Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di un'Interfaccia CAD per Endoscopie Gastrointestinali

Relatori

Prof. Monica Visintin

Prof. Guido Pagana

Candidato

Matteo Quarta

Anno Accademico 2022/2023

Sommario

Il tratto gastro-intestinale umano (GI) è soggetto allo sviluppo di molteplici patologie, che possono presentarsi come la comparsa anomala di mucosa o alterazioni del tessuto che lo compongono. Tali patologie possono variare in grande misura nella loro gravità, da lievi disturbi trattabili fino a forme letali di cancro. Tra di esse troviamo le metaplasie gastrointestinali (MGI), che ricadono nel mezzo della precedente scala trattandosi di un'alterazione reversibile del tessuto e sono da considerare una forma pre-tumorale. La *World Health Organization* (WHO) stima che ogni anno 3.5 milioni di pazienti incorrano in condizioni tumorali nel tratto intestinale, che con un tasso di mortalità del 63% causa circa 2.2 milioni di morti all'anno. [1]

In questo contesto è possibile inserire le intelligenze artificiali. L'interesse verso questo settore ha permesso a ricercatori e studiosi di realizzare architetture via via più specializzate nella risoluzione di problemi specifici, come l'analisi di immagini e l'estrazione di conoscenza da dati clinici. Il raggiungimento di risultati via via più soddisfacenti ha dunque portato a chiedersi come sia possibile impiegare queste tecnologie in ambiti reali, non solo in contesti a bassa specializzazione per la risoluzione di problemi ripetitivi ma anche in ambiti più complessi a supporto di decisioni da prendere di fronte di situazioni complesse.

Questo lavoro dunque si propone di studiare l'impiego di intelligenze artificiali in ambito medico, studiando le opportunità offerte da due

approcci di riferimento ed analizzandone le principali criticità, al contempo introducento delle interfacce grafiche per avvicinare modelli precedentemente preclusi da competenze informatiche specifiche per poterli utilizzare.

Indice

Capitolo 1

Introduzione

1.1 Cosa Sono le Metaplasie

Prima di entrare nei dettagli tecnici degli strumenti e delle possibilità oggi a disposizione per lo sviluppo di un applicativo, è necessario conoscere il problema di riferimento. Prima di formalizzare il probelma ed una sua possibile soluzione, vediamo dunque come esso si presenta nella sua forma originale, ovvero il problema reale.

Metaplasia è uno dei termini medici più impropriamente utilizzati nel linguaggio comune, se non addirittura in modo erroneo. La tendenza è infatti quella di ritenere la comparsa di questa condizione medica al pari della comparsa di un tumore, condizione medica ben differente che è detta invece neoplasia. Nel caso del tratto intestinale, entrambe le condizioni sono accomunate dai sintomi con i quali si presentano, ma le implicazioni in un caso e nell'altro sono ben diverse.

Le due condizioni sono fortemente collegate. Una metaplasia puo' infatti diventare se non identificata e trattata per tempo una neoplasia, essendo la prima uno stadio intermedio tra una condizione sana ed una patologica. Una sostanziale differenza tra le due è che la metaplasia,

a differenza della neoplasia, è una condizione reversibile. La comparsa della metaplasia è infatti dovuta all'introduzione di uno stimolo esterno che porta il tessuto originale a cambiare la sua composizione cellulare, portandolo dunque ad essere una metaplasia. È fondamentale però tener conto che un'alterazione della composizione cellulare non implica necessariamente un'alterazione del genoma del tessuto, cosa che rende possibile la reversibilità della metaplasia. Se lo stimolo in questione viene rimosso, il tessuto ritorna dunque allo stato originale, senza alcuna alterazione.

Lo stimolo in questione è solitamente un cambio d'ambiente, che porta le cellule a cambiare per adattarsi al nuovo. L'esempio più classico di stimolo che porta alla comparsa di metaplasia è forse l'irritazione dovuto al fumo di sigaretta. Esso porta il tessuto a cambiare per adattarsi al nuovo ambiente, diventando dunque più resistente all'irritazione e inibendo la sensibilità al fumo. Tuttavia l'inibizione della sensibilità al fumo porta con sé gravi conseguenze, in quanto esso prevede la trasformazione dell'epitelio cilindrico con uno pluristratificato, che va anche ad inibire i meccanismi protetittivi dell'epitelio stesso e lasciandolo dunque più vulnerabile a infezioni e sostanze tossiche provenienti dall'esterno.

1.2 Metaplasie Gastrointestinali

Le metaplasie gastrointestinali sono dunque, come il nome suggerisce, delle metaplasie che si vengono a sviluppare nello stomaco. L'interesse verso questa specifica parte del corpo umano nasce dal fatto che essa

è già normalmente soggetta allo sviluppo di condizioni patologiche ad alto tasso di mortalità, comea ad esempio l'adenocarcinoma gastrico, che secondo studi ha una possibilità di svilupparsi fino a dieci volte maggiore in soggetti che hanno sviluppato una metaplasia gastrica.

Utilizzando il metodo diagnostico di Jass e Felipe è possibile distinguere tre tipi di metaplasie gastrointestinali, di cui due sono dette incomplete in quanto introducono alcune distorsioni ghiandolari nel tessuto epiteliale, mentre la terza è detta completa poiché replica perfettamente il fenotipo dell'intestino tenue. È necessario notare che le tipologie incomplete sono quelle che hanno un maggior probabilità di portare a forme patologiche più severe. Lo sviluppo della neoplasia viene indotto mediante quel che è nota come cascata di Correa, in cui ogni alterazione conduce alla seguente in maniera via via più severe fino a sfociare definitivamente in una neoplasia.

Il metodo diagnostico principale per l'identificazione di tali alterazioni è dunque l'endoscopia intestinale. La convenzionale endoscopia a luce bianca non fornisce tuttavia abbastanza informazioni per poter distinguere in maniera sufficientemente precisa le lesioni gastrointestinali, e per questo motivo gli strumenti moderni sono attrezzati con più filtri ottici a diverse lunghezze d'onda. Il *Narrow Band Imaging* ad esempio filtra la luce bianca andando a bloccare alcune specifiche lunghezze d'onda, da cui il nome, permettendo di esaltare le caratteristiche della mucosa cellulare. Inoltre, la strumentazione moderna offre varie modalità di acquisizione, che consistono nell'impiego di diversi

filtri applicabili all'immagine, come la rilevazione di superfici ed il miglioramento del tono dell'immagine, al fine di fornire più prospettive su cui basarsi durante l'analisi.

Seppur importante, l'analisi endoscopica non è l'unico strumento a disposizione per la diagnosi di metaplasie, ed anzi è in discussione la sua rilevanza come metodo diagnostico. La biopsia è un metodo diagnostico che permette di ottenere campioni di tessuto per analizzarne la composizione cellulare, ed è da considerarsi il metodo diagnostico per eccellenza per la determinazione della presenza di metaplasie gastrointestinali. Questo è dovuto al fatto che anche per un medico esperto non è sempre possibile accertare con abbastanza sicurezza la presenza o meno di metaplasie gastrointestinali solo attraverso l'analisi endoscopica. La biopsia è però un metodo più invasivo e dalle tempistiche più lunghe, cose che rende la sua applicazione in larga scala poco sostenibile per i sistemi sanitari. L'interesse di un'analisi endoscopica assistita dall'intelligenza artificiale nasce dunque qui, dalla possibilità di ottenere teniche di *screening* che possano contemporaneamente essere più sostenibili e più affidabili.

Capitolo 2

Dataset

Allenando un modello di intelligenza artificiale è necessario rifelttere sul tipo di dati che si vogliono utilizzare per l'apprendimento. Nel mondo accademico sono disponibili vari *dataset* che vengono impiegati come banco di prova nel momento in cui si vuole produrre una nuova architettura o tipologia di modello, ma tali dati spesso non sono adatti l'architettura è stata già dimostratata essere adatta e la si vuole invece integrare in un applicativo reale, rendendo quindi necessario l'impiego di una diversa tipologia di dati. Questi *dataset* possono tuttavia essere impiegati non come principale fonte di informazioni ma come supporto ausiliario ad essi, come nel nostro caso.

Nella fase di *training* del modello sono stati impiegati due *dataset* , uno pubblico ed impiegato largamente per studiare le applicazioni dell'intelligenza artificiale in ambito medico ed un altro che è stato invece creato *ad-hoc* per queso progetto a partire da alcune immagini fornite dall'Ospedale Mauriziano di Torino, che chiameremo *dataset* Mauriziano per semplicità. Il primo è il dataset HyperKvasir 2.4, che raccoglie una moltitutine di immagini di vari riscontri patologici di più parti dello stomaco. La più grande criticità di questo *dataset* , che lo rende

inadatto ad essere impiegato come unica fonte di dati, è che nonostante l'elevato numero di campioni da cui è composto, in termini numerici sia dal punto di vista della quantità di immagini che di patologie riportate, non contiene rilevazioni di metaplasie gastrointestinali. Il *dataset* Mauriziano si trova invece in una situazione diametralmente opposta, poiché le immagini contenute in esso sono state acquisite specificatamente per questo progetto ma sono disponibili in quantità estremamente limitata.

2.1 Dataset Mauriziano

Il dataset si presenta come una raccolta di quasi 1200 acquisizioni endoscopiche raccolte dall'Ospedale Mauriziano su circa 200 pazienti, suddiviso in immagini dette "originali", in quanto non presentano alcuna annotazione, ed immagini "alterate", in quanto è stata aggiunta dal personale medico una o più tracce che indicano la zona considerata patologica. Di queste immagini tuttavia solo 49 contengono rilevazioni patologiche, che sono le immagini realmente utilizzabili per l'apprendimento dei modelli, un numero decisamente troppo basso in casi in cui si voglia ricorrere a tecniche di *deep learning*.

Tuttavia, i modelli di *shallow learnging* non operano operano direttamente sulle immagini in *input* ed hanno anzi bisogno che queste siano trasformate in un formato che possa essere utilizzato per l'apprendimento. Tale trasformazione, descritta di seguito, rende il nostro *dataset* di dimensioni molto più generose, e puo' in parte utilizzare anche le acquisizioni totalmente sane.

Come punto di partenza si prendono le immagini patologiche originali

e quelle alterate. Va fatto notare che per alcune immagini alterate non è stata fornita la corrispondente immagine originale. Per la creazione del modello di *shallow learning*3.2 questo non è un problema, in quanto le tracce non sono distinguibili se non guardando l'immagine nella sua interezza. Le immagini vengono dunque ritagliate in modo da rimuovere i bordi superflui provenienti dall'interfaccia dell'ISCAN, dati circa l'acquisizione stessa che sono presenti in sovraimpressione e la visuale ausiliaria che non è utilizzata nel modello. Un esempio di immagine dopo il ritaglio è mostrato in figura 2.1. Come anticipato possiamo in questa fase utilizzare alcune immagini non patologiche, e se ne utilizzano per la precisione 25 per non creare un eccessivo scompenso nei dati.

Le immagini alterate contengono delle tracce che delimitano le zone patologiche. Tale tracce non sono direttamente impiegabili per l'apprendimento, ed è quindi necessario andare a costruire delle maschere che raccolgano l'intera area di interesse.

