UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Dell'Informazione

Corso di Laurea in

Ingegneria Informatica

LOSS FUNCTION PER IMAGE SEGMENTATION

Relatore:
Prof. Loris Nanni
Correlatrice:
Dott.ssa Daniela Cuza

Laureando:
Nicola Lorenzon

Anno accademico 2021/2022

Abstract

This discussion has two purposes:
The first purpose consists in the analysis of the behavior of a neural network for the segmentation of images of colon polyps, which is the process of accurately delineating and discriminating the region that identifies the polyp from the background. Particularly, it will be studied the behavior of the model as it changes the *loss function*.
The second one, on the other hand, consists in implementing and testing new loss functions in order to *improve the performance* of the network itself.

Abstract

Questa trattazione si pone due obiettivi:
Il primo obiettivo consiste nell'analisi del comportamento di una rete neurale volta alla segmentazione di immagini di polipi al colon, ovvero il processo di delineare e discriminare accuratamente la regione che identifica il polipo dallo sfondo. In particolare, ne verrà studiato il comportamento al variare della *funzione di loss*.
Il secondo, invece, consiste nell'implementare e testare nuove loss function al fine di *migliorare le prestazioni* della rete stessa.

Contents

Abstract ii

Introduzione 1

Nomenclature 3
0.0.1 Symbols . . . 3
0.0.2 Abbreviations . . . 3

1 RETI NEURALI 5
1.1 Neuroni . . . 5
1.2 Deep Learning e CNN . . . 6
1.3 Calcolo del gradiente: l'algoritmo di back-propagation . . . 9
1.3.1 Stochastic Gradient Descent . . . 10
1.4 Funzioni obiettivo - Loss Functions . . . 10
1.4.1 Dice Loss . . . 10
1.4.2 Tversky Loss . . . 11
1.4.3 Binary Cross Entropy Loss . . . 11
1.4.4 Balanced Cross Entropy . . . 11
1.4.5 Focal Loss . . . 12
1.4.6 Focal Tversky Loss . . . 12
1.4.7 Log-Cosh Dice Loss . . . 12
1.4.8 Top-K Loss . . . 12
1.4.9 Hausdorff Distance Loss . . . 13

2 SPIEGAZIONE PROBLEMA ED IMPLEMENTAZIONE LOSS FUNCTION 15
2.1 Caso di studio . . . 15
2.2 Image Segmentation . . . 15
2.3 Skip Connection . . . 16
2.4 Encoder-Decoder . . . 17
2.5 Dilated Convolutions . . . 17
2.6 Implementazione Modello: ResNet-18 . . . 18
2.7 Dataset . . . 18
2.8 Metriche di valutazione . . . 19
2.8.1 TP - TN - FP - FN . . . 19
2.8.2 mIoU - Mean Intersection over Union . . . 19
2.8.3 mDice . . . 19

2.8.4 Precision 20
2.8.5 Recall 20
2.8.6 Accuracy 20
2.9 Risultati 20

3 IMPLEMENTAZIONE NUOVE LOSS 22
3.1 LogCosh Tversky 22
3.2 Exponential Logarithmic Dice 22
3.3 Exponential Logarithmic Loss 23
3.4 Ensemble di Loss 23
3.5 Risultati 24
3.6 Validazione 25

4 CONCLUSIONI 27

List of Figures

1.1 Neurone biologico vs modello neurone matematico 6
1.2 ConvLayer - Convolutional Layer 7
1.3 ReLU Layer - Rectified Linear Units Layer 7
1.4 Pooling Layer - Max Pooling 8
1.5 Struttura CNN 8
1.6 Learning Rate - Valore ottimo 9

2.1 Polyph segmentation 16
2.2 Skip Connection 16
2.3 Encoder Decoder Structure 17
2.4 Dilated Convolution 17
2.5 SX Polyph Image, DX Polyph Mask 18
2.6 Matrice Di Confusione 2 Classi 19

3.1 Output Rete 25

Introduzione

Lo scopo di questo elaborato è quello di fornire, dopo un'accurata analisi delle reti neurali, in particolare le reti neurali convoluzionali, prima una panoramica delle loss function presenti in letteratura applicate al caso di studio (polyph segmentation) e di spiegare poi, il lavoro sperimentale fatto implementando nuove loss function.
La tesi sarà articolata in quattro capitoli. I primi tre avranno le seguenti funzioni:
Nel primo verranno illustrate le reti neurali, con particolare attenzione alle reti neurali convoluzionali, ampiamente usate in problemi di segmentazione di immagini, come quello trattato da questa tesi.
Nel secondo capitolo verrà fatto focus sullo stato dell'arte. Verrà spiegato il caso di studio, ovvero il problema della segmentazione di immagini mediche relative ai polipi al colon.
Verrà spiegata la rete neurale utilizzata.
Saranno spiegate le principali loss functions presenti in letteratura, esponendone i risultati ottenuti mediante il loro utilizzo.
Nel terzo capitolo verranno esposte nuove loss functions, implementate modificando le loss functions presenti in letteratura ed illustrate nel capitolo 2, oppure implementate ex-novo.
Mentre il quarto ed ultimo capitolo ha lo scopo di spiegare, tramite una descrizione analitica e verbale, i risultati raggiunti.

