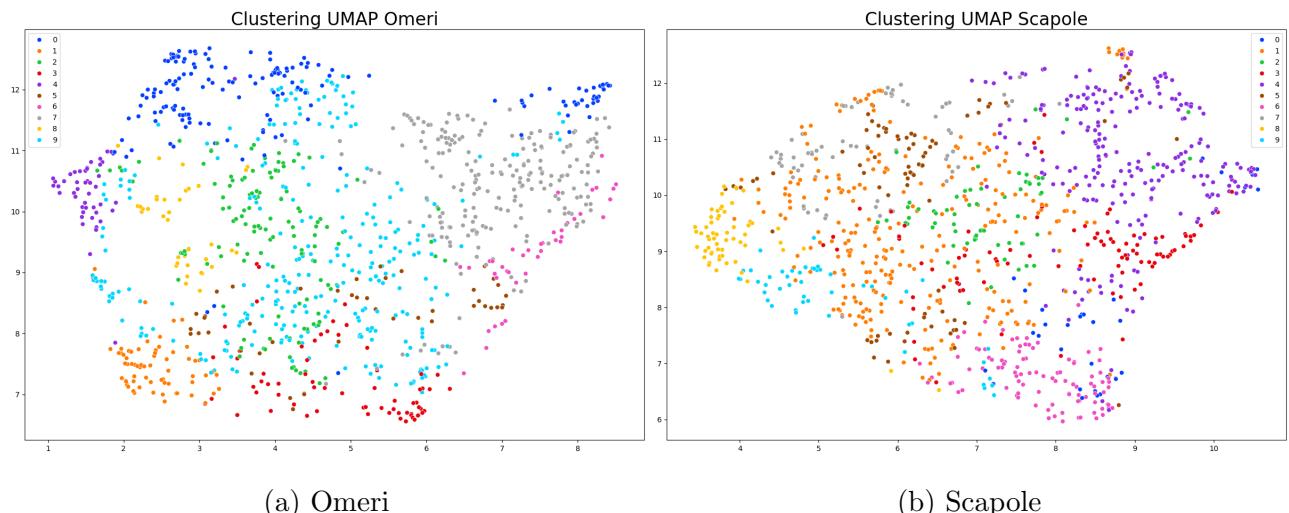


Figura 5.3: Visualizzazione dei 10 cluster trovati per omeri e scapole sul dataset modificato

Osservando la figura 5.3, si può infatti notare che il grafico riguardante le scapole non presenta più i due grandi gruppi presenti precedentemente, segno del fatto che tramite il ribaltamento delle scapole è stato possibile eliminare l'informazione sull'orientamento dalle feature estratte, producendo così una nuvola di punti più uniforme. Anche il grafico degli omeri risulta essere differente dal precedente, ma a livello visivo non si notano particolari diversità.

Una volta ottenuto il risultato del clustering, è necessario ora produrre degli strumenti che ne permettano una consultazione rapida ed efficiente. A questo scopo, sono stati implementati due tool di visualizzazione interattivi con obiettivi differenti che permettono di analizzare più nel dettaglio alcuni aspetti importanti dell'output di questo algoritmo. Essi saranno oggetto della prossima sezione.

5.2 Tool di Visualizzazione

In questa sezione verranno presentati i tool implementati allo scopo di permettere la consultazione di alcuni aspetti fondamentali scaturiti dall'applicazione dell'algoritmo di clustering. L'obiettivo finale è quello di permettere una comoda analisi del risultato del metodo in modo da valutare la bontà della suddivisione ottenuta. Il primo tool consiste nel rendere interattivi i grafici mostrati nella sezione 5.1.3, in modo da fornire informazioni aggiuntive sui punti al passaggio del cursore sopra gli stessi, così come la possibilità di escludere o isolare dalla visualizzazione determinati cluster. Il secondo tool permette invece la visualizzazione tridimensionale degli omeri e delle scapole più vicini ai centroidi per ognuno dei cluster. Essi sono considerati i più rappresentativi per i cluster stessi e quindi permettono un facile confronto tra ossa appartenenti allo stesso cluster e a cluster differenti.

5.2.1 Grafico interattivo

Il primo tool prodotto permette un'analisi interattiva dei grafici che mostrano il risultato del clustering tramite l'applicazione dell'algoritmo UMAP. Esso è stato implementato tramite l'utilizzo di Dash [13] e Plotly [14], due librerie Python che lavorano congiuntamente. La prima consiste in un Framework che permette il facile sviluppo di applicazioni web interattive in Python. È molto utilizzata per creare applicazioni di visualizzazione dei dati in quanto lavora congiuntamente ad altre librerie, come Plotly. Essa permette infatti di creare in modo molto semplice un ampio insieme di grafici differenti la cui potenzialità risulta essere l'elevata interattività. Unendo queste due librerie è stato quindi possibile sviluppare due semplici applicazioni web ognuna delle quali contiene il grafico relativo all'osso in esame riportato in figura 5.2 reso interattivo grazie all'utilizzo di Plotly.

Per brevità, di seguito verrà analizzata solamente l'applicazione relativa alle scapole, in quanto perfettamente analoga a quella degli omeri.

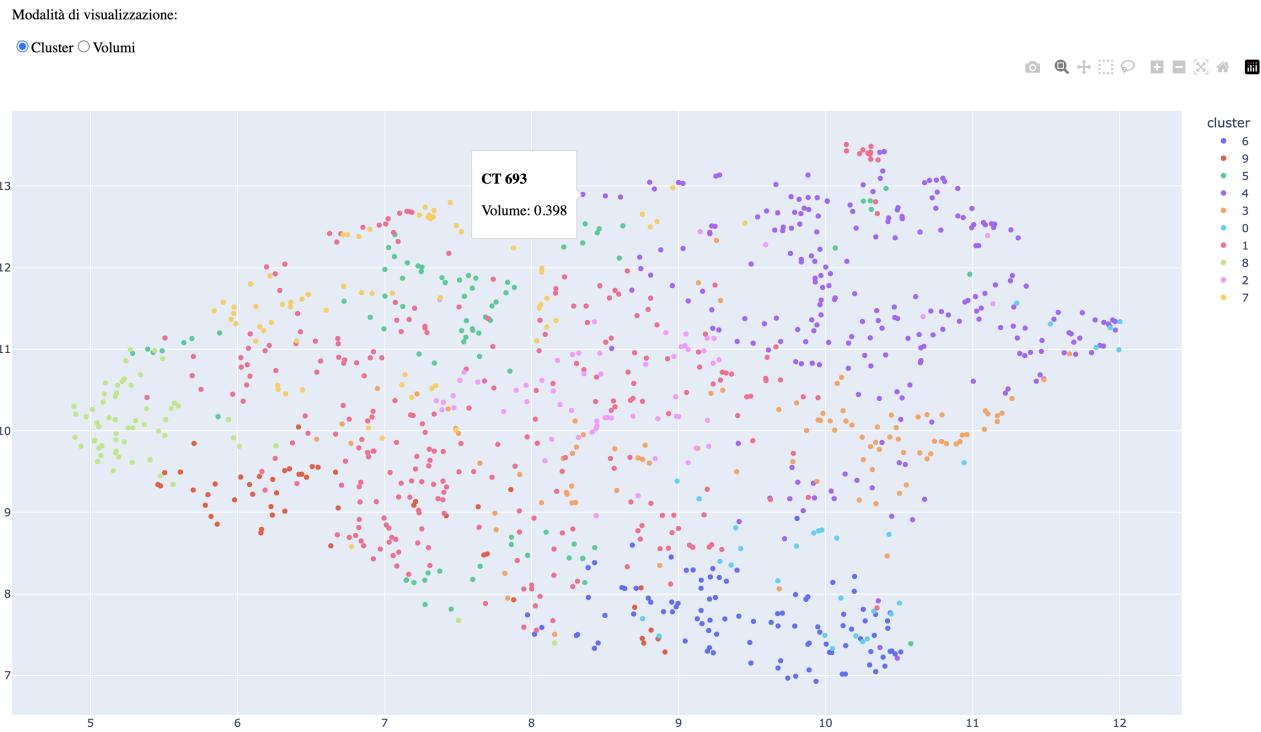


Figura 5.4: Pagina dell'applicazione web delle scapole relativa ai cluster

La figura 5.4 mostra la pagina web relativa alla visualizzazione dei cluster sul dataset delle scapole. In alto a destra è presente un menù tramite il quale si possono effettuare zoom sul grafico in modo da visualizzare meglio alcune parti di interesse. A destra del grafico è presente la legenda dei cluster, ognuno dei quali possiede un numero da 0 a 9 che lo identifica. Non avendo ancora la possibilità di comprendere l'eventuale significato dei cluster, i numeri non possiedono alcun significato specifico, ma sono solamente degli identificatori univoci. Cliccando sui vari numeri dei cluster è possibile escluderli dalla visualizzazione, mentre facendo doppio click è possibile isolare solamente quel cluster ed escludere tutti gli altri. Infine, quando il cursore passa su un punto del grafico, viene mostrato un tooltip contenente il numero della TC corrispondente a quel punto e il volume della scapola in essa contenuto. La misura sul volume è stata introdotta in modo da fornire un'informazione aggiuntiva durante la consultazione del grafico. Si può infatti notare che in cima alla pagina è presente la possibilità di cambiare la modalità di visualizzazione del grafico da cluster a volumi.