Il punto di partenza è una maschera totalmente vuota. Possiamo intendere per maschera una matrice binaria $H \times W$, con H la larghezza dell'immagine e W la sua altezza, dove ogni *bit* rappresenta un pixel nell'immagine originale. Se quel *bit* è posto a 1, allora il pixel corrispondente è considerato un pixel appartenente all'area patologica. Sfruttando il colore utilizzato per le annotazioni, un verde acceso come indicato nella figura 2.1, è possibile porre a 1 tutti i pixel che hanno quel colore ed ottenere lo stesso anello nella nostra maschera. Si procede dunque applicando alla nostra maschera un operatore di dilatazione

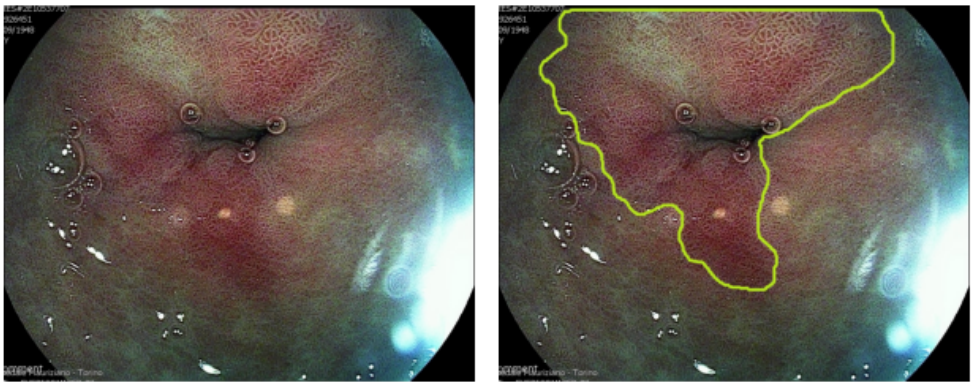

Figura 2.1: Esempio di immagine originale (sinistra) e di immagine alterata (destra)

per 3 iterazioni, allo scopo di assicurarsi che l'anello sia chiuso, fondamentale per l'operatore successivo. Un secondo operatore, questa volta di chiusura, riempie il nostro anello ed otteniamo così una maschera che racchiude l'intera area. Per annullare gli effetti della prima dilatazione, viene infine applicato un operatore di erosione, sempre per 3 iterazioni. Una raffigurazione del risultato è visibile nella figura 2.2.

L'annotazione viene dunque salvata per essere utilizzata in un secondo momento. Per praticità ne vengono salvate due versioni, una in formato *npy* che consente di essere caricata ed ottenere direttamente la maschera binaria calcolata, e una in formato *png* che è invece il formato utilizzato dalla libreria `PyTorch`. In ogni caso, è necessario salvare

le maschere con una codifica *lossless* altrimenti i processi di *training* produrrebbero errori, cosa che entrambi i formati permettono di fare.

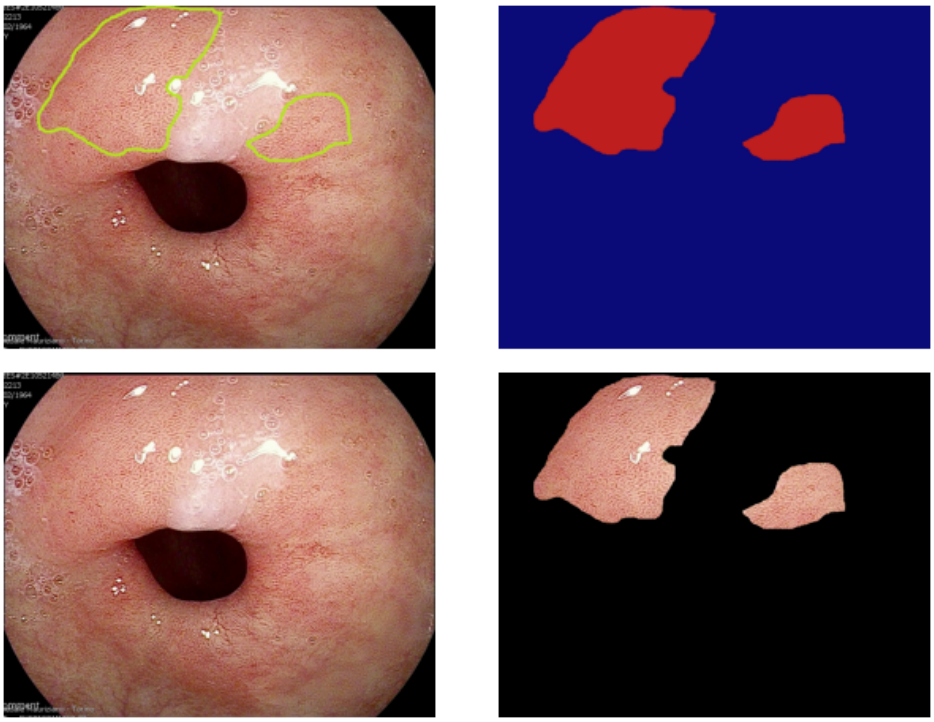

Figura 2.2: Processo di estrazione delle maschere.
In senso orario: immagine annotata, maschera calcolata, immagine originale, ritaglio della sola parte patologica.

Per i modelli di *deep learning* non sono necessari ulteriori passaggi, ma per il nostro modello di *shallow learning* sono necessarie ulteriori fasi di *pre-processing* che discutiamo di seguito. Il primmo passo è la trasformazione dell'immagine in scala di grigi, che porta ogni *pixel* in un valore tra 0 e 255, dove 0 indica un *pixel* totalmente nero e 255 uno totalmente bianco. Si procede dunque selezionando tutti i *pixel* di intensità superiore a 40, generando quella che potremmo definire una

maschera temporanea. A questa maschera viene applicato un operatore morgologico di erosione per due iterazioni e uno di dilatazione per 20. Vengono dunque spenti tutti i *pixel* che non fanno parte di questa maschera. Questa operazione serve per rimuovere eventuale rumore presenti nell'immagine e le zone non a fuoco. Si procede dunque con la rimozione dei riflessi di luce che avviene tramite algoritmo *K-Means*, con un numero di *clusters* fissato a 30. I *clusters* vengono inizializzati come punti equispaziati tra 0 e 255 e l'algoritmo, inteso come convergenza, viene iterato solo una volta. Questo accade poiché si ha già un'idea dei valori che si stanno cercando, e non sono quindi necessarie inizializzazioni casuali o iterazioni multiple. A questo punto si spengono, ovvero vengono posti a 0, i pixel che ricadono nei 3 *clusters* corrispondenti ai valori più luminosi. All'immagine viene dunque applicato il gradiente di Sobel al fine di accentuare venature e discontinuità nell'immagine.

A questo punto per estrarre i campioni statistici dalla nostra immagine è necessario andare a dividere l'immagine in piccole zone denominate *Region of Interest* (ROI). Queste ROI hanno una dimensione fissa di 50×50 *pixel*, con una sovrapposizione del 40% tra una ROI e la successiva in entrambe le direzioni, ovvero di 20 pixel. In fase di estrazione vengono filtrate via le ROI che non contengano effettivamnte del tessuto andando a scartare quelle che siano composte per più del 60% da pixel neri. Per decidere se una ROI sia patologica o sana, si procede a confrontare la posizione della ROI con la corrispondente zona nella maschera di predizione ricavata in precedenza: se la ROI è contenuta per il 50% o più nella maschera la ROI viene considerata patologica,

altrimenti viene considerata sana.

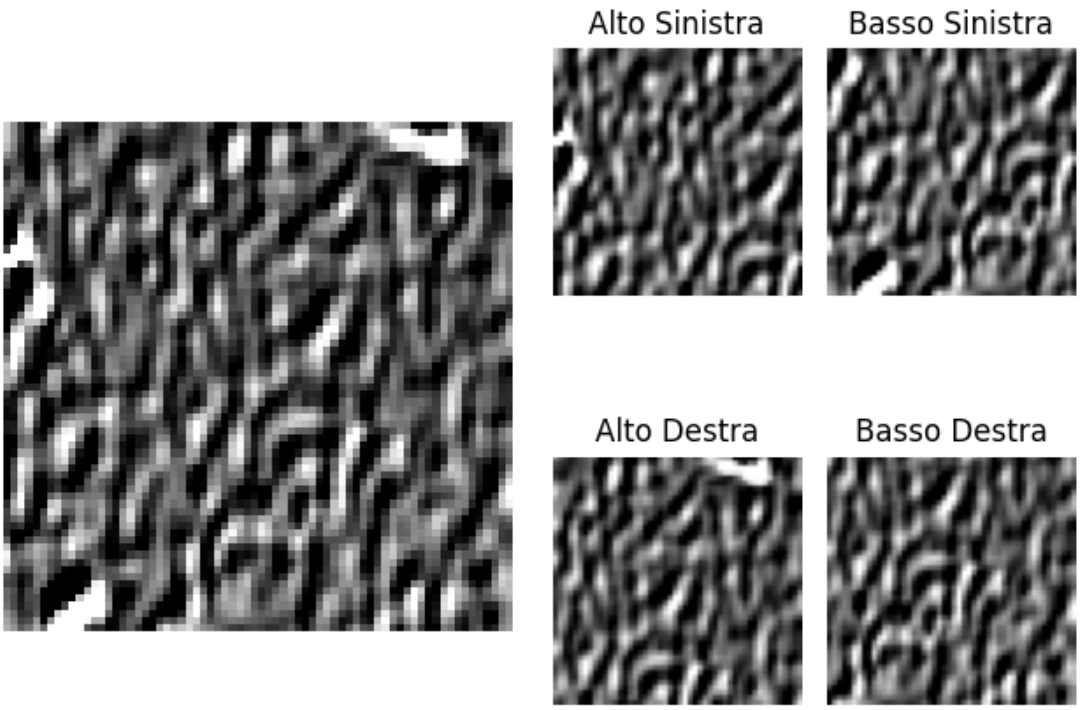

Figura 2.3: Esempio di sovrapposizione delle ROI

Appare evidente che essendo le zone di tessuto patoligico solo una piccola parte dell'immagine e utilizzando anche immagini totalmente sane, il numero di ROI sane sia molto più alto di quelle patologiche: per questo motivo le ROI patologiche vengono salvate anche rotate di 90° in senso orario e salvate nuovamente, in modo da aumentare la quantità di dati a disposizione. A questo punto è possibile estrarre le prime *features* calcolati sui valori dei pixel di ogni singola ROI, che sono:

- Media aritmetica

- Mediana
- 10 Percentile
- 90 Percentile
- Differenza Interquartile
- Media Quadratica
- Deviazione Standard
- Varianza
- Uniformità
- *Skewness*

Per le prossime *features* è invece necessario trasformare le ROI da bianco e nero a livello di grigi. La trasformazione in livelli di grigio significa assegnare ogni pixel non un valore di grigio ma un valore che rappresenti un intervallo di grigio, con gli intervalli che sono equispaziati. Il numero di intervalli è fissato a 4, e una rappresentazione grafica, che non corrisponde ad una vera rappresentazione matematica, è visibile nella figura 2.4.

Questa trasformazione è necessaria in quanto le prossime *features* vengono estratte da due matrici che assumono una rappresentazione quantizzata delle informazioni. Le due matrici in questione sono la *Gray Level Co-occurrence Matrix* (GLCM) e la *Gray Level Run Length Matrix* (GLRLM), che hanno dimostrato di essere molto efficaci in operazioni di *pattern analysis* [20]. La GLCM è una matrice quadrata di dimensione $N \times N$, dove N è il numero di livelli di grigio utilizzati, nel nostro

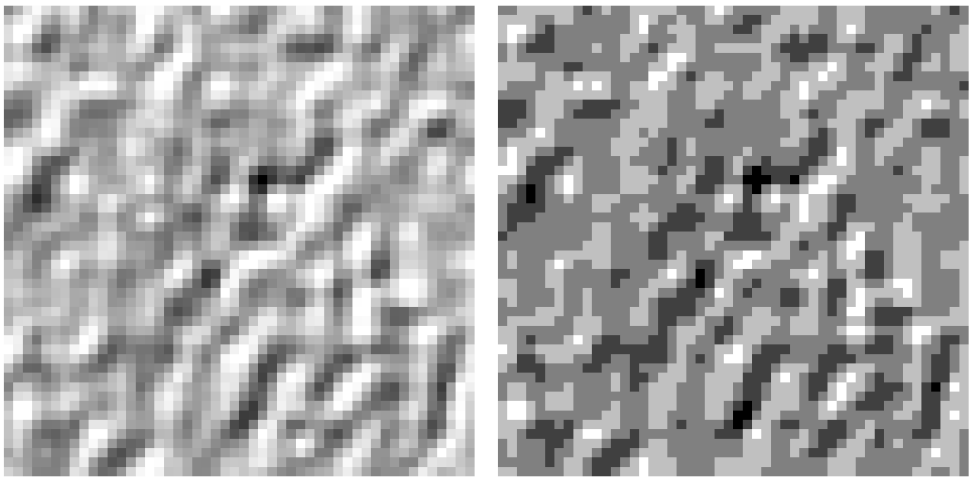

Figura 2.4: Esempio di ROI estratta per algoritmo di *shallow learning* (sinistra) e di versione digitalizzata su 5 livelli di grigio per il calcolo di GLCM e GLRLM (destra).

caso dunque 4×4. I due parametri da considerare per calcolarla sono la distanza d e l'angolo θ. L'elemento m_{ij} della matrice rappresenta quante volte un *pixel* con valore i è stato trovato a distanza d in direzione θ rispetto ad un *pixel* con valore j nella ROI. Nel nostro caso la distanza è fissata a $d = 1$ e l'angolo a $\theta = 0°$. Una rappresentazione del calcolo della GLCM è visibile nella figura 2.5.