Nomenclature

0.0.1 Symbols

Y = insieme di pixel classificati in output dal modello.
y = singolo pixel classificato in output dal modello.
T= insieme di pixel classificati nella ground truth.
t = singolo pixel classificato presente nella ground truth.
$\bar{y} = 1 - y$
$\bar{t} = 1 - t$
a_i^k = i-esimo neurone al k-esimo layer della rete.
w_i^k = peso associato al i-esimo neurone del k-esimo layer della rete.
W^k = insieme di pesi associati ai neuroni del k-esimo layer.
α = learning rate, coefficiente moltiplicativo per aggiustare la velocità di convergenza nell'algoritmo di back-propagation

0.0.2 Abbreviations

CNN = Convolutional Neural Network = Rete Neurale Convoluzionale
TP = True positive = Positivi veri
Numero di pixel facenti parte della classe *1*, classificati *1*.
TN = True negative = Negativi veri
Numero di pixel facenti parte della classe *0*, classificati *0*.
FP = False positive = Positivi falsi
Numero di pixel facenti parte della classe *0*, classificati *1*.
FN = False negative = Negativi falsi
Numero di pixel facenti parte della classe *1*, classificati *0*.

1 RETI NEURALI

Le reti neurali, in informatica, sono un insieme di neuroni e collegamenti atti ad imparare, modificando autonomamente i propri pesi (=coefficienti moltiplicativi degli input), a mappare dei vettori in ingresso, ai rispettivi vettori target in uscita.

1.1 Neuroni

Il **neurone** è l'elemento basilare del sistema nervoso, ed anche il più importante, tanto che può essere considerato l'unità di calcolo primaria alla base della nostra intelligenza. Il sistema nervoso umano ne è costituito approssimativamente da 86 miliardi, connessi fra di loro da un numero di *sinapsi* dell'ordine di $10^{(15)}$. La figura [1.1] mostra sulla sinistra la rappresentazione biologica di un neurone, mentre sulla destra un modello matematico nel quale: ogni neurone riceve in input il segnale dai suoi *dendriti* e, una volta elaborato mediante una funzione prestabilita, produce un segnale di output che andrà ad alimentare il neurone successivo.
Questa attività biologica può essere rappresentata da un modello matematico nel quale i segnali che viaggiano attraverso gli *assoni* (x_i) interagiscono attraverso un prodotto ($x_i w_i$) con i dendriti dei neuroni con i quali sono collegati tramite sinapsi (w_i). L'idea è che il valore delle delle sinapsi (i pesi w_i) possa essere appreso autonomamente e sia in grado di controllare, in base al proprio modulo e segno, l'influenza di ogni neurone sui successivi. Il modello matematico che ne risulta è il seguente:

$$N_k = f_{kl}(\sum_{i=1}^{n} x_i * w_i + b) \tag{1.1}$$

Un neurone ha il compito di applicare una funzione predefinita al prodotto vettoriale tra gli input x_i moltiplicati per i relativi pesi w_i, ci somma un valore "correttivo" ed infine applica una funzione definita al valore trovato.

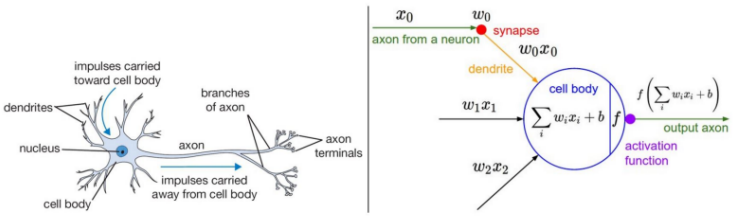

Figure 1.1: Neurone biologico vs modello neurone matematico

1.2 Deep Learning e CNN

Il **Deep Learning**, letteralmente apprendimento profondo, è una branca del machine learning che a sua volta fa parte del più ampio mondo dell'intelligenza artificiale.
I sistemi di apprendimento profondo trovano spazio in molti settori, come per esempio il riconoscimento oggetti nelle immagini e nei video, la trascrizione del parlato in testo, la raccomandazione di risorse online e molti altri.
Per far ciò, i sistemi di deep learning utilizzano varie tipologie di reti, tra le quali, spiccano sicuramente le *Convolutional Neural Network* e le *Recurrent Neural Networks*. In questa trattazione verrà analizzata solamente la prima tipologia.
Le **Convolutional Neural Network** (alle quali si farà riferimento da ora in poi, per brevità, con l'acronimo CNN) sono un tipo di rete neurale *feed-forward* composta da vari layer (= insieme di neuroni). Con feed-forward si intende che le connessioni tra le unità (neuroni) non formano cicli e l'output di un livello risulta essere l'input per il livello successivo.
I layer che solitamente compongono una CNN sono:

- Un **layer di input**, costituito da una sequenza di neuroni in grado di ricevere le informazioni da elaborare. A questo livello, infatti, verrà passato il vettore di dati che, per esempio, nel campo dell'image segmentation trattato in questa tesi, rappresenta i pixel dell'immagine di ingresso.