Nella figura 5.5 è presente la visualizzazione dello stesso grafico suddiviso in base al volume delle ossa. In questa visualizzazione i diversi colori indicano volumi differenti: più il colore è

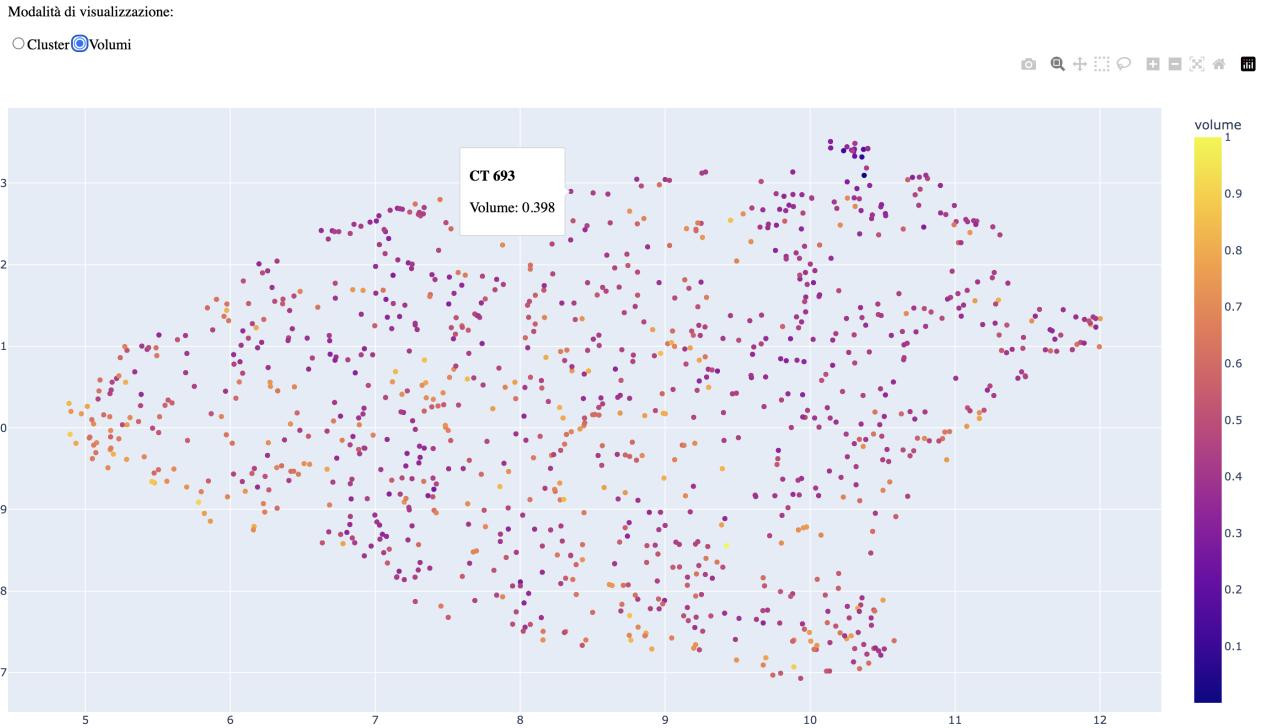


Figura 5.5: Pagina dell'applicazione web delle scapole relativa ai volumi

chiaro, più il volume è alto, come mostrato anche nella legenda a destra. Si può facilmente notare che in questo caso i colori rappresentano valori continui compresi tra 0 e 1, mentre nel grafico precedente erano presenti solamente 10 colori, ognuno rappresentante un cluster diverso. Essi sono stati ottenuti dividendo il volume di ogni osso rispetto a quello dell'osso più grande, che avrà un valore pari ad 1. Il calcolo del volume è stato eseguito semplicemente contando il numero di voxel pari a 1 nella maschera di segmentazione prodotta al primo step in quanto, in seguito al resampling delle immagini effettuato all'inizio del progetto (sezione 2.2.2), ogni voxel possiede la stessa dimensione in ognuna delle immagini. In questo modo si fornisce una rappresentazione alternativa degli stessi dati tramite una colorazione basata su un aspetto puramente geometrico dell'osso, fornendo informazioni aggiuntive per la consultazione che potrebbero fornire alcune utili idee sull'effettiva classificazione finale, come ad esempio la possibile correlazione tra l'individuazione di alcuni cluster e il dato relativo al volume delle ossa che vi appartengono. Facendo un confronto tra la figura 5.4 e la figura 5.5, infatti, è possibile cercare di capire se alcuni cluster siano stati più o meno condizionati dal volume delle ossa che li compongono. Ad esempio, il cluster 8 sembra formato prevalentemente da ossa piuttosto grandi, a segno del fatto che il volume in quel caso potrebbe essere stato un fattore determinante per la separazione di quel cluster dagli altri. La maggior parte dei cluster, però, sembra contenere ossa dal volume piuttosto variabile, ad indicare che probabilmente in quei

casi questa informazione non è risultata rilevante per la suddivisione.

5.2.2 Rendering 3D

Il seguente tool è stato creato per permettere la visualizzazione tridimensionale delle ossa più rappresentative di ogni cluster, consentendo ad esempio di confrontarle con altre appartenenti a un cluster differente. In questo modo risulta essere più semplice notare eventuali similarità tra la conformazione delle ossa appartenenti allo stesso cluster o differenze con ossa appartenenti a cluster diversi.

Sono state per questo scopo prodotte due ulteriori semplici applicazioni web, una per le scapole e l'altra per gli omeri, tramite l'utilizzo di Dash. In particolare, è stato necessario utilizzare un'estensione di Dash chiamata Dash-VTK. VTK (The Visualization Toolkit) [25] è un software che consente di manipolare e visualizzare dati scientifici di molti tipi diversi. Esso fornisce una serie di funzionalità tra cui il rendering e l'interazione con dati tridimensionali. Dash mette a disposizione alcuni componenti da implementare nelle applicazioni web che utilizzano proprio questo visualizzatore per il rendering di immagini tridimensionali. Di seguito verrà descritta l'applicazione prodotta per le scapole che nuovamente è analoga a quella degli omeri.

Come mostrato nella figura 5.6, la pagina web è composta da due righe di immagini tridimensionali, ognuna delle quali rappresenta un cluster e contiene le tre scapole più significative dello stesso. Le scapole qui rappresentate sono state ottenute applicando la maschera di segmentazione alle TC originali e ribaltando le immagini necessarie in modo tale da eliminare anche da questa rappresentazione l'informazione sull'orientamento della scapola e potersi concentrare solamente sulla forma delle stesse. Ogni riga contiene un menù a discesa tramite il quale è possibile cambiare il cluster visualizzato in quella specifica riga. Inoltre, ognuna delle finestre contenente i rendering è completamente interattiva e permette la rotazione e il movimento dell'osso in qualsiasi posizione desiderata. È possibile anche cambiare il colore del rendering e altri parametri tramite il pulsante in alto a sinistra di ognuna delle visualizzazioni. Ogni finestra ha anche associato un titolo che ne descrive il cluster di appartenenza ed il numero della TC da cui l'osso è stato estratto. In questo modo è possibile consultare lo stesso dato anche tramite tool differenti, come ad esempio XNAT (sezione 2.1). La decisioni riguardanti il numero di righe e il numero di visualizzazione per riga sono state prese in questo modo per

