UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

INSIEME DI APPROCCI PER LA SEGMENTAZIONE 3D DEL TUMORE CEREBRALE

Relatore: Prof. Loris Nanni

Laureando: Christian Francesco Russo

ANNO ACCADEMICO 2021 – 2022
Data di laurea 22/09/2022

Abstract

Il riconoscimento di oggetti all'interno delle immagini richiede abilità complesse quali la conoscenza del contesto e la capacità di identificare i bordi degli oggetti.

In computer vision, questo compito è chiamato segmentazione semantica e consiste nella classificazione di ogni pixel di un'immagine. Tale compito è di primaria importanza in molti scenari della vita reale: nella guida autonoma consente l'identificazione degli oggetti che circondano il veicolo; nella diagnosi medica migliora la capacità di individuare precocemente patologie pericolose, in questo modo si riesce a prevenire il rischio di conseguenze gravi che sarebbero causate da uno stato di avanzamento della malattia [1].

In questo lavoro di tesi viene proposto un nuovo insieme di approcci di segmentazione binaria 3D di immagini mediche cerebrali, ottenute mediante risonanza magnetica (MRI), con lo scopo di provare a migliorare quello che ad oggi è lo stato dell'arte delle attuali tecniche di riconoscimento del tumore cerebrale.

In prima istanza vengono proposti due primi approcci 'weak'. Il primo metodo proposto esegue la segmentazione a livello di slice 2D di volume, questo metodo è basato su DeepLabV3+ e sfrutta una rete ResNet come backbone. Il secondo metodo proposto esegue la segmentazione a livello di patch di volume 3D, questo metodo è basato su U-Net 3D.

Al fine di andare a migliorare le prestazioni ottenute singolarmente dai due metodi precedenti, viene proposto un terzo approccio 'strong' che consiste nell'ensemble dei primi due approcci.

Le prestazioni e la qualità dell'ensemble sono fortemente legate ad alcuni fattori; uno dei più importanti è la diversità tra i singoli modelli. Per questo motivo le reti 'weak', ottenute dai primi due approcci proposti, vengono addestrate utilizzando una pletora di loss function differenti. Successivamente vengono testate le prestazioni di tutti gli ensemble ottenuti combinando ciascuna delle due reti 'weak', che sono state addestrate per ciascuna delle loss function proposte.

Contenuti

Introduzione

I tumori cerebrali sono forme tumorali che colpiscono il sistema nervoso centrale, ossia l'insieme di encefalo, midollo allungato e cervelletto. Essi si distinguono in tumori primitivi, che si sviluppano direttamente nel tessuto nervoso centrale, e in tumori secondari, che hanno origine da tumori che crescono in altri organi (es. polmone o mammella) e che poi si diffondono al tessuto nervoso [2].

I tumori primitivi sono abbastanza rari, in media in Italia rappresentano l'1.6% di tutti i tumori, mentre sono invece più frequenti i tumori secondari. Il tasso di incidenza di tumori cerebrali registrati in questi ultimi anni è andato progressivamente aumentando. Questo è legato non solo alla maggiore diffusione di tecniche di imaging, quali MRI (risonanza magnetica) e TC (tomografia computerizzata), ma anche alla maggiore esposizione a campi elettromagnetici a cui tutti noi oggi siamo sempre più sottoposti, in particolare quelli emessi dai cellulari (anche se ad oggi non esistono prove certe che ciò aumenti il rischio di sviluppare un cancro) che, insieme alle esposizioni a radiazioni ad alto dosaggio cancerogene, sono tra le principali cause dello sviluppo di tale patologia. Altre cause dell'insorgenza di un tumore cerebrale sono legate allo sviluppo di mutazioni, spesso casuali, del DNA durante il processo di divisione cellulare [3].

Il miglioramento delle tecniche di machine learning in questi ultimi anni, in particolare per quanto riguarda le tecniche di deep learning nell'ambito del computer vision, ne ha permesso l'utilizzo in diversi ambiti applicativi. Per esempio, in ambito di diagnosi medica, la segmentazione semantica di immagini mediche viene pesantemente utilizzata per diagnosticare patologie sin dai primi stadi, o ancora prima che queste si manifestino, questo con una precisione molto maggiore di quella di un operatore esperto umano e con un notevole risparmio di tempo, permettendo in questo modo ai medici di intervenire preventivamente per curare ed evitare il degenerare di tali patologie.

D'altra parte però, è noto che l'addestramento di reti neurali profonde richieda un enorme quantitativo di dati, cosa che in ambito medico è molto difficile ottenere, specie per quanto riguarda le patologie rare o le nuove patologie. Per questo motivo, per poter utilizzare in maniera proficua questa tipologia di reti, si ricorre a tecniche di transfer learning, quali fine-tuning. In questi ultimi anni è stato inoltre compiuto un grande lavoro per quanto riguarda la costruzione e la pubblicazione di dataset per diversi specifici settori di applicazione. Ne è un esempio il dataset BraTS che è stato utilizzato per questo lavoro e che contiene volumi di scansioni MRI cerebrali per l'identificazione del tumore cerebrale primale.

In questo lavoro di tesi viene proposto un nuovo insieme di approcci di segmentazione binaria 3D di scansioni MRI cerebrali, con lo scopo di provare a migliorare quello che ad oggi è lo stato dell'arte delle attuali tecniche di riconoscimento del tumore cerebrale.

In prima istanza vengono proposti due primi approcci 'weak'. Il primo metodo proposto esegue la segmentazione a livello di slice 2D di volume, questo metodo è basato su DeepLabV3+ e sfrutta una rete ResNet come backbone. Il secondo metodo proposto esegue la segmentazione a livello di patch di volume 3D, questo metodo è basato su U-Net 3D.

Il noto teorema no-free lunch per il machine learning afferma che non può esistere un singolo modello che lavora bene su qualsiasi dataset. Per questo motivo, al fine di andare a migliorare le prestazioni ottenute singolarmente dai due metodi precedenti, viene proposto un terzo approccio 'strong' che consiste nell'ensemble dei primi due approcci. In un ensemble le singole reti 'weak' vengono addestrate sullo stesso training set, in questo modo ogni modello dovrebbe generalizzare in maniera differente nello spazio di training. Gli ensemble forniscono lo stato dell'arte in molti domini di utilizzo, ma è importante assicurare alcune proprietà. Una di queste è quella di imporre qualche tipo di diversità tra il set di reti 'weak'.

Per questo motivo, ciascuna rete ottenuta dai primi due approcci proposti viene addestrata utilizzando una diversa loss function, scelta tra una pletora di loss function proposte. Successivamente vengono testate le prestazioni di tutti gli ensemble ottenuti combinando ciascuna coppia di reti 'weak', allo scopo di trovare quella con prestazioni maggiori.

La struttura di questa tesi è la seguente.

Nel Capitolo 1 vengono trattati in maniera generale tutti i concetti e gli aspetti fondamentali che sono alla base del deep learning.

Nel Capitolo 2 viene trattata la segmentazione semantica, in particolare si entra più nel dettaglio di quelli che sono i concetti teorici e gli aspetti pratici che sono stati utilizzati per questo lavoro di tesi. In particolare, viene dapprima introdotto il dataset BraTS, le sue caratteristiche e il suo utilizzo, poi vengono introdotte, prima da un punto di vista teorico, poi implementativo, le architetture di rete utilizzate per implementare i diversi approcci di segmentazione proposti, infine vengono descritte le metriche utilizzate per misurare le performance dei metodi di segmentazione proposti.

Nel Capitolo 3 viene trattata ancora più nel dettaglio l'esperienza svolta, in particolare viene descritta l'implementazione in linguaggio MATLAB del codice che è stato implementato per ciascuno dei diversi approcci proposti, i parametri di addestramento utilizzati e soprattutto vengono trattate nel dettaglio tutte le loss function che sono state testate e i relativi risultati sperimentali raccolti.

Infine, nel capitolo Conclusioni vengono analizzati i risultati ottenuti, vengono tratte le considerazioni finali sul lavoro svolto e vengono riportati quelli che potrebbero essere gli ulteriori sviluppi futuri per questo lavoro di tesi.

Capitolo 1

Deep Learning

Il deep learning è una branca del machine learning che utilizza le reti neurali artificiali per risolvere gli stessi problemi, oppure problemi più complessi, di quelli che normalmente vengono risolti con le classiche tecniche di machine learning. A differenza di queste ultime però, le reti neurali profonde sono in grado di raggiungere prestazioni assai migliori e normalmente non sembrano presentare un limite superiore alle prestazioni massime raggiungibili. Un particolare tipo di reti neurali profonde sono le CNN che trovano largo impiego nell'analisi di immagini. Le reti neurali profonde, così come le CNN, per poter essere addestrate efficacemente richiedono una grande quantità di dati, cosa che spesso non si ha a disposizione. Per questo motivo spesso si ricorre a tecniche di Transfer Learning, come il fine-tuning, per riutilizzare reti pre-addestrate su dataset contenenti classi di oggetti diverse da quelle del dataset che si ha a disposizione e le si vanno a modificare in modo da adattarle per il tipo di problema che si vuole risolvere.

1.1 Neuroni

1.1.1 Neurone artificiale

Fin dagli albori dell'informatica si è cercato di creare un sistema intelligente che fosse in grado di simulare il funzionamento del cervello umano.

Alla base del funzionamento del cervello umano si trova una rete di circa 86 miliardi di neuroni, i quali sono collegati tra loro mediante circa 10^{15} sinapsi. Ogni neurone è formato da un corpo centrale detto soma, dal quale partono i dentriti che permettono al neurone di ricevere uno o più segnali in ingresso. Una volta elaborati questi segnali, il neurone emette un segnale in uscita mediante l'assione; questo segnale costituisce il segnale di ingresso per altri neuroni.

Il neurone perciò, altro non è che una funzione di trasferimento che mappa uno o più segnali di ingresso in un segnale di uscita. La prima formalizzazione del modello matematico del neurone è stata data nel 1943 da McCulloch e Pittis ed è rappresentato in Figura 1.1:

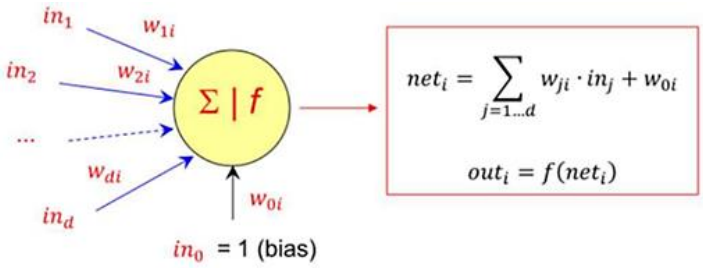

Figura 1.1 - Rappresentazione del modello matematico di un neurone

dove si ha che:

- $in_1, in_2, \dots, in_d$ sono i d segnali che costituiscono il pattern (o feature vector) di ingresso al neurone i-esimo
- $w_{1i}, w_{2i}, \dots, w_{di}$ sono i pesi di ogni ingresso j-esimo
- w_{0i} è detto bias ed è un ulteriore peso collegato ad un ingresso fittizio che permette di tarare il punto di lavoro ottimale del neurone
- net_i è il livello di eccitazione globale (o potenziale interno) del neurone
- $f(\cdot)$ è la funzione di attivazione che determina il comportamento del neurone in funzione del suo livello di eccitazione net_i; essa rappresenta pertanto l'output del neurone

1.1.2 Percettrone

Nel 1956 viene proposto da Rosenblatt un nuovo modello di neurone chiamato percettrone, rappresentato in Figura 1.2, con il quale viene introdotto anche il concetto di addestramento:

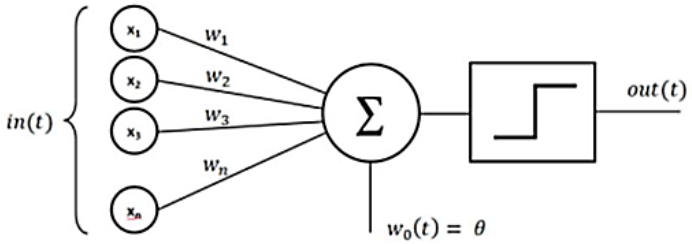

Figura 1.2 - Rappresentazione del modello matematico di un percettrone

dove si ha che:

- L'ingresso del percettrone è:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = \sum_{i=1}^{n} w_i x_i + b$$

- L'uscita del percettrone è da una semplice funzione di attivazione a scalino:

$$\begin{cases} 1 & \text{se } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0 \\ 0 & \text{altrimenti} \end{cases}$$

Per modificare i pesi della rete durante l'addestramento si utilizza la delta rule:

$$\Delta_{w_{ij}} = \alpha(t_i - y_i)x_i$$

dove si ha che:

- α è il learning rate
- t_j è l'output che si vuole ottenere (o target) dal percettrone j-esimo
- y_j è l'output predetto dal percettrone j-esimo
- x_i è l'input i-esimo del percettrone j-esimo

perciò si osserva che se il pattern in input al percettrone è classificato correttamente ($y_j = t_j$) allora i pesi del percettrone non vengono modificati, altrimenti i pesi vengono modificati di un valore $\Delta_{w_{ij}}$.

1.2 Reti neurali

1.2.1 Reti feedforward e recurrent

Una rete neurale artificiale è una rete formata da gruppi di neuroni organizzati in livelli (o layer). In base al tipo di collegamenti tra i neuroni dei vari livelli della rete, si possono distinguere due tipologie di reti neurali: le reti feedforward e le reti recurrent.

Nelle reti feedforward ogni neurone è collegato con i neuroni del livello successivo, perciò non sono presenti cicli all'interno del grafo della rete, ossia non esistono connessioni tra neuroni dello stesso livello o connessioni orientate da un livello a quello precedente.

Al contrario le reti recurrent contengono, oltre alle connessioni di tipo feedforward, anche connessioni di feedback tra neuroni dello stesso livello o con neuroni del livello precedente; perciò questo tipo di reti, rispetto a quelle precedenti, possiede la proprietà dell'"effetto memoria" che è utile in tutti quei casi in cui il pattern in input è una sequenza (es. video, audio, frasi del linguaggio naturale, ecc.).

1.2.2 Multilayer perceptron

Una rete multilayer perceptron (MLP) è una rete feedforward con almeno tre livelli, ossia avente un livello di input (o input layer), un livello di output (o output layer) e almeno un livello interno (o hidden layer):

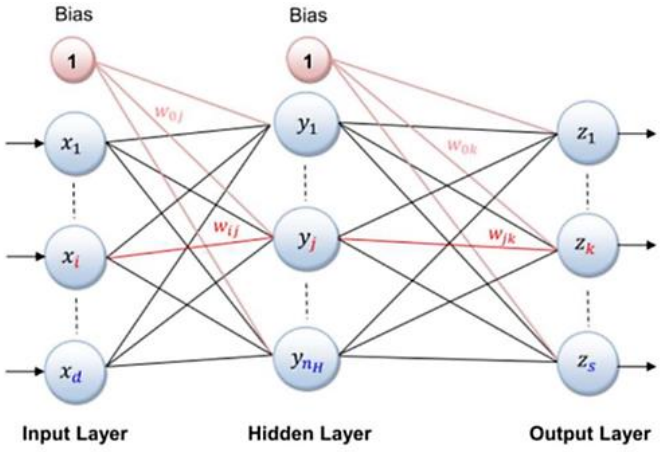

Figura 1.3 - Rappresentazione di una rete multilayer perceptron a tre livelli

Dal teorema di universal approximation si dimostra che una MLP avente un solo livello nascosto, come quella rappresentata in Figura 1.3, è in grado di approssimare qualsiasi funzione continua che mappi l'ingresso $\mathbb{R}^n$ in qualsiasi uscita $\mathbb{R}^m$ (che può essere rappresentata graficamente come un iperquadratica).

La validità di questo teorema non implica tuttavia che nella pratica esista un modo semplice per ottenere tale risultato, in particolare per ipotesi sono necessari un dataset privo di rumore e con dimensionalità infinita e un algoritmo di ottimizzazione dei pesi ottimale.

Per questo motivo, per risolvere il primo problema, in questi ultimi anni si è passati dallo sviluppo di semplici reti con pochi livelli interni al deep learning, ossia allo sviluppo di reti complesse con centinaia o anche migliaia di livelli interni. A questo punto rimane però ancora il problema dell'addestramento inefficiente di queste reti che, come descritto più avanti, fu risolto nel 1986 da Hinton.

1.2.3 Forward propagation

Durante la fase di esecuzione, dopo aver addestrato la rete, ossia dopo aver tarato tutti i suoi pesi, la rete neurale può essere utilizzata per processare i pattern in ingresso e calcolare in questo modo l'output attraverso la tecnica forward propagation (o propagazione dell'input in avanti).

Tornando alla rete in Figura 1.3, è possibile osservare che per poter calcolare l'output della rete è necessario calcolare l'output di ogni singolo neurone, partendo dall'input layer e spostandosi di livello in livello verso l'output layer, come:

$$y_j = f\left(\sum_{i=1,\dots,d} w_{ij} \cdot x_i + w_{0j}\right)$$

dove, come già visto, $f(\cdot)$ è la funzione di attivazione del neurone j-esimo.