Da questa matrice è possibile estrarre le seguenti *features*:

- *Contrasto*
- *Correlazione*
- *Energia*
- *Omogeneità*

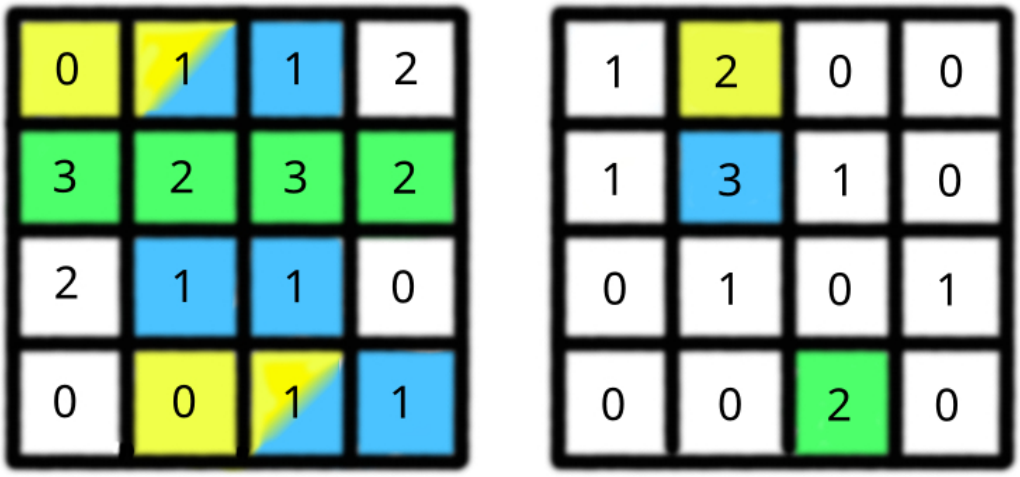

Figura 2.5: Una matrice 4×4 (sinistra) e la GLCM associata (destra), con $d = 1$ e $\theta = 0°$. I colori indicano il contributo di ogni pixel alla GLCM.

Analogamente viene calcolata la GLRLM, anch'essa una matrice di dimensione $N \times N$. A differenza della GLCM, la GLRLM ha come unico parametro l'angolo θ lungo cui percorrere una *run*. Si intende per *run* una sequenza di pixel in direzione θ con lo stesso valore. Dunque, l'elemento m_{ij} della matrice rappresenta il numero di *run* di lunghezza j con valore i nella ROI. Anche qui, un esempio di calcolo è visibile nella figura 2.6.

Le *features* estratte dalla GLRLM sono:

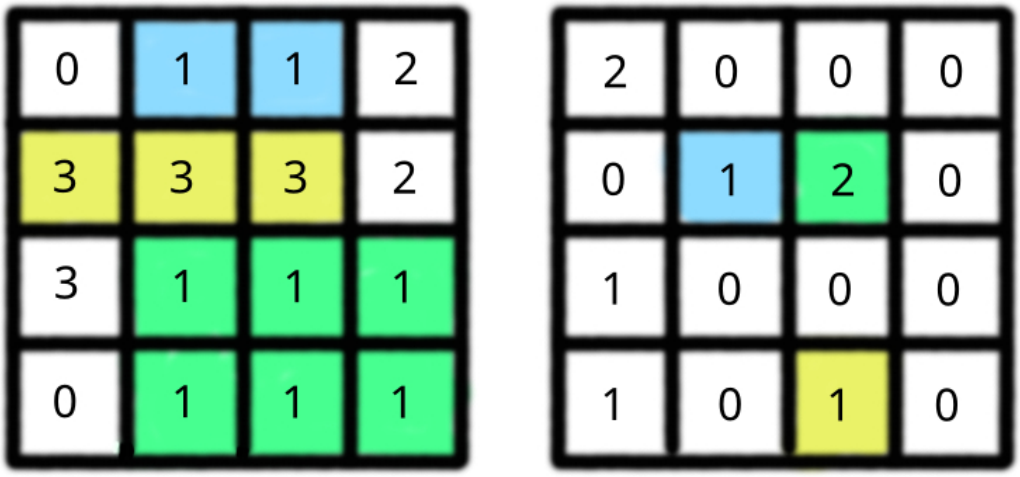

Figura 2.6: Una matrice 4×4 (sinistra) e la GLRLM associata (destra), con angolo $\theta = 0°$. I colori indicano il contributo di ogni pixel alla GLRLM.

- *Gray Level Non Uniformity* (GLN)
- *Gray Length Non Uniformity Normalized* (GLNN)
- *High-Gray Level Run Emphasis* (HGLRE)
- *Low-Gray Level Run Emphasis* (LGLRE)
- *Long Run Emphasis* (LRE)
- *Long Run High Gray Level Emphasis* (LRHGE)
- *Long Run Low Gray Level Emphasis* (LRLGE)

- *Mean Absolute Deviation* (MAD)
- *Run Length Non Uniformity* (RLN)
- *Run Length Non Uniformity Normalized* (RLNN)
- *Run Percentage* (RP)
- *Short Run Emphasis* (SRE)
- *Short Run High Gray Level Emphasis* (SRHGLE)
- *Short Run Low Gray Level Emphasis* (SRLGLE)

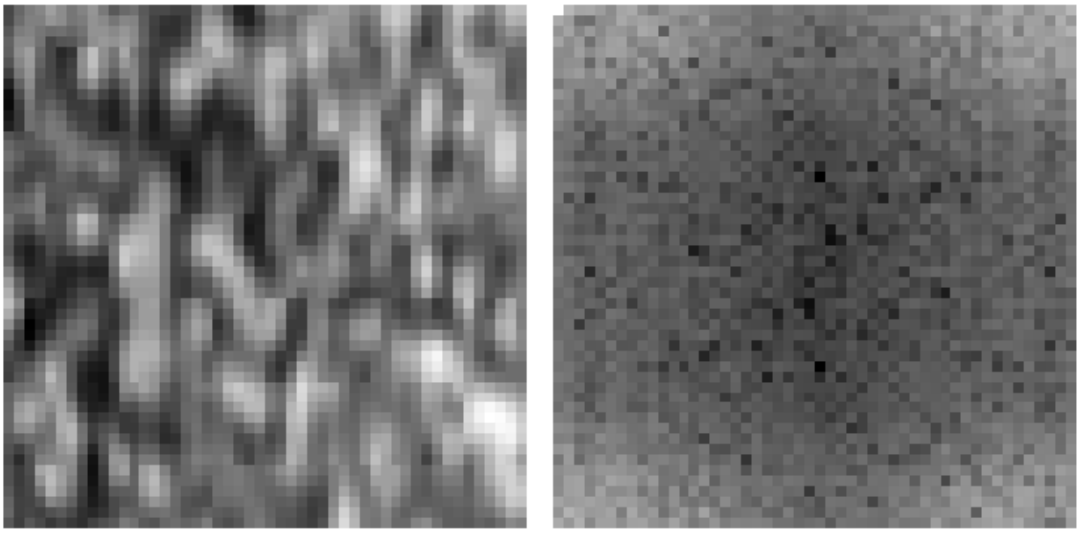

Figura 2.7: Esempio di una ROI (sinistra) e una sua visualizzazione in dominio della frequenza (destra) su scala logaritmica

Per le ultime *features* è necessario portare le ROI nel dominio della frequenza, utilizzando la *Discrete Fourier Transform* (DFT), calcolata con algoritmo *Fast Fourier Transform* (FFT) implementato dalla libreria `scipy` [6]. In questa fase viene utilizzata la versione originale della ROI, ovvero con valori tra 0 e 255. Tuttavia, per calcolare efficacemente i valori, è necessario riscalare i valori tra 0 e 1, rendendoli dunque valori decimali. Dalla ROI trasformata vengono prelevate le seguenti *features*:

- Picco di Frequenza
- Banda
- Frequenza di Kurtosis

Un sommario di tutte le *features* estratte è disponibile nella tabella 2.1. Le *features* estratte da ogni roi vengono dunque salvate in dei *file* di formato CSV, separatamente per *train* e *test* a seconda della ROI da cui sono estratte. La parte di validazione è estratta dal *train* al momento del *training* , e corrisponde al 10% dei dati. A questi *file* viene poi applicata la tecnica *min-max normalization* per normalizzare i dati tra valori compresi tra 0 e 1 e ridurre la variabilità dei dati. Un sommario delle immagini utilizzate e delle ROI/campioni estratti è riportato nella tabella 2.2.

Come detto in precedenza, l'algoritmo di *deep learning* non necessita di *pre-processing* delle immagini antecedente alla fase di training, e si riporta quindi solo la ripartizione delle immagini nella tabella 2.3.

Primo Ordine	**GLCM**	**GLRLM**	**DFT**
Media	Contrasto	GLN	Freq. Massima
Mediana	Correlazione	GLNN	Banda
10 Percentile	Energia	HGLRE	Freq. di Kurtosis
90 Percentile	Omogeneità	LGLRE	
Diff. Interquantile		LRE	
Media Quadratica		LRHGLE	
Dev. Standard		LRLGE	
Varianza		MAD	
Uniformità		RLN	
Skewness		RLNN	
		RP	
		SRE	
		SRHGLE	
		SRLGLE	

Tabella 2.1: Valori statistici estratti dalle ROI.

	Train	Test	Totale
Immagini Sane	20	5	25
Imagini Patologiche	40	9	49
ROI Sane	93.757	22.739	116.469
ROI Patologiche	52.840	11.602	64.442
Totale	60	13	73
	146'597	34.341	180.938

Tabella 2.2: Composizione del dataset Mauriziano per algoritmo di Shallow Learning

Tuttavia, delle 49 immagini patologiche alterate, di solo 37 si ha a disposizione la corrispondente immagine originale, e nel caso di algoritmi di *deep learning* è rischioso utilizzare immagini alterate per l'allenamento, e si utilizzano dunque meno immagini. Le immagini sono ripartite

in maniera ingenerosa verso lo *split* di *training* , ma questo è necessario altrimenti i risultati di precisione non sarebbero poi attendibili, anche perché questo *dataset* è impiegato nell'algoritmo di *deep-learning* solo per la fase di *fine-tuning* . In ultimo, tramite il *software* `ffmpeg` tutte le immagini alterate sono utilizzate per creare un video su cui poter testare l'applicativo. Non è corretto utilizzare immagini impiegate in fase di validazione o di allenamento per testare un modello, tuttavia non andando a valutare le prestazioni su tale video ma solo la funzionalità questo comportamento risulta accettabile. Il video è realizzato ad un *frame-rate* di 10 FPS, non per limiti del modello utilizzato ma per rendere apprezzabili le predizioni in quanto le immagini non vanno a comporre un vero video ma sono semplici immagini che vengono mostrate in sequenza.

Split	Immagini
Train	18
Test	10
Validation	9
Totale	37

Tabella 2.3: Divisione del dataset Mauriziano per algoritmo di Deep Learning

2.2 Il Dataset HyperKvasir

A differenza del *dataset* Mauriziano, il dataset HyperKvasir [2] viene fornito in maniera (quasi) pronta all'uso. Parliamone brevemente ed andiamo a descrivere le poche azioni necessarie.

Il *dataset* nasce per sopperire alla mancanza di dati medici per lo studio e l'addestramento di modelli di intelligenza artificiale. La raccolta di dati è, in ogni ambito, un lavoro oneroso in termini di tempo e risorse umane necessarie. Lavorando in ambito medico è dunque necessario di annotatori con un livello di specializzazione estremamente elevato, dunque di difficile reperibilità. Altri *dataset* , come *Common Object and Context* [15] (COCO) ed *ImageNet* [9] rappresentano invece contesti comuni e giornalieri, e quindi il loro principale costo deriva dalla dimensione del *dataset* stesso. Il personale medico dell'ospedale Maruiziano ha fatto inoltre notare che questo *dataset* è appena sufficiente, in quanto se per contesti comuni è possibile dare un'interpretazione oggettiva delle entità raffigurate (la segmentazione di un cane non dovrebbe creare particolari dibattiti), nell'ambito medico potrebbero esserci invece più opinioni su come e se annotare o meno una zona patologica, cosa di cui il *dataset* è sprovvisto.