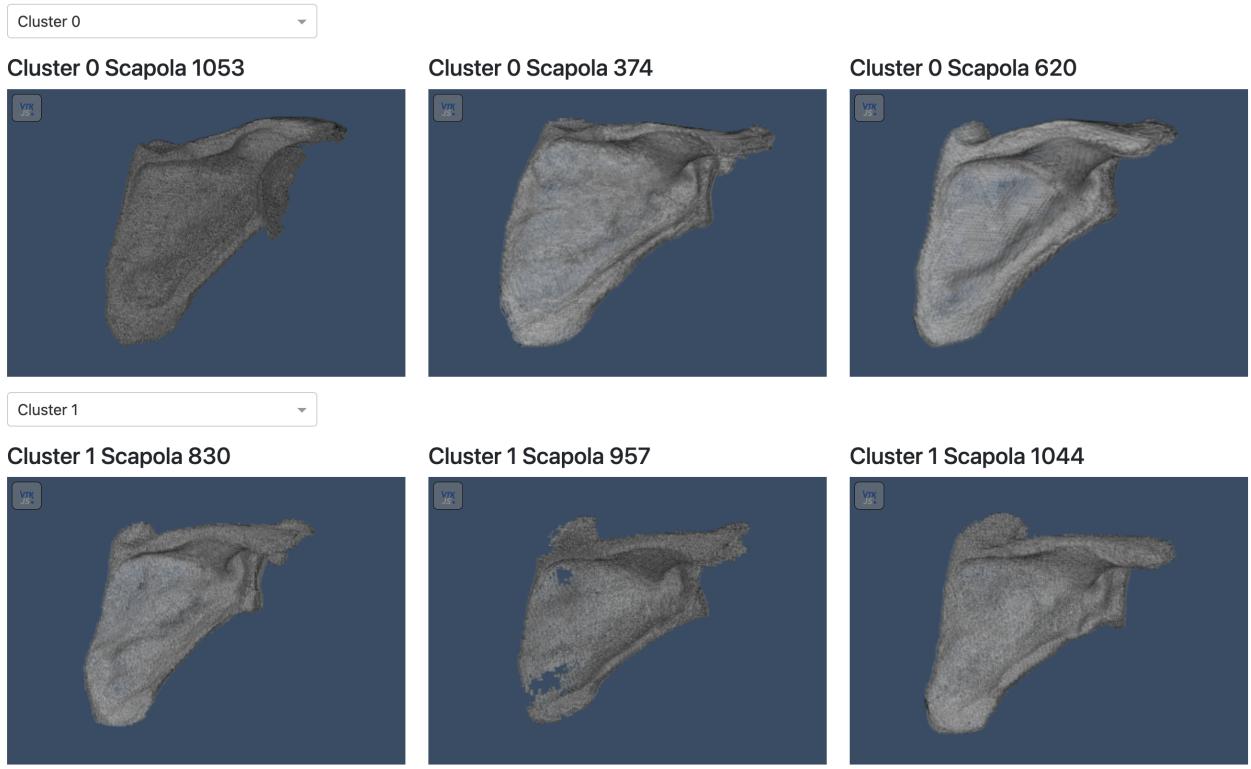


Figura 5.6: Pagina dell'applicazione web che mostra i rendering 3D dei volumi delle scapole

svariati motivi. In primo luogo, il tempo di caricamento di pagine web contenenti rendering 3D risulta essere piuttosto lungo, in quanto la mole di dati da visualizzare è elevata. L'aggiunta di nuove righe o nuove visualizzazioni renderebbe il caricamento iniziale della pagina ancora più lungo. Inoltre, è stato deciso di presentare solamente due cluster alla volta per rendere il confronto tra gli stessi più immediato. In questo modo, infatti, si visualizzano le ossa dei due cluster da comparare esattamente una sotto l'altra e ciò permette una comparazione più rapida. È possibile però estendere facilmente questi numeri, in quanto l'applicazione è stata sviluppata in modo da essere facilmente modificabile. L'aggiunta di nuove righe o nuovi dati da visualizzare sarebbe infatti implementabile in modo piuttosto rapido nella stessa in base alle necessità.

Le scapole più significative per ogni cluster sono state ottenute nuovamente tramite l'algoritmo k-nearest neighbors (KNN), già utilizzato nella sezione 5.1.3 per l'analisi delle scapole a scopo di suddivisione tra parte destra e sinistra. Nell'applicazione implementata, le ossa visualizzate sono ordinate da sinistra a destra in ordine crescente di distanza dal centroide (che di solito è un punto nello spazio non corrispondente a nessun dato fisico): la prima a sinistra corrisponde alla scapola con distanza minima, quella centrale corrisponde alla seconda scapola più vicina mentre la terza è quella con la distanza maggiore tra le tre visualizzate.

Lo scopo di questo tipo di visualizzazione è quindi quello di permettere una selezione variabile dei cluster da confrontare e di conseguenza visualizzare le ossa più rappresentative di ognuno di essi in modo da ottenere un confronto immediato tra scapole appartenenti allo stesso cluster e scapole appartenenti a cluster differenti. Effettuando diverse visualizzazioni, con la possibilità di movimento delle scapole in una qualsiasi posizione desiderata, è possibile indagare le caratteristiche principali di ognuna di esse. Ciò è stato creato con lo scopo di permettere l'individuazione di cluster differenti che in realtà rappresentano lo stesso morfotipo e che quindi potrebbero essere uniti, oppure per la ricerca di caratteristiche comuni alle scapole dello stesso cluster che ne permettano un'effettiva identificazione. Nel caso di dubbi su alcuni cluster, è possibile inoltre utilizzare questo tool congiuntamente al precedente, isolando dal grafico solamente i cluster interessati e visualizzando ulteriori ossa dello stesso cluster tramite altri strumenti per confermare o smentire determinate ipotesi scaturite dall'analisi degli stessi tramite l'applicazione. I due tool prodotti sono infatti stati pensati per essere utilizzati congiuntamente in quanto forniscono informazioni differenti sugli stessi dati consentendone un'analisi più completa.

5.3 Analisi dei Risultati

In questa sezione verranno analizzati i risultati prodotti dal clustering finale svolto sulle feature delle scapole e degli omeri ottenute rimuovendo l'informazione sull'orientamento delle stesse. Verranno innanzitutto mostrate alcune metriche utili calcolate sui cluster che evidenziano alcuni aspetti e differenze tra i cluster ottenuti. In seguito, verrà utilizzato il tool di visualizzazione tridimensionale per trarre alcune considerazioni pratiche sulle differenze o analogie della forma delle scapole tra cluster differenti. Questo secondo step non verrà riportato per gli omeri in quanto possiedono una forma molto più complicata da distinguere e sono meno importanti per l'effettiva classificazione della spalla.

5.3.1 Metriche sui Cluster Finali

In questa prima parte saranno mostrate alcune metriche calcolate sul risultato finale del clustering per entrambi i dataset che consentono di ottenere alcune informazioni utili riguardanti ciascun specifico cluster prodotto e di conseguenza confrontare i vari cluster. Le metriche calcolate sono di due tipi: metriche inter-cluster, cioè che valutano dei parametri scaturiti dal-

l’interazione tra i vari cluster, e metriche intra-cluster, cioè riguardanti informazioni interne ad ogni cluster. Saranno di seguito presentate le principali metriche con gli adeguati commenti.

Omeri

Il primo clustering analizzato è quello riguardante le feature degli omeri. Essi, come già detto, rappresentano ossa piuttosto difficili da distinguere a livello strutturale in quanto tutte piuttosto simili tra di loro. Inoltre, la loro utilità nel riconoscimento dei morfotipi della spalla è più limitata di quella delle scapole proprio per questo motivo. È però opportuno analizzare anche i risultati ottenuti dal clustering di questi ultimi, in quanto potrebbero fornire delle utili informazioni.

In primo luogo verranno analizzate due metriche intra-cluster. La prima indica la cardinalità di ogni cluster, cioè il numero di punti da cui esso è formato. La seconda indica invece la distanza media di ogni punto del cluster dal suo centroide. L’investigazione di queste due metriche permette di ricavare una rappresentazione generale della grandezza dei cluster e della loro variabilità.