L'output complessivo della rete può quindi essere calcolato come:

$$z_k = f\left(\sum_{j=1,\dots,n_H} w_{jk} \cdot y_j + w_{0k}\right) =$$

$$= f\left(\sum_{j=1,\dots,n_H} w_{jk} \cdot f\left(\sum_{i=1,\dots,d} w_{ij} \cdot x_i + w_{0j}\right) + w_{0k}\right)$$

Dato l'output $\boldsymbol{z}$ della rete, ottenuto a partire dal pattern $\boldsymbol{x}$ in input, nel seguito verrà descritto come realizzare un ottimizzatore che permetta di settare i pesi della rete in modo da minimizzare l'errore tra l'output desiderato $\boldsymbol{t}$ e l'output $\boldsymbol{z}$ previsto dalla rete.

1.3 Addestramento

1.3.1 Loss function

L'addestramento di una rete neurale consiste nel determinare il valore dei suoi pesi interni $\boldsymbol{w}$ che permettono di mappare il pattern in ingresso $\boldsymbol{x}$ nell'uscita $\boldsymbol{z}$ della rete.

Come già anticipato in precedenza, nel 1986 Hinton e Williams hanno sviluppato un algoritmo efficiente per effettuare l'addestramento delle reti neurali profonde chiamato algoritmo di error backpropagation (o gradient descent), il quale si basa sull'applicazione della regola di derivazione a catena. Questo algoritmo ancora oggi è alla base degli algoritmi di ottimizzazione utilizzati nelle moderne reti neurali profonde.

Prima di poter introdurre l'algoritmo di ottimizzazione backpropagation, è necessario definire un altro concetto importante che è il concetto di loss function. Sia $\boldsymbol{z} = [z_1, \dots, z_s]$ l'output prodotto dalla rete (attraverso forward propagation), in corrispondenza di un dato pattern di ingresso $\boldsymbol{x} = [x_1, \dots, x_d]$, e sia $\boldsymbol{t} = [t_1, \dots, t_s]$ l'output desiderato; definiamo la funzione di ottimizzazione da minimizzare, detta loss function, come:

$$J(\boldsymbol{w}, \boldsymbol{x}) \stackrel{\text{def}}{=} \frac{1}{2} \sum_{c=1,\dots,s} (t_c - z_c)^2$$

essa quantifica quanto l'output prodotto dalla rete si discosta da quello desiderato.

L'errore $J(\boldsymbol{w})$ sull'intero training set è dato dalla media di tutti i $J(\boldsymbol{w}, \boldsymbol{x})$, per ogni pattern $\boldsymbol{x}$ del training set. Nel seguito viene descritto come minimizzare la loss function $J(\boldsymbol{w}, \boldsymbol{x})$ andando a modificare i pesi $\boldsymbol{w}$ della rete in direzione opposta al gradiente di J.

1.3.2 Algoritmo backpropagation e aggiornamento dei pesi

L'algoritmo di back propagation, detto anche gradient descent, è un algoritmo di ottimizzazione dei pesi che consiste nell'andare a calcolare il gradiente di ogni loss function $J(\boldsymbol{w}, \boldsymbol{x})$ (in funzione dei pesi della rete), in modo da trovare la direzione di massima crescita

dell'errore. Dopo aver determinato la direzione di massima crescita dell'errore, quello che si fa è andare ad aggiornare i pesi della rete muovendosi in direzione opposta a quella del gradiente calcolato, ossia nella direzione di massima decrescita dell'errore, in questo modo si riesce a minimizzare l'errore $J(\boldsymbol{w}, \boldsymbol{x})$.

Nello specifico, facendo sempre riferimento alla rete in Figura 1.3, abbiamo che:

- Per calcolare il gradiente delle loss function dei pesi tra i neuroni dei livelli hidden-output il procedimento è il seguente:

$$\frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial w_{jk}}\left(\frac{1}{2}\sum_{c=1,\dots,s}(t_c - z_c)^2\right) = -\delta_k \cdot y_j$$

  dove per ottenere il risultato soprariportato è stata utilizzata la regola della catena, i passaggi sono stati omessi.

  A questo punto per aggiornare i pesi w_{jk}, come già accennato, quello che si fa è muoversi in direzione opposta al gradiente, ossia nella direzione $+\delta_k \cdot y_j$ come segue:

$$w_{jk} = w_{jk} + \eta \cdot \delta_k \cdot y_j$$

  dove chiamiamo learning rate il parametro η che verrà definito nel seguito.

- Allo stesso modo per calcolare il gradiente delle loss function dei pesi tra i neuroni dei livelli input-hidden il procedimento è il seguente:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial w_{ij}}\left(\frac{1}{2}\sum_{c=1,\dots,s}(t_c - z_c)^2\right) = -\delta_j \cdot x_i$$

  dove anche in questo caso per ottenere il risultato soprariportato è stata utilizzata la regola della catena, i passaggi sono stati omessi.

  A questo punto per aggiornare i pesi w_{ij}, come prima, quello che si fa è muoversi in direzione opposta al gradiente, ossia nella direzione $+\delta_k \cdot x_i$ come segue:

$$w_{ij} = w_{ij} + \eta \cdot \delta_j \cdot x_i$$

Si noti che sia l'algoritmo di backpropagation che l'aggiornamento dei pesi vengono eseguiti durante la fase di addestramento della rete, ma l'algoritmo di backpropagation per il calcolo del gradiente viene eseguito per primo e si applica a partire dall'output layer spostandosi di livello in livello verso l'input layer, l'aggiornamento dei pesi invece viene eseguito subito dopo e si applica a partire dall'input layer spostandosi di livello in livello verso l'output layer.

1.3.3 Learning rate

Il learning rate η definisce l'"aggressività" con la quale ci si muove nella direzione di convergenza della funzione J. Esso è pertanto uno degli iperparametri del sistema, ossia uno di quei parametri il cui valore viene definito in fase di progettazione della rete.

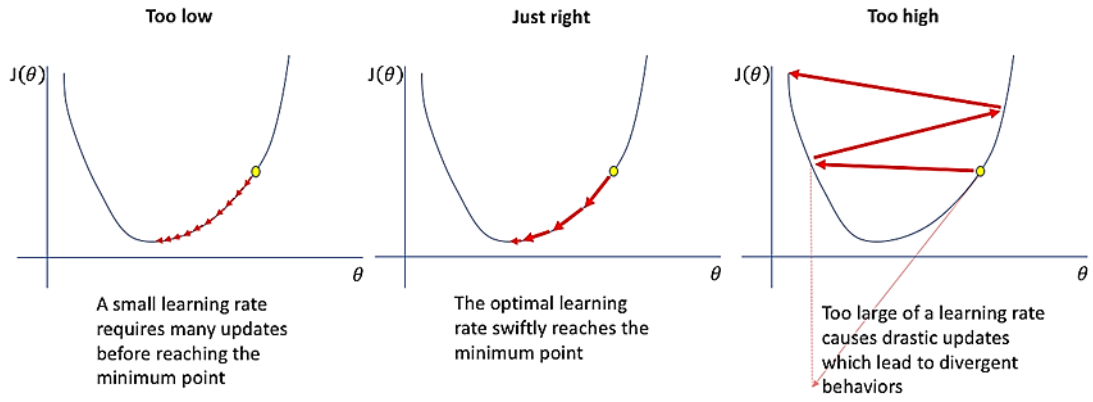

Figura 1.4 - Rappresentazione della convergenza della loss in funzione del learning rate

Come mostrato in Figura 1.4 un learning rate troppo alto può portare ad avere grandi oscillazioni della convergenza, rendendo il sistema instabile o portando addirittura alla divergenza; al contrario un learning rate troppo basso porta a una convergenza molto lenta, pertanto bisognerà eseguire più epoche di addestramento sull'intero training set prima che il sistema riesca a raggiungere la convergenza.

1.3.4 Funzioni di attivazione

Le funzioni di attivazione, come detto in precedenza, rappresentano le funzioni di output di ciascun neurone della rete, in particolare sono funzioni non lineari che mappano l'input nell'output del neurone. In base alla funzione di attivazione scelta, l'output della rete può essere uno score di similarità o una vera e propria probabilità. Di seguito sono riportate alcune delle più note funzioni di attivazione [8]:

- Sigmoide: questa funzione è utilizzata principalmente per i problemi di classificazione binaria:
$$f(z) = \frac{1}{1 + e^{-z}}$$
- Tangente iperbolica: questa funzione è simile alla sigmoide, con la differenza che è centrata in zero:
$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$
- ReLU (Rectified Linear Unit): questa funzione è migliore delle due precedenti in quanto risolve il problema del vanishing descent, ossia il rischio che, durante l'esecuzione dell'algoritmo gradient descent, la rete non modifichi più i propri pesi a causa delle ripetute moltiplicazioni di piccoli valori, dovute all'applicazione della regola della catena, che possono portare il gradiente a zero; questa funzione viene di solito utilizzata per i livelli nascosti della rete:
$$f(z) = \max(0, z)$$
- Leaky ReLU: questa funzione è una versione aggiornata della ReLU e risolve il problema della ReLU morente, ossia il fatto che le attivazioni nella regione $z < 0$

hanno pendenza 0 e di conseguenza in quella regione i pesi non vengono più aggiornati:

$$f(z) = \max(0.1x, x)$$

- Softmax: questa funzione di attivazione consiste nella combinazione di molte sigmoidi e permette di calcolare la probabilità che un dato pattern appartenga a ciascuna classe del training set, per questo motivo è utilizzata per problemi di classificazione multi-classe e viene usata solo per l'ultimo livello della rete:

$$f(net_k) = \frac{e^{net_k}}{\sum_{c=1,\dots,s} e^{net_c}}$$

1.4 Algoritmi di ottimizzazione

1.4.1 Stochastic gradient descent (SGD)

L'algoritmo Stochastic Gradient Descent (SGD) [4] è uno degli approcci ad oggi più utilizzati per implementare l'algoritmo Gradient Descent (GD).

Nella discesa stocastica del gradiente si va a sostituire il vettore del gradiente di GD con una sua stima stocastica, dove questa stima stocastica è ottenuta dal gradiente della loss function per una singola istanza di dati.

Sia f_i la loss function della rete per l'istanza i-esima:

$$f_i = J(x_i, y_i, \boldsymbol{w})$$

La funzione da minimizzare è f che è la loss totale su tutte le istanze:

$$f = \frac{1}{n}\sum_{i=1}^{n} f_i$$

In SGD i pesi vengono aggiornati in base al gradiente di f_i (anziché alla loss totale f):

$$w_{k+1} = w_k - \eta_k \nabla f_i(w_k)$$

dove se i è scelto casualmente, allora f_i è una stima di f rumorosa, ma non distorta, e può essere riscritta matematicamente come:

$$E[\nabla f_i(w_k)] = \nabla f(w_k)$$

Come risultato di ciò, il passo k-esimo di SGD è in media identico al passo k-esimo di GD calcolato su tutte le istanze:

$$E[\mathrm{w_{k+1}}] = w_k - \eta_k E[\nabla f_i(w_k)] = w_k - \eta_k \nabla f(w_k)$$

quindi in media ogni aggiornamento di SGD è equivalente ad un aggiornamento calcolato su tutti i dati.

Oltre a essere più rapido, SGD fornisce risultati migliori di una discesa del gradiente completa, inoltre il rumore di SGD permette di evitare di rimanere bloccati in minimi locali,

in questo modo si è in grado di trovare un minimo locale migliore (più profondo). Questo fenomeno è chiamato Annealing.

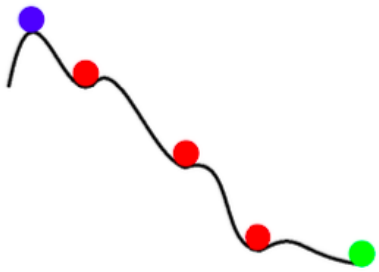

Figura 1.5 - Rappresentazione del fenomeno Annealing con SGD

Riassumendo i vantaggi di SGD sono:

- Previene la computazione di istanze ridondanti
- Per le prime iterazioni il rumore è piccolo se comparato all'informazione trasmessa dal gradiente, quindi un passo di SGD è virtualmente tanto buono quanto un passo di GD
- Annealing: il rumore nell'aggiornamento del SGD può prevenire la convergenza verso minimi locali poco profondi
- È drasticamente meno dispendioso dal punto di vista computazionale in quanto il gradiente non deve essere ricalcolato per tutte le istanze

1.4.2 Mini-batching

La tecnica del mini-batching [4] prevede che la loss function venga calcolata su più istanze del training set alla volta, anziché su una sola istanza, questo permette di ridurre il rumore nel passo di aggiornamento:

$$w_{k+1} = w_k - \eta_k \frac{1}{|B_i|} \sum_{j \in B_i} \nabla f_j(w_k)$$

dove B_i è il mini-batch i- esimo.

I mini-batch permettono di utilizzare appieno l'hardware a disposizione, infatti le GPU risultano essere sottoutilizzate quando, per l'addestramento, vengono utilizzate singole istanze. È importante notare che la discesa del gradiente non dovrebbe mai utilizzare batch grandi quanto il training set.

Con epoca si indica l'addestramento eseguito utilizzando mini-batch che coprono l'intero gruppo di istanze del training set, in particolare con numero di epoche si indica il numero di volte in cui l'intero training set è stato utilizzato per eseguire l'addestramento.

1.4.3 Stochastic gradient descent con momento (sgdm)

L'algoritmo Stocastic Gradient Descent con Momento (sgdm) [4], a differenza dell'algoritmo SGD, durante la fase di aggiornamento pesi utilizza due quantità, anziché una sola, che vengono aggiornate iterativamente:

$$p_{k+1} = \beta_k p_k + \nabla f_i(w_k)$$

$$w_{k+1} = w_k - \eta_k p_{k+1}$$

dove p è chiamato momento del SGD e $\beta \in [0,1]$ è detto fattore di smorzamento. p può essere pensato come una media a valori mobili del gradiente, ad ogni iterazione w viene mosso nella direzione del nuovo momento p_{k+1}.

La forma precedente può essere riscritta in forma matematicamente equivalente come segue:

$$w_{k+1} = w_k - \eta_k \nabla f_i(w_k) + \beta_k (w_k - w_{k-1})$$

questa forma prende il nome di "metodo stocastico della palla pesante".

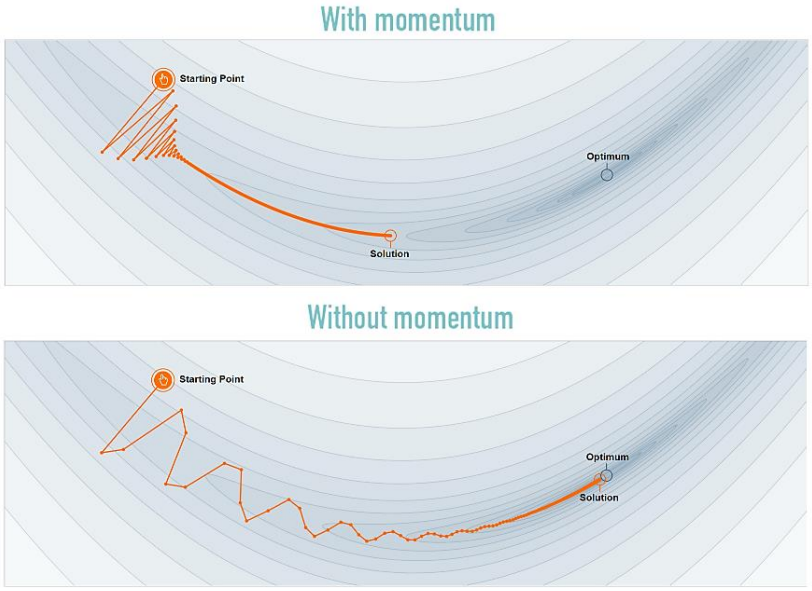

Figura 1.6 - Rappresentazione dell'effetto del momento

L'intuizione che sta dietro al concetto di momento è simile a quella di momento fisico. Il processo di ottimizzazione può essere assimilato a quello del rotolamento di una palla pesante giù per una collina. Il momento mantiene la palla in movimento nella stessa direzione nella quale si sta muovendo. Il gradiente può invece essere pensato come una forza che spinge la palla in un'altra direzione.

Come mostrato in Figura 1.6, il momento smorza le oscillazioni che sono molto comuni con SGD. Per quanto riguarda il fattore di smorzamento β si ha che: per $\beta = 0$ si ottiene SGD, per $0 < \beta < 1$ si ottiene un tempo di cambio di direzione che è tanto maggiore quanto maggiore è il valore di β.

1.4.4 Adam

L'algoritmo SGD con momento rappresenta lo stato dell'arte dei metodi di ottimizzazione per molti problemi di machine learning. Ma esistono altri metodi, chiamati metodi adattivi, che vengono utilizzati nel caso in cui SGD con momento non funzioni.

A differenza di SGD in cui ogni singolo peso viene aggiornato utilizzando un'equazione con lo stesso learning rate η globale, i metodi adattivi adattano il learning rate individualmente per ogni peso, per farlo utilizzano l'informazione fornita dai gradienti per ciascun peso.