- Tanti **layer intermedi** nascosti, composti solitamente dalla tripletta di *Convolutional layer*, *ReLU layer* e *Pooling layer*

  - **ConvLayer** è il cuore della rete, dal quale prende anche il nome. Ha lo scopo di eseguire la convoluzione di uno o più filtri mxm con m dispari, tutti della stessa dimensione, con la matrice di input al fine di creare aggregazioni di dati utili per rilevare pattern di diversa natura.
    Definendo I la matrice di input ed F il filtro, l'operazione di convoluzione fra matrici come:

$$p(i,j) = \sum_{i=1}^{m}\sum_{j=1}^{m}(I[i + \lceil\frac{m}{2}\rceil - y, j + \lceil\frac{m}{2}\rceil - x] * F[y,x]) \tag{1.2}$$

    Durante il passo feed-forward ogni filtro viene fatto convolvere lungo la lunghezza e l'altezza della features map.
    Nel caso di immagini con D canali-colore, questa operazione viene svolta D volte, tenendo così conto della profondità.

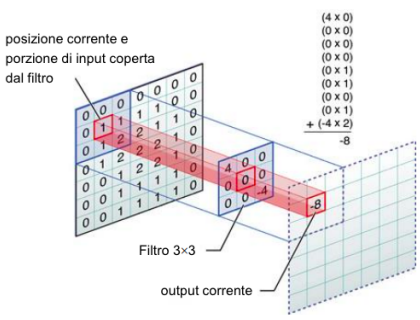

Figure 1.2: ConvLayer - Convolutional Layer

Output del ConvLayer:
La dimensionalità dell'output di questo layer dipende, oltre che dalle dimensioni dei filtri utilizzati per la convoluzione, anche da due iperparametri, lo *stride* e il *padding*:

* **Stride:** specifica il numero di pixel di cui si vuol traslare il filtro sulla features map ad ogni spostamento. Più alto è lo stride, meno preciso sarà l'output, più basso è lo stride, più sarà computazionalmente pesante gestire la features map in output.
* **Zero padding:** nell'operazione di convoluzione, può essere utile avere un dei bordi inizializzati ad uno specifico valore (solitamente zero) così da non perdere il focus sui primissimi pixel significativi vicino ai bordi.

Le dimensioni dell'output di questo layer possono essere calcolate come segue, tenendo conto di *stride* e *padding*:

$$W_{out} = \frac{W_{in} - F + 2 * P}{S} + 1 \tag{1.3}$$

– **ReLU Layer** (= Rectified Linear Units Layer), posto subito dopo il precedente, si occupa dell'annullamento dei valori non significativi ai fini della classificazione, ottenuti con la convoluzione.

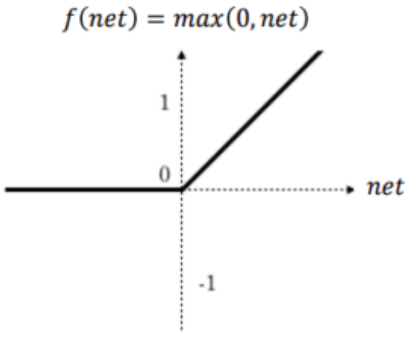

Figure 1.3: ReLU Layer - Rectified Linear Units Layer

– **Pooling Layer:** esegue un'aggregazione delle informazioni generando così una feature map di dimensione inferiore al fine di ridurre il carico computazionale al layer successivo. Il pooling layer opera applicando un filtro di dimensioni

FxF che esegue operazioni di aggregazione come (ad esempio la funzione max che, dato un vettore, restituisce l'elemento di dimensione maggiore) secondo la metrica definita.
Anche in questo caso le dimensioni della features map in output dipendono dallo stride (S) con il quale il filtro viene fatto scorrere sull'input.
Definiti F e S è possibile calcolare le dimensioni di output del pooling layer:

$$\begin{aligned} W_2 &= \frac{W_1 - F}{S} + 1 \\ H_2 &= \frac{H_1 - F}{S} + 1 \\ D_2 &= D_1 \end{aligned} \tag{1.4}$$

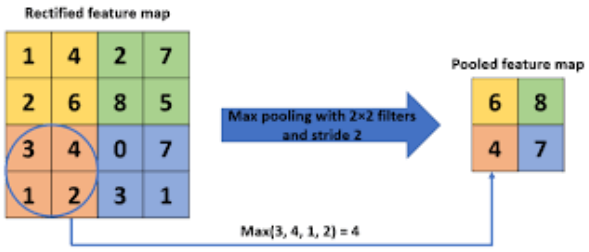

Figure 1.4: Pooling Layer - Max Pooling

- Un **layer di output** che, ricevute le informazioni elaborate dai layer precedenti, ed effettua la classificazione vera e propria utilizzando una funzione prestabilita.