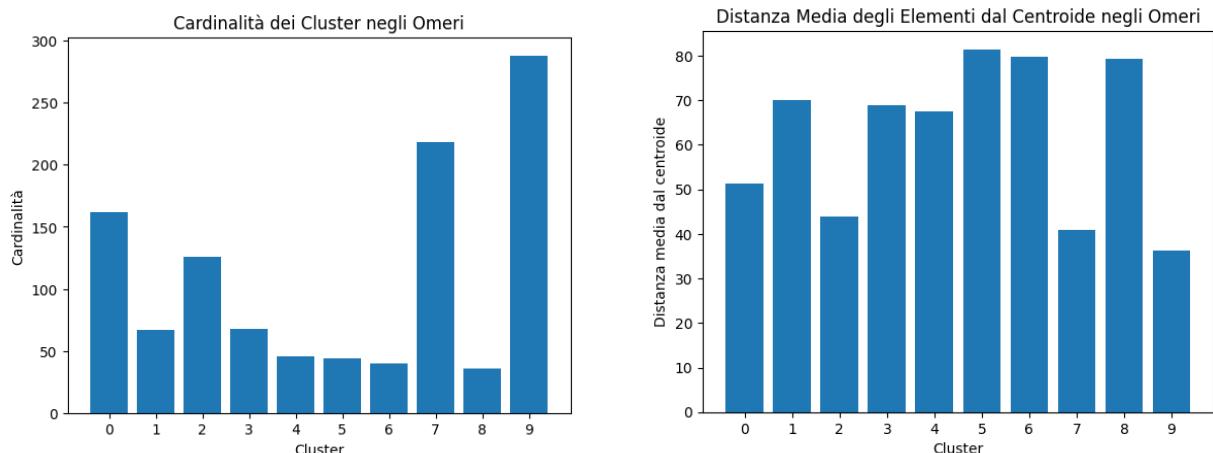


Figura 5.7: Metriche intra-cluster degli omeri

La figura 5.7 mostra i grafici relativi alle due metriche intra-cluster considerate. Sull’asse x sono presenti le label dei cluster mentre sull’asse y i valori delle metriche. È possibile notare come ci sia un grande squilibrio nelle cardinalità dei cluster. In particolare due di essi, il cluster 7 e il 9, possiedono molti più elementi degli altri. Si nota in particolare una netta suddivisione tra i cluster con più di 100 elementi e gli altri che invece ne contengono molti meno. L’utilità di questa metrica emerge soprattutto se analizzata insieme alla successiva, che rappresenta la distanza media dei punti dal centroide di ogni cluster. Quest’ultima fornisce

una misura di variabilità del cluster. Più la distanza media è alta, più il cluster possiede al suo interno valori variabili e risulta essere formato da una nuvola di punti meno densa. Questo fattore è strettamente legato al primo, in quanto la variabilità di un cluster dipende solitamente dal numero di dati che lo compongono. Quando si considerano queste metriche, è importante esaminare soprattutto i cluster che contengono pochi dati, in quanto potrebbero contenere degli outlier o osservazioni con caratteristiche particolari. In questo specifico caso, infatti, si può notare come i cluster più popolosi possiedano una distanza media più bassa mentre quelli più piccoli possiedano una distanza più elevata. Ciò è segno del fatto che i cluster piccoli sono formati da dati molto variabili e piuttosto distanti gli uni dagli altri, che corrisponderanno molto probabilmente a ossa con caratteristiche particolari. I cluster con cardinalità più elevata, invece, sono più compatti in quanto possiedono una distanza media inferiore. Essi saranno probabilmente formati da dati più uniformi e che quindi riusciranno a rappresentare meglio le caratteristiche morfologiche generali dell'osso. I cluster piccoli, invece, saranno in grado di fornire delle informazioni utili per quanto riguarda i casi particolari di queste ultime, come ad esempio delle lievi deformazioni che solamente poche ossa possiedono.

Dopo aver analizzato le metriche intra-cluster, verranno mostrati i risultati provenienti dal calcolo della metrica inter-cluster presa in considerazione, che consiste nella matrice delle distanze dei centroidi. Essa contiene, per ogni riga e ogni colonna, le distanze di un determinato centroide da tutti gli altri. Si otterrà così una matrice simmetrica 10x10, con valori pari a 0 sulla diagonale principale.

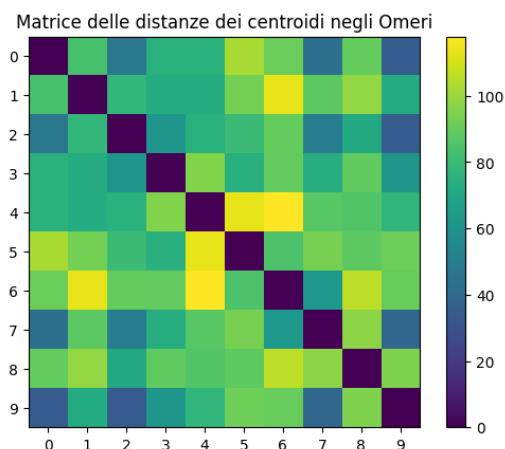


Figura 5.8: Matrice che rappresenta le distanze inter-cluster tra i cluster degli omeri

La figura 5.8 contiene una rappresentazione visiva della matrice appena descritta. L'elemento (i,j) della matrice rappresenta la distanza euclidea tra il centroide del cluster i e quello del cluster

j. Questo tipo di rappresentazione risulta essere molto utile in quanto fornisce a livello visivo una rappresentazione delle distanze tra i vari cluster, aiutando a comprendere quali saranno più simili tra loro e quali invece probabilmente più differenti basandosi sulla distanza tra i loro centroidi nello spazio d'origine. La legenda sui colori è rappresentata dalla barra verticale a destra. Più il colore è chiaro, più la distanza tra i centroidi è alta.

Da questa rappresentazione è possibile ricavare dati utili sulla struttura generale dei risultati del clustering e sul rapporto tra di essi. Dalla figura emerge che i cluster più distanti risultano essere il 4 e il 6, così come anche il 4 e il 5. In particolare si può notare come i cluster più piccoli siano in media più distanti dagli altri. Questo è segno che probabilmente nello spazio originale essi sono locati agli estremi dello stesso. Il cluster 0 è invece quello che possiede più cluster vicini, così come il 7 e il 9, che risultano effettivamente essere i più popolosi. Questa metrica fornisce quindi una visione della struttura dello spazio originale, che dai risultati qui scaturiti risulta essere rappresentata piuttosto fedelmente dalla proiezione in due dimensioni rappresentata nel grafico 5.3a ottenuto tramite UMAP.

In definitiva, sembra che per gli omeri il clustering sia stato piuttosto complicato, in quanto essi possiedono una struttura morfologica molto simile gli uni agli altri e le loro differenze risiedono in pochi dettagli concentrati soprattutto nella testa dell'omero. Le feature dell'autoencoder infatti faticano a catturare queste diversità, come si può notare anche dalle rappresentazioni dello spazio ottenute e presentate nelle sezioni precedenti. Per questo motivo, le feature ottenute dalle scapole e la loro suddivisione rappresentano l'informazione più rilevante per questo lavoro. Nella prossima parte verranno riportate le stesse metriche applicate ai risultati del clustering ottenuti proprio dalle feature delle scapole.

Scapole

Le feature estratte dalle scapole e la loro successiva suddivisione in cluster rappresentano il risultato più importante in quanto la loro forma risulta essere fondamentale per l'identificazione della tipologia di spalla in esame. Le metriche di seguito riportate sono le stesse già descritte per gli omeri e quindi il loro significato non sarà nuovamente esplicitato. Saranno però riportati i relativi grafici con le opportune considerazioni.

La figura 5.9 mostra la cardinalità e la media delle distanze per quanto riguarda i 10 cluster ottenuti dalle feature delle scapole. È possibile notare che anche in questo caso, come per