Uno dei metodi adattivi più utilizzati è ADAM (ADAptive Moment estimation) [5] in cui l'aggiornamento del momento è convertito in una media mobile esponenziale. Con ADAM i pesi vengono aggiornati come segue:

$$m_{t+1} = \beta m_t + (1-\beta)\nabla f_i(w_t)$$

$$v_{t+1} = \alpha v_t + (1-\alpha)\nabla f_i^2(w_t)$$

$$w_{t+1} = w_t - \eta \frac{m_t}{\sqrt{v_t + 1} + \epsilon}$$

dove η è il learning rate globale, ϵ è una costante molto piccola utilizzata per evitare errori di divisione per zero, v_t è una stima del secondo momento, $\alpha \in (0,1)$ è un fattore costante ad ogni passo t-esimo che smorza i valori meno recenti finché non sono più una parte importante nella media mobile esponenziale, m_{t+1} è la media mobile esponenziale del momento.

1.4.5 Batch normalization

Nelle reti neurali normalmente si alternano operazioni lineari (es. operazioni di convoluzione nelle CNN) con operazioni non lineari (ossia funzioni di attivazione). È pratica comune inserire uno layer di batch normalization prima del layer lineare o dopo la funzione di attivazione. I layer di normalizzazione influenzano i dati, ma non modificano la potenza della rete, nel senso che una rete non normalizzata è sempre in grado di fornire lo stesso output della medesima rete normalizzata.

L'operazione di normalizzazione è definita semplicemente come:

$$\boldsymbol{y} = \frac{a(\boldsymbol{x} - \boldsymbol{\mu})}{\boldsymbol{\sigma}} + \boldsymbol{b}$$

dove $\boldsymbol{x}$ è il vettore in input, $\boldsymbol{y}$ è il vettore in output, $\boldsymbol{\mu}$ è la stima della media di $\boldsymbol{x}$, $\boldsymbol{\sigma}$ è la stima della deviazione standard di $\boldsymbol{x}$, a è il fattore di scala e $\boldsymbol{b}$ è il termine di distorsione. Sia a che $\boldsymbol{b}$ vengono appresi durante l'addestramento.

La batch normalization [5] si applica ad un solo canale dell'input e funziona bene per problemi di computer vision.

Il motivo per cui si utilizzano layer di batch normalization è che in questo modo le reti diventano più facili da ottimizzare, permettendo in questo modo l'utilizzo di valori di learning rate più elevati, in questo modo si riesce a velocizzare l'addestramento della rete.

1.5 CNN

1.5.1 Differenze tra CNN e MLP

Nel 1998 dal gruppo di LeCun vengono introdotte le Convolutional Neural Network (CNN) che hanno avuto grande successo solo di recente grazie all'avvento del big data e allo sviluppo del GPU computing che hanno permesso l'addestramento delle CNN e quindi il loro utilizzo.

A differenza delle reti MLP, le reti CNN hanno i seguenti vantaggi:

- Processing locale: ossia ogni neurone è connesso con solo alcuni dei neuroni del livello successivo (connessioni locali) e non più a tutti i neuroni come nelle reti MLP, questo porta a una notevole riduzione del numero di connessioni.
- Pesi condivisi: le connessioni tra neuroni sono condivise a gruppi, mentre nelle MLP ciascun arco aveva un peso proprio, questo porta ad una notevole riduzione del numero di pesi.

La riduzione del numero di connessioni e di pesi permette di avere reti più leggere e di conseguenza di poterne aumentarne la dimensionalità.

1.5.2 Struttura generale di una CNN

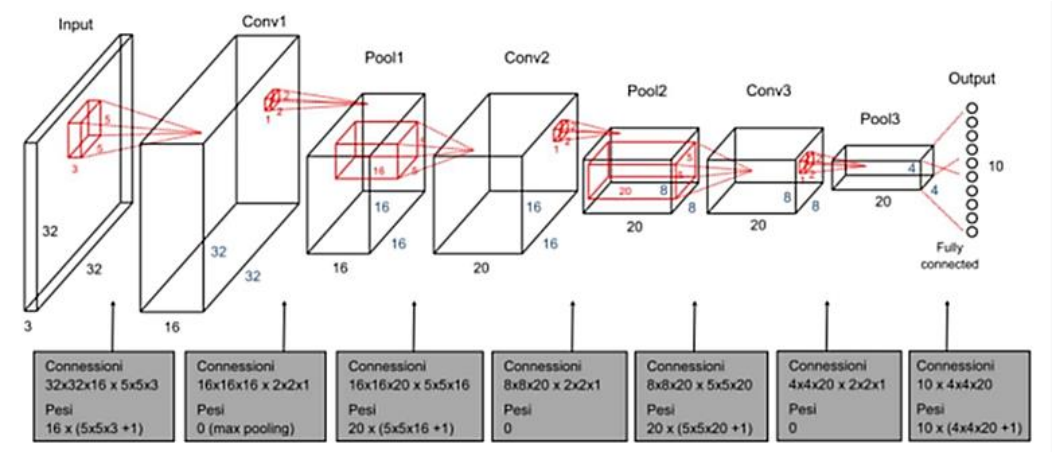

Figura 1.7 - Rappresentazione dell'esempio di una rete CNN

In Figura 1.7 è rappresentato lo schema di una generica rete CNN, in particolare si può vedere che ogni CNN contiene i seguenti strati (o layer):

- Input layer: è collegato direttamente ai pixel dell'immagine di input

- Hidden layers: sono costituiti dall'alternanza di due tipi di layer: convolutional layer e pooling layer e hanno connessioni locali e pesi condivisi
- Fully-connected layer: gli ultimi layer della rete sono generalmente fully-connected e operano come un classificatore MLP
- Output layer: una scelta comune per il livello di output, quando si vuole sviluppare un classificatore, è quella di utilizzare la funzione di attivazione SoftMax

1.5.3 Convolutional layer

Alla base delle CNN troviamo l'operazione di convoluzione con un filtro, in particolare, similmente alle reti classiche, nelle reti CNN i pesi delle connessioni tra neuroni sono dati dai pesi dei filtri presenti nei vari livelli della rete.

Un filtro digitale è una maschera di pesi che viene fatta scorrere su porzioni diverse dell'immagine di input. Il risultato di tale operazione di convoluzione è dato dal prodotto scalare tra il filtro e la porzione dell'immagine (avente la stessa dimensione del filtro), come mostrato in Figura 1.8:

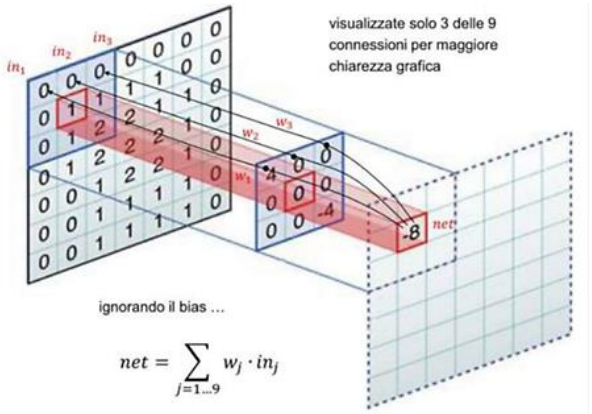

Figura 1.8 - Rappresentazione dell'operazione di convoluzione

Si osservi che i filtri che operano su volumi 3D di input sono rappresentati come dei tensori composti da più strati, detti feature map, dove ogni feature map rappresenta uno specifico filtraggio.

Quando un filtro viene fatto scorrere su un'immagine (o volume) di input, questo può essere fatto scorrere con passo unitario, oppure con uno Stride (o passo) maggiore. Questa operazione riduce la dimensione delle feature map nel volume di output e conseguentemente il numero di connessioni. Sui livelli iniziali della rete per valori piccoli di stride si ha un elevato guadagno in termini di efficienza a discapito di una leggera penalizzazione in accuratezza.

Un'ulteriore possibilità per regolare la dimensione delle feature map è quella di aggiungere un bordo (di valori 0) al volume di input. Con il parametro Padding si denota lo spessore (in pixel) di tale bordo.

Sia W_{out} la dimensione della feature map di output orizzontale/verticale e sia W_{in} la corrispondente dimensione dell'input orizzontale/verticale, sia inoltre F la dimensione del filtro orizzontale/verticale. Vale la seguente relazione:

$$w_{out} = \frac{(W_{in} - F + 2 \cdot \text{Padding})}{\text{Stride}} + 1$$

1.5.4 Pooling layer

Il layer di pooling esegue un'operazione di aggregazione delle informazioni del volume di input, generando in output lo stesso numero di feature map di quelle in input, ma di dimensione inferiore. L'obiettivo è quello di conferire invarianza rispetto a semplici trasformazioni dell'input, mantenendo al tempo stesso le informazioni significative ai fini della discriminazione dei pattern.

Gli operatori di aggregazione più utilizzati sono media (avg) e massimo (max), in quanto entrambi sono piuttosto invarianti per piccole traslazioni. Questo tipo di aggregazione non ha pesi da apprendere.

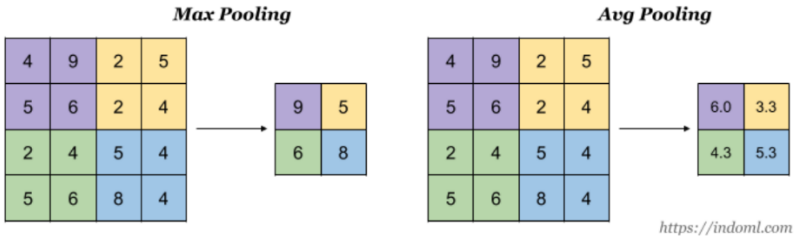

Figura 1.9 - Rappresentazione dell'operazione di pooling

1.6 Transfer Learning

Il training di CNN complesse su dataset di grandi dimensioni è molto dispendioso dal punto di vista temporale, anche se eseguito su GPU, e richiede di avere a disposizione un enorme dataset di dati, cosa che spesso non si ha.

Per fortuna esistono delle tecniche che permettono di effettuare il transfer learning, ossia il riutilizzo di una rete preaddestrata su un certo dataset, per risolvere un dato problema. Esistono due tecniche per effettuare il transfer learning: riutilizzo di features e fine-tuning.

Il riutilizzo di features (o features extraction) consiste nell'estrarre da una rete preaddestrata i livelli intermedi durante il passo forward in modo da utilizzarli come feature vector di input per un altro classificatore (es. SVM).

Il fine-tuning è invece una tecnica più interessante ed è quella che, come descritto meglio nel seguito, è stata utilizzata in questo lavoro per costruire le reti che sono state utilizzate per svolgere la segmentazione del tumore cerebrale. Questa tecnica consiste nel:

- Rimpiazzare il livello di output, che era stato progettato per il problema per cui la rete era stata inizialmente addestrata, con un nuovo livello di output (in genere SoftMax) che si adatti alla dimensione del nuovo problema che si sta affrontando.
- Come valori iniziali dei pesi della rete si utilizzano quelli della rete preaddestrata, tranne che per il fully-connected layer, dove i pesi vengono settati in maniera random.
- Effettuare nuove iterazioni di addestramento per ottimizzare i pesi della rete preaddestrata alle peculiarità del nuovo dataset.

Il fatto di partire da pesi già pre-settati, anziché da zero (pesi random), permette di ottenere un notevole risparmio di tempo durante la fase di addestramento, in quanto la rete utilizzata di solito è stata preaddestrata per un problema simile al nuovo problema che si vuole risolvere, quindi si arriverà molto prima alla convergenza. Perciò questa tecnica permette di ottenere ottimi risultati anche con dataset di dimensioni molto più ridotte rispetto ad un addestramento da zero.

Capitolo 2

Segmentazione semantica

La segmentazione semantica permette di identificare oggetti, con i loro relativi bordi, all'interno di un'immagine o di un volume. Questa tecnica consiste in particolare nell'etichettare ogni pixel di un'immagine 2D, o ogni voxel di un volume 3D, con una determinata classe.

I problemi di segmentazione possono essere principalmente di due tipologie: problemi di segmentazione binaria, consistono nell'identificare un solo oggetto, in genere chiamato foreground, rispetto a tutto il resto presente all'interno dell'immagine, in genere chiamato background; oppure possono essere problemi di segmentazione multi-classe se si vogliono identificare più oggetti appartenenti a classi diverse all'interno dell'immagine.

In questo lavoro di tesi, come accennato precedentemente, vengono proposti diversi approcci per andare ad eseguire la segmentazione semantica binaria di volumi 3D ottenuti mediante scansioni MRI cerebrali, in modo da etichettare come 'tumor' la massa di tumore cerebrale presente all'interno di ciascun volume, andandola a discriminare da tutto il resto (la parte di cervello sana e lo sfondo), che viene etichettato come 'background'.

2.1 Dataset

2.1.1 BraTS

Il dataset BraTS (scaricabile da qui [6]) contiene scansioni di risonanze magnetiche (MRI) di tumori cerebrali, in particolare di gliomi, che sono le neoplasie cerebrali primarie più comuni.

La dimensione del dataset è di circa 110 GB e contiene 750 volumi 4D, di cui 484 sono etichettati e 266 non sono etichettati. Ciascun volume rappresenta una pila di immagini 3D e ha dimensioni $240 \times 240 \times 155 \times 4$, dove le prime tre dimensioni corrispondono rispettivamente ad altezza, larghezza e profondità di un'immagine volumetrica 3D, la quarta dimensione corrisponde invece alle diverse modalità di scansione MRI (o canali) [7]. In

Figura 2.1 è riportato un esempio di rappresentazione grafica 3D di un volume cerebrale del dataset BraTS.

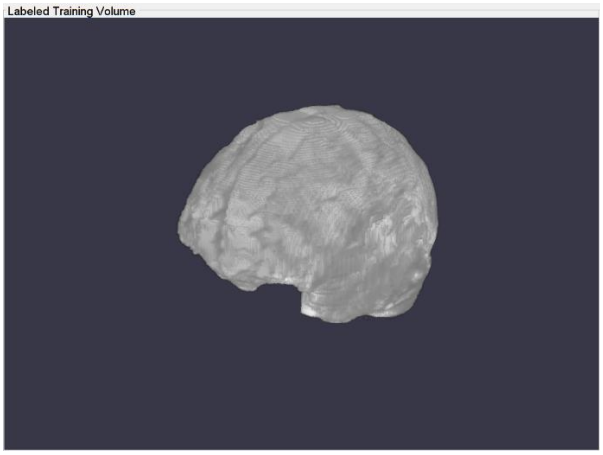

Figura 2.1 - Rappresentazione di un volume 3D del dataset BraTS

2.1.2 Pre-processing dei dati

Per addestrare le reti in maniera più efficiente, i volumi sono stati pre-elaborati utilizzando la funzione di supporto *preprocessBraDataset.m*, presa e descritta nel dettaglio da [7]. Questa funzione esegue le seguenti operazioni:

- Ritaglio delle parti superflue dai volumi e dalle relative etichette, in modo da ridurne la dimensione, pur conservando la parte principale che è quella del cervello.
- Normalizzazione delle modalità di scansione (o canali) di ogni volume in modo indipendente, sottraendo la media e dividendo per la deviazione standard della regione cerebrale ritagliata.
- Suddivisione del data set in training set, validation set e test set, come descritto meglio nella sezione successiva.

2.1.3 Training, Validation e Test sets

In machine learning per le tecniche di apprendimento supervisionato, ossia per quelle tecniche in cui è previsto l'utilizzo di dati etichettati, il dataset deve essere suddiviso in 3 parti:

- Training set: è l'insieme dei pattern che servono per addestrare il sistema.
- Validation set: viene utilizzato durante la fase di progettazione per settare gli iperparametri del sistema e durante la fase di addestramento sul training set per testare le prestazioni e per verificare che il sistema non vada in overfitting. Si parla di overfitting quando il sistema impara a lavorare perfettamente sul training set, ma ottiene prestazioni molto scarse quando va a lavorare su un altro set di dati (si dice che il sistema impara a memoria). Durante la fase di addestramento del

sistema, nel momento in cui le prestazioni misurate sul training set dovessero risultare ottime, ma quelle misurate sul validation set dovessero iniziare a calare, si hanno due soluzioni:
    - Se le prestazioni del sistema sono sufficientemente buone si arresta l'addestramento per evitare che il sistema vada in overfitting.
    - Altrimenti si vanno a modificare gli iperparametri della rete, ossia quei parametri che vengono modificati durante la fase di progettazione della rete e che devono poi rimanere inalterati; dopodiché si va a ripetere l'addestramento finché le prestazioni del sistema non risulteranno accettabili.
- Test set: deve essere utilizzato solo ed esclusivamente per il test finale della rete. È errore purtroppo comune e molto grave andare ad utilizzare il test set per modificare i parametri di rete, migliorando in questo modo, in maniera solo fittizia, le prestazioni in fase di testing del sistema. Così facendo le prestazioni misurate risulteranno falsate e questo errore emergerà solo quando si andrà ad utilizzare la rete in contesti di applicazioni reali.