Così facendo, la struttura di una CNN risulta essere la seguente:

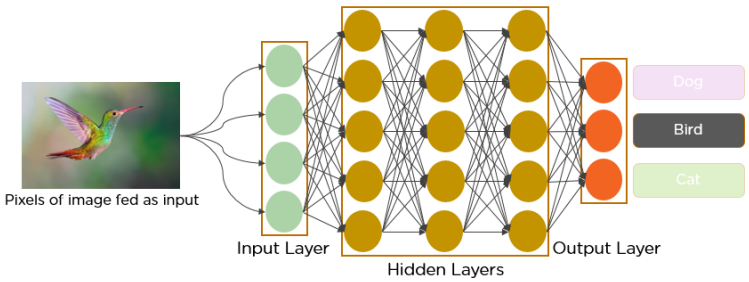

Figure 1.5: Struttura CNN

1.3 Calcolo del gradiente: l'algoritmo di back-propagation

La **back-propagation** è un algoritmo di discesa del gradiente utilizzato per aggiornare i pesi di una rete neurale al fine di migliorarne le prestazioni.
Data una generica osservazione (x, y) l'algoritmo di back-propagation prevede di effettuare un primo passo in avanti lungo l'intera rete (feed-forward step) e salvare i valori che si creano ad ogni nodo a_i^k di ogni layer, incluso quello di output. La responsabilità di ogni nodo nella previsione del vero valore di y viene quindi misurata attraverso il calcolo di un termine d'errore δ_i^k.
Avendo dunque:

$$y = f(x, W) \tag{1.5}$$

come output della rete e:

$$f_l = L[y, f(x, W^k)] \tag{1.6}$$

come funzione di perdita, è possibile definire una funzione gradiente che misura le variazioni della funzione obiettivo in relazione alle variazioni dei pesi come:

$$\Delta W^k = \frac{1}{n} \sum_{i=1}^{n} \frac{\delta L[y, f(x, W^k)]}{\delta w_i} \tag{1.7}$$

Una volta calcolato il gradiente come media dei singoli gradienti (riferiti al singolo neurone), è possibile procedere con l'aggiornamento dei pesi del sistema come segue:

$$W^k \leftarrow W^k - \alpha(W^k + \lambda W^k), k = 1, ..., K - 1 \tag{1.8}$$

Grazie al calcolo del gradiente, viene ricavata la direzione lungo la quale, muovendoci, andremmo eventualmente a diminuire l'errore, tuttavia non è nota la grandezza del passo da fare.
Nella formula 1.8 infatti, viene introdotto un coefficiente α, iperparametro solitamente chiamato *learning-rate*.
Il **learning-rate** rappresenta sicuramente un punto chiave per le prestazioni di una CNN dato che esso influenza notevolmente la convergenza verso un risultato ottimo:
Valori troppo grandi, tendono ad avere convergenza più velocemente ma sono rischiosi dal momento che potrebbero non trovare il minimo assoluto e/o girarci attorno.
Valori troppo piccoli potrebbero portare alla convergenza molto lentamente ed eventualmente a non uscire da un minimo locale.

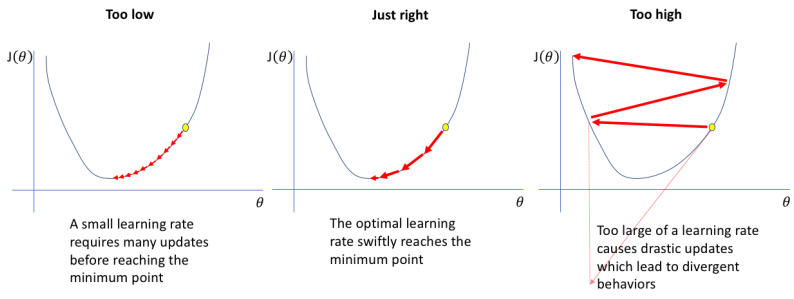

Figure 1.6: Learning Rate - Valore ottimo

1.3.1 Stochastic Gradient Descent

Diversi algoritmi di apprendimento automatico utilizzano la back-propagation per l'aggiornamento dei pesi della rete neurale, tra i quali Stochastic Gradient Descent.
Per motivi di tempi computazionali, spesso si preferisce non tener conto di tutti i pattern a disposizione per l'aggiornamento dei pesi, per questo motivo, Stochastic Gradient Descent applica la 1.7 solamente ad un sottoinsieme di dati organizzati in gruppi, chiamati *batch*.

1.4 Funzioni obiettivo - Loss Functions

Come accennato sopra, per il calcolo del gradiente c'è bisogno di una funzione obiettivo: la **loss function**.
La loss function di una CNN è una metrica che fornisce un'indicazione su quanto la predizione fatta all'ultima iterazione dell'algoritmo si discosta dal ground truth, ovvero, l'output che ci si aspetterebbe.
Si cerca quindi, nell'implementazione di un modello di machine learning, di trovare una loss function che ci permetta di minimizzare l'errore della rete.
In letteratura sono proposte svariate loss functions per l'ottimizzazione dei più ricorrenti problemi di machine learning come l'image segmentation.
Tra queste troviamo [3]:

- Dice Loss
- Tversky Loss
- Binary Cross Entropy Loss
- Balanced Cross Entropy
- Focal Loss
- Focal Tversky Loss
- Log-Cosh Dice Loss
- Top-K Loss
- Hausdorff Distance Loss

1.4.1 Dice Loss

Dice Loss è una funzione che utilizza il coefficiente Sørensen-Dice, metrica usata per stimare le peformance di modelli di image segmentation. Il coefficiente Sørensen-Dice, date in input due immagini, determina quanto simili sono, fornendo in output un valore nel range $[0, 1]$.