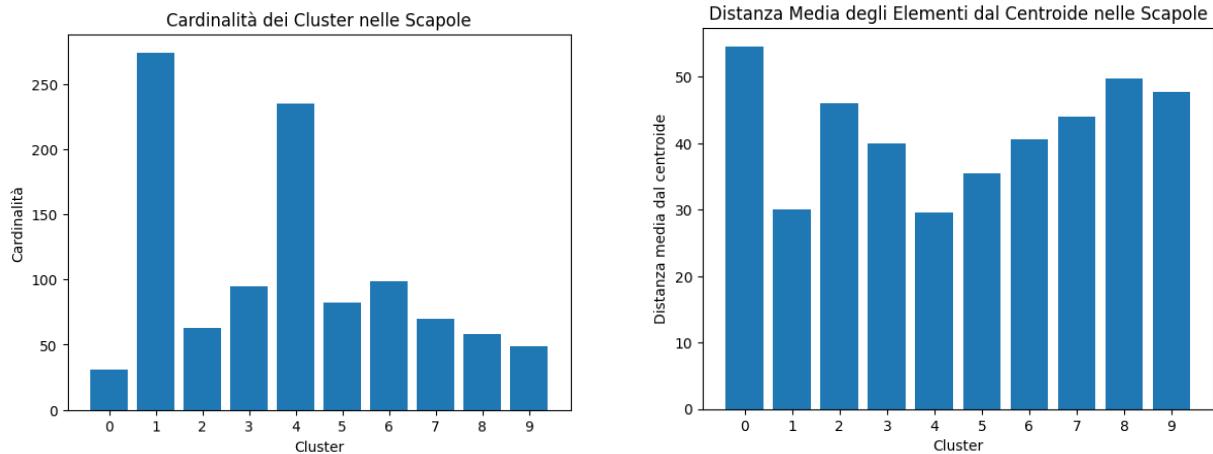


Figura 5.9: Metriche intra-cluster delle scapole

gli omeri, è immediatamente visibile uno squilibrio tra i cluster per quanto riguarda la loro cardinalità. In questo grafico però due cluster in particolare spiccano su tutti gli altri, mentre i rimanenti risultano essere circa della stessa dimensione senza grosse differenze tra di essi. Per quanto riguarda il rapporto con la metrica rappresentante le distanze medie, anche in questo caso i cluster più grandi sono quelli aventi la media delle distanze inferiore. I cluster più piccoli, invece, possiedono una media maggiore e quindi una maggiore variabilità dei dati al loro interno. Una piccola eccezione è rappresentata dal cluster 5, che pur essendo più piccolo dei 6, possiede una media delle distanze inferiore. Ciò potrebbe segnare che quel cluster contenga meno outlier pur possedendo un numero di elementi più contenuto. Gli altri cluster più piccoli, invece, soprattutto il numero 0, presentano molto probabilmente alcune caratteristiche inusuali e proprie di poche scapole ed è quindi buona norma prenderli in considerazione. È necessario ricordare, però, che la scelta del numero di cluster è stata appositamente presa considerando un numero di cluster superiore al numero teorico di morfotipi che dovrebbero esistere. I dati ottenuti dalle metriche sono molto utili anche allo scopo di fornire una prima idea riguardante quali cluster potrebbe essere opportuno agglomerare in quanto riferiti a scapole appartenenti a spalle dello stesso morfotipo. I cluster più grandi, infatti, rappresentano probabilmente già un tipo di spalla più consolidato, mentre i più piccoli potrebbero essere più facilmente propensi a un'agglomerazione.

Un'altra indicazione utile per comprendere le possibili agglomerazioni tra i cluster, ma anche molte altre informazioni, è data dalla matrice delle distanze, riportata in figura 5.10. Osservando la struttura generale, si nota una zona più scura verso il centro dell'immagine, che diventa più chiara all'esterno. Infatti, osservando i colori, si può notare come la riga rappresentante il

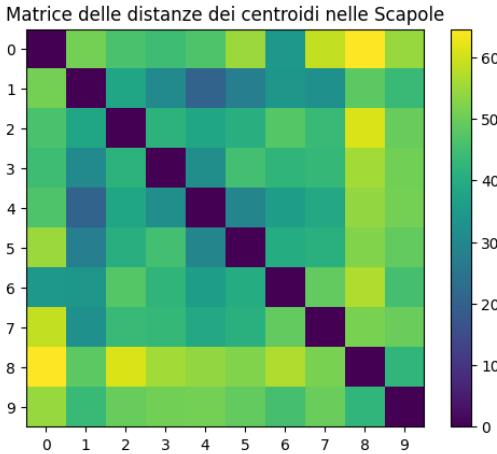


Figura 5.10: Matrice che rappresenta le distanze inter-cluster tra i cluster delle spalle

cluster 8 sia quella con i colori più chiari. Ciò è segno che esso è probabilmente localizzato nei contorni dello spazio e piuttosto isolato dagli altri. Questo aspetto indica il fatto che l'algoritmo sia stato in grado di isolare bene questo cluster dagli altri. Inoltre esso possiede una cardinalità ridotta ed è quindi più probabile che il suo centroide risulti lontano dai rimanenti. La zona rappresentante i cluster dal numero 1 al 7 è invece più scura, a segno del fatto che quei cluster sono più vicini tra loro e quindi le differenze saranno più difficilmente individuabili. Anche in questo caso, come per gli omeri, i cluster 1 e 4, che sono quelli con la cardinalità più elevata, sono anche quelli che possiedono una distanza in media minore da tutti gli altri. Anche alcuni cluster più piccoli però, come il 2 o il 5, presentano delle distanze contenute. Dai dati ottenuti sembra quindi che lo spazio sia diviso in una parte più compatta formata dai cluster dal numero 1 al 7, mentre gli altri 3 rimanenti siano collocati agli estremi e più isolati dai restanti. L'uso combinato di questi risultati con le precedenti metriche intra-cluster può fornire ulteriori indicazioni sull'identificazione delle possibili agglomerazioni tra cluster. I cluster più piccoli e più vicini potrebbero infatti rappresentare un'unica classe di spalle che è stata divisa a seguito della scelta del numero di cluster. Ad esempio, il cluster 0 risulta essere lontano da tutti gli altri e vicino solamente al cluster 6. Essendo entrambi cluster piccoli, potrebbe essere che essi in realtà facciano parte della stessa tipologia di spalla.

Questa rappresentazione risulta anche essere molto utile per la fase successiva che riguarda la ricerca di differenze o similarità nella forma delle scapole tramite una loro ispezione visiva utilizzando il tool realizzato che permette la visualizzazione tridimensionale delle stesse. Infatti, esso permette di confrontare i cluster a coppie ed è quindi opportuno trovare le coppie più rappresentative da confrontare. La matrice delle distanze inter-cluster permette una prima

valutazione in questo senso sulla base delle distanze tra i cluster.

Come già accennato, è possibile che i cluster trovati con il metodo sviluppato non corrispondano per nulla o solo in parte a morfotipi clinicamente significativi. Tuttavia, la metodologia qui sviluppata fornisce delle utili informazioni e può essere utilizzata come base per ulteriori raffinamenti, ad esempio utilizzando metodi che prevedano una suddivisione automatizzabile della scapola in sotto regioni da analizzare separatamente tramite le metodologie di autoencoding e clustering descritte in questo lavoro. Per ottenere maggiori riscontri, occorre effettuare come ultima attività un’analisi visiva delle scapole, che sarà descritta nella prossima sezione.

5.3.2 Considerazioni sui Rendering delle Scapole

In questa sezione verranno commentati alcuni aspetti legati alla conformazione delle scapole scaturiti dai risultati del clustering, utilizzando lo strumento di visualizzazione implementato e cercando di individuare alcune caratteristiche di diversità o similarità tra le ossa di cluster differenti.

Come già affermato, lo strumento di visualizzazione permette di confrontare a coppie le tre scapole più significative di ogni cluster. Per selezionare quali coppie di cluster siano più interessanti da esaminare risulta essere molto utile la matrice delle distanze riportata in figura 5.10. Sulla base dei risultati della metrica citata, è possibile individuare alcune coppie di cluster degne di un confronto diretto. L’obiettivo di questo confronto consiste nell’individuare a livello visivo delle differenze o similarità tra le scapole più caratteristiche di alcune coppie di cluster, considerate rappresentative, per fornire un’analisi dei risultati ottenuti. Le considerazioni svolte di seguito si basano sugli output delle segmentazioni, che potrebbero presentare degli errori rispetto ai dati reali. L’obiettivo rimane comunque quello di evidenziare le differenze puramente morfologiche tra le stesse, considerando le segmentazioni come corrette.

La figura 5.11 mostra il confronto tra il cluster 0 e il cluster 8, che risultano essere i più distanti. Prima di iniziare il confronto, è opportuno evidenziare che le scapole 841 e 842 del cluster 8 sono identiche, a segno del fatto che probabilmente nei dati originali erano presenti alcune TC duplicate. Questo non risulta essere un grande problema, visto l’elevata mole di dati, ma suggerisce anzi il fatto che il modello sia stato in grado di posizionare vicine tra loro nello spazio delle scapole effettivamente identiche, che sono risultate appartenere allo stesso cluster.