Per addestrare le reti utilizzate per questo lavoro di tesi, del dataset BraTS, sono stati utilizzati solo i primi 484 volumi etichettati che, dopo essere stati pre-elaborati come descritto nella sezione precedente, sono stati suddivisi come segue:

- Training set: 400 volumi
- Validation set: 29 volumi
- Test set: 55 volumi

Il motivo per cui sono stati utilizzati solo i dati etichettati è che, in questo modo, è possibile andare a misurare le prestazioni ottenute dai diversi approcci di segmentazione proposti in fase di testing, comparando il risultato previsto dalla rete, ottenuto mediante la segmentazione di ciascun volume del test set, con il risultato previsto dal corrispondente volume etichettato, che prende il nome di ground truth.

2.1.4 Patching e slicing dei dati

Un problema che si presenta quando si va ad addestrare una rete neurale profonda su immagini mediche è che spesso le quantità di memoria e le risorse di calcolo, necessarie per archiviare ed elaborare i volumi 3D, non sono sufficienti. L'addestramento di una rete sull'intero volume di input non è pratico a causa dei vincoli imposti dalle risorse limitate della GPU.

Per risolvere questo problema la rete basata su U-Net 3D, utilizzata per implementare il secondo approccio di segmentazione proposto, è stata addestrata a livello di patch (o porzioni) di volume, che vengono estratte in maniera random da ciascun volume e dal corrispondente volume ground truth, come mostrato in Figura 2.2 (dove le patch sono rappresentate in formato bidimensionale, ma in realtà sono dei sotto-volumi tridimensionali

del volume di partenza). Patch diverse possono contenere la stessa regione di volume, quindi, in generale, l'intersezione tra due patch non è vuota.

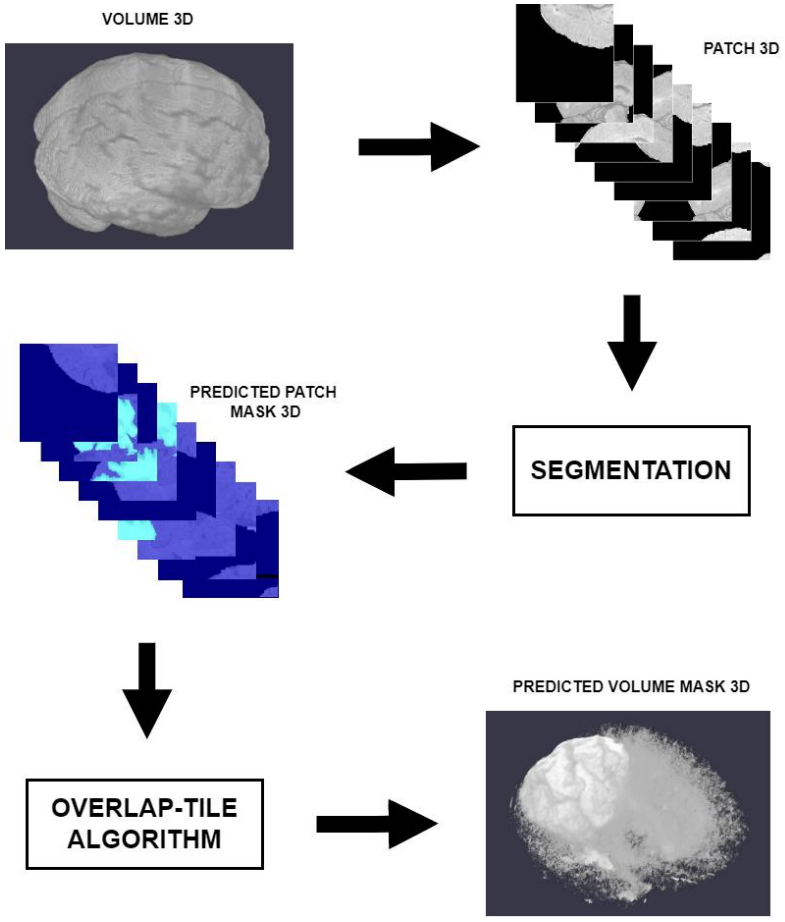

Figura 2.2 - Rappresentazione del patching di un volume 3D

Successivamente viene utilizzato un algoritmo di sovrapposizione delle tessere, preso e spiegato nel dettaglio da [7], che permette di unire le patch segmentate in modo da ottenere il volume di partenza segmentato, come mostrato in Figura 2.2.

Questo problema non si presenta invece per l'addestramento della rete basata su DeepLabV3+, utilizzata per implementare il primo approccio di segmentazione proposto. In questo caso si lavora intrinsecamente su sotto-porzioni del volume di partenza, in particolare si lavora a livello di immagini 2D ottenute mediante deep slicing dei volumi 3D iniziali (estrazione di fette di volume ottenute muovendosi ortogonalmente al piano xy del volume, cioè parallelamente all'asse z), come mostrato in Figura 2.3.

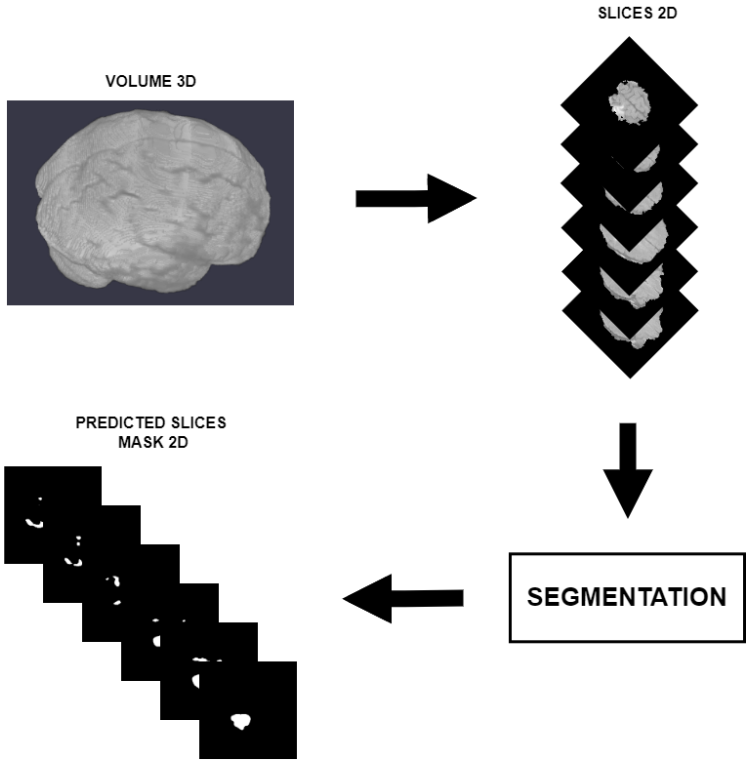

Figura 2.3 – Rappresentazione dello slicing di un volume 3D

Si osservi che sia il patching che lo slicing dei volumi 3D avviene parallelamente per tutti e 4 canali del volume. In particolare, U-Net 3D richiede volumi 3D di ingresso a 4 canali, mentre DeepLabV3+ richiede immagini 2D di ingresso a 3 canali, perciò in questo caso l'ultimo canale viene scartato. In Figura 2.4 è mostrato l'esempio di una slice 2D a 3 canali ottentuta dallo slicing di un volume.

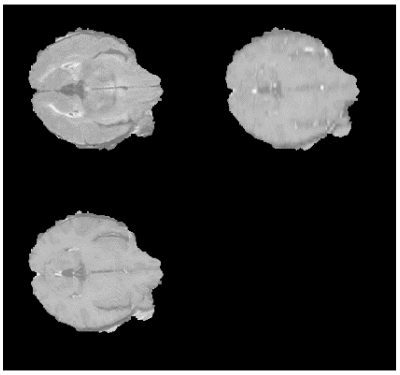

Figura 2.4 - Rappresentazione della slice di volume a 3 canali

È quindi importante osservare che, nonostante si parli di slice o immagini 2D, queste sono in realtà rappresentate mediante dei tensori 3D di dimensioni: larghezza × altezza × numero di canali, allo stesso modo quando si parla di volumi 3D, questi sono in realtà rappresentati mediante dei tensori 4D di dimensioni: larghezza × altezza × profondità × numero canali.

2.1.5 Filtraggio dei dati

Per quanto riguarda il primo approccio di segmentazione proposto, è stata aggiunta qualche riga di codice per implementare il filtraggio delle slice di volume, la cui slice ground truth corrispondente contiene un numero di pixel di foreground inferiore a una certa soglia. Il motivo per cui viene fatto ciò è che, in questo modo, quando si va ad eseguire la segmentazione delle slice di volume si evita di ottenere maschere (slice binarie) in output alla rete con 0 pixel di foreground, questo infatti provocherebbe errori dovuti alle divisioni per 0 nel calcolo di molte loss function, oltre che errori nel calcolo delle metriche di performance delle reti.

2.1.6 Data Augmentation

Le reti neurali profonde devono essere invarianti rispetto a una vasta gamma di variazioni dell'input. Per questo motivo, prima di eseguire l'addestramento della rete, dopo la fase di pre-processing viene spesso eseguita un'ulteriore fase di manipolazione dei dati che è la fase di Data Augmentation, la quale ha come obbiettivo quello di rendere la rete più robusta alle variazioni dell'input.

Per il secondo approccio di segmentazione proposto, dopo la fase di pre-processing, i dati del training set e del validation set subiscono una fase di Data Augmentation attraverso la funzione di supporto *augmentedAndCrop3dPatch.m*, presa e descritta nel dettaglio da [7], la quale esegue le seguenti operazioni:

- Ruota in modo casuale e specchia i dati di training (non quelli di validation) in modo da rendere l'allenamento più robusto.
- Ritaglia le patch in base alla dimensione dell'output di U-Net 3D.

2.2 Architetture per la segmentazione

2.2.1 Architetture encoder-decoder

Le reti neurali completamente convoluzionali (FCN) a differenza delle reti CNN utilizzano come ultimo layer un fully convolutional layer al posto di un fully connected layer. Questo tipo di reti sono state largamente utilizzate per anni con successo per attività di segmentazione semantica. Un problema che però si presenta quando si esegue la segmentazione di immagini utilizzando una FCN è che le feature map in input diventano più piccole man mano che attraversano i vari livelli convoluzionali (a causa dello stride) e di pooling della rete, questo provoca la perdita di informazioni sulle immagini, di conseguenza i risultati della previsione in output risultano essere a bassa risoluzione e i confini degli oggetti risultano essere sfocati [15].

Per risolvere questo problema, in questi ultimi anni, per implementare algoritmi di segmentazione semantica, si è passati ad architetture encoder-decoder che sono così strutturate:

- Nella parte dell'encoder si utilizza una rete neurale backbone (es. ResNet) per eseguire un downsampling (o convoluzione) dell'immagine in input, in modo da ridurre la risoluzione, e quindi la dimensione, delle feature map in input, permettendo tuttavia di distinguere efficacemente le diverse classi di oggetti all'interno dell'immagine con un costo computazionale ridotto.
- Nella parte del decoder viene effettuato un upsampling (o deconvoluzione) delle classi di oggetti individuati all'interno dell'immagine, in modo da riportarle a piena risoluzione nella mappa di segmentazione predetta dalla rete (recupero delle informazioni spaziali).

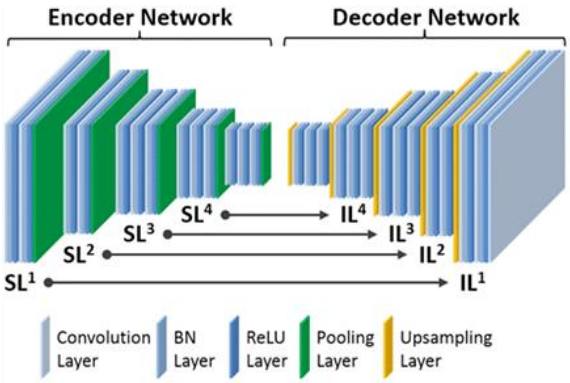

Figura 2.5 - Rappresentazione di un esempio di architettura encoder-decoder

Dalla rappresentazione in Figura 2.5 si può vedere che ogni layer convoluzionale è accoppiato con un layer deconvoluzionale, entrambi i layer hanno la stessa dimensione e la stessa profondità, in questo modo l'immagine restituita dalla rete viene riportata alle stesse dimensioni dell'immagine originale in input [9][10].

Questo tipo di architettura è quella adottata dai più noti algoritmi di segmentazione semantica di immagini, quali DeepLab e U-Net che verranno descritti nel seguito.

2.2.2 Atrous Convolution

Come accennato precedentemente, le reti FCN riducono in maniera significativa la risoluzione e la dimensione delle feature map dell'immagine in ingresso alla rete. Un rimedio a questo problema è quello di utilizzare un ulteriore layer di deconvoluzione (o decoder), cosa che però richiede memoria e tempo aggiuntivo.

Come citato da [12] una soluzione migliore è quella di utilizzare la convoluzione dilatata o Atrous Convolution (introdotta per lo sviluppo dell'architettura DeepLab), dove Atrous sta per "Algoritmo con buchi".

Questo tipo di convoluzione permette di calcolare l'output di qualsiasi layer con qualsiasi risoluzione di campionamento attraverso due parametri (oltre ai classici parametri della

convoluzione stride e padding): kernel, che definisce il campo visivo della convoluzione, e atrous rate, che definisce la spaziatura tra i valori di un dato kernel. In Figura 2.6 è mostrato un esempio di applicazione della convoluzione dilatata per diversi valori dell'atrous rate per un determinato valore fissato del kernel.

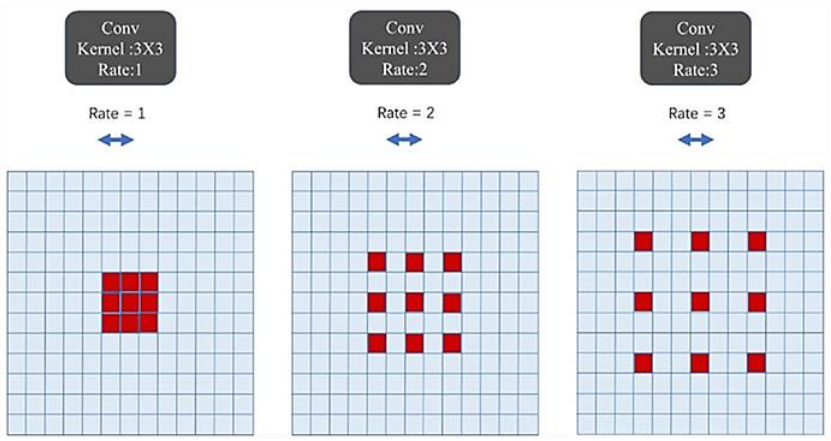

Figura 2.6 - Rappresentazione dell'Atrous Convolution a diversi rate

Questo metodo è migliore rispetto alla convoluzione classica, in quanto, utilizzando un solo layer, si è in grado di ottenere più informazioni dalla stessa immagine semplicemente variando il campo visivo del filtro; inoltre è computazionalmente più veloce.

Nel caso delle reti FCN si possono andare a rimpiazzare gli ultimi layer convolutivi, che di solito tendono a diminuire la risoluzione dell'immagine in output, con una catena di Atrous Convolution layer, in questo modo si riesce ad ottenere un'immagine di output con una risoluzione maggiore rispetto a quella della rete originale. Estendendo questo approccio a tutti i layer convolutivi della rete si riesce invece ad ottenere un'immagine in output avente la stessa risoluzione dell'immagine in input alla rete. A differenza dell'approccio con layer deconvolutivi, questo approccio converte le immagini in feature map a densità maggiore senza richiedere parametri extra e velocizzando considerevolmente la fase di addestramento [11][12].

2.2.3 Spatial Pyramid Pooling (SPP)

Uno Spatial Pyramid Pooling (SPP) layer è un livello di pooling che permette di rimuovere il vincolo di dimensione fissa dell'immagine in input a una rete CNN. Per farlo basta semplicemente andare ad anteporre un layer SPP prima dell'ultimo layer convoluzionale.

Come mostrato in Figura 2.7, il layer SPP esegue più volte l'operazione di max-pooling su scale diverse delle feature map dei livelli convolutivi precedenti, ossia per diversi valori incrementali del kernel. Successivamente ogni operazione di polling viene trasformata in una rappresentazione vettoriale e tali rappresentazioni vettoriali vengono concatenate tra loro per produrre una rappresentazione vettoriale finale a lunghezza fissa.

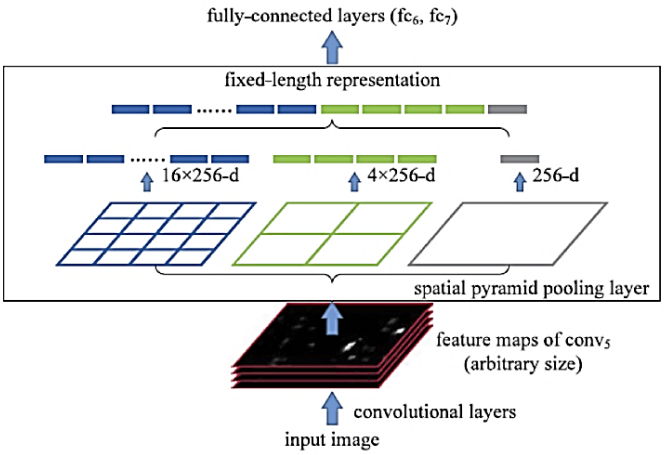

Figura 2.7 - Rappresentazione del metodo SPP

Il layer SPP è quindi in grado di acquisire informazioni su scale diverse dell'immagine e di generare un feature vector multi-scala che può essere utilizzato come input per un layer fully connected (o per altri classificatori) [13][14].