$$DiceCoefficient = D(Y, T) = \frac{2|Y \cap T|}{|Y| + |T|} \tag{1.9}$$

Utilizzando la 1.9 ed applicandola al campo dell'image segmentation, otteniamo la Dice Loss Function così definita:

$$L_{Dice}(Y,T) = 1 - D(Y,T) = 1 - \frac{2Y\bar{T} + \epsilon}{Y + \bar{T} + \epsilon} \tag{1.10}$$

È stato aggiunto $+\epsilon$ a numeratore e denominatore per gestire il caso limite nel quale $y = \bar{t} = 0$

1.4.2 Tversky Loss

L'indice di Tversky, utilizzato per il calcolo della Tversky Loss Function, può essere interpretato come una generalizzazione del coefficiente Dice. Tversky Index, a differenza del Dice Index, aggiunge un peso a FP (false positive) e FN (false negative) aggiungendo un coefficiente moltiplicativo β [1].
Definito quindi il Tversky Index come:

$$TI(Y,T) = \frac{YT}{YT + \beta(1-Y)T + (1-\beta)Y(1-T)} \tag{1.11}$$

Si può quindi definire la Tversky Loss Function utilizzando la 1.11 come segue:

$$L_{Tverski}(Y,T) = 1 - \frac{YT + \epsilon}{YT + \beta(1-Y)T + (1-\beta)Y(1-T) + \epsilon} \tag{1.12}$$

Anche in questo caso, è stato aggiunto $+\epsilon$ a numeratore e denominatore per gestire il caso limite nel quale $Y = \bar{T} = 0$.
Si può notare inoltre che, per $\beta = \frac{1}{2}$, Tversky Loss risulta analoga a Dice Loss.

1.4.3 Binary Cross Entropy Loss

Binary Cross Entropy deriva dalla distribuzione di Bernoulli e misura la differenza tra due distribuzioni di probabilità.

$$L_{BCE}(Y,T) = -(T \log(Y) + (1-T) \log(1-Y)) \tag{1.13}$$

1.4.4 Balanced Cross Entropy

La Balanced Cross Entropy non è altro che la Binary Cross Entropy con un coefficiente moltiplicativo $\alpha \in [0,1]$ che permette di correggere i problemi legati alle classi non bilanciate.
Si definisce quindi α_t come:

$$\alpha_t = \begin{cases} \alpha & \text{se} \quad T = 1 \\ 1-\alpha & \text{altrimenti} \end{cases} \tag{1.14}$$

Così facendo otterremo quindi:

$$L_{BaCE}(Y,T) = \alpha_t CE(Y_t) = -\alpha_t \log(Y_t) \tag{1.15}$$

1.4.5 Focal Loss

Focal Loss può essere vista come una variante della Binary Cross Entropy. L'obiettivo è quello di dare meno importanza ai pattern semplici così da lasciare più spazio a quelli difficili.
Questa funzione di loss funziona bene se si hanno classi molto sbilanciate, per esempio molto più sfondo rispetto all'elemento d'interesse.
Per capire meglio Focal Loss, si può pensare alla Cross Entropy:

$$L_{CE}(Y,T) = \begin{cases} -\log Y & \text{se} \quad T=0 \\ -\log(1-Y) & \text{altrimenti} \end{cases} \tag{1.16}$$

Per semplificare la notazione è possibile definire Y_t come:

$$Y_t = \begin{cases} Y & \text{se} \quad T=0 \\ 1-Y & \text{altrimenti} \end{cases} \tag{1.17}$$

È quindi possibile, grazie alla 1.17, ridefinire la 1.16:

$$L_{CE}(Y,T) = CE(Y_t) = -\log(Y_t) \tag{1.18}$$

1.4.6 Focal Tversky Loss

Focal Tversky Loss nasce dall'esigenza di assegnare un peso maggiore ai falsi negativi così da aumentare la metrica recall (spiegata in seguito).
Per far ciò si utilizza l'indice Tversky (1.12). Ad esso viene aggiunto ad esponente un coefficiente $\frac{1}{\gamma}$, dopo varie prove è stato riscontrato che le performance migliori si ottengono con $\gamma = \frac{4}{3}$.

$$L_{FT} = -\sum_c (1 - L_{Tversky_c})^{\frac{1}{\gamma}} \tag{1.19}$$

1.4.7 Log-Cosh Dice Loss

L'approccio Log-Cosh (= Logaritmo e Coseno iperbolico) è molto usato in problemi basati sulla regressione, in particolare per smussare gli spigoli che si formano nei grafici delle funzioni utilizzate.
Si può dimostrare che $f = \log(\cosh(x))$ rimane differenziabile e continua dopo la derivata prima.
Utilizzando la [1.10] è possibile definire:

$$L_{LC_{Dice}} = \log(\cosh(DiceLoss)) \tag{1.20}$$

1.4.8 Top-K Loss

Top-K Loss mira a forzare la rete ad imparare dai pattern difficili durante il training, dando meno importanza a quelli facili.
È definita in questo modo:

$$L_{TK}(Y,T) = \frac{\sum_{i=1}^{N} 1(y_{i,j=j \wedge p_{i,j}<t}) \log p_{i,j}}{\sum_{i=1}^{N} 1(y_{i,j=j \wedge p_{i,j}<t})} \tag{1.21}$$

Dove:
$t \in]0,1]$ è un iperparametro.