HyperKvasir è dunque la più grande raccolta di immagini e video di esami di endoscopia digestiva, rilasciato proprio allo scopo di facilitare la ricerca in questo ambito. Va fatto notare però che questo *dataset* , nonostante le sue dimensioni, presenti comunque molti problemi e limitazioni. Delle circa 110'000 immagini disponibili infatti, 99'000 sono sprovviste di qualunque tipo di annotazioni, ovvero nè segmentate in alcun modo, e delle restanti 11'000 solo 1'000 sono immagini segmentate e dunque di nostro interesse. Infine, le classi segmentate non rappresentano metaplasie gastrointestinali ma bensì polipi, una patologia molto diversa da quella in esame qui. Un sommario della composizione del

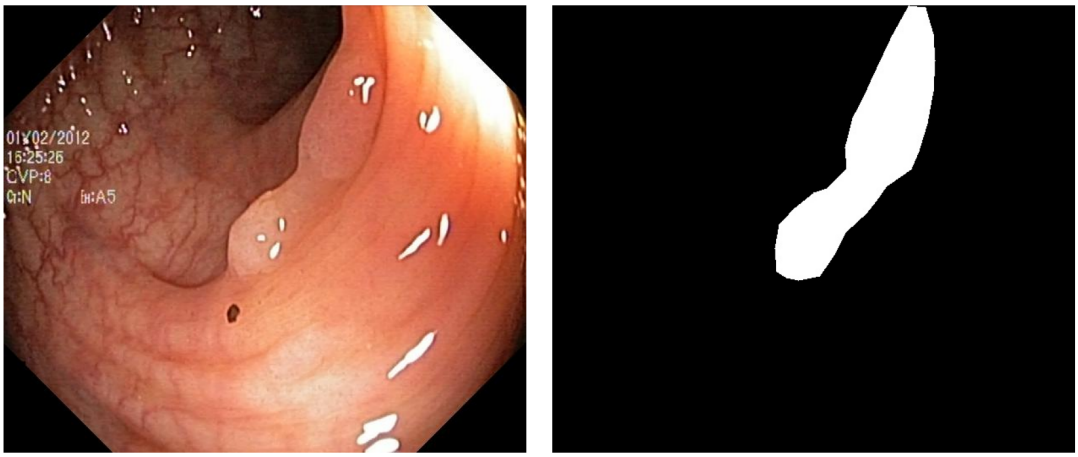

Figura 2.8: Esempio di immagine del dataset HyperKvasir e relativa maschera di segmentazione

dataset è riportato nella tabella 2.4.

Tipologia	# Campioni	# Classi	Dimensione (MB)
Classificate	10'662	23	3'960
Segmentate	1'000	1 (Polyp)	57
Non annotate	99'417	0	29'940
Video	374	30	32'539
Totale	111'453	—	66'496

Tabella 2.4: Composizione del dataset HyperKvasir

È ovviamente improprio l'utilizzo di un dataset che raffiguri entità diverse da quelle che si possono riscontrare nel contesto di applicazione del modello, tuttavia avendo a che fare con un numero di campioni estremamente limitato si è scelto di utilizzarlo in congiunzione con il

dataset Mauriziano. Questa tecnica è nota col nome di *transfer learning* [11], che consiste nell'usare come modello di partenza per un nuovo modello uno precedentemente allenato su un *dataset* anche molto diverso da quello che si utilizzerà in seguito. Ciò avviene solitamente per velocizzare la fase di *training* o ottenere risultati migliori, ma è anche molto utile nel caso in cui si disponga di pochi dati come nel nostro caso. È bene notare che l'impiego del *transfer learning* non è dunque un tentativo disperato o un'ultima risorsa ma bensì la norma, vista la pervasività che ha dimostrato nei vari ambiti di applicazione.

In ultimo, il dataset HyperKvasir è anche molto eterogeno nella forma in cui si presentano le immagini. A differenza del dataset in 2.1 infatti, dove le acquisizioni sono state effettuate tutte con la medesima macchina, nel caso di HyperKvasir infatti abbiamo con immagini provenienti da diverse fonti, di diversa risoluzione e nettamente inferiore a quelle del dataset Mauriziano e che non utilizzano molteplici filtri come nel caso di acquisizoni tramite ISCAN.

Essendo queste immagini impiegate solo nel modello di *deep learning*, esse non necessitano di alcuna trasformazione antecedente alla fase di training. Ciò che ha necessitato invece di piccole elaborazioni sono le maschere, che vengono fornite in formato JPEG e dunque non *lossless* . Questo causa ad esempio la presenza di valori diversi da 0 e da 1 nelle maschere, che corrispondono a classi che non esistono e la loro presenza è dovuta alla compressione. Una veloce quanto efficace modifica è stata dunque quella di modificare il valore dei *pixel* della maschera affinché ogni *pixel* con un valore maggiore o uguale a 1 venga impostato a 1, e

riesportarle in formato PNG.

Capitolo 3

Modelli

3.1 Lavori Precedenti

Con termine intelligenza artificiale, alternativamente anche *machine learning*, si indicano quelle tecniche che permettono di estrarre informazioni a partire dai dati, e che si diramano in una vastissima gamma di metodi di apprendimento e tipologie di problemi. Un primo distinguo che si puo' è quello tra *shallow learning* e *deep learning*. Seppure le differenze tra i due siano tante e non di poco conto, in entrambi i casi abbiamo un algoritmo che dato un input $\vec{x}$, la cui realizzazione concreta varia a seconda dell'approccio utilizzato come già visto nel capitolo 2, e ne calcolano un output $\vec{y} = f(\vec{x})$, che corrisponde alla predizione dell'algoritmo. Anche per $\vec{y}$ la realizzazione concreta puo' differire molto: nel nostro specifico caso per esempio quel che vogliamo ottenere è una maschera che indichi quali *pixel* dell'immagine contengono l'area considerata patologica, e quindi rappresenta in realtà una matrice di dimensioni pari a quelle dell'immagine in input. Generalmente parlando il valore di ogni singolo elemento di tale matrice avrà un valore compreso tra 0 ed N_c, dove N_c è il numero di classi del nostro problema,

nel nostro caso 2.

Figura 3.1: Esempio di *Semantic Segmentation* eseguita su una foto di tre cani [10].

Il nostro problema di riferimento è noto col nome di *semantic segmentation* [7], di cui un esempio è visibile nella figura 3.1, che è un caso specifico della *image classification*. Nella *image classification* infatti quel che si fa è dedurre cosa l'immagine in input rappresenti, senza specificare quale parte dell'immagine contenga la classe rilevata. Nella *semantic segmentation* invece si vuole rilevare quale zona dell'immagine contenga le classi rilevate. Qui di seguito dunque andiamo a proporre due approcci, dove il primo riprende un lavoro basato su un algoritmo di *shallow learning* [16] ed il secondo invece è basato su un approccio di *deep learning* [13], e per entrambi i modelli si prova a valutare non solo le loro prestazioni in termini di precisione, in parte già assodati,

ma anche la loro possibile applicazione in un contesto reale.

3.2 Modello Shallow Learning

Il nostro primo modello di riferimento [16] è basato su alberi decisionali, modelli molto popolari nel mondo del *machine learning* per vari motivi, uno tra i tanti la capacità di poter interpretare il loro funzionamento. Se infatti con modelli *deep learning* è possibile solo parzialmente capire il processo di inferenza, gli alberi decisionali sono per costruzione molto vicini al modo di ragioanre umano, e quindi più facilmente interpretabili. [14] Modelli di questo tipo prendono anche il nome di modelli *white box*, mentre modelli non interpretabili prendono il nome di modelli *black box*. Questa affermazione ha tuttavia una validità relativa, in quanto nel momento in cui i il significato delle componenti del valore in *input* salgono di numero o complessità in termini di cosa rappresentino comprendere tale processo diventa molto più complicato.

Il lavoro in questione non ha però semplicemente utilizzato degli decisionali, bensì un *ensamble* di alberi decisionali, che prende il nome di *random forest*, ossia un insieme di alberi decisionali. Questa tecnica viene impiegata spesso quando, per la complessità del problema o per la scarsità dei dati, non è possibile costruire un singolo albero decisionale sufficientemente preciso. Si utilizzano dunque più alberi decisionali in modo combinato, in un modo che prevede che le decisioni prese dal singolo albero siano molto carenti, ma che la decisione presa in maniera collettiva da tutti gli alberi risulti nettamente superiore. Gli alberi singoli facenti parte di una *random forest* sono anche detti *weak*

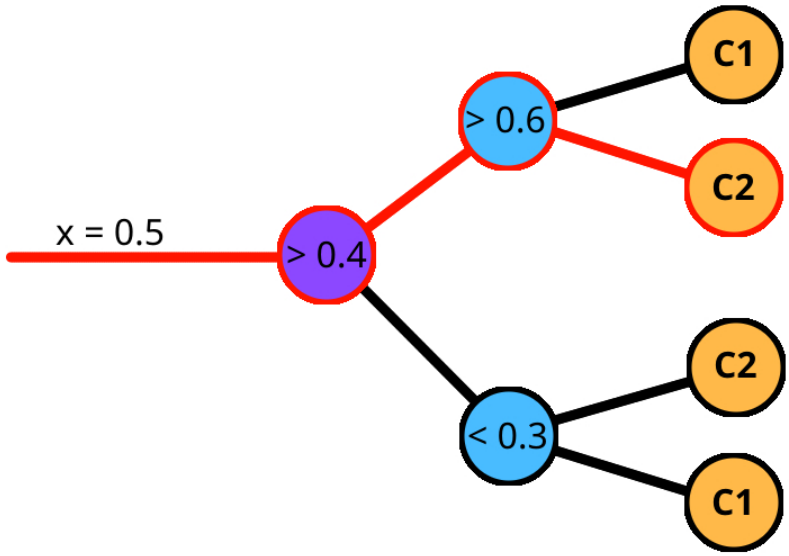

Figura 3.2: Esempio semplificato del processo di inferenza di un albero decisionale

learners.

La costruzione di un *ensamble* di *weak learners* prevede di decidere come tale *ensamble vada costruito*. Le foreste casuali sono solitamente costruite tramite *bagging*, dove ogni albero viene costruito su un sottoinsieme scelto casualmente dai dati di training. Questo consentirebbe anche il *training* parallelo degli alberi. Nel nostro caso però si utilizza una tecnica detta *Histogram Boosted*. L'implementazione impiegata è quella offerta dalla libreria `sciki-learn` [19]. Il *boosting* prevede che l'i-esimo albero venga costruito utilizzando come base l'$i-1$-esimo albero, e avviene quindi in maniera sequenziale. La differenza tra le due tecniche riside nel modo in cui viene eseguito lo *split* dei dati. Si intende per *split* il modo in cui si sceglie la soglia che fa da separatore tra

due rami di un albero. Questo viene solitamente calcolato in modo da massimizzare la suddisivione delle classi nei rami. Utilizzando una tecnica *gradient based* tuttavia, quel che si fa a fare è invece minimizzare l'errore di classificazione dovuto alla suddivisione, calcolato mediante una funzione di costo. La tecnica è inoltre detta *Histogram Boosted* [8] in quanto il criterio di suddivisione viene calcolato non sui valori del nodo ma sull'istogramma di tali valori. Questo accade in quanto all'aumentare del numero di valori univoci della *feature* sotto esame e delle *features* stesse, il calcolo della soglia puo' risultare oneroso. Decidendo la soglia in base all'istogramma invece si puo' scegliere una soglia che riduca il fattore di perdita in maniera più efficiente.

Ottenute le predizioni di tutti gli alberi della foresta, per ottenere la predizione finale si utilizza uno schema di voto, che si occupa di unire le predizioni di tutti gli alberi in un'unica predizione finale. I meccanismi per fare ciò sono molti e variano a seconda del problema, ma in questo caso è stato utilizzato il *majority vote*, ovvero una ROI viene classificata come patologica se la maggioranza degli alberi l'ha classificata come tale.