2.2.4 Atrous Spatial Pyramid Pooling (ASPP)

Un altro approccio per costruire una rappresentazione multi-scala di un'immagine è Atrous Spatial Pyramid Pooling (ASPP). Questo approccio, proposto da [12], è ispirato a Spatial Pyramid Pooling (SPP) e permette di costruire una rappresentazione multi-scala utilizzando Atrous Convolution, a diversi rate di campionamento, anziché max-pooling.

Le feature estratte da ogni campionamento vengono successivamente processate e fuse per generare il risultato finale, come mostrato in Figura 2.8.

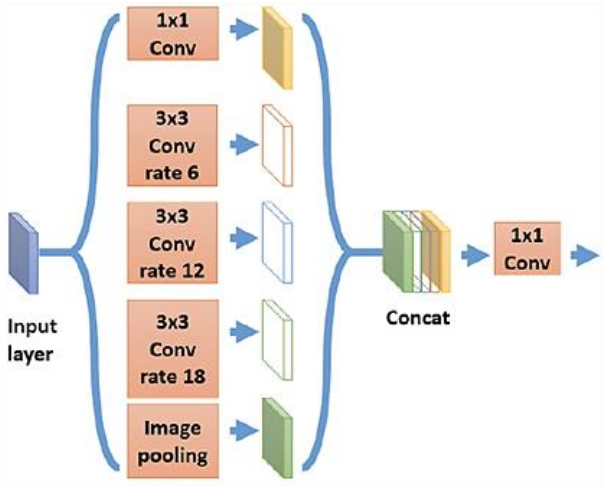

Figura 2.8 - Rappresentazione del metodo ASPP

In questo modo si evita che la risoluzione tra immagine di input e immagine di output venga ridotta, infatti, mentre l'output di SPP era un vettore, l'output di ASPP è una feature map di dimensioni uguali alla feature map del layer convolutivo in input [13].

2.3 Segmentazione 2D: DeepLabV3+

2.3.1 DeepLab

Il modello DeepLab è un'architettura di segmentazione semantica sviluppata da Google che utilizza le Atrous Convolution e i layer ASPP (Atrous Spatial Pyramid Pooling) per ottenere immagini ad alta risoluzione e per fare in modo che i bordi degli oggetti identificati risultino meno sfocati. Questa architettura si è evoluta nel corso delle diverse versioni come segue [15]:

- DeepLabV1: utilizza Atrous Convolution e Fully Connected Conditional Random Fiels (CRF) per controllare la risoluzione alla quale vengono elaborate le feature dell'immagine.
- DeepLabV2: utilizza Atrous Spatial Pyramid Pooling (ASPP) per considerare oggetti su scale e segmenti diversi con una precisione molto migliorata.
- DeepLabV3: oltre all'utilizzo di Atrous Convolution, utilizza un modulo ASPP migliorato, includendo la normalizzazione dei batch, funzionalità a livello di immagine ed elimina il CRF (Conditional Random Fiels).

In questo lavoro di tesi, come già accennato in precedenza, per implementare il primo approccio di segmentazione proposto, è stata utilizzata una versione migliorata dell'architettura DeepLabV3, che è l'architettura DeepLabV3+, che verrà meglio descritta nel seguito.

2.3.2 Architettura DeepLabV3

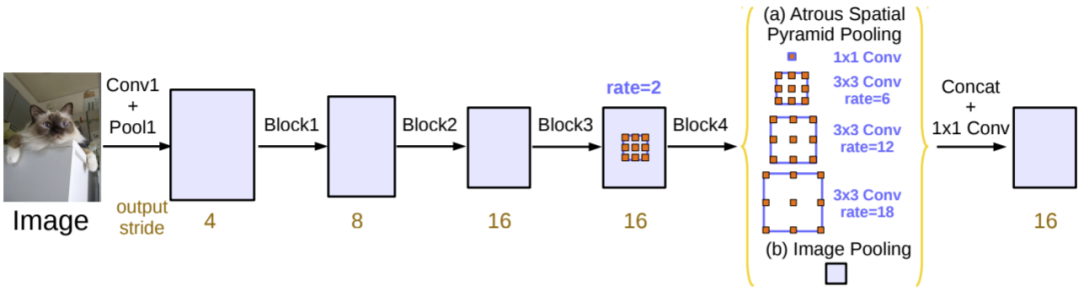

Figura 2.9 - Rappresentazione architettura DeepLabV3

L'architettura di segmentazione semantica DeepLabV3, rappresentata in Figura 2.9, è così strutturata [16][17][18]:

- Le feature map vengono estratte dalla rete backbone.
- L'Atrous Convolution viene utilizzata negli ultimi blocchi della rete backbone per controllare la dimensione delle feature map, in questo modo si riesce a mantenere l'output stride costante e allo stesso tempo ad avere un campo visivo maggiore. Ciò si traduce in una miglior conservazione della risoluzione e della dimensione dell'immagine, senza però dover andare ad aumentare il numero dei parametri o il costo computazionale.

- Oltre alle feature map estratte dalla rete backbone, viene aggiunta una rete ASPP (con batch normalization inclusa), la quale restituisce in output una feature map ottenuta dalla concatenazione dei vari branch in (a) e dalla concatenazione con (b), rappresentati in Figura 2.9, dove questa feature map contiene la previsione della classe di ogni pixel.
- L'output della rete ASPP viene infine fatto passare attraverso una convoluzione 1×1 che ne riduce il numero di canali in modo che la dimensionalità effettiva della maschera segmentata in output risulti uguale alla dimensionalità dell'immagine in input alla rete.

Si noti che in Figura 2.9 viene indicato il parametro output stride che sta ad indicare il rapporto tra le dimensioni della risoluzione dell'immagine in input e le dimensioni della feature map in output al dato layer convolutivo.

2.3.3 Architettura DeepLabV3+

Come visto nella sezione precedente, DeepLabV3 utilizza il modulo ASPP, nel quale sono presenti numerosi livelli paralleli di Astrous Convolution a rate differenti, per catturare informazioni di contesto multi-scala.

D'altra parte però, molte informazioni semantiche legate ai bordi degli oggetti sono codificate nelle ultime feature map e vengono perse a causa delle operazioni di stride all'interno della rete backbone. Questo problema può essere in parte risolto utilizzando l'Atrous Convolution per estrarre feature map più dense. Tuttavia, a causa dei limiti di memoria delle GPU, è computazionalmente proibitivo estrarre feature map con risoluzione 8, o anche 4, volte più piccola di quella dell'immagine in ingresso.

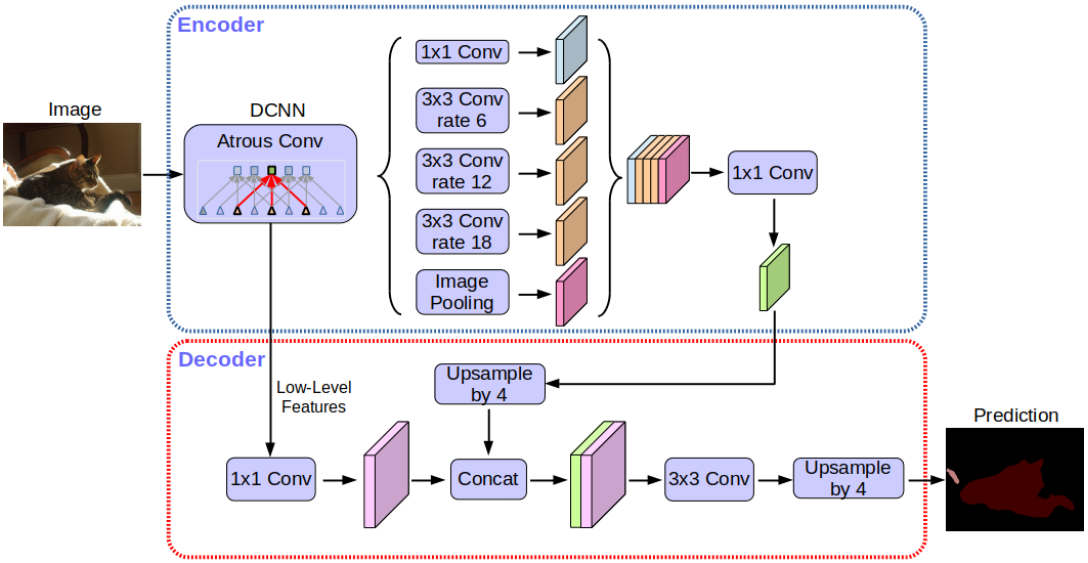

Figura 2.10 - Rappresentazione architettura DeepLabV3+

D'altra parte, i modelli encoder-decoder sono noti per essere computazionalmente più efficienti. Per questo motivo il modello di architettura di segmentazione semantica

DeepLabV3+ combina i vantaggi dei due metodi, andando ad aggiungere un modulo decodificatore che va a migliorare i risultati della segmentazione, specialmente per quanto riguarda i bordi degli oggetti [20].

L'architettura DeepLabV3+, rappresentata in Figura 2.10, è così strutturata [19]:

- L'encoder è formato dalla rete backbone e dal modulo ASPP. L'encoder può ricevere in input immagini di qualsiasi dimensione e viene utilizzato per estrarre le feature map.
- Il decoder viene utilizzato per recuperare il dettaglio del bordo degli oggetti.

2.3.4 ResNet

Secondo l'universal approximation theorem una rete feedforward con un singolo hidden layer è sufficiente a rappresentare qualsiasi funzione. D'altra parte però, l'hidden layer potrebbe risultare enorme e la rete sarebbe incline a overfitting.

Per questo motivo l'architettura di rete ha bisogno di essere più profonda. Tuttavia l'incremento della profondità della rete non funziona solo impilando layer tra loro, in quanto maggiore è la profondità della rete e più difficile ne diventa l'addestramento, a causa del noto problema del vanishing descent. Il vanishing descent si presenta durante la fase di backpropagation del gradiente quando, dopo diverse applicazioni della regola della catena, la moltiplicazione ripetuta può rendere infinitamente piccolo il gradiente della loss function, di conseguenza i pesi rischiano di non venire più aggiornati e la rete smette di imparare. Ciò significa che con un'architettura standard le prestazioni della rete tendono a degradare rapidamente man mano che questa diventa più profonda.

L'intuizione a questo punto è la seguente: sappiamo che reti meno profonde hanno performance migliori di quelle più profonde aventi qualche layer aggiuntivo. Perciò perché non saltare questi layer extra in modo da ottenere un accuracy simile a quella delle corrispondenti reti meno profonde?

L'idea centrale di ResNet è l'introduzione di una cosiddetta "identity shortcut connection", o connessione residua, la quale permette di saltare uno o più layer della rete. Utilizzando le connessioni identità, la rete continua a funzionare allo stesso modo, ma non produce più errori di addestramento maggiori rispetto alle controparti meno profonde.

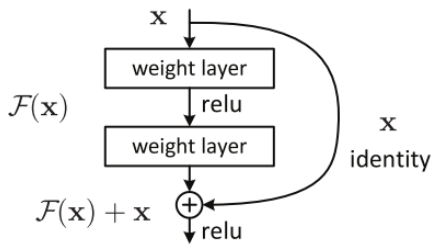

Figura 2.11 - Rappresentazione di un residual block

Per capire perché questo funziona, si consideri un blocco di rete neurale, come quello mostrato in Figura 2.11, avente come input x. Si vuole fare in modo che la rete impari la vera distribuzione $H(x)$.

È noto che le reti neurali riescano ad approssimare molto bene le funzioni, pertanto sono in grado di risolvere facilmente la funzione identità:

$$F(x) = x$$

Seguendo la stessa logica, se bypassiamo l'input del primo layer del blocco con l'output dell'ultimo layer del blocco, attraverso una connessione identità, la rete dovrebbe essere in grado di predire qualsiasi funzione $H(x)$, anche se gli viene sommato l'input x:

$$H(x) = F(x) + x$$

dove $F(x) = H(x) - x$ prende il nome di residuo, da cui questo tipo di blocco prende il nome di blocco residuo. Si osservi che questo tipo di blocchi è in grado di imparare la funzione identità semplicemente impostando $F(x) = 0$.

Tornando alla Figura 2.11, si osserva che: il blocco residuo sta complessivamente cercando di imparare il vero output $H(x)$, mentre i layer interni del blocco stanno cercando di imparare il residuo $F(x)$.

Perciò ricapitolando si ha che i layer di una rete tradizionale imparano il vero output $H(x)$, mentre i layer della rete residua imparano il residuo $F(x)$.

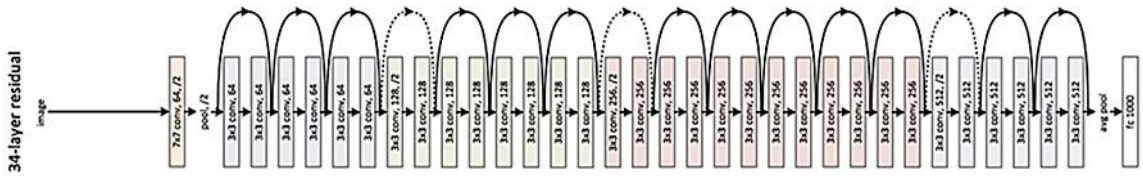

Figura 2.12 - Rappresentazione dell'esempio di una rete ResNet

Le reti ResNet, come quella mostrata in Figura 2.12, sono reti composte da blocchi residui impilati tra loro, anziché da blocchi normali, in questo modo si risolve il problema del vanishing descent. Ciascun blocco residuo effettua salti di 2-3 layer, in questo modo le connessioni residue permettono di propagare gradienti maggiori ai livelli inziali della rete, di conseguenza i layer inziali della rete possono imparare tanto velocemente quanto i layer finali; ciò rende possibile l'addestramento di reti più profonde [21][22][23][24].

Esistono molte varianti e diverse versioni del framework ResNet, ogni versione viene indicata con ResNet-X, dove il numero X è la profondità della rete, ossia il numero di layer da cui questa è composta.

2.3.5 Implementazione architettura per segmentazione 2D

Il primo approccio di segmentazione binaria 2D proposto è stato realizzato mediante transfer learning, in particolare mediante fine-tuning, utilizzando l'architettura DeepLabV3+ che utilizza come rete backbone ResNet-101. I layer della rete sono stati ottenuti mediante la funzione *designDeepLabV3.m* (che a sua volta utilizza la funzione *deeplabv3plusLayersAle.m*).

La rete ResNet-101 è una rete CNN con 101 layer di profondità fornita da Matlab. Questa rete è ottima per la segmentazione semantica di immagini, in quanto è stata pre-addestrata

su più di un milione di immagini del database ImageNet e può classificare oltre 1000 categorie di oggetti [25].

2.4 Segmentazione 3D: U-Net 3D

2.4.1 Architettura U-Net

L'idea alla base delle CNN è quella di apprendere le feature map di un'immagine e sfruttarle per creare una feature map più sfumata. Questo funziona bene nei problemi di classificazione, poiché l'immagine viene convertita in un vettore, ma nella segmentazione semantica non dobbiamo solo convertire le feature map in un vettore, ma dobbiamo anche ricostruire un'immagine a partire da questo vettore.

Questo compito è molto complesso in quanto è molto più difficile convertire un vettore in un'immagine che viceversa. L'intera idea che sta alla base di U-Net ruota attorno a questo problema.

Durante la conversione dell'immagine in vettore viene appresa la mappatura della feature map dell'immagine. L'idea di U-Net è quella di utilizzare la stessa mappatura per riconvertire questa mappatura in immagine, in questo modo si dovrebbe riuscire a preservare l'integrità strutturale dell'immagine, riducendone enormemente la distorsione.