$1(f(x))$ vale 1 se $f(x)$ è verificata, 0 altrimenti.
In sostanza, i pixels che hanno una probabilità di predizione sotto la soglia t sono ininfluenti al fine del calcolo poiché sono facilmente classificabili.

1.4.9 Hausdorff Distance Loss

La *Hausdorff Distance* è una metrica che misura l'errore di segmentazione date in input la ground truth e la segmentazione predetta [4].
Per due set di punti Y e T, la distanza di Hausdorff può essere calcolata come:

$$hd(Y,T) = \max_{y \in Y} \min_{t \in T} ||y - t||_2 \tag{1.22}$$

Nel calcolo della Loss Function, la Distanza di Hausdorff viene calcolata bi-direzionalmente, ovvero prendendo il massimo $hd(Y,T)$ e $hd(T,Y)$:

$$L_{HD}(Y,T) = \max\left(hd(Y,T), hd(T,Y)\right) \tag{1.23}$$

Nella definizione di 1.22 è stata usata la distanza euclidea, nulla vieta tuttavia di utilizzare altre metriche.

2 SPIEGAZIONE PROBLEMA ED IMPLEMENTAZIONE LOSS FUNCTION

In questo capitolo, dopo aver spiegato il caso di studio e il modello usato, verranno discusse le performance ottenute da esso, al variare della loss function.

2.1 Caso di studio

Le immagini sono la rappresentazione visiva degli oggetti, descritte come matrici discrete all'interno di un calcolatore, ecco perché, se la capacità di vedere attraverso di esse viene applicata alle moderne tecnologie, è possibile progettare e sviluppare sistemi avanzati ed innovativi spendibili in innumerevoli campi di applicazione.
L'**image recognition**, tecnica volta all'analisi "intelligente" di immagini da parte di un calcolatore, trova spazio sia in ambiti industriali o scientifici, che più comunemente in ciò che riguarda la vita di tutti noi. Un esempio, ormai comune, di tecnologia già in uso che appartiene all'Intelligenza Artificiale ed è strettamente legato alla *Computer Vision*, è il riconoscimento di oggetti quali volti, macchine, persone, animali, nelle immagini.
In questa trattazione verrà studiato il comportamento delle tecniche di Computer Vision mediante una rete neurale convoluzionale, applicate al contesto della segmentazione di polipi al colon, al variare della loss function della rete stessa.

2.2 Image Segmentation

La **segmentazione** di un'immagine, nel campo di elaborazione digitale di matrici, è il processo grazie al quale un'immagine viene *partizionata* in regioni significative.
La segmentazione è di solito utilizzata per localizzare oggetti e bordi (linee, curve, ecc.).
Più precisamente, la segmentazione è il processo con il quale si classificano i pixel dell'immagine che hanno caratteristiche comuni, pertanto ciascun pixel in una regione è simile agli altri della stessa regione per una qualche proprietà o caratteristica (colore, intensità o texture).
Regioni adiacenti si distinguono le une dalle altri per differenze, più o meno significative,

di almeno una di queste caratteristiche. Il risultato di un'immagine segmentata è un insieme di regioni che, collettivamente, coprono l'intera immagine.
Nel caso di studio, l'obiettivo è segmentare la regione che identifica il polipo, data in input l'immagine di una colonscopia:

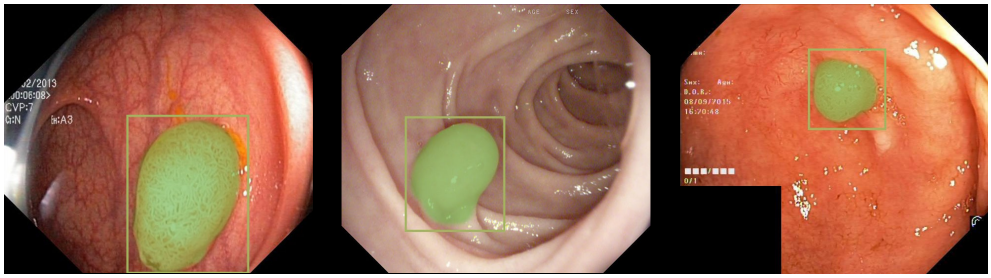

Figure 2.1: Polyph segmentation

2.3 Skip Connection

Come spiegato precedentemente, i livelli più alti di una CNN sono costituiti da filtri in grado, tramite l'operazione di convoluzione, di individuare semplici pattern all'interno dell'immagine come contorni, spigoli ecc. mentre i livelli più profondi imparano a riconoscere pattern più complessi come texture.
L'algoritmo di retro propagazione del gradiente dovrà, applicato come da definizione, ripercorrere tutta la rete per aggiornare i pesi portando così ad una notevole perdita di prestazioni per quanto riguarda il tempo di addestramento del modello in questione.
Un metodo usato, al fine di velocizzare il passo in questione è quello di definire delle **Skip Connection**, ovvero: fornendo l'output di un layer i, non solo al layer successivo $i+1$ ma anche ad uno o più layer $i+k$ con $(k>1)$ successivi.