La procedura del *majority voting* viene anche re-iterata sui singoli *pixel*. Essendo infatti le ROI estratte con una sovrapposizione spaziale del 40%, potremmo avere singoli *pixel* con classificazioni discordanti derivanti da ROI diverse ma sovrapposte. Per calcolare la maschera finale dunque si utilizzano due matrici: nella prima il singolo elemento indica quante volte il corrispondente *pixel* è stato sottoposto a valutazione,

mentre nella seconda si conteggia il numero di volte che è stato classificato come patologico. Dal momento che alcune ROI non vengono analizzate, come quelle di sfondo per esempio, si potrebbero avere *pixel* con un conteggio di valutazioni pari a 0. Per evitare errori numerici, a questi *pixel* viene assegnato un numero di valutazioni pari a 1. Dalla divisione elemento per elemento si ottiene la matrice delle frazioni in cui il *pixel* è stato classificato come patologico, e vengono confermati come tali quelli che hanno una frazione maggiore di 0.5.

Il modello, valutando le singole ROI, è carente nella valutazione d'insieme dell'immagine, e per questo motivo le maschere vengono sottoposte a *post-processing*. Nello specifico alla maschera binaria finale vengono applicati i seguenti operatori morfologici:

- Erosione $\times$ 10
- Dilatazione $\times$ 20
- Chiusura $\times$ 10
- Erosione $\times$ 10

Il primo operatore serve a rimuovere quel che potremmo chiamare rumore di classificazione, ovvero singoli *pixel* o micro-aree di *pixel* classificati come patologici ma che non compongono una parte signifiativa della maschera. Il secondo e il terzo operatore vanno invece a riempire eventuali buchi presenti nella maschera, mentre il quarto annulla l'effetto della prima erosione solo sulle maschere di dimensione abbastanza estesa.

3.3 BiSeNet

Discutiamo ora del secondo approccio considerato, ispirato a un lavoro nello stesso ambito [13] e basato su reti neurali. A differenza del modello di *shallow learning* tali modelli sono molto poco interpretabili ma tuttavia essi sono generalmente anche molto più performanti, cosa che li rende molto appetibili in contesti dove un'alta accuratezza è richiesta.

Un generico modello di *semantic segmentation* prevede varie componenti e fasi, che prendono l'*input* originale e vanno contemporaneamnete a estrarre e comprimere le informazioni in esso contenute. Qui si notano le prime due differenze molto importanti rispetto al precedente modello, ovvero tali modelli riescono ad elaborare le immagini nella loro interezza, senza dover valutare e poi combinare sotto-aree della stessa, e sono in grado di lavorare con immagini a colori. Va fatto notare che sarebbe tecnicamente possibile adattare il modello di *shallow learning* per accettare immagini a colori, ma questo significherebbe applicare il metodo descritto separatamente per ogni canale di colore, che richiederebbe un ulteriore adattamento.

L'approccio scelto è basato sull'architettura *Bilateral Segmentation Network* (BiSeNet) [23], nello specifico BiSeNet v2 [22], il cui scopo è combinare le informazioni spaziali con quelle contetuali contenute nell'immagine al fine di ottenere risultati migliori. Uno schematico è visibile nella figura 3.3. I due principali strumenti impiegati dalle reti neurali per l'estrazione delle informazioni sono infatti i *convolutional layers* e i *pooling layers*. I primi sono in grado di estrarre informazioni locali e sono generalmente lenti da calcolare, mentre i secondi sono

in grado di estrarre informazioni globali e sono molto più veloci. Per un'accurata segmentazione è necessario avere entrambe le tipologie di informazioni e all'aumentare del numero di livelli si riesce generalmente ad ottenere risultati migliori, ma questo porterebbe anche ad un aumento del tempo di inferenza che deve rimanere contenuto. Per questo motivo, come suggerito dal nome, BiSeNet impiega due rami di estrazione paralleli, uno denominato *Detail Branch* (DB) e l'altro *Semantic Branch* (SB). Inoltre, invece di utilizzare filtri convoluzionali classici, BiSeNet sostituisce i *pooling layers* con filtri convoluzionali *depth-wise* (DWConv) [24], più veloci da calcolare. I due risultati vengono infine combinati tramite un *Bilateral Aggregation Layer*.

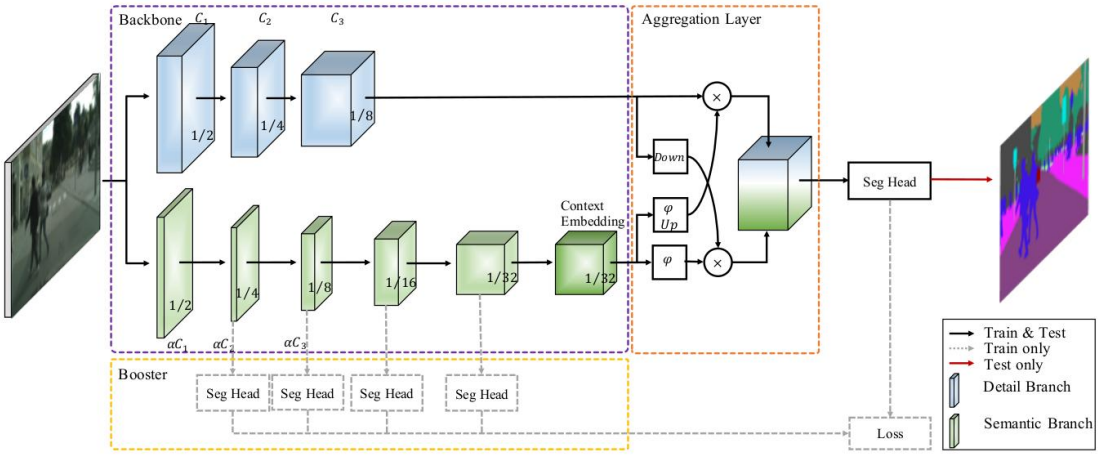

Figura 3.3: Architettura generale di BiSeNet v2. Fonte: Paper [22]

Il DB dunque si compone di tre livelli, ognuno dei quali applica due *set* di filtri convoluzionali, il primo con *stride* 2 e il secondo con *stride* 1, seguito da *batch normalization* e da una funzione di attivazione *ReLU*. Si intende per *stride* la dimensione del passo che si fa nel calcolo

della convoluzione. L'impiego di uno *stride* pari a 2 con filtri di dimensione 3×3 permette di dimezzare la dimensione dell'*input*. La *batch normalization* consiste nel normalizzare le *feature maps* lungo la dimensione di profondità, in modo da avere una media nulla e una deviazione standard unitaria. La funzione di attivazione $ReLU$ è una funzione lineare che azzera i valori negativi e li lascia inalterati per quelli positivi. Applicando questo compinente tre volte, si ottengono *feature maps* di dimensione pari a 1/8 dell'*input* originale.

Il SB invece è composto da quattro livelli, uno denominato *STEM* e tre detti *gather-and-expand* (GE). Il primo riduce di un fattore 4 la dimensione dell'*input* originale, mentre i successivi di un fattore 2, pertanto le *feature maps* ottenute sono di dimensione pari a 1/32 dell'*input* originale. Lo *STEM* applica un primo livello convoluzionale con *stride* 2, per poi diramarsi e applicare allo stesso altri due livelli convoluzionali che dimezzano a loro volta questo input, facendono infine la somma, da cui la riduzione di un fattore 4. Il GE è composto a sua volta da due livelli: il primo funziona in modo simile ad un blocco residuale, ovvero applica dei filtri convoluzionali e poi somma l'*input* originale alla sua uscita, lasciando dunque inalterata la dimensione delle *feature maps* ottenute. Il secondo livello opera in modo simile ma utilizzando nei punti intermedi dei filtri con *stride* 2, dunque dimezzando la dimensione delle *feature maps* ottenute. I filtri convoluzionali usati dal blocco GE sono del tipo DWConv.

L'AL, visibile in figurea 3.4, fonde le informazioni ottenute dal DB e dal SB. Quest'operazione avviene due volte, nuovamente in due rami

che questa volta vanno a inrociarsi. Una visu Il DB viene infatti nuovamente sottoposto a due livelli convoluzionali con *stride* 2, riducendo la dimensione delle *feature maps* a una compatibile con quelle del SB, mentre il SB viene sottoposto ad *upsampling* bilineare, che quadruplicando le dimensioni diventa compatibile con le feature maps del DB. I due risultati vengono quindi moltiplicati elmento per elemento e si ottengono due *set* di *feature maps* che vengono sommate. Visto che uno dei due *set* di *feature maps* è comunque di dimensione 1/4 rispetto all'altro, si adatta il minore tramite proiezione lineare.

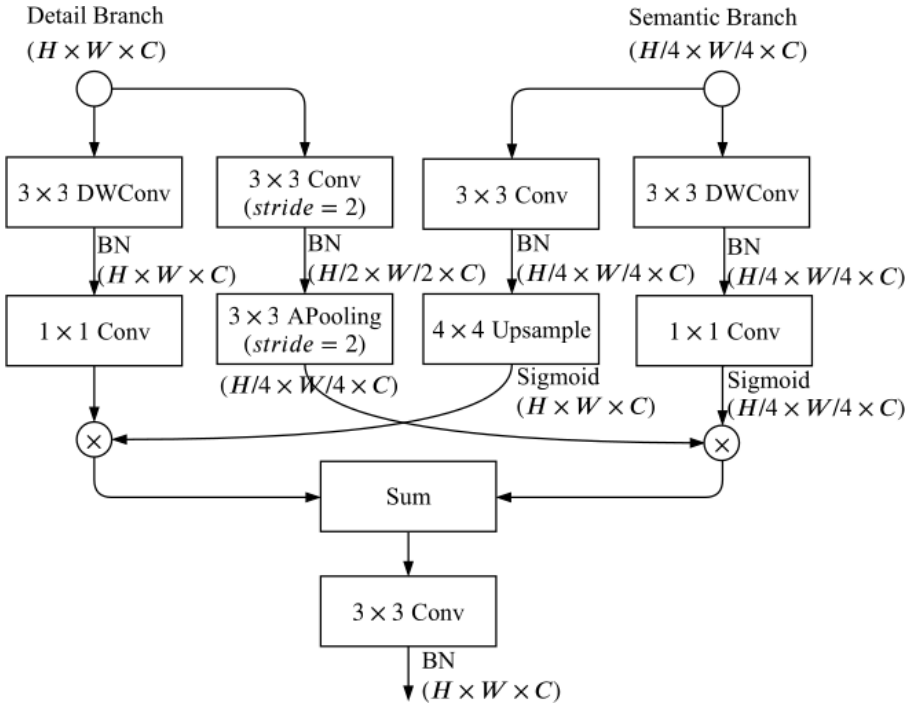

Figura 3.4: Schematico del *Bilateral Aggregation Layer* (AL). Fonte: Paper [22]

Il risultato dell'AL viene dunque passato alla testa di segmentazione, che applica dei filtri convoluzionali per modificare le dimensioni ed il numero delle *feature maps* ottenute. In particolare, avendo N *feature*

maps di dimensione $H \times W$, si ottengono C *feature maps* di dimensione $sH \times sW$. Il fattore s è tale che $sH \times sW$ sia la dimensione originale dell'immagine in input, mentre C è il numero di classi del problema. Trattandosi di un problema di segmentazione binaria è possibile utilizzare un valore di C pari a 2. Tuttavia un valore di C pari a 1 è anche possibile ed è l'opzione scelta in questo caso. Il termine di perdita è calcolato sui singoli *pixel* tramite la funzione sigmoide, e valori di predizione maggiori o uguali a 0.5 vengono considerati come appartenenti alla classe 1, ovvero quella patologica.

3.4 Training e Risultati

3.4.1 Training

Il modello di *shallow learning* descritto nella sezione 3.2 viene allenato con tecnica *K-Fold Cross-Validation*, che consiste nella divisone del dataset in K parti di uguale dimensione. Si esegue quindi K volte l'allenamento del modello, usando $K - 1$ parti del dataset, detti *fold*, per l'allenamento ed il restante per il *testing*. Il 10% dei dati di *training* viene utilizzato come *validation set*. Vi sono vari parametri che possono essere modificati. Il primo è il *learning rate*, che regola quanto modificare le soglie di divisone delle foglie ad ogni nuovo albero. Segue poi il numero massimo di alberi da impiegare nella costruzione della foresta, probabilmente il fattore più determinante del modello. Il training viene svolto con *early stopping*, ovvero si ferma l'allenamento quando il modello non migliora le prestazioni dopo aver costruito un certo numero di alberi. Altro parametro legato agli alberi è la profondità massima

che possono raggiungere, anch'essa sottoposta a *early stopping*. Infine è possibile modificare il fattore di *L2-regularization*, che serve per ridurre l'*overfitting* del modello.