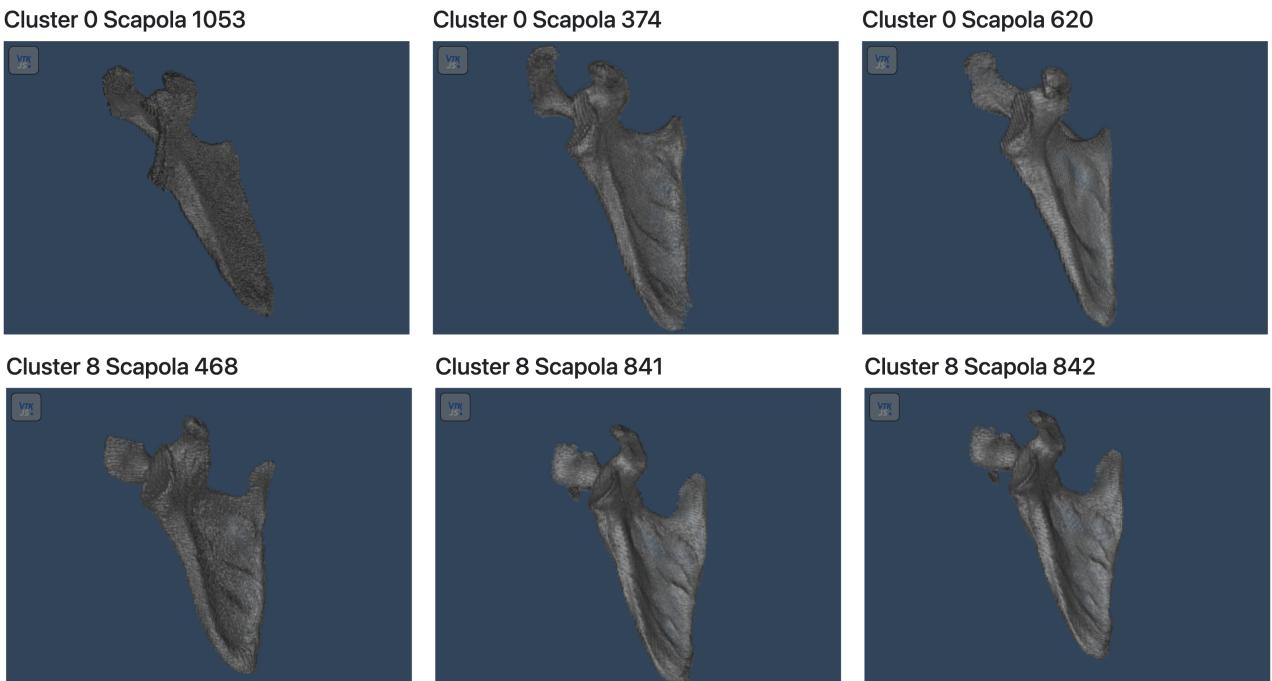


Figura 5.11: Confronto tra scapole del cluster 0 e del cluster 8

È possibile invece notare alcune differenze abbastanza evidenti tra i due gruppi di scapole. In primo luogo, l'acromion sembra essere più corto e largo nelle scapole del cluster 8 rispetto a quelle del cluster 0, in cui esso possiede una forma più allungata. Inoltre un'ulteriore differenza molto evidente riguarda la fossa del sovraspinato, che come già evidenziato nella sezione 2.4.2 rappresenta uno dei problemi di segmentazione in quanto assente da moltissime delle scapole. Si può qui notare però che, pur mancando questa parte a tutte le scapole, l'angolo superiore delle stesse nel cluster 8 risulta essere molto più accentuato rispetto a quello nel cluster 0. È importante anche evidenziare che il confronto è avvenuto tra cluster piuttosto piccoli, cioè dotati di pochi elementi ciascuno, in particolare il cluster 0 che contiene poco più di 25 scapole. Come già affermato, essi potrebbero contenere dei casi particolari di scapole e non rappresentare veri e propri morfotipi. Infatti, dato il numero di cluster scelto, non ci si aspetta di riconoscere un morfotipo per ognuno di essi.

Osservando ora il confronto tra i due cluster più grandi, cioè l'1 e il 4, riportato in figura 5.12, risulta essere molto più complicato notare delle differenze morfologiche. I due cluster, infatti, possiedono una distanza minima tra i rispettivi centroidi ed è quindi normale che le diversità tra le ossa che li compongono siano minime. Si può notare però che la suddivisione non è probabilmente in questo caso basata né sull'angolo superiore né sulla forma dell'acromion, in quanto questi due aspetti possiedono differenze anche tra scapole all'interno dello stesso cluster.

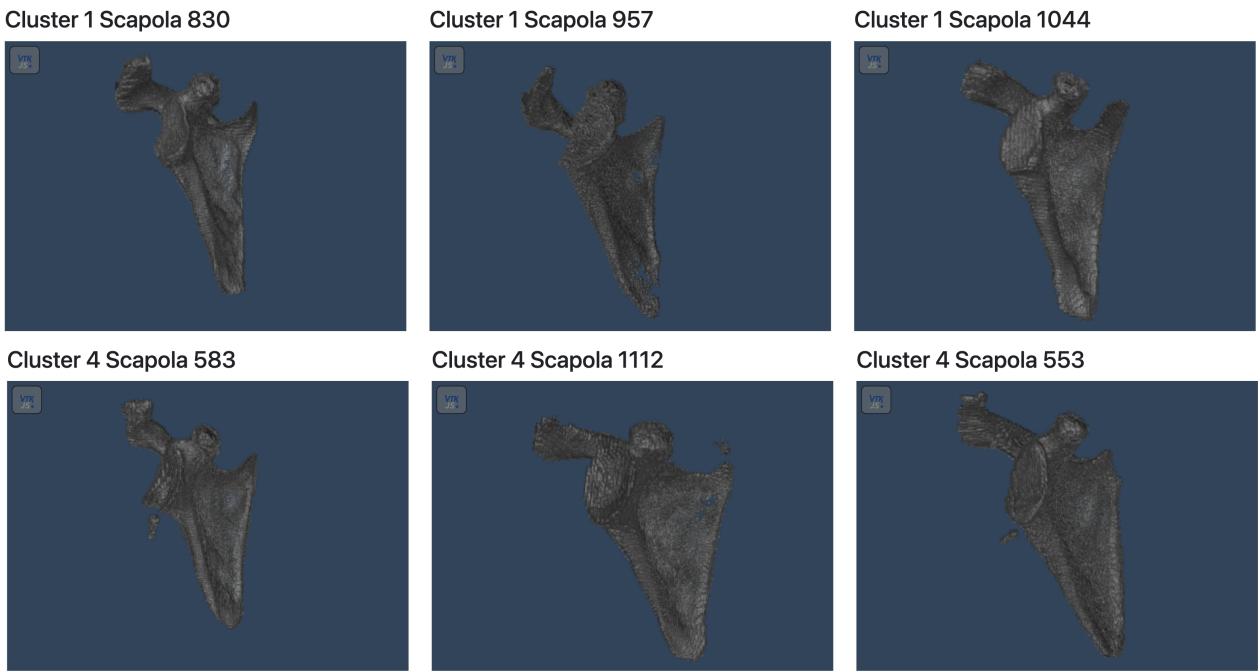


Figura 5.12: Confronto tra scapole del cluster 1 e del cluster 4

La prima scapola del cluster 4, ad esempio, ha un acromion molto più allungato della seconda e della terza. Le differenze tra questi cluster potrebbero quindi essere legate ad altri aspetti, ad esempio dalla forma e dimensione del processo coracoideo o della glena, anche se quest'ultima in alcune scapole, come la seconda del cluster 1, viene segmentata in modo non ottimale. Queste parti della scapola sono infatti le più evidenti a livello visivo e sono le uniche di cui è possibile notare a colpo d'occhio delle differenze. Il clustering potrebbe però essere basato anche su aspetti più difficilmente individuabili, che necessitano di conoscenza aggiuntiva nel settore. Bisogna anche evidenziare che essendo cluster molto grandi, con un numero di scapole superiore a 200, il confronto su tre sole scapole, pur essendo esse le più rappresentative, non può essere esaustivo. Questa rappresenta una delle limitazioni del lavoro, che andrà corretta e ampliata in modo appropriato in lavori successivi. Essi prevederanno lo sviluppo di tecniche di ispezione e visualizzazione più complete e fruibili, che permetteranno la visualizzazione arbitraria di un certo insieme di ossa personalizzato per ognuno dei cluster.