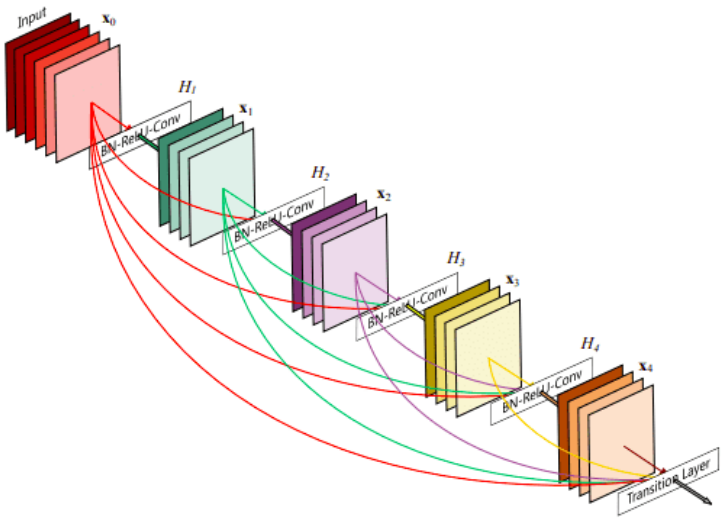

Figure 2.2: Skip Connection

2.4 Encoder-Decoder

Nell'image segmentation, una tipologia di CNN che ha trovato largo impiego, e l'unica che ad ora ha dato risultati tangibili, è l'autoencoder convoluzionale [6].
Il modello punta a fare prima una compressione delle immagini in input attraverso dei layer convoluzionali, seguita poi da una decompressione utilizzando dei layer de-convoluzionali.
La struttura della CNN risulta quindi essere:

$$Conv. \rightarrow ReLU \rightarrow Pooling \rightarrow ... \rightarrow Unpooling \rightarrow ReLU \rightarrow Deconv. \tag{2.1}$$

Ogni strato convoluzionale e il corrispondente deconvoluzionale avranno la stessa dimensione e la stessa profondità. La particolarità delle reti encoder-decoder risiede nell'uguaglianza tra dimensioni di input e di output, necessaria alla classificazione pixel-level [2].

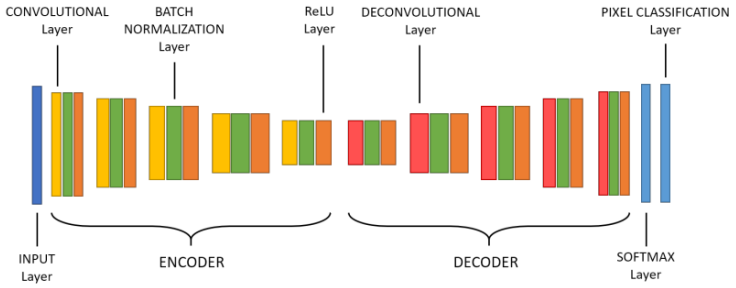

Figure 2.3: Encoder Decoder Structure

2.5 Dilated Convolutions

Dilated Convolutions è una tecnica che permette di aumentare il range di informazioni ottenibili da un layer convoluzionale senza perdita di informazione.
"Espandendo" il kernel infatti, è possibile coprire un'area più vasta dell'immagine di input.
Questo metodo è migliore rispetto ad usare una normale convoluzione poiché permette di ottenere più informazioni da un solo layer ed è computazionalmente più veloce.

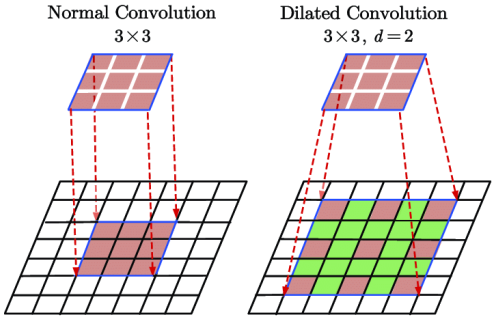

Figure 2.4: Dilated Convolution

2.6 Implementazione Modello: ResNet-18

Il modello usato per affrontare il problema in questione è **ResNet-18**, una rete neurale convoluzionale di 18 strati studiata per avere buone performance su problemi di segmentazione di immagini.
ResNet-18 è stata implementata in MATLAB tramite l'ausilio dell'applicazione *designDeepLabV3plus*.
designDeepLabV3plus permette di creare un modello DeepLab v3+ per problemi di semantic segmentation, usa un'architettura encoder-decoder, dilated convolutions, e skip connections ed è utilizzabile scaricando un'estensione chiamata *Deep Learning Toolbox*.
Al modello "base" di ResNet-18 fornito da designDeepLabV3plus, sono state fatte le opportune modifiche al fine di adattarlo al caso di studio.

2.7 Dataset

Per addestrare una CNN è richiesto in input un data-set composto da due distinti insiemi di immagini.
Il primo insieme è composto dalle immagini da segmentare, il secondo insieme è invece composto dalle label mask, ovvero matrici della stessa dimensione degli input, i cui elementi sono le etichette associate al pixel corrispondente nell'immagine da segmentare.
Le maschere sono ottenute dall'operazione di labeling, ossia l'etichettatura dei pixel, condotta manualmente su ogni singola immagine del dataset. Il data-set utilizzato per l'addestramento della rete in questione è **Kvasir**.
Kvasir mette a disposizione 880 differenti immagini di polipi al colon e le relative maschere binarie che discriminano le regioni rappresentanti il polipo dal background.