La tecnica di *training* prevede un meta-parametro che è il fattore di *over/under fitting*. Esso va a regolare la quantità di campioni sani del dataset da utilizzare in fase di *training*. Valori minori di 1 indicano che alcuni campioni sani non verranno utilizzati, mentre valori maggiori di 1 indica che alcuni campioni verranno ripetuti più volte.

Nella ricerca dei parametri ottimali è stata impiegata la strategia di *grid-search*, ovvero provando tutte le possibili combinazioni di parametri tra quelle definite, che sono visibili nella tabella 3.1. Il training viene svolto su CPU.

L'ottimizzazione del modello di *deep learning* di cui in 3.3 è invece basata su *Stochastic Gradient Descent* (SGD) con apprendimento adattivo. La formulazione dell'aggiornamento dei pesi dopo un'iterazione del modello è data da:

$$w(t) = \beta w(t-1) + (1-\beta)\eta(t)(\frac{\partial L}{\partial w} - \lambda w(t-1)) \tag{3.1}$$

dove termine $w(t)$ indica i pesi del modello alla t-esima iterazione. Il termine β è il parametro di *momentum* che è impostato a 0.9, mentre $\eta(t)$ è il *learning-rate*, che è una funzione del tempo in quanto esso ha un valore fissato a 0.5 con *warm-up* lineare per le prime 1000 iterazioni e successivamente ha un'andamento polinomaile con potenza 0.9, fino a un minimo di 10×10^{-4}. L'allenamento è compiuto su una singola GPU con un numero di campioni per GPU pari a 8, che da una *batch-size* pari

Parametro	Valori
Learning Rate	0.01 0.05 0.1 0.5
Numero Massimo Alberi	75 100 150 200 250
Profondità Massima	50 100 150
Fattore Regolarizzazione	0.1 0.5 1.0
Fattore di Sampling	0.9 1.0 1.1
Totale Possibili Combinazioni	540

Tabella 3.1: Parametri utilizzati nella grid-search

a 8. Il termine $\frac{\partial L}{\partial w}$ è il termine di perdita calcolato sull'intero *batch* di dati, che viene calcolato mediante la funzione di perdita *Cross-Entropy* (CE) con l'utilizzo dell'operatore sigmoide. Si intende per iterazione uno *step* di dimensione pari a *batch-size* numero di campioni. Il training in questa fase viene svolto sul dataset HyperKvasir descritto in 2.2 per 20.000 iterazioni, usando come dataset di validazione quello ribortato in tabella 2.3, eseguito ad intervalli regolari di 2.000 iterazioni in cui si va a valutare la precisione in termini di *Intersection over Union*

(IoU) e la precisione in termini percentuali, salvando di volta in volta un *checkpoint* del modello. Dopo le iniziali 40′000 iterazioni, viene selezionato il modello con la accuracy maggiore e viene sottoposto ad una fase di *fine-tuning* sulle immagini di training di cui alla tabella 2.3. Si sceglie come metrica discrimante la precisione e non l'IoU perché essendo le due classi diverse è più ragionevole l'utilizzo della prima. Nella fase di *fine-tuning* si utilizza un learning-rate costante pari a 0.001, il numero di iterazioni è pari a 2.000 e se ne salva un *checkpoint* ogni 200 iterazioni. Alla fine di questa fase viene prelevato nuovamente il modello con la migliore accuracy sempre valutata sulla porzione di test della tabella 2.3. In entrambe le fasi vengono applicate all'immagine in ingresso le trasformazioni, volte ad aumentare la quantità e l variabilità dei dati descritte nella tabella 3.2.

Trasformazione	Probabilità	Valore	Split
Resize	1.0	800×600 px	Test Train
Rotazione	0.5	180°	Train
Normalizzazione	1.0	$\mu = [123, 116, 103]$ $\sigma = [58, 57, 57]$	Test Train
Luminosità Casuale	0.5	–	Train
Contrasto Casuale	0.5	–	Train
Saturazione Casuale	0.5	–	Train

Tabella 3.2: Trasformazioni applicate alle immagini per algoritmo di *deep learning*

3.4.2 Risultati

In questa fase riportiamo i risultati numerici dei modelli. Per il modello di *shallow-learning* si riportano i risultati dei 5 modelli che hanno ottenuto la migliore prestazione secondo l'F1-Score per la sola classe patologica. F1 è una metrica che cerca di bilanciare i risultati di precisione e di richiamo. Se da un lato infatti è desiderabile un modello ad alto richiamo, ovvero che sia in grado di individuare tutte le aree potenzialmente patologiche presenti nell'immagine, dall'altro è desiderabile che questo obiettivo non venga raggiunto classificando con eccessiva leggerezza aree non patologiche come tali. I risultati sono riportati in tabella 3.3. Il tempo di esecuzione medio per tutti i modelli è di circa 10 secondi, che dipende per lo più dalla fase di estrazione delle componenti di GLCM e GLRLM e non è intaccato significativamente da altri fattori come la profondià o il numero di alberi impiegati nella foresta.

Parametri					**Prestazioni**		
SR	**LR**	**MD**	**MT**	**REG**	**F1**	**PREC**	**REC**
1.1	0.1	50	250	1.0	**63.46**	68.35	59.22
1.1	0.1	50	200	0.1	63.35	67.47	**59.69**
1.1	0.1	50	250	0.1	63.27	68.24	58.98
1.0	0.1	100	250	1.0	63.27	68.09	59.08
1.1	0.1	100	200	0.1	63.18	**70.12**	57.48

Tabella 3.3: Milgiori risultati del modello di shallow-learning.
SR = Sampling Rate, LR = Learning Rate, MD = Massima Profondità Alberi, MT = Massimo Numero di Alberi, REG = Fattore di Regolarizzazione. I risultati fanno riferimento alla sola classe patologica.

Dai risultati si nota come l'idea di adottare l'*over-sampling* abbia effettivamnte portato a risultati migliori per il modello e che la scelta del *learning rate* non sembra essere di particolare rilievo, probababilmente per la presenza del fattore di regolarizzazione. Focalizzandoci sulle metriche di interesse invece notiamo come i risultati non siano esattamente brillanti, in con valori di F1 che si aggirano attorno ai 63 punti percentuali. Questo numero non si discosta eccessivamente dai valori di precisone e di richiamo, lasciando indicare una reale difficoltà del modello nel distinguere tra le due classi. Un esempio di queste prestazioni altalenanti è visibile nell'immagine 3.5.

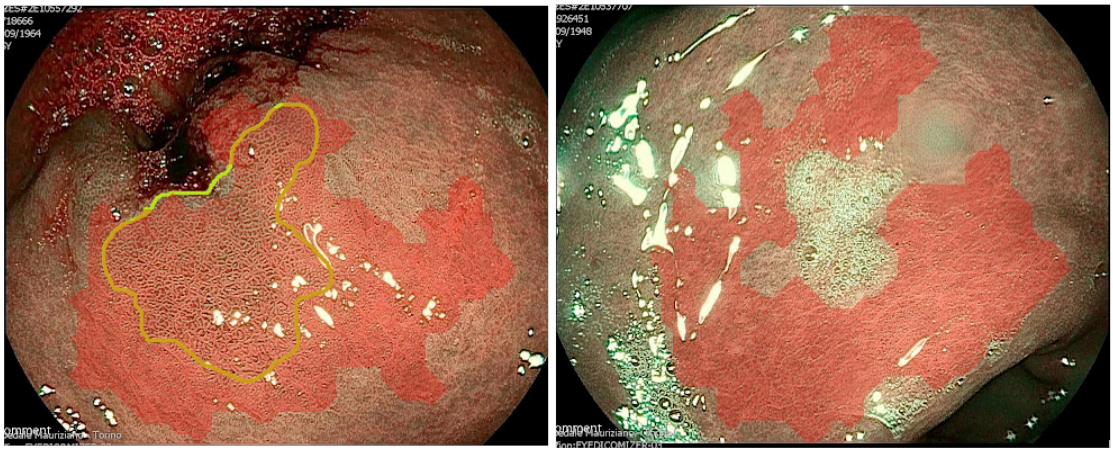

Figura 3.5: Esempi di errori grossolani nel modello di *shallow learning.*
Sinistra: immagine patologica, eccesiva regione evidenziata;
Destra: misclassificazione catastrofica

Queste prestazioni congiuntamente al lungo tempo di elaborazione, circa 10 secondi per un fotogramma, lo rendono difficilmente utilizzabile in un contesto reale. Tuttavia la lentezza del modello risiede principalmente nella necessità di calcolare le matrici GLCM e GLRLM per ogni

ROI. Per calcolare queste componenti è stata impiegata la libreria Pyradiomcs [12], che al contrario di altre librerie non è ottimizzata per eseguire i calcoli in maniera efficiente e parallela. Un miglioramento di queste API e un futuro sviluppo sulle prestazioni potrebbe rendere appetibile questo tipo di modello.

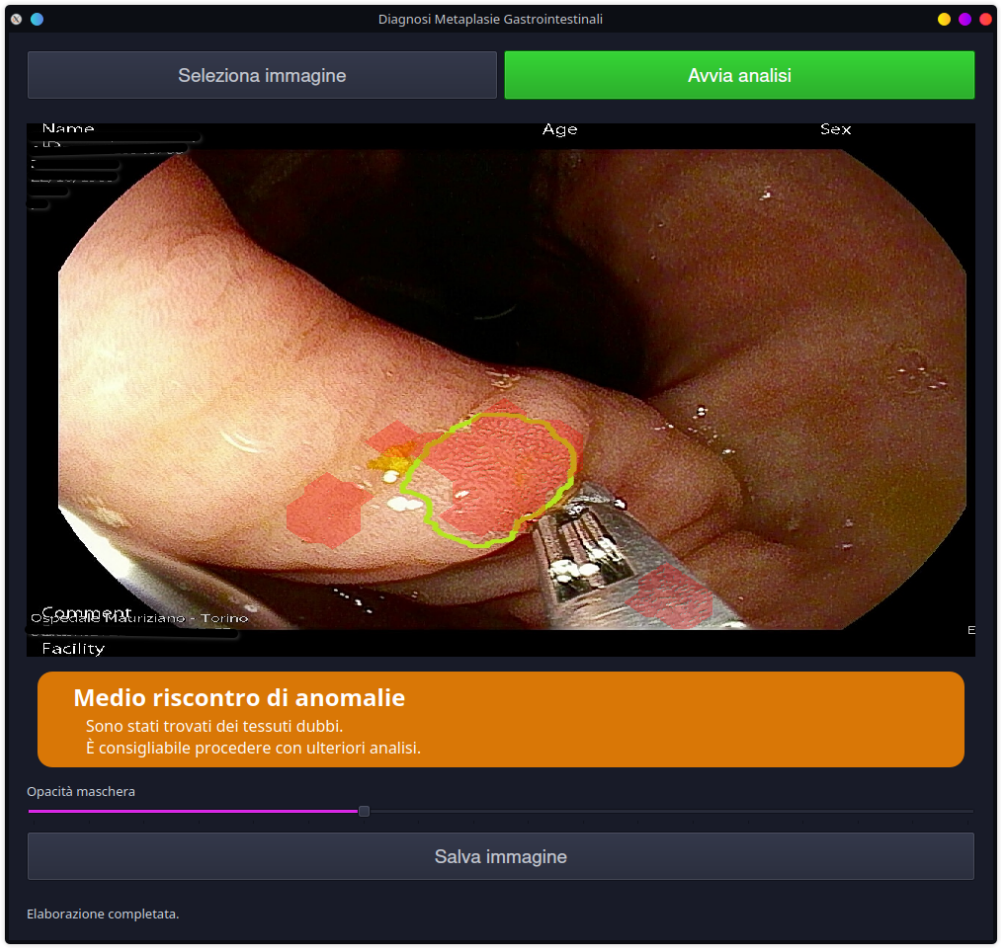

Figura 3.6: Interfaccia grafica per la classificazione delle ROI. La zona cerchiata in verde indica l'area che desidereremmo rilevare, le zone a sovrapposizione rossa indicano le zone rilevate come anomale.