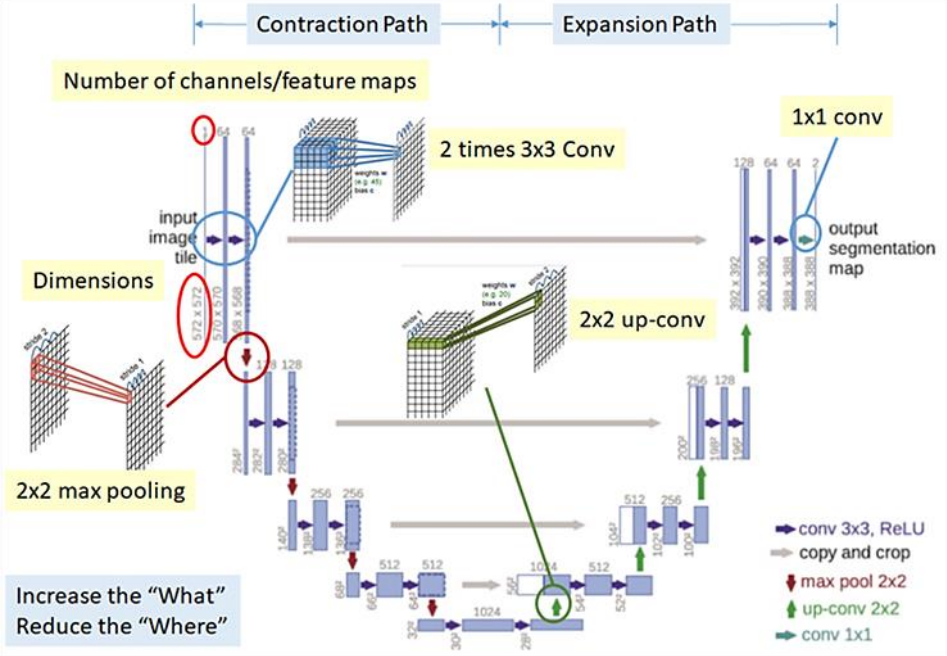

Figura 2.13 - Rappresentazione architettura U-Net

L'architettura di U-Net, rappresentata in Figura 2.13, ha la forma di una "U", da cui prende il nome. Questa architettura è costituita da tre sezioni:

- Sezione di contrazione: è composta da molti blocchi di contrazione, ogni blocco accetta un input, applica due layer di convoluzione 3 × 3, ciascuno seguito da una ReLU, e applica un layer max-pooling 2 × 2 per il down-sampling.
  Dopo ogni blocco il numero di feature map raddoppia, in questo modo l'architettura è in grado di apprendere strutture complesse.
- Sezione bottleneck: agisce da mediatrice tra la sezione di contrazione e la sezione di espansione. In questa sezione sono presenti un layer di max-pooling 2 × 2, due layer di convoluzione 3 × 3 e un up convolution layer 2 × 2.
- Sezione di espansione: similmente alla sezione di contrazione, consiste in numerosi blocchi di espansione, ogni blocco passa l'input a due layer di convoluzione 3 × 3 seguiti da due up convolution layer 2 × 2 per l'up-sampling. Dopo ogni blocco il numero di feature map si dimezza, in modo da mantenere la simmetria. Tuttavia, alle feature map ottenute ad ogni livello della sezione di espansione, vengono aggiunte in coda le feature map del corrispondente livello di contrazione. Questo fa in modo che le feature apprese durante la contrazione dell'immagine vengano utilizzate per ricostruirla.

Per simmetria, il numero di blocchi di espansione è uguale al numero di blocchi di contrazione [26].

2.4.2 Architettura U-Net 3D

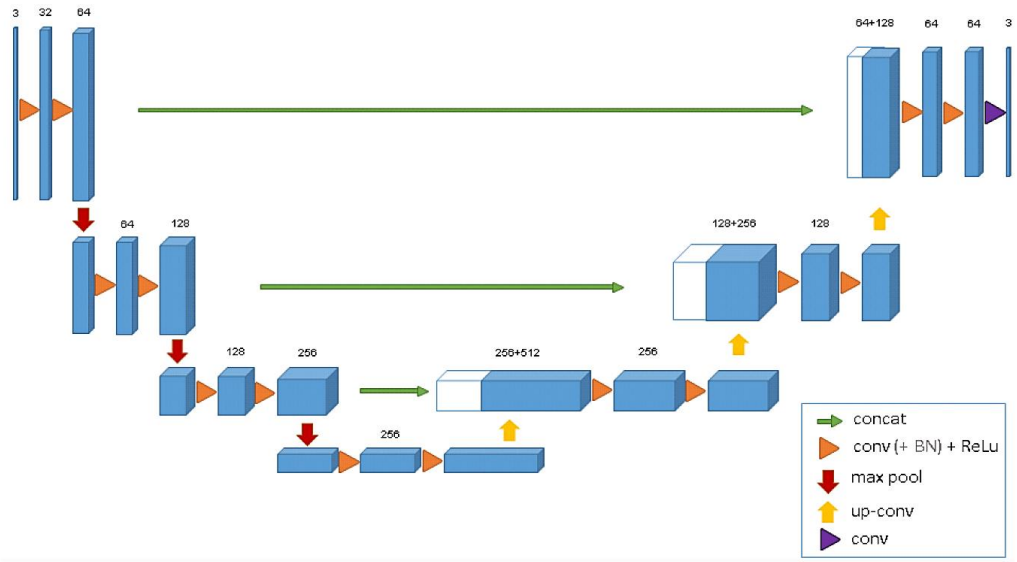

Figura 2.14 - Rappresentazione architettura U-Net 3D

L'architettura U-Net 3D, mostrata in Figura 2.14, è abbastanza simile a quella di U-Net, con l'unica differenza che in questo caso la rete è strutturata per eseguire la segmentazione di volumi anziché di immagini. Perciò, come per U-Net, vediamo che anche in questo caso sono presenti le tre medesime sezioni:

- Sezione di contrazione: in questa sezione viene eseguita l'analisi del volume di ingresso. Ogni blocco di contrazione contiene due layer convolutivi $3 \times 3 \times 3$, ciascuno seguita da una ReLU, e un layer max-pooling $2 \times 2 \times 2$ con stride di 2 per ogni dimensione.
- Sezione bottleneck: questa sezione agisce da mediatrice tra la sezione di contrazione e la sezione di espansione. In questa sezione sono presenti un layer di max-pooling $2 \times 2 \times 2$, due layer di convoluzione $3 \times 3 \times 3$ e un up convolution layer $2 \times 2 \times 2$.
- Sezione di espansione: in questa sezione viene eseguita la sintesi del volume di ingresso. Ogni blocco di espansione contiene un up-convolution layer $2 \times 2 \times 2$ con stride di 2 per ciascuna dimensione, e due layer convolutivi $3 \times 3 \times 3$, ciascuno seguita da una ReLU. Come per U-Net, alle feature map ottenute ad ogni livello della sezione di espansione, vengono aggiunte in coda le feature map del corrispondente livello di contrazione. Questo fa in modo che le feature apprese durante la contrazione dell'immagine vengano utilizzate per ricostruirla.
  Infine è presente un ultimo layer di convoluzione $1 \times 1 \times 1$ che riduce il numero di canali in uscita a 3.

In ogni layer convolutivo, prima del layer della funzione di attivazione ReLU, è presente un layer di batch normalization.

Come per U-Net anche in questo caso, per simmetria, il numero di blocchi di espansione è uguale al numero di blocchi di contrazione [27][28].

2.4.3 Implementazione architettura per segmentazione 3D

Il secondo approccio di segmentazione binaria 3D proposto è stato realizzato utilizzando l'architettura U-Net 3D. I layer della rete sono stati ottenuti mediante la funzione *unet3dLayers.m* fornita dal Computer Vision Toolbox di Matalab.

Questa rete è ottima per la segmentazione semantica di volumi ed è molto utilizzata per la segmentazione volumetrica di immagini mediche.

Si noti che in questo caso, a differenza del primo approccio proposto, non si ricorre a fine-tuning, quindi in questo caso la rete ottenuta deve essere addestrata da zero.

2.5 Ensemble: architettura e implementazione

Il terzo approccio di segmentazione binaria 3D proposto è stato ottenuto dall'ensemble dei primi due approcci di segmentazione 'weak' che sono stati descritti precedentemente, rispettivamente ai capitoli 2.3.5 e 2.4.3.

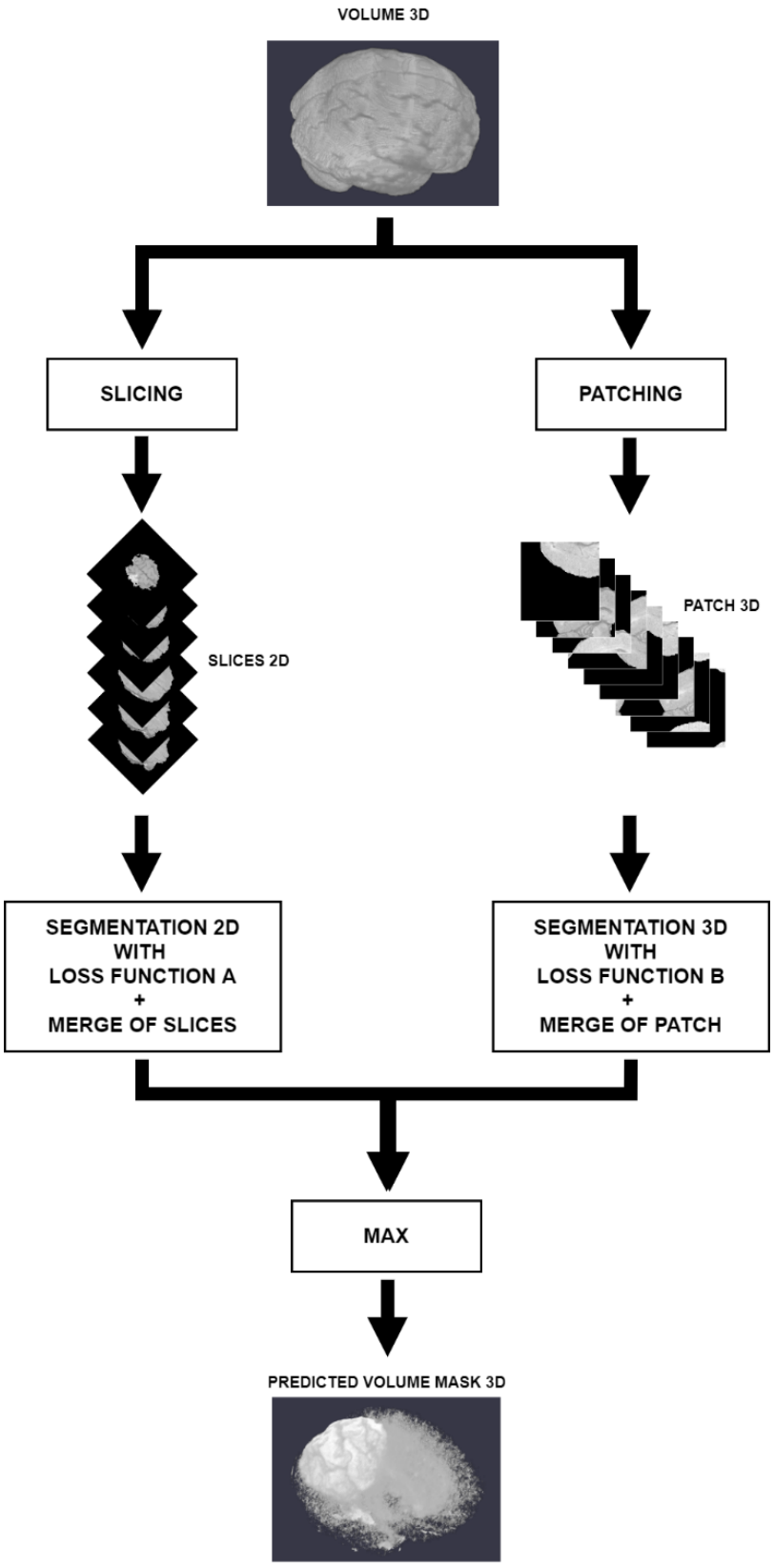

Figura 2.15 - Rappresentazione ensemble per segmentazione 3D

In Figura 2.15 è rappresentato lo schema complessivo dell'ensemble, il cui funzionamento è il seguente:

- In input troviamo i volumi 3D del dataset BraTS, ottenuti mediante MRI cerebrale.
- Ciascun volume viene diviso indipendentemente in slice e in patch, come descritto nel capitolo 2.1.4.
- Le slice vengono processate e segmentate attraverso il primo approccio di segmentazione 2D 'weak', la cui rete utilizza una certa loss function A. Dopo aver effettuato il merge delle slice viene restituita in output la maschera 3D della segmentazione predetta.
- Le patch vengono processate e segmentate attraverso il secondo approccio di segmentazione 3D 'weak', la cui rete utilizza una certa loss function B. Dopo aver effettuato il merge delle patch viene restituita in output la maschera 3D della segmentazione predetta.
- Le due previsioni ottenute vengono fuse insieme mediante il criterio del massimo che restituisce in output una maschera 3D, la quale altro non è che la sovrapposizione (o unione) delle due previsioni 'weak'.

Si osservi che il test set utilizzato per testare l'ensemble è stato ottenuto mediante il pre-processing dei dati, come descritto nel capitolo 2.1.2.

2.6 Metriche di valutazione delle performance

2.6.1 TP, TN, FP, FN

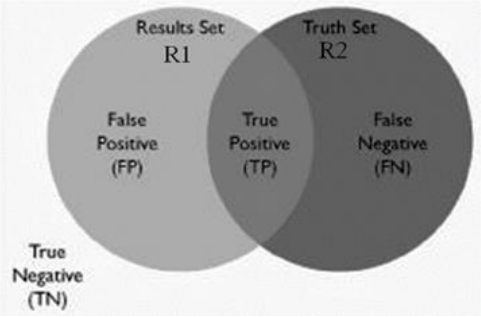

Figura 2.16 - Rappresentazione TP, TN, FP, FN

In questa sezione viene mostrato come vengono calcolati in pratica i noti coefficienti TP, TN, FP e FN, rappresentati in Figura 2.16, che sono utilizzatissimi in machine learning e che saranno utili nel seguito per andare a definire le metriche di misura delle performance degli algoritmi di segmentazione:

- True Positive (TP):

$$TP = \sum_i \sum_j (gt(i,j) == 1 \;\&\; pred(i,j) > 0)$$

- True Negative (TN):

$$TN = \sum_i \sum_j (gt(i,j) == 0 \,\&\, pred(i,j) == 0)$$

- False Positive (FP):

$$FP = \sum_i \sum_j (gt(i,j) == 0 \,\&\, pred(i,j) > 0)$$

- False Negative ():

$$FN = \sum_i \sum_j (gt(i,j) == 1 \,\&\, pred(i,j) == 0)$$

2.6.2 IoU score e Dice score

Per misurare le prestazioni di un algoritmo di segmentazione la metrica più immediata sembra essere l'accuracy, che è definita semplicemente come la percentuale di pixel correttamente classificati ed è ottenuta confrontando i pixel della maschera dell'immagine predetta dal dato algoritmo di segmentazione con i pixel della maschera ground truth. Il motivo per cui questa non è una buona metrica per la segmentazione è dovuto al problema delle classi fortemente sbilanciate. Questo problema si presenta quando una classe occupa una porzione molto grande dell'immagine, mentre le altre classi occupano porzioni molto più ridotte; ciò accade molto di frequente in questo tipo di problemi.

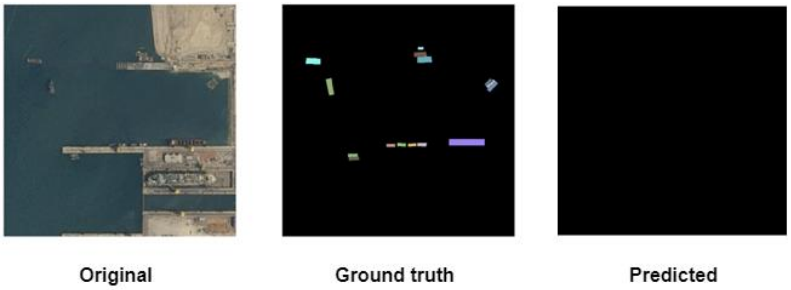

Figura 2.17 - Rappresentazione esempio di cattiva segmentazione

In Figura 2.17 è mostrato un esempio di cattiva segmentazione. In questo caso se si va a misurare l'accuracy tra la maschera predetta e la maschera ground truth si otterrà sicuramente un accuracy nell'ordine del 95%, anche se è evidente che tale predizione non è affatto buona [29].

Le metriche che si utilizzano per valutare le performance di un algoritmo di segmentazione semantica sono principalmente due: IoU score e Dice score. Queste metriche rappresentano approcci leggermente diversi per misurare quanto sono simili i risultati prodotti dall'algoritmo di segmentazione e quelli delle corrispondenti immagini ground truth.

La metrica Intersection over Union (IoU), nota anche come indice di Jaccard, è definita come l'area di intersezione tra la maschera prevista dalla rete e la maschera ground truth, divisa per l'unione delle due aree:

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|} \equiv \frac{|A \cap B|}{|A| + |B| - |A \cap B|} = \frac{TP}{TP + FP + FN}$$

dove l'indice di Jaccard assume valore minimo 0 per indicare che non vi è alcuna area di sovrapposizione e valore massimo 1 per indicare una segmentazione perfettamente sovrapposta. Nel caso di problemi di segmentazione multi-classe l'IoU score è dato dalla media aritmetica dell'IoU score di ogni classe.

La metrica Dice score, o coefficiente di Sørensen-Dice, è definita come 2 volte l'area di intersezione tra la maschera prevista dalla rete e la maschera ground truth, divisa per il numero totale di pixel tra le due immagini:

$$Dice(A,B) = \frac{2|A \cap B|}{|A| + |B|} = \frac{2TP}{2TP + FP + FN}$$

I due coefficienti IoU e Dice sono correlati positivamente, perciò anche in questo caso il Dice score assume valore tra 0 e 1, dove 1 indica la massima somiglianza tra predizione e ground truth. Nel caso di problemi di segmentazione multi-classe il Dice score è dato dalla media aritmetica del Dice score di ogni classe.

Per calcolare mIoU (IoU medio) e mDice (Dice score medio) sull'intero test set basta semplicemente calcolare la media aritmetica dei rispettivi coefficienti calcolati per ogni pattern del test set.