Infine, l'ultima coppia scelta per il confronto è quella formata dal cluster 0 e dal cluster 6. Essi infatti, sono due cluster piccoli in termini di cardinalità che però risultano essere molto vicini tra loro. Infatti, è piuttosto evidente nella riga del cluster 0 della figura 5.10, come esso sia piuttosto lontano da tutti gli altri e risulti essere vicino solamente al cluster 6, con una differenza di distanza non irrilevante. Questo aspetto, insieme al fatto che esso rappresenti il cluster più

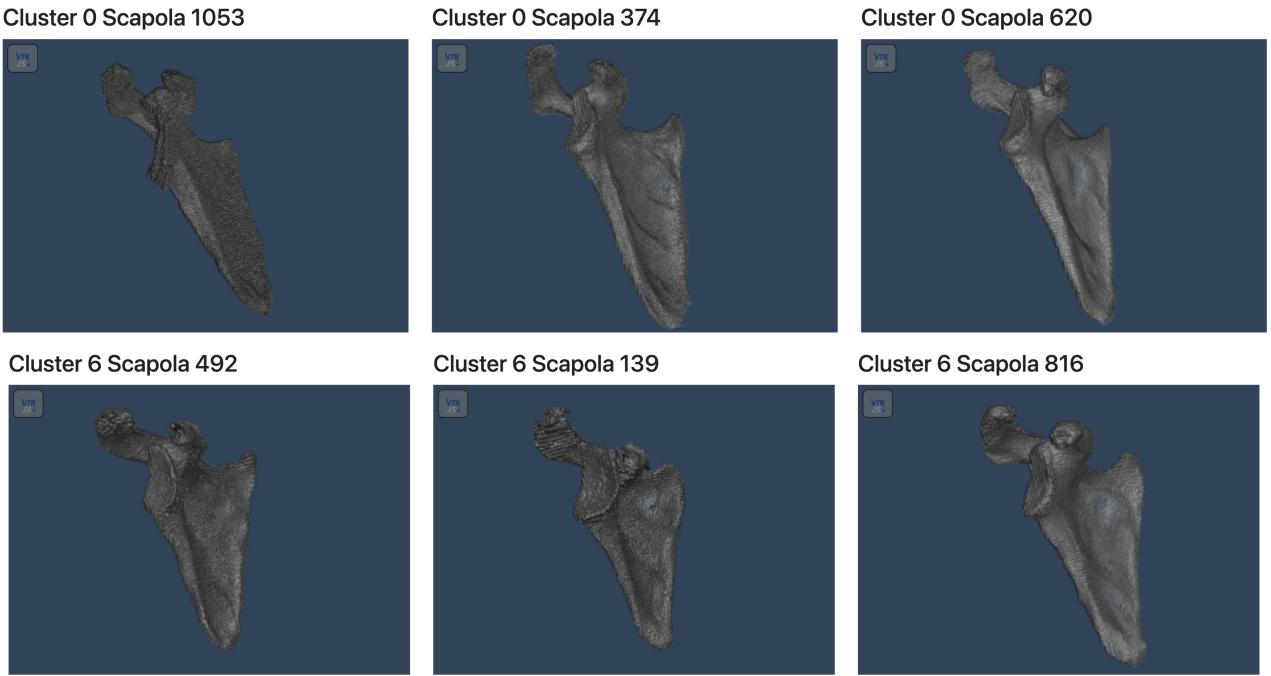


Figura 5.13: Confronto tra scapole del cluster 0 e del cluster 6

piccolo, può far pensare che questi due siano cluster molto propensi all'agglomerazione. Anche dal confronto visivo, presente in figura 5.13, si può notare come ci siano scapole molto simili tra i due cluster, come ad esempio la terza di ognuno dei due. Questo conferma anche a livello morfologico la possibilità di unione di questi due cluster in uno solo, che probabilmente riuscirà a rappresentare meglio una certa tipologia di spalla evitando di considerare gruppi di spalle che possiedano caratteristiche troppo specifiche.

Per quanto riguarda gli altri cluster qui non presentati, le differenze tra gli stessi sono piuttosto difficilmente individuabili. Il cluster 8 sembra essere quello che contiene scapole dalla morfologia più particolare e unica, aspetto consistente con la lontananza dello stesso da tutti gli altri. In generale, i dettagli più facili da identificare sono comunque spesso sempre riguardanti l'angolo superiore, la forma dell'acromion e quella del processo coracoideo, che sono le parti più facilmente individuabili e dalla forma più caratteristica. Le particolarità sulla glena, che come anche evidenziato dagli articoli presentati nel capitolo 1 rappresenta una delle parti fondamentali della scapola, sono molto più complicate da vedere a livello macroscopico e necessiterebbero di misurazioni particolari riguardo ad alcune sue caratteristiche come la dimensione o l'inclinazione. Inoltre, alcune informazioni non sono individuabili dal rendering in questione, come ad esempio il volume totale della scapola. Infatti, data la possibilità di permettere lo zoom e il movimento delle ossa offerta dalle finestre di visualizzazione, risulta difficile intuire

solamente a livello visivo il reale volume delle ossa in esame, in quanto non appena uno di essi viene zoomato si perde immediatamente la concezione del suo reale volume in relazione agli altri. Questa informazione è però stata riportata nel grafico presente nel tool di visualizzazione dello spazio in due dimensioni e quindi facilmente ricavabile utilizzando lo stesso.

Concludendo, in questo capitolo è stata presentata e analizzata la fase finale del progetto che riguarda una prima suddivisione delle scapole e degli omeri basata sulle feature estratte dalle immagini delle stesse. L'analisi finale dei risultati ha mostrato alcune interessanti particolarità dei cluster, che hanno permesso di trarre alcune conclusioni sulle possibili differenze tra le ossa che li compongono e sulla loro qualità, così come sulla possibilità di agglomerare alcuni di essi basandosi sulle metriche e sull'ispezione visiva delle stesse. L'obiettivo finale è infatti quello di permettere un'analisi approfondita dell'output del clustering, in modo da rendere la sua valutazione il più semplice e completa possibile fornendo utili informazioni e strumenti per la sua analisi. Tutto ciò è stato implementato in modo da consentire una facile modifica dei parametri allo scopo di riuscire in futuro ad ottenere, se esiste, una vera e propria classificazione delle spalle nei loro morfotipi.

Capitolo 6

Conclusioni e Sviluppi Futuri

Questo lavoro di tesi ha permesso la realizzazione di strumenti intuitivi e facilmente utilizzabili per la consultazione dei risultati ricavati dalla classificazione ottenuta basandosi su dati provenienti da TC di spalle di pazienti patologici. Il raggiungimento di questo obiettivo ha coinvolto una serie di step, che hanno portato infine alla realizzazione dei tool sopracitati. Avendo come unici dati a disposizione delle TC di spalle provenienti da pazienti patologici, è stato inizialmente necessario svolgere una fase di segmentazione delle ossa principali componenti la spalla, non prima di aver effettuato una serie di operazioni di preprocessing sui dati delle TC per renderli uniformi e processabili dal tool di segmentazione. Le maschere così prodotte sono state anch'esse uniformate e i dati riguardanti gli omeri e le scapole suddivisi in due differenti dataset. Ognuno di essi ha permesso l'addestramento di un modello di Deep Learning creato ad hoc che ha consentito l'estrazione di alcune feature, cioè delle caratteristiche riassuntive, ottenute basandosi sulle sole maschere di segmentazione contenute nei dataset stessi. Una volta estratte queste feature da ognuna delle immagini tramite l'applicazione del modello opportunamente addestrato, è stato possibile suddividere le immagini di ogni dataset in gruppi, ottenuti sulla base delle caratteristiche estratte, utilizzando un algoritmo di clustering. Questi gruppi, in numero fisso, rappresentano ossa con caratteristiche morfologiche simili tra loro e consentono una prima classificazione delle spalle originali. Per permettere una consultazione rapida ed efficiente dei cluster così ottenuti, sono stati implementati due tool complementari tra di loro che forniscono dei metodi interattivi per l'analisi di alcune caratteristiche fondamentali del risultato ottenuto e che permettono una valutazione del risultato della classificazione. Ciò rappresenta l'output finale di questo lavoro di tesi, che si è quindi occupato della parte più tecnica e onerosa del

processo di riconoscimento dei morfotipi della spalla, cioè quella che si occupa di tutte le fasi necessarie al processing delle immagini delle TC d'ingresso fino al raggiungimento di una loro prima classificazione, creando i modelli e utilizzando gli algoritmi necessari allo scopo.

Il processo ha coinvolto una lunga serie di step, alcuni dei quali hanno presentato delle limitazioni. Esse riguardano maggiormente le fasi iniziali, che coinvolgono i dati originali e il tool di segmentazione. I dati, infatti, non sempre sono risultati essere di sufficiente qualità e ciò ha dato origine a segmentazioni non perfette in certi casi. Inoltre, per quanto riguarda il tool di segmentazione, esso è risultato non essere sempre ottimale, producendo nella maggior parte dei casi una segmentazione piuttosto precisa ma mancante, nelle scapole, di una parte delle stesse chiamata fossa del sovraspinato che non è stata segmentata. Le altre fasi, invece, si sono svolte in modo più robusto in quanto sono stati implementati modelli e algoritmi creati ad hoc per il problema. I risultati ottenuti si possono considerare soddisfacenti. Il modello prodotto infatti è stato in grado di estrarre delle feature dalle immagini ad esso passate, come evidenziato dal primo clustering nel quale le scapole sono state chiaramente suddivise tra destra e sinistra. Inoltre, anche durante l'analisi del secondo clustering, sono emerse alcune importanti caratteristiche che hanno portato a una suddivisione finale dello spazio in gruppi piuttosto eterogenei tra di loro, ma rappresentanti ossa di forme differenti. Una delle parti più complicate risulta essere quella riguardante la comprensione del metodo adottato dall'algoritmo per suddividere un gruppo dall'altro, anche se alcune delle diversità nella forma delle ossa sono state piuttosto facilmente individuabili.