Grazie a un confronto con il personale medico dell'Ospedale Mauriziano ci sono stati dei suggerimenti in merito a quanto discusso fin'ora. Tralasciando le considerazioni sulle prestazioni, sicuramente troppo basse per un impiego reale, si è discusso di come l'analisi in separata sede di campioni raccolti durante l'endoscopia non risulti di particolare utilità al personale medico. È stato inoltre fatto notare come le *feature* colore siano in realtà di grande importanza in quanto alcune deformazioni sono identificabili anche mediante esse. Un'altro esempio di come le feature colore possano essere d'aiuto è visibile nell'immagine 3.6, dove un corpo esterno allo stomaco dal colore atipico viene comunque identificato come tessuto patologico.

Per la valutazione dei risultati del modello learning descritto in 3.3, considerando la situazione più precaria in termini dei dati a disposizione, valutiamo i risultati dopo entrambi i passaggi di training, prendendo come riferimento tre *checkpoint* diversi. I risultati sono riportati in tabella 2.3 e riportano i valori medi non pesati di IoU e precisione valutati sullo split di validazione, con l'ultima che è stata scelta essere la metrica discriminante per selezionare il modello ottimale. Questo metodo di selezione è stato scelto poiché considerando le problematiche legate alla scarsa quantità di dati portano inevitabilmente a una situazione in cui la variabilità nei risultati di allenamento e nella valutazione dei *checkpoint* è estremamente elevata.

Commentiamo i risultati ottenuti prima e dopo la fase di *fine-tuning*. La parte superiore della tabella 3.4 confermano alcune nostre supposizioni iniziali. Nello specifico notiamo come il modello abbia in qualche

Checkpoint	Iterazione	mIoU (%)	mAccuracy (%)
Prima del fine-tuning			
C1	14'000	41.64	**69.11**
C2	18'000	**43.67**	66.91
C3	2'000	43.47	67.51
Dopo fine-tuning			
C1	400	50.99	71.99
C2	1'200	**54.85**	**72.70**
C3	1'400	44.83	71.66

Risultati su Test Set		
Checkpoint	mIoU (%)	mAccuracy (%)
C1	51.89	70.38
C2	**58.83**	**74.06**
C3	49.17	68.99

Tabella 3.4: Risultati del modello di deep-learning.
mIoU = media aritmetica delle IoU per classe, mAccuracy = accuratezza media aritmetica per classe

modo imparato a discriminare tra le due classi in termini di accuratezza, raggiungendo valori paragonabili a quelli del modello di *shallow learning*. Potremmo considerare questo come una conferma della sua validità non avendo in realtà avuto accesso al *dataset* che modella il dominio di applicazione. Tuttavia, la differenza in termini di forma con cui le anomalie si presentano è molto evidente in termini di IoU, che si assesta per i tre *checkpoint* al di sotto del 50%. Una soglia di segmentazione accettabile sarebbe una almeno del 60%. Dopo il *fine-tuning* si osserva un miglioramento notevole in termini di IoU, che riesce a raggiungere nel caso del modello C2 un valore prossimo a tale soglia. Il processo ha anche giovato alla precisione del modello che si assesta

attorno al 70%, un valore anche in questo caso non impressionante e prossimo a quelli visti per il modello di *shallow learning*. Considerate le condizioni totalmente avverse in termini di dati a disposizione per l'allenamento di un modello di questo tipo, potremmo considerare questi risultati come accettabili, ma va considerata l'elevata variabilità che si puo' incontrare in essi visto la ridotta dimensione del *test set*. Il miglioramento più sostanziale si ha nella velocità di inferenza, che raggiunge i 9.8 FPS su una singola GPU NVIDIA Tesla T4 Tensor. Alcuni esempi sono visibili nella figura 3.7

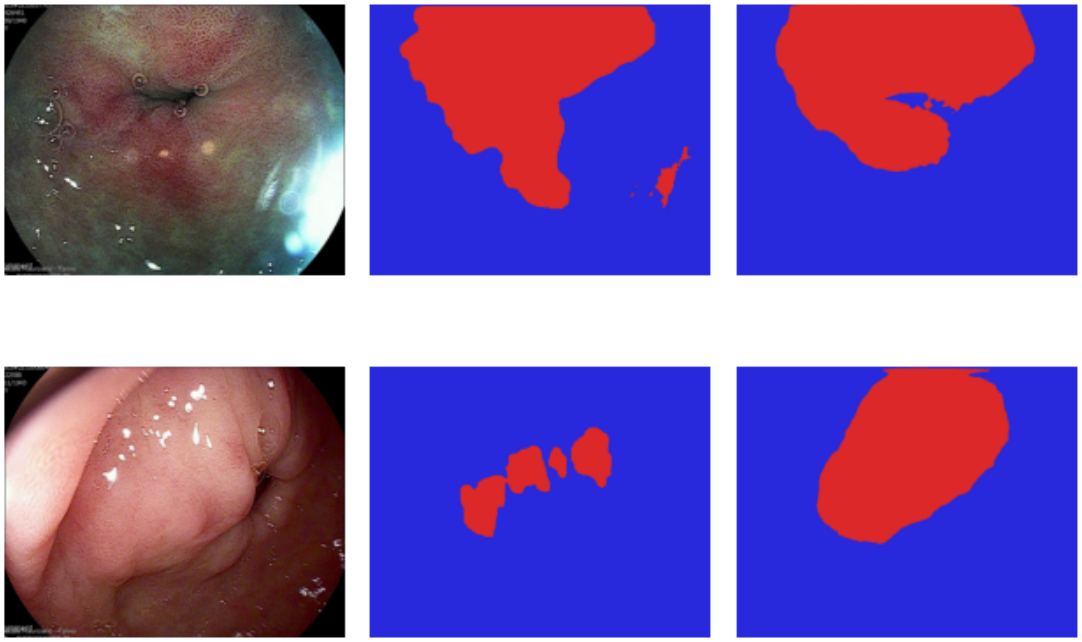

Figura 3.7: Predizioni del modello C2 dopo il *fine-tuning*.
Da sinistra a destra: immagine originale, maschera di segmentazione, predizione del modello C2.

Capitolo 4

Applicativo

4.1 Struttura

L'interfaccia è stata realizzata utilizzando il linguaggio di programmazione Python e la libreria PyQt5. PyQt si occupa di fare da *wrapper* alla libreria Qt, ovvero ne espone le funzionalità in modo da poterle utilizzare in Python. Qt è un *framework* multi-piattaforma per lo sviluppo di applicazioni grafiche, scritto in C++, che mette a disposizione una serie di classi e funzioni per la creazione di interfaccie grafiche. Questo permette di impiegare lo stesso codice per l'interfaccia su diversi sistemi operativi.

Al fine di testare l'applicativo sarebbe necessario avere a disposizione un dispositivo ISCAN da poter collegare al computer e vedere il comportamento dell'applicativo con la sorgente video catturata. Questo è ovviamente impratico per vari motivi, dal dover impegnare lo strumento per lunghi periodi di tempo alla necessità di avere un paziente che si stia sottoponendo ad un esame in modo da avere un responso. Quindi, per fini puramente dimostrativi, utilizzando il *framework* multimediale `ffmpeg` [21] è stato realizzato un video che simula la sorgente

video dell'ISCAN. Questo avviene concatenando le immagini del *dataset* Mauriziano in un video, con una durata di mezzo secondo per immagine. La scelta del tempo è per rendere apprezzabile le predizioni mentre vengono eseguite, visto che le immagini non vanno a comporre un video reale.

Infine è necessario che tale video risulti disponibile come se fosse proveniente da un dispositivo fisico collegato alla macchina su cui si esegue il programma. Video4Linux (`V4L`) è una collezione di *driver* video che espone, solo su piattaforma Linux, varie *API* per gestire i dispositivi video collegati al computer. In particolare espone anche una *API* per la creazione di un dispositivo video virtuale, ovvero un dispositivo non realmente esistente ma che il sistema operativo puo' interfacciarsi come fosse tale. Nuovamente tramite `ffmpeg` è possibile impostare il video precedentemente realizzato come flusso video in uscita da questo dispositivo virtuale, andando quindi a mimare il comportamento che si avrebbe con una webcam o lo stesso ISCAN. Va fatto notare che in questo caso il video creato non contiene i bordi neri che si avrebbero con un dispositivo ISCAN reale, come discusso nella sezione 2.1. Tuttavia sarebbe possibile all'avvio dell'applicativo impostare il flusso video di `ffmpeg` per ritagliare via questi bordi o eseguire la stessa procedura via codice Python nell'applicativo.

Ultimo componente esterno che è necessario avere a mente per comprendere il funzionamento dell'applicativo è il software `Docker` [17]. Esso consente di fare quel che è nota come *lightweight virtualization*, ovvero di creare un ambiente virtuale in cui eseguire un programma

in modo isolato dal resto del sistema. Questo permette di avere un ambiente pulito e prevedibile, in cui eseguire un determinato applicativo non dovendosi preoccupare di eventuali conflitti o che l'ambiente in questione sia adeguato. Esso è il *software* impiegato da `MMDeploy` per il *serving* dei modelli. `MMDeploy` è una libreria che fa parte dell'ecosistema `OpenMMLab`, che offre le funzionalità per esporre i modelli sviluppati con `MMSegmentation` , ovvero la libreria con cui è stato allenato il nostro modello di *deep learning*. `MMSegmentation` è infatti un livello di astrazione al di sopra della nota libreria `PyTorch` [18], libreria che permette di sviluppare modelli di *machine learning* con facilità. I modelli sviluppati con `MMSegmentation` [5] sono tuttavia non compatibili con `PyTorch`, per cui `MMDeploy` offre le funzionalità per convertire tali modelli in un formato compatibile con `PyTorch` e di eseguirne il *serving* tramite `Docker`, rendendo la procedura molto più semplice.

Un modello servito tramite `MMDeploy` [4] è dunque reso disponibile tramite un apposito *container* `Docker` che crea un ambiente virtuale Linux nel sistema che si sta utilizzando. Notare che il sistema ospite non deve necessariamente essere un sistema Linux. La comuniazione con tale *container* , e conseguentemente col nostro modello, è resa possibile tramite delle *API REST*. Esse sono un protocollo di comunicazione reso popolare dalla sua semplicità d'uso e flessibilità. Nello specifico, ad ogni richiesta ad un *endpoint* si otterrà una risposta che sarà indipendente dalle precedenti. Il container in questione espone tre *endpoint* , ma quello di nostro interesse è quello che permette di inviare un'immagine e ricevere la predizione come risposta. Tale *endpoint*

è esposto sulla porta 8080 all'indirizzo locale. Pertanto, l'endpoint finale sarà `http://localhost:8080/nome_modello/predictions`, con `nome_modello` che è il nome del *file* che viene utilizzato per salvare il modello. Questo procedimento, che puo' risultare complicato per persone non avvezze alla tecnologia e di poco interesse per il personale medico, puo' essere resto totalmente automatizzato, in modo che si possa lanciare l'applicativo senza dover eseguire ogni volta questa procedura.

Il funzionamento interno dell'applicativo è dunque piuttosto semplice. Esso acquisice in maniera periodica un fotogramma dalla sorgente video selezionata e lo invia tramite API REST al *container* `Docker`, al quale il nostro modello risponderà con la maschera di segmentazione corrispondente. Il timer è impostato a 40 millisecondi, che corrisponde a circa 24 fotogrammi al secondo, una frequenza video standard considerata sufficientemente fluida. Tale frequenza è superiore a quella riportata nella sezione 3.4.2, ma questo non è un problema. Infatti l'applicativo non farà nuove richieste se la precedente non è stata ancora soddisfatta, di fatto ponendo 24 fotogrammi al secondo come limite massimo. Inoltre, i fotogrammi del modello sono stati calcolati su *hardware* carente, ed è quindi molto probabile che reale velocità di inferenza su un sistema adeguato sia drasticamente superiore.