Capitolo 3

Esperienza svolta

Tutto il codice sviluppato per questo lavoro di tesi per: elaborare i dati del dataset BraTS, implementare i tre approcci di segmentazione 3D proposti, eseguire i training e i test delle reti, implementare le loss function utilizzate per l'addestramento delle reti e il codice delle funzioni di supporto utilizzate, sono state scritte in MATLAB.

MATLAB (matrix laboratory) è un linguaggio di programmazione multi-paradigma proprietario e un'ambiente di calcolo sviluppato da MathWorks. Questo linguaggio consente la manipolazione di matrici, la tracciatura di funzioni e di dati, l'implementazione di algoritmi, la creazione di interfacce utente e l'interfacciamento con programmi scritti in altri linguaggi di programmazione [30].

Nonostante MATLAB sia destinato principalmente al calcolo numerico, esso mette a disposizione tantissime altre funzionalità attraverso toolbox aggiuntivi. Per questo lavoro di tesi sono stati utilizzati i seguenti toolbox:

- Deep Learning Toolbox e Statistics and Machine Learning Toolbox per andare a lavorare con modelli di deep learning
- Computer Vision Toolbox e Image Processing Toolbox per eseguire la segmentazione semantica e l'elaborazione del dataset
- Parallel Computing Toolbox per eseguire l'addestramento delle reti utilizzando la potenza di calcolo della GPU, anziché quella della CPU

3.1 Dati, Training e Test

In questa sezione viene descritto schematicamente il funzionamento dei tre script principali che sono stati sviluppati per implementare i tre approcci di segmentazione proposti:

Main_2D_segmentation.mlx: implementa il primo approccio di segmentazione 2D 'weak' proposto; il suo funzionamento è il seguente:

- Definizione dei percorsi delle cartelle, degli iperparametri e delle variabili/flag del programma

- Preprocessing del dataset, slicing del dataset, filtraggio dei dati e creazione di training set, validation set e test set, mediante la funzione di supporto sviluppata *createSliceSets.m*, che a sua volta utilizza la funzione di supporto fornita *preprocessBraDataset.m* (vedi sezioni 2.1.1, 2.1.2, 2.1.3, 2.1.4 e 2.1.5)
- Training della rete (ciclo for: per ogni loss function):
    - Definizione della loss function da utilizzare (vedi sezione 3.3)
    - Costruzione di una nuova rete mediante la funzione di supporto fornita *designDeepLabV3.m* (vedi sezione 2.3.5)
    - Modifica del primo layer della rete per adattarlo alla dimensione dell'immagine che si vuole segmentare, modifica dell'ultimo layer della rete per utilizzare la loss function definita (vedi sezione 1.6)
    - Definizione dei parametri di addestramento (vedi sezione 3.2)
    - Addestramento della rete
    - Salvataggio della rete addestrata
- Test della rete (ciclo for: per ogni loss function):
    - Definizione della loss function
    - Caricamento della rete preaddestrata che utilizza la loss function definita
    - Segmentazione delle slice dei volumi di test utilizzando la rete caricata
    - Calcolo e salvataggio degli score mDice e mIoU

Main_3D_segmentation.mlx: implementa il secondo approccio di segmentazione 3D 'weak' proposto; il suo funzionamento è il seguente:

- Definizione dei percorsi delle cartelle, degli iperparametri e delle variabili/flag del programma
- Preprocessing del dataset e creazione di training set, validation set e test set, mediante la funzione di supporto fornita *preprocessBraDataset.m* (vedi sezioni 2.1.1, 2.1.2, 2.1.3 e 2.1.6)
- Patching di training set, validation set e test set (vedi sezione 2.1.4)
- Training della rete (ciclo for: per ogni loss function):
    - Definizione della loss function da utilizzare (vedi sezione 3.3)
    - Costruzione di una nuova rete mediante la funzione di supporto fornita *unet3dLayers.m* (vedi sezione 2.4.3)
    - Modifica del primo layer della rete per adattarlo alla dimensione del volume che si vuole segmentare, modifica dell'ultimo layer della rete per utilizzare la loss function definita (vedi sezione 1.6)
    - Definizione dei parametri di addestramento (vedi sezione 3.2)
    - Addestramento della rete
    - Salvataggio della rete addestrata
- Test della rete (ciclo for: per ogni loss function):
    - Definizione della loss function
    - Caricamento della rete preaddestrata che utilizza la loss function definita

- Segmentazione dei volumi di test a livello di patch, utilizzando la rete caricata e l'algoritmo di sovrapposizione delle tessere per ricostruire l'intero volume segmentato (vedi sezione 2.1.4)
- Calcolo e salvataggio degli score mDice e mIoU

Main_ensemble_3D_segmentation.mlx: implementa il terzo approccio di segmentazione 3D proposto, ottenuto mediante ensemble dei due approcci 'weak' precedenti; il suo funzionamento è il seguente:

- Definizione dei percorsi delle cartelle e delle variabili/flag del programma
- Caricamento del test set (quello creato da *Main_3D_segmentation.mlx*)
- Slicing e patching del test set
- Test dell'ensemble (ciclo for: per ogni coppia di loss function (A, B)):
  - Definizione della loss function A per la prima rete 'weak'
  - Definizione della loss function B per la seconda rete 'weak'
  - Caricamento della prima rete preaddestrata 'weak' che utilizza la loss function A definita
  - Caricamento della seconda rete preaddestrata 'weak' che utilizza la loss function B definita
  - Segmentazione delle slice dei volumi di test, utilizzando la prima rete caricata e merge delle slice per ricostruire l'intero volume segmentato
  - Segmentazione delle patch dei volumi di test, utilizzando la seconda rete caricata e algoritmo di sovrapposizione delle tessere per ricostruire l'intero volume segmentato
  - Unione, per ogni volume di test, delle due maschere predette, ottenute dalla segmentazione a livello di slice e dalla segmentazione a livello patch, mediante il criterio del massimo (vedi sezione 2.5)
  - Calcolo e salvataggio degli score mDice e mIoU

In Figura 3.1 è rappresentato il confronto tra un'immagine predetta, ottenuta mediante segmentazione durante la fase di test, e la corrispondente immagine ground truth:

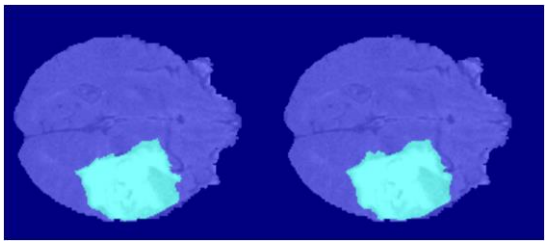

Figura 3.1 - Rappresentazione del confronto tra immagine predetta e ground truth

Nel seguito, nella sezione 3.2 verranno descritti i parametri di addestramento che sono stati utilizzati nei due script *Main_2D_segmentation.mlx* e *Main_3D_segmentation.mlx* per addestrare le reti; nella sezione 3.3 verranno invece descritte tutte le loss function

implementate e testate che sono state utilizzate per addestrare le reti nei due script corrispondenti.

3.2 Parametri di addestramento

Primo approccio 'weak' proposto per segmentazione 2D:

- inputSize: 240
- threshold: 10
- maxDataToUse: 484
- reteBackbone: ResNet-101
- MaxEpochs: 10
- InitialLearnRate: 0.001
- MiniBatchSize: 10
- Optimizer: 'sgdm':
- Momentum: 0.9
- LearnRateSchedule: piecewise
- LearnRateDropPeriod: 5
- LearnRateDropFactor: 0.2
- L2Regularization: 0.005

Secondo approccio 'weak' proposto per segmentazione 3D:

- inputSize: 132
- patchPerImage: 132
- maxDataToUse: 484
- MaxEpochs: 20
- InitialLearnRate: $5 \cdot 10^{-4}$
- MiniBatchSize: 5
- Optimizer: 'sgdm'
- Momentum: 0.9
- LearnRateSchedule: piecewise
- LearnRateDropPeriod: 5
- LearnRateDropFactor: 0.95

3.3 Loss function testate

3.3.1 Dice Loss

Il coefficiente Dice, descritto nel capitolo 2.6.2, è ampiamente utilizzato per calcolare la similarità tra due immagini, in particolare, date due immagini Y (prevista dalla rete) e *T*

(ground truth), restituisce un valore nell'intervallo $[0,1]$ che è tanto maggiore quanto maggiore è la similarità tra le due immagini.

$$Dice(Y,T) = \frac{2|Y \cap T|}{|Y| + |T|}$$

Il coefficiente Dice può anche essere utilizzato come loss function andandolo a ridefinire come segue [31]:

$$L_{Dice}(Y,T) = 1 - Dice(Y,T) = 1 - \frac{2|Y \cap T| + \epsilon}{|Y| + |T| + \epsilon}$$

dove è stata aggiunta una piccola costante ϵ a numeratore e a denominatore per gestire i casi limite in cui $Y = T = 0$.

3.3.2 Binary Cross-Entropy

La Binary Cross Entropy deriva dalla distribuzione di Bernoulli, in particolare è definita come misura della differenza tra due distribuzioni di probabilità. Questa loss è ampiamente utilizzata per la classificazione di oggetti e lavora molto bene per la segmentazione come pixel level classification.

La Binary Cross-Entropy è definita come segue [31]:

$$L_{BCE}(Y,T) = -(T \log(Y) + (1-T)\log(1-Y))$$

dove Y è il valore predetto dalla rete e T è il valore ground truth.

3.3.3 Weighted Cross-Entropy

La Weighted Cross-Entropy è una variante della Binary Cross Entropy, in cui gli esempi positivi vengono pesati da un certo coefficiente β. Questa loss è ampiamente utilizzata in caso di dati distorti.

La Weighted Cross-Entropy è definita come segue [31]:

$$L_{W-BCE}(Y,T) = -(\beta \cdot T \log(Y) + (1-T)\log(1-Y))$$

Si noti che il coefficiente β può essere utilizzato per regolare i FN e i FP. Per esempio, per ridurre il numero di FN basta settare $\beta > 1$, similmente per ridurre il numero di FP basta settare $\beta < 1$.

3.3.4 Balanced Cross-Entropy

La Balanced Cross-Entropy è simile alla Weighted Cross-Entropy, con l'unica differenza che, oltre agli esempi positivi, vengono pesati anche gli esempi negativi.

La Balanced Cross-Entropy è definita come segue [31]:

$$L_{B-BCE}(Y,T) = -(\beta \cdot T \log(Y) + (1-\beta) \cdot (1-T)\log(1-Y))$$

dove in questo caso β è definito come:

$$\beta = 1 - \frac{1}{|T|}\sum_{t \in T} t$$

dove $t \in T$ è il valore di ogni singolo pixel all'interno di T.

3.3.5 Focal Loss

La Focal Loss può essere vista come una variante della Binary Cross-Entropy, in particolare pesa meno gli esempi facili, consentendo al modello di concentrarsi maggiormente sull'apprendimento di esempi difficili. Questa loss funziona bene nei casi in cui si hanno classi fortemente sbilanciate.

Per capire meglio come venga ottenuta la Focal Loss, diamo la seguente definizione di Binary Cross-Entropy, equivalente a quella data nella sezione 3.3.2:

$$CE(Y,T) = \begin{cases} -\log Y & \text{se } T = 0 \\ -\log(1-Y) & \text{altrimenti} \end{cases}$$

Per semplificare la notazione definiamo la stima di probabilità di classe come:

$$Y_t = \begin{cases} Y & \text{se } T = 0 \\ 1 - Y & \text{altrimenti} \end{cases}$$

in questo modo possiamo riscrivere la Cross-Entropy come:

$$CE(Y,T) = CE(Y_t) = -\log(Y_t)$$

A questo punto è possibile definire la Focal Loss come segue [31]:

$$L_{FL}(Y_t) = -\alpha_t (1 - Y_t)^{\gamma} \log(Y_t)$$

dove $\gamma > 0$; in particolare per $\gamma = 1$ la Focal Loss lavora come la Cross-Entropy. Similmente $\alpha \in [0,1]$ e può essere impostato come l'inverso della frequenza della classe o trattato come iperparametro.

3.3.6 Tversky Loss

L'indice Twersky può essere visto come una generalizzazione del coefficiente Dice, in particolare aggiunge un peso β ai FP e ai FN.

$$Twersky(Y,T) = \frac{YT}{YT + \beta(1-Y)T + (1-\beta)Y(1-T)}$$

dove osserviamo che per $\beta = 1/2$ otteniamo il coefficiente Dice.

Similmente al coefficiente Dice, l'indice Twersky può essere utilizzato come loss function andandolo a ridefinire come segue [31]:

$$L_{Twersky}(Y,T) = 1 - \frac{1 + YT}{1 + YT + \beta(1-Y)T + (1-\beta)Y(1-T)}$$

3.3.7 Focal Twersky Loss

Similmente alla Focal Loss che si concentra sugli esempi difficili andando a pesare meno gli esempi più facili/comuni, la Focal Twersky Loss cerca di imparare esempi difficili aventi per esempio piccole ROIs (region of interest) con l'aiuto del coefficiente γ.

La Focal Twersky Loss è definita come segue [31]:

$$L_{FT}(Y,T) = \sum_{c} \left(1 - Twersky_c(Y,T)\right)^{\gamma}$$

dove $\gamma \in [1,3]$.

3.3.8 Focal Dice Loss

La Focal Dice Loss è stata sviluppata proprio per la segmentazione del tumore cerebrale. Prima di vedere la definizione della Focal Dice Loss bisogna però andare a definire alcuni termini utili.

Sia B la maschera binaria predetta della rete e T il volume ground truth. Si definisce Hard Dice Coefficient il coefficiente:

$$D_H = \frac{2|B \cap T|}{|B| + |T|}$$

dove questo coefficiente è definito solo per dati binari.

Si consideri la probabilità stimata per il volume predetto Y, il quale è composto da N voxel. Ogni elemento $y_i \in [0,1]$, con $y_i \in Y$, mentre ogni elemento $g_i \in \{0,1\}$, con $g_i \in G$. Si definisce Soft Dice Coefficient il coefficiente:

$$D_S = \frac{2\sum_{i=1}^{N} p_i g_i + \epsilon}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} g_i + \epsilon}$$

dove ϵ è un termine per la stabilità numerica che evita le divisioni per 0 ed evita di ottenere valori troppo piccoli.

L' Hard Dice Coefficient D_H è utile per assestare il risultato finale, mentre il Soft Dice Coefficient D_S è necessaria durante la fase di ottimizzazione, dove è d'obbligo avere una funzione differenziabile.

Si definisce Soft Dice Coefficient Loss:

$$L_{D_S} = 1 - D_S$$

Tutte queste definizioni sono valide per il caso binario, ma possono essere facilmente generalizzate per $t = 1, \dots, k$ classi. Per farlo basta calcolare la L_{D_S} binaria per ogni valore di t, in questo modo il Soft Multi-Dice Coefficient Loss pesato può essere scritto come:

$$L_{MD_S} = \sum_{t=1}^{k} \omega_t L_{D_{S_t}}$$

dove $\omega_t \geq 0$ è il parametro pesato per la classe t-esima e $L_{D_{S_t}}$ è la Soft Dice Loss binario per la classe t. Un modo comune per impostare i parametri ω_t è quello di sfruttare il logaritmo dell'inverso della frequenza della classe.

A partire da queste definizioni, la Focal Dice Loss può essere definita come segue [32]:

$$L_{FD}^{W} = \sum_{t=1}^{k} \omega_t \left(1 - D_{S_t}^{\frac{1}{\gamma}}\right)$$

dove D_{S_t} è il Soft Dice Coefficient per la classe t e il fattore $1/\gamma$, con $\gamma \geq 1$, è applicato all'esponente del D_{S_t} per ogni classe.

Stando a quanto riportato dagli autori di questa loss, questa formulazione ha tre proprietà:

- Questa loss in pratica non è influenzata dai pixel che vengono classificati in maniera scorretta come classe t con un valore di D_{S_t} alto. Al contrario la loss decresce significativamente quando D_{S_t} è piccolo.
- Il rate con il quale le classi segmentate correttamente è pesato di meno è governato dal parametro γ. Per $\gamma = 1$ si ha che L_{FD} corrisponde a L_{MD_S}. All'aumentare di γ la rete è forzata a focalizzarsi sulle classi con una cattiva segmentazione.
- Poiché L_{FD} è basata sulla definizione di coefficiente Dice, si suggerisce di utilizzare tale metrica per valutarne le prestazioni.

Si noti che, se si vuole approfondire la Focal Dice Loss, nel paper [32] vengono riportate altre due versioni modificate con relativa analisi del funzionamento e delle prestazioni ottenute.

3.3.9 Log-Cosh Dice Loss

La Log-Cosh Dice Loss permette di sfruttare le caratteristiche della Dice Loss rendendola più trattabile.

La Dice Loss, a causa della sua natura non convessa, potrebbe non riuscire a raggiungere il risultato ottimo. Altre loss function, come Lovasz-Softmax Loss, mirano ad affrontare questo problema aggiungendo lo "smoothing" attraverso l'estensione Lovasz.

L'approccio Log-Cosh è ampiamente utilizzato in deep learning nei problemi di regressione per smussare la curva. Le funzioni iperboliche, come $\cosh x$, introducono la non linearità e sono facilmente differenziabili. Il problema del $\cosh x$ è che può crescere fino ad infinito, per evitare ciò tale funzione viene incapsulata all'interno della funzione $\log y$ in modo da ottenere una funzione risultante continua, finita e differenziabile.