Il compito di capire le modalità con cui le ossa sono state suddivise nei vari cluster fa parte delle prime fasi degli sviluppi futuri del progetto e spetta ai medici ortopedici. Infatti, il motivo per cui sono stati prodotti i tool di visualizzazione è proprio quello di permettere ai medici di utilizzare gli stessi per valutare la bontà della classificazione ottenuta in modo intuitivo. Lo scopo di questa valutazione sarà quello di ricevere dei feedback da esperti del settore sull'effettiva praticità della classificazione ottenuta, cercando di evidenziare i problemi e le caratteristiche dei cluster. L'idea è quella di riuscire ad ottenere informazioni sul corretto numero di cluster da produrre tramite l'ispezione di alcuni aspetti degli stessi da parte di professionisti che possano indicare la giusta direzione da intraprendere. Queste interazioni con i medici permetteranno una serie di correzioni riguardanti alcuni aspetti del metodo che possano portare ad ottenere ad ogni iterazione una suddivisione differente, fino a quando essa non verrà giudicata dagli esperti come una classificazione sensata delle spalle in morfotipi. Le correzioni saranno pro-

babilmente concentrate inizialmente sul metodo di clustering e in particolare sulla ricerca del corretto numero di cluster da ricavare. Potrebbe però essere necessario successivamente raffinare anche altri aspetti del metodo di clustering, ad esempio provando ad utilizzare algoritmi differenti. Inoltre, potrebbe risultare utile modificare anche il modello implementato sulla base delle considerazioni dei medici, in quanto essi potrebbero fornire utili informazione per guidare il training del modello verso parti o caratteristiche specifiche delle ossa che potrebbero essere considerate più significative per la classificazione finale. Il modello attuale, infatti, riceve in ingresso solamente le immagini delle segmentazioni e quindi si basa interamente sulla forma delle ossa in generale, senza aver alcuna informazione aggiuntiva sulle parti più importanti da considerare.

Dopo una serie di fasi di confronto con i medici e conseguente correzione incrementale di alcuni aspetti del processo, sarà infine possibile ottenere un risultato vero e proprio che potrà essere positivo o negativo. Infatti, ci si aspetta infine di ottenere una classificazione delle spalle che abbia senso dal punto di vista clinico e che riguarda particolari caratteristiche morfologiche delle ossa che le compongono che potrebbero addirittura non essere mai state considerate prima. Non è però detto che ciò si riesca ad ottenere. Infatti, potrebbe anche risultare che una classificazione delle spalle sulla base delle immagini disponibili non sia possibile. È opportuno considerare che i dati in ingresso provengono da TC di pazienti patologici e non è quindi scontato che una vera e propria classificazione possa scaturire dalle stesse. Potrebbe infatti accadere che l'insieme di spalle prese in considerazione non risulti abbastanza rappresentativo per una classificazione di questo tipo, anche considerando il fatto che le ossa disponibili appartengono a pazienti patologici e quindi possiedono tutte qualche deformazione, seppur leggera.

Lo step finale del processo consisterebbe poi, nel caso in cui una classificazione scaturisse, nel trovare un collegamento tra eventuali classi di spalle in pazienti patologici e le stesse in spalle di pazienti sani. Ovviamente, questo passo necessita della possibilità di accedere a TC di pazienti sani, che al momento non sono disponibili. Nel caso in cui non fosse emersa alcuna classificazione significativa delle spalle patologiche, sarebbe possibile anche applicare gli stessi step descritti in questo lavoro a un insieme di spalle sane, con le opportune modifiche, che potrebbero invece fornire una classificazione opportuna. Basandosi sulle caratteristiche della stessa, potrebbe essere possibile di conseguenza l'ottenimento di una classificazione correlata in spalle patologiche. La presenza di questa correlazione risulterebbe essere un grande aiuto per gli ortopedici che sarebbero in grado di classificare la spalla del paziente in esame e, tramite

la correlazione tra la classe di appartenenza della spalla sana e la corrispondente della spalla patologica, comprendere a priori le possibili modalità di operazione da effettuare.

Michele Signor

Bibliografia

- [1] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*, pages 132–149, 2018.
- [2] Stijn J Casier, Robin Van den Broecke, Jan Van Houcke, Emmanuel Audenaert, Lieven F De Wilde, and Alexander Van Tongel. Morphologic variations of the scapula in 3-dimensions: a statistical shape model approach. *Journal of shoulder and elbow surgery*, 27(12):2224–2231, 2018.
- [3] Min Chen, Xiaobo Shi, Yin Zhang, Di Wu, and Mohsen Guizani. Deep feature learning for medical image analysis with convolutional autoencoder neural network. *IEEE Transactions on Big Data*, 7(4):750–758, 2017.
- [4] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Deep convolutional autoencoder-based lossy image compression. In *2018 Picture Coding Symposium (PCS)*, pages 253–257. IEEE, 2018.
- [5] Jun Kang Chow, Zhaoyu Su, Jimmy Wu, Pin Siang Tan, Xin Mao, and Yu-Hsing Wang. Anomaly detection of defects on concrete structures with the convolutional autoencoder. *Advanced Engineering Informatics*, 45:101105, 2020.
- [6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, pages 226–231, 1996.
- [7] Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Christophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dominique Jennings, Fiona Fennessy, Milan Sonka, et al. 3d slicer. <https://www.slicer.org/>.

- [8] Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Christophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dominique Jennings, Fiona Fennessy, Milan Sonka, et al. 3d slicer as an image computing platform for the quantitative imaging network. *Magnetic resonance imaging*, 30(9):1323–1341, 2012.
- [9] Open Health Imaging Foundation(OHIF). Ohif viewer. <https://ohif.org/>.
- [10] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th international conference on data mining workshops (ICDMW)*, pages 241–246. IEEE, 2016.
- [11] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part II 24*, pages 373–382. Springer, 2017.
- [12] Benjamin Hohlmann, Malte Asseln, Jiacheng Xu, and Klaus Radermacher. Investigation of morphotypes of the knee using cluster analysis. *The Knee*, 35:157–163, 2022.
- [13] Plotly Technologies Inc. Dash. <https://dash.plotly.com/>.
- [14] Plotly Technologies Inc. Plotly. <https://plotly.com/python/>.
- [15] Fabian Isensee, Paul F Jaeger, Simon AA Kohl, Jens Petersen, and Klaus H Maier-Hein. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature methods*, 18(2):203–211, 2021.
- [16] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChe journal*, 37(2):233–243, 1991.
- [17] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*, pages 52–59. Springer, 2011.
- [18] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.
- [19] Medacta. Medacta international. <https://www.medacta.com/>.

- [20] The Medical Imaging Technology Association (MITA). Dicom. <https://www.dicomstandard.org/>.
- [21] Neuroinformatics Research Group (NRG). Xnat. <https://www.xnat.org/>.
- [22] Katrien Plessers, Peter Vanden Berghe, Christophe Van Dijck, Roel Wirix-Speetjens, Philippe Debeer, Ilse Jonkers, and Jos Vander Sloten. Virtual reconstruction of glenoid bone defects using a statistical shape model. *Journal of shoulder and elbow surgery*, 27(1):160–166, 2018.
- [23] Lior Rokach and Oded Maimon. Clustering methods. *Data mining and knowledge discovery handbook*, pages 321–352, 2005.
- [24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.
- [25] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit (4th ed.)*. Kitware, 2006.
- [26] Daniel J Trosten and Puneet Sharma. Unsupervised feature extraction—a cnn-based approach. In *Image Analysis: 21st Scandinavian Conference, SCIA 2019, Norrköping, Sweden, June 11–13, 2019, Proceedings 21*, pages 197–208. Springer, 2019.
- [27] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
- [28] Filip Verhaegen, Alexander Meynen, Jonathan Pitocchi, Philippe Debeer, and Lennart Scheyns. Quantitative statistical shape model-based analysis of humeral head migration, part 2: Shoulder osteoarthritis. *Journal of Orthopaedic Research*, 41(1):21–31, 2023.
- [29] Jakob Wasserthal, Hanns-Christian Breit, Manfred T Meyer, Maurice Pradella, Daniel Hinck, Alexander W Sauter, Tobias Heye, Daniel T Boll, Joshy Cyriac, Shan Yang, et al. Totalsegmentator: Robust segmentation of 104 anatomic structures in ct images. *Radiology: Artificial Intelligence*, 5(5), 2023.