4.2 Interfaccia

L'interfaccia si presenta in maniera semplice e scarna, in modo da risultare di facile utilizzo e raggiungere le funzionalità principali con pochi passaggi. Uno di questi è una fase di *setup* minimale. All'avvio dell'applicazione viene mostrata una finestra che richiede all'utente di selezionare la sorgente video da utilizzare per catturare il flusso video. La cattura di tale flusso avviene mediante la libreria `OpenCV` [3] che permette di catturare singoli fotogrammi da un dispositivo e di riprodurli in maniera continua per dare la sensazione di star guardando un video. Questa fase è necessaria dal momento che in un determinato istante potrebbero esserci più dispositivi video collegati al computer su cui si sta eseguendo l'applicazione, e bisogna quindi scegliere quello che si vuole utilizzare, ovvero l'ISCAN.

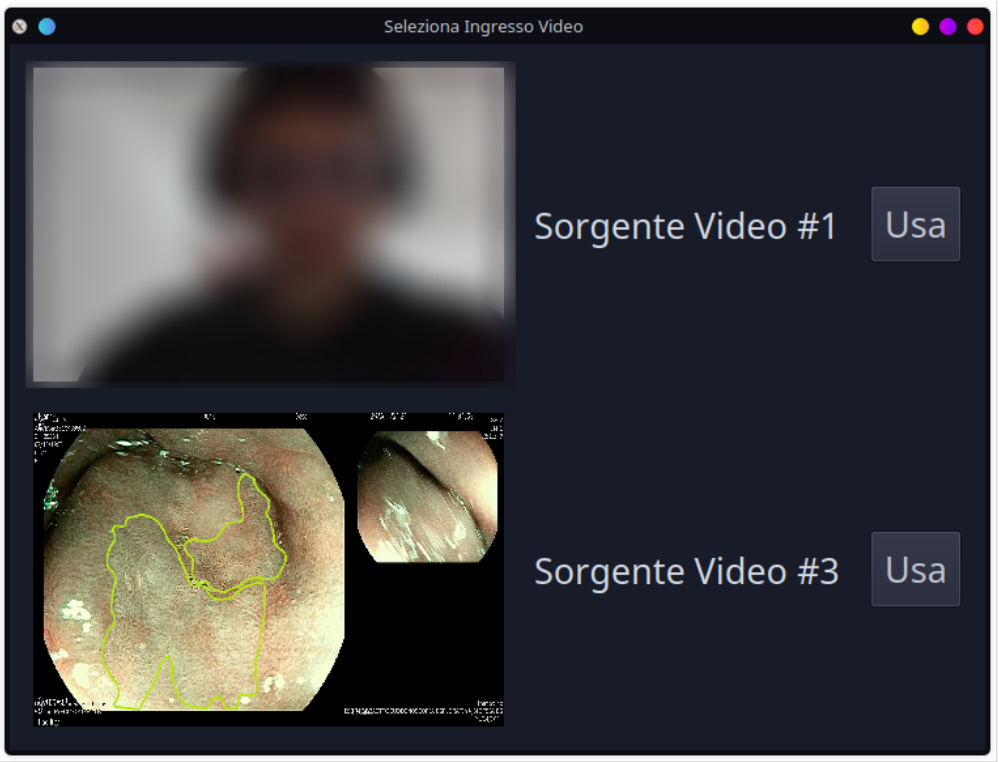

Figura 4.1: Finestra selezione ingresso video

Come mostrato in figura 4.1 vengono mostrate dunque tutti gli ingressi video disponibili, assieme ad una piccola anteprima del flusso video in ingresso da quella sorgente. Vengono filtrati in automatico i flussi video non validi, come si puo' notare dall'ID numerico mancante nell'immagine. Accanto ad ogni opzione è posto un bottone "Usa" che permette di selezionare la sorgente e chiude dunque questa prima finestra.

Chiusa la prima finestra se ne aprirà una seconda, che è la finestra principale dell'applicativo. Tale finestra alla sua creazione viene informata del flusso video selezionato e comincerà dunque a ritrasmetterlo per permettere all'utente di visualizzare le immagini in ingresso con le

relative predizioni. Non appena il flusso video viene agganciato viene anche avviato il *timer* che scadendo periodicamente cattuererà la corrente immagine e la convertirà in colori RGBA, visto che `OpenCV` utilizza il formato GBR. Tale immagine verà dunque inviata al modello che ne eseguirà la predizione e risponderà con una maschera di segmentazione dove i *pixel* di un valore pari a 1 corrispondono alla classe patologica.

A partire da tale maschera si genererà l'immagine da sovrapporre all'immagine in questione. Ciò avviene trasformando la maschera di predizione ottenuta in una maschera binaria ed applicando un operatore morfologico di dilatazione per un numero di iterazioni pari al valore di spessore di maschera impostato dall'interfaccia. A questo punto vengono rimossi da tale maschera tutti i *pixel* corrispondenti alla maschera binaria originale, in modo da ottenere non una maschera di segmentazione totalmente riempita ma piuttosto un bordo che ne evidenzia la zona, in modo da risultare meno invadente. Viene dunque creata un'immagine a partire da questa maschera, dove i it pixel del bordo verranno posti a un colore rosso con un'opacità pari a quella impostata dall'interfaccia, mentre i rimanenti saranno del tutto trasparenti, e questa foto viene dunque fusa all'immagine originale ottenuta dalla sorgente video. Un esempio del risultato finale è visibile in figura 4.2.

Come anticipato i due valori impostabili sono dunque quello dell'opacità della maschera di segmentazione e del suo spessore. Ciò avviene tramite due *slider* , dove nel caso dell'opacità si puo' scegliere tra un valore minimo del 10% ed uno massimo del 100%. Nel caso dello spessore

invece l'intervallo di valori va dalle 2 alle 15 iterazioni.

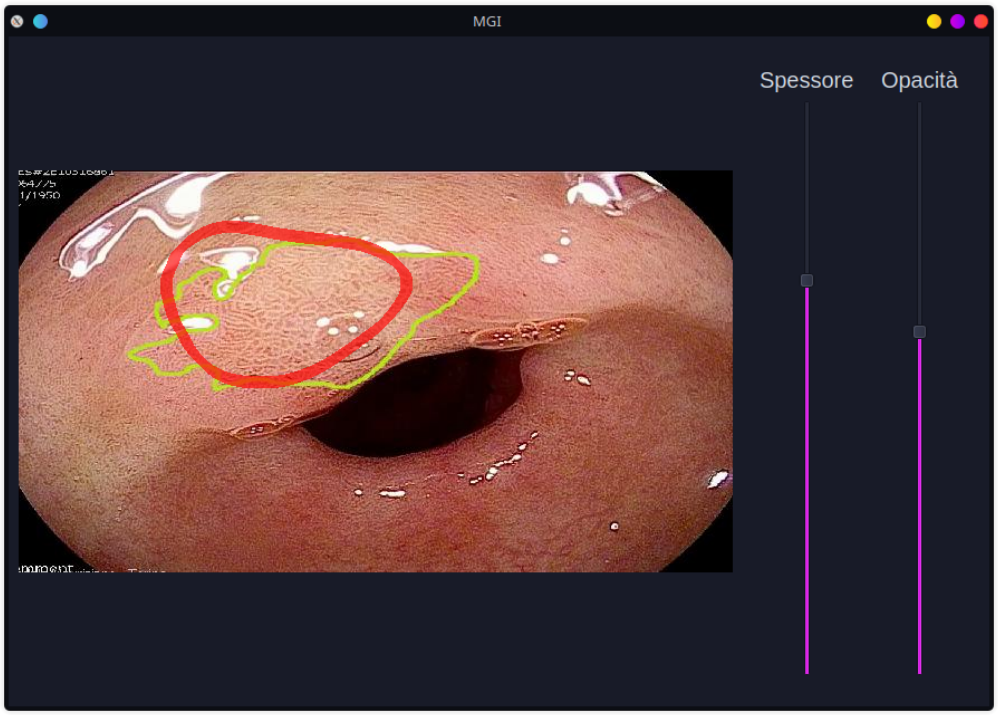

Figura 4.2: Interfaccia del *tool* per riconoscimento di metaplasie con metodo di deep learning.
In verde: area annotata come patologica
In rosso: area riconosciuta come patologica

Capitolo 5

Conclusioni

Quanto discusso in questa tesi ha permesso di accertare quali siano le possibilità e i limiti dell'applicazione dell'intelligenza artificiale a strumenti diagnostici visivi. La ricerca in merito ad architetture per l'elaborazione di immagini in tempo reale è da considerarsi tutt'altro che conclusa, tuttavia lavori recenti hanno messo a disposizione degli sviluppatori modelli già sufficientemente adeguati per poter implementare delle prime versioni funzionanti degli algoritmi. Ancora una volta, come spesso accade in ambiti di applicazione reale, il vero collo di bottiglia resta la quantità di dati a disposizione, che seppur calmierata attraverso varie tecniche di *training* e mediante l'espansione artificiale, quando numericamente troppo scarsa rende impossibile il raggiungimento di risultati soddisfacenti. Anche se intraprendente, i dati raccolti da un singolo ospedale in merito ad una patologia dalla bassa incidenza resta una fonte di dati insufficiente. Sebbene *dataset* pubblici stiano iniziando ad essere diffusi allo scopo di permettere la sperimentazione e l'affinamento dei modelli, ogni condizione medica ed ogni patologia restano un problema a sé stante, richiedendo dunque enromi sforzi dal punto di vista della raccolta e dell'annotazione dei dati, problema reso

ancor più oneroso in un ambito come quello medico dove entrambe le operazioni necessitano di personale dall'elevatissimo grado di specializzazione per essere portate a termine. D'altro canto, è stato possibile confermare come l'impiego di modelli di *deep learning* si sia dimostrato nuovamente almeno al pari di approcci più classici, dato non di poco conto consideranta appunto la voracità dei modelli in termini di quantità di informazione richiesta.

Bibliografia

[1] Melina Arnold, Christian C. Abnet, Rachel E. Neale, Jerome Vignat, Edward L. Giovannucci, Katherine A. McGlynn, and Freddie Bray. Global burden of 5 major types of gastrointestinal cancer., 2020.

[2] Hanna Borgli, Vajira Thambawita, Pia H Smedsrud, Steven Hicks, Debesh Jha, Sigrun L Eskeland, Kristin Ranheim Randel, Konstantin Pogorelov, Mathias Lux, Duc Tien Dang Nguyen, Dag Johansen, Carsten Griwodz, Håkon K Stensland, Enrique Garcia-Ceja, Peter T Schmidt, Hugo L Hammer, Michael A Riegler, Pål Halvorsen, and Thomas de Lange. Hyperkvasir, a comprehensive multi-class image and video dataset for gastrointestinal endoscopy. *Scientific Data*, 7(1):283, 2020.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] MMDeploy Contributors. Openmmlab's model deployment toolbox. https://github.com/open-mmlab/mmdeploy, 2021.

[5] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020.

[6] SciPy 1.0 Contributors. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

[7] Gabriela Csurka, Riccardo Volpi, and Boris Chidlovskii. Semantic image segmentation: Two decades of research, 2023.

[8] Jingyi Cui, Hanyuan Hang, Yisen Wang, and Zhouchen Lin. Gbht: Gradient boosting histogram transform for density estimation, 2021.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] Foto di Anoir Chafik. Elaborazione personale.

[11] Rahim Entezari, Mitchell Wortsman, Olga Saukh, M. Moein Shariatnia, Hanie Sedghi, and Ludwig Schmidt. The role of pre-training data in transfer learning, 2023.

[12] Joost J.M et al. Pyradiomics, a library for clinical image processing, 2017.

[13] Siripoppohn V. et al. Real-time semantic segmentation of gastric intestinal metaplasia using a deep learning approach, 2022.

[14] Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On explaining decision trees, 2020.

[15] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[16] Toma M. Metodi di machine learning per il riconoscimento di metaplasie gastrointestinali a partire da immagini gastroscopiche, 2022.

[17] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Shruti Singh, Divya Srivastava, and Suneeta Agarwal. Glcm and its application in pattern recognition. In *2017 5th International Symposium on Computational and Business Intelligence (ISCBI)*, pages 20–25, 2017.

[21] Suramya Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006.

[22] Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation, 2020.

[23] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. *CoRR*, abs/1808.00897, 2018.

[24] Pengfei Zhang, Eric Lo, and Baotong Lu. High performance depthwise and pointwise convolutions on mobile devices, 2020.

Elenco delle figure

Elenco delle tabelle