La Log-Cosh Dice Loss è definita come segue [31]:

$$L_{LC-Dice}(Y, T) = \log\left(\cosh\left(L_{Dice}(Y, T)\right)\right)$$

3.3.10 Log-Cosh Twersky Loss

Essendo la Twersky Loss una generalizzazione della Dice Loss, valgono gli stessi ragionamenti visti per la Log-Cosh Dice Loss, pertanto la Log-Cosh Twersky Loss può essere definita come segue:

$$L_{LCTwersky}(Y,T) = \log\Big(\cosh\Big(L_{Twersky}(Y,T)\Big)\Big)$$

3.3.11 Exponential Logarithmic Loss

La Exponential Logarithmic Loss migliora la segmentazione di piccole strutture ed è definita come segue [34]:

$$L_{EL}(Y,T) = \omega_{Dice} L_{EL-Dice} + \omega_{CE} L_{CE}$$

dove $\omega_{Dice}, \omega_{CE} \in [0,1]$, con $\omega_{Dice} + \omega_{CE} = 1$, sono i rispettivi pesi di $L_{EL-Dice}$ e L_{CE} che sono così definite:

$$L_{EL-Dice}(Y,T) = \exp\big[(-\ln\big(Dice(Y,T)\big)^{\gamma_{Dice}}\big]$$

$$L_{EL-CE}(Y,T) = \exp\big[w_l(-\ln\big(p_l(\boldsymbol{x})\big)^{\gamma_{Cross}}\big]$$

dove $p_l(x)$ è la probabilità softmax, $\boldsymbol{x}$ è la posizione all'interno dell'immagine ground truth l-esima, γ_{Dice} e γ_{Cross} permettono di avere più controllo sulla non linearità della loss e il peso ω_l permette di bilanciare tra esempi semplici e difficili.

3.3.12 Dual Cross-Entropy

La Dual Cross-Entropy è simile alla Cross-Entropy, ma aggiunge un ulteriore termine di regolarizzazione, in particolare è definita come segue [33]:

$$L_{nDCE}(Y,T) = -\sum_{i=1}^{c} T_{i,n}\log\big(Y_{i,n}\big) + \beta\sum_{i=1}^{c}\big(1-T_{i,n}\big)\log\big(\alpha + Y_{i,n}\big)$$

dove $\beta \geq 0$ e $\alpha > 0$ sono degli iperparametri.

Il termine di regolarizzazione $\beta\sum_{i=1}^{c}\big(1-T_{i,n}\big)\log\big(\alpha + Y_{i,n}\big)$ pone un vincolo sulle classi negative, in modo da ridurre l'effetto del vanishing descent.

3.3.13 Dual Focal Loss

La Dual Focal Loss è ottenuta come combinazione delle versioni migliorate delle loss function: Focal Loss e Dual Cross-Entropy:

- Per migliorare le prestazioni della Dual Cross-Entropy, viene modificato il termine di regolarizzazione $\beta\big(1-T_{i,n}\big)\log\big(\alpha + Y_{i,n}\big)$ che diventa $\beta\big(1-T_{i,n}\big)\log\big(\rho - Y_{i,n}\big)$, dove $\beta \geq 1$ e $\rho \geq 1$. In questo modo, al contrario della

versione originale, la rete viene penalizzata proporzionalmente quando si hanno dei FN.

- Per migliorare le prestazioni della Focal Loss, viene modificato il fattore dinamico $\alpha(1 - Y_{i,n})^{\gamma}$ che diventa $\alpha(|y_{i,n} - z_{i,n}|)^{y}$, dove $\alpha, \gamma \geq 1$. In questo caso però questo fattore di peso non viene più moltiplicato, ma sommato alla Cross-Entropy. In questo modo, si riescono ad ottenere risultati e gradienti migliori della Cross-Entropy, inoltre si consente alla loss di prendere in considerazione esplicitamente (attraverso feedback) l'errore sulle classi negative, rendendo più robusto il gradiente.

La Dual Focal Loss è pertanto definita come segue [33]:

$$L_{nDFL}(Y,T) = -\sum_{i=1}^{c}\left(T_{i,n}\log(Y_{i,n}) + \beta(1 - T_{i,n})\log(\rho - Y_{i,n}) - \alpha(|T_{i,n} - Y_{i,n}|)^{y}\right)$$

3.3.14 Noise Robuste Dice Loss

La Dice Loss, vista nella sezione 3.3.1, si comporta bene anche per classi fortemente sbilanciate. Essa può essere vista come una variante del Mean Square Error (MSE), il quale tuttavia non si dimostra essere robusto quando le label presentano del rumore.

È stato dimostrato teoricamente che il Mean Absolute Error (MAE) può ottenere una robustezza al rumore maggiore di MSE e della Cross-Entropy sotto certe ipotesi. Il MAE può essere riformulato come loss function per la segmentazione come segue:

$$L_{MAE}(Y,T) = \frac{1}{N}\sum_{i=1}^{N}|Y_i - T_i|$$

dove N è il numero di pixel nell'immagine.

Nonostante la tolleranza alle label che presentano rumore, L_{MAE} tratta tutti i pixel allo stesso modo, per questo motivo ottiene prestazioni scadenti con le CNN e non è in grado di gestire le classi sbilanciate.

La Noise Robuste Dice Loss è una generalizzazione delle funzioni L_{Dice} e L_{MAE}, in particolare ne sfrutta i vantaggi e va a superare i loro rispettivi limiti.

La Noise Robuste Dice Loss è definita come segue [35]:

$$L_{NR-Dice}(Y,T) = \frac{\sum_{i=1}^{N}|Y_i - T_i|^{\gamma}}{\sum_{i=1}^{N}{Y_i}^2 + \sum_{i=1}^{N}{T_i}^2 + \epsilon}$$

dove $\gamma \in [1,2]$ è un iperparametro e $\epsilon = 10^{-5}$ è un numero piccolo per la stabilità numerica. Per $\gamma = 2$ si ha che $L_{NR-Dice}$ equivale a L_{Dice}, mentre per $\gamma = 1$ si ha che $L_{NR-Dice}$ equivale a L_{MAE}.

3.3.15 Sensitivity Specificity Loss

Similmente al coefficiente Dice, sensitività e specificità sono largamente utilizzati come metriche per valutare le predizioni di segmentazione.

La Sensitivity Specificity Loss è definita come segue [31]:

$$L_{SS} = \omega \cdot Sensitivity + (1 - \omega) \cdot Specificity$$

dove ω è un iperparametro utilizzato per gestire il problema delle classi sbilanciate e le funzioni Sensitivity e Specificity sono così definite:

$$Sensitivity = \frac{TP}{TP + FN}$$

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

3.3.16 Combinazioni di loss

Un approccio molto utilizzato per definire nuove loss function è quello di andare combinare due o più loss function esistenti tra loro, semplicemente facendone una media pesata. Il motivo per cui viene fatto ciò è che, in questo modo, si riescono a combinare le caratteristiche e i vantaggi di diverse loss function allo scopo di ottenere prestazioni della rete migliori.

Spesso però, il comportamento delle loss function non è molto prevedibile come può sembrare, ciò significa che loss function che sono state definite per un certo tipo di problema, possono ottenere pessimi risultati per un problema molto simile, o possono ottenere ottimi risultati per un problema molto diverso. Per questo motivo, dopo aver testato le combinazioni di loss che sulla carta sembrano dare risultati migliori, spesso si vanno a testare altre combinazioni in maniera random, allo scopo di vedere se una di queste dà risultati migliori di quelli delle loss precedentemente testate.

In questo lavoro di tesi, dato il gran numero di loss function già definite, vengono proposte solo un paio di combinazioni di loss function che sono le seguenti:

- Combinazione 1: Focal Twersky Loss + Focal Dice Loss:
- Combinazione 2: Log-Cosh Dice Loss + Focal Dice Loss + Log-Cosh Focal Twersky Loss

3.4 Risultati

Per motivi di tempo non è stato possibile addestrare e testare le reti, ottenute dai diversi approcci di segmentazione proposti, su server. Per farsi un'idea, l'addestramento di una rete ottenuta con il secondo approccio di segmentazione, con i parametri di addestramento definiti nella sezione 3.2, richiede più di 20 ore di addestramento su server multi-GPU. L'addestramento di una rete ottenuta con il primo approccio di segmentazione, con i

parametri di addestramento definiti nella sezione 3.2, nonostante sia più leggera richiede comunque più di 4 ore di addestramento su server multi-GPU. Ogni rete va addestrata e testata per ogni loss function definita (che sono circa una ventina). Inoltre, bisogna testare tutte le combinazioni di reti, ottenute da ogni loss function, che vanno a costituire l'ensemble. Si può quindi facilmente stimare che tale lavoro può richiedere più di due settimane di computazione.

A causa delle ridotte capacità di calcolo e della ridotta quantità di memoria, sia RAM che quella della GPU, disponibile sul mio computer, su cui sono installati: una GPU Geforce GTX 970, un processore Intel i7-11700k e 16GB di RAM, è stato possibile solo verificare che il codice implementato funzionasse correttamente. Tutte le reti che ho addestrato sul mio computer hanno richiesto un settaggio degli iperparametri con valori molto bassi, per non saturare le risorse del computer. Per questo motivo i risultati ottenuti dai test, nonostante siano validi, non sono degni di nota e pertanto ho deciso di non riportarli all'interno di questa sezione.

In attesa che l'addestramento e i test delle reti vengano eseguiti su server, in modo da ottenere risultati più realistici su quelle che sono le reali prestazioni degli approcci di segmentazione implementati, riporto in Figura 3.2 una tabella di quello che è lo stato dell'arte delle prestazioni di segmentazione ottenute sul dataset BraTS nel 2019 [36], in modo da avere un riferimento con cui comparare i risultati che saranno ottenuti.

	Dice		
	WT	ET	TC
Model in [56]	0.905	0.764	0.820
Model in [60]	0.890	0.765	0.811
Model in [61]	0.896	**0.766**	0.790
Model in [62]	0.900	0.750	0.794
Model in [63]	0.894	0.737	0.807
Our model	**0.936**	0.760	**0.851**

Figura 3.2 - Stato dell'arte su BraTS 2019 in termini di dice score

Conclusioni

In questo lavoro di tesi sono stati proposti tre nuovi approcci per eseguire la segmentazione binaria 3D di scansioni MRI cerebrali, con lo scopo di provare a migliorare quello che ad oggi è lo stato dell'arte delle attuali tecniche di riconoscimento del tumore cerebrale.

All'interno di questa tesi sono stati inizialmente trattati in maniera generica quelli che sono gli aspetti fondamentali che sono alla base del deep learning. Successivamente sono stati trattati più nel dettaglio il pre-processing del dataset BraTS, la data augmentation, la creazione di training set, validation set e test set; successivamente sono state introdotte le architetture di rete DeepLabV3+ e U-Net 3D; dopodiché sono state trattate le metriche di misura delle performance per la segmentazione.

Infine, nell'ultima parte, è stato descritto il codice implementato, le loss function utilizzate e sono stati riportati i risultati sperimentali raccolti.

Come visto nella sezione Risultati, l'addestramento e il testing di una rete progettata per eseguire la segmentazione semantica ha un costo computazionale molto elevato e richiede moltissimo tempo, pertanto, per poter essere eseguiti con successo, richiedono le capacità di calcolo di un server multi-GPU. Purtroppo per motivi di tempo non siamo riusciti ad addestrare e a testare le reti progettate su server, tuttavia, il corretto funzionamento del codice implementato è stato testato sul mio computer.

In attesa di riscontri sperimentali, è possibile iniziare a fare una previsione su quali loss function ci si aspetta che daranno risultati migliori. In particolare, è stato visto che la zona tumorale (foreground) copre un'area, o un volume, molto inferiore rispetto al resto dell'immagine (background), pertanto la classe 'tumor' risulta sbilanciata rispetto alla classe 'background'; inoltre la classe 'tumor' ha una forma complessa e molto irregolare. Mi aspetto pertanto che le loss function che daranno risultati migliori saranno quelle progettate per lavorare su classi fortemente sbilanciate e/o quelle progettate per lavorare su regioni di foreground molto irregolari. Prevedo quindi che le loss function che daranno risultati migliori saranno: Dice Loss, Focal Loss, Focal Twersky Loss e Focal Dice Loss.

Sviluppi futuri:

- Training e test delle reti su server per verificare la bontà dei metodi proposti
- Ricerca di nuove loss function e creazione di più loss function composte da testare per i diversi approcci proposti
- Il mio relatore, il prof. Nanni, ha suggerito di sfruttare la caratteristica del secondo approccio di segmentazione proposto, che è quella di lavorare su patch di volume

casuali, per creare un ensemble; sarebbe quindi interessante implementare e testare tale intuizione

- Per migliorare ulteriormente l'ensemble si possono provare a combinare più reti ottenute dagli approcci 'weak' proposti o ad implementarne di nuovi

Bibliografia

[1] *Uno studio empirico sull'insieme degli approcci di segmentazione.* Nanni L, Lumini A, Loreggia A, Formaggio A, Cuza D., pp. 341-358 [link]

[2] *Tumori cerebrali*, IRCCS Humanitas Research Hospital, [link]

[3] *Tumore del cervello*, AIRC [link]

[4] *Tecniche di ottimizzazione I*, Aaron Defazio, [link]

[5] *Tecniche di ottimizzazione II*, Aaron Defazio, [link]

[6] *Dataset BraTS*, Medical Decathon, [link]

[7] *3D Brain Tumor Segmentation Using Deep Learning*, MathWorks, [link]

[8] *Funzioni di Attivazione nel deep learning la Guida Completa*, Intelligenza Artificiale Italia [link]

[9] *An Encoder-Decoder Network Based FCN Architecture for Semantic Segmentation*, Yongfeng Xing, Luo Zhong, and Xian Zhong, [link]

[10] *Learning Deconvolution Network for Semantic Segmentation*, Hyeonwoo Noh, Seunghoon Hong, Bohyung Han, [link]

[11] *An Introduction to different Types of Convolutions in Deep Learning*, Paul-Louis Pröve, [link]

[12] *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*, L. -C. Chen, G. Papandreou, I. Kokkinos, K. Murphy e AL Yuille, pp. 834-848, [link]

[13] *Convolution and Pooling Variants (Dilated Convolution, SPP, ASPP*, jun94, [link]

[14] *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*, Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, [link]

[15] *How DeepLabV3 Works*, ArcGIS API for Python, [link]

[16] *DeepLabv3: Semantic Image Segmentation*, Madeline Schiappa, [link]

[17] *Review: DeepLabv3 - Atrous Convolution (Semantic Segmentation)*, Sik-Ho Tsang, [link]

[18] *Rethinking Atrous Convolution for Semantic Image Segmentation,* Liang-Chieh Chen George Papandreou Florian Schroff Hartwig Adam, [link]

[19] *An Encoder-Decoder Network Based FCN Architecture for Semantic Segmentation*, Yongfeng Xing, Luo Zhong, and Xian Zhong, [link]

[20] *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam, [link]

[21] *An Overview of ResNet and its Variants*, Vincent Feng, [link]

[22] *Deep Residual Learning for Image Recognition*, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, [link]

[23] *Understanding and visualizing ResNets*, Pablo Ruiz, [link]

[24] *Residual blocks - Building blocks of ResNet*, Sabyasachi Sahoo, [link]

[25] *ResNet-101 convolutional neural network*, MathWorks [link]

[26] *Review: U-Net (Biomedical Image Segmentation)*, Sik-Ho Tsang, [link]

[27] *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation*, Ozg¨un Cicek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger, [link]

[28] *Semantic Segmentation - U-Net*, Kerem Turgutlu [link]

[29] *Metrics to Evaluate your Semantic Segmentation Model*, Ekin Tiu, [link]

[30] *MATLAB*, Wikipedia [link]

[31] *A survey of loss functions for semantic segmentation*, Shruti Jadon [link]

[32] *Focal Dice Loss-Based V-Net for Liver Segments Classification*, Berardino Prencipe, Nicola Altini, Giacomo Donato Cascarano, Antonio Brunetti, Andrea Guerriero, and Vitoantonio Bevilacqua, [link]

[33] *Dual Focal Loss to address class imbalance in semantic segmentation*, Md SazzadHossain, John M.Betts, Andrew P.Paplinski, [link]

[34] *3D Segmentation with Exponential Logarithmic Loss for Highly Unbalanced Object Sizes*, Ken C. L. Wong, Mehdi Moradi, Hui Tang, and Tanveer Syeda-Mahmood, [link]

[35] *A Noise-Robust Framework for Automatic Segmentation of COVID-19 Pneumonia Lesions From CT Images*, Guotai Wang , Xinglong Liu, Chaoping Li, Zhiyong Xu, Jiugen Ruan, Haifeng Zhu, Tao Meng, Kang Li, Ning Huang, and Shaoting Zhang, [link]

[36] *Nature-Inspired Level Set Segmentation Model for 3D-MRI Brain Tumor Detection*, Oday Ali Hassen, Sarmad Omar Abter, Ansam A. Abdulhussein, Saad M. Darwish, Yasmine M. Ibrahim and Walaa Sheta [link]