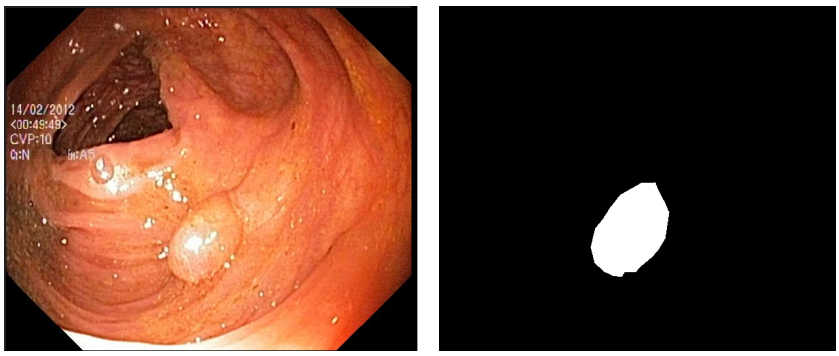

Figure 2.5: SX Polyph Image, DX Polyph Mask

2.8 Metriche di valutazione

La valutazione del modello, al variare della loss function, è fatta tramite alcune tra le più rilevanti metriche di valutazione di modelli di image segmentation. Le funzioni ricevono in input le label di ground truth e quelli derivate dalla predizione del modello su di un *test set* di input. In particolare, sono state usate: mIoU, mDice, Precision, Recall, Accuracy.

2.8.1 TP - TN - FP - FN

Per semplificare la definizione delle metriche di valutazione, conviene definire la matrice di confusione come prodotto logico tra ground truth e predicted value. Il risultato, nel caso di due classi, sarà il seguente:

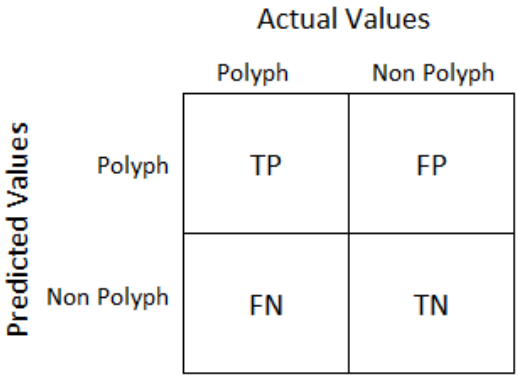

Figure 2.6: Matrice Di Confusione 2 Classi

2.8.2 mIoU - Mean Intersection over Union

Intersection over Union, noto anche come coefficiente di similarità di Jaccard, è una metrica molto usata per problemi di segmentazione. Per ogni classe, l'IoU è il rapporto tra pixel correttamente classificati e la somma del numero complessivo di pixel etichettati e di pixel classificati:

$$IoU = \frac{|Y \cap T|}{|Y \cup T|} = \frac{TP}{TP + FP + FN} \tag{2.2}$$

Verrà quindi eseguito il calcolo della metrica per ogni classe, per tutte le immagini del test set e ne verrà fatta una media aritmetica. Nel caso il dataset fosse molto sbilanciato (una classe è molto più presente dell'altra), è possibile normalizzare mIoU con un coefficiente che tiene conto del numero di pixel totale di ogni singola classe.

2.8.3 mDice

La metrica Dice è definita come media su tutto il test set della 1.9, coefficiente di Sørensen-Dice:

$$DiceCoefficient = \frac{|Y \cap T|}{|Y| + |T|} = \frac{2TP}{2TP + FP + FN} \tag{2.3}$$

2.8.4 Precision

Metrica molto usata per la lettura dei risultati di un modello è la **precision**, essa ci dà un'idea della porzione di pixel classificati di classe i correttamente, sul totale dei pixel classificati come i:

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

2.8.5 Recall

Recall, a differenza di Precision, ci da un'idea della porzione di pixel classificati correttamente di classe i, sul totale dei pixel di classe i:

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

2.8.6 Accuracy

L'**accuracy** globale di un modello è la frazione di pixel correttamente classificati, indipendentemente dalla classe, sul totale.
In generale, l'accuratezza non è molto precisa ed affidabile, soprattutto in presenza di dataset sbilanciati.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

Dalla 2.6 si può notare che il modello è un classificatore ideale se FP = FN = 0.

2.9 Risultati

Fra le metriche sopra citate è stata scelta la mDice come più significativa per fornire un punteggio al modello e per poter, in questo modo, valutarne le performance.
Sono state implementate in **MATLAB R2021b** le loss-functions presentate nel capitolo 1.4, applicandole al modello presentato al capitolo 2.6 ed i risultatati sono i seguenti:

Loss Function utilizzata	**Score mDice**
Dice Loss	0,8271
Focal_Tversky	**0,8413**
Focal_Dice	0,8377
Focal Loss	0,6253
Binary Cross Entropy	0,8
Logcosh Dice	0,7745
Logcosh Focal Tversky	0,8067
Top-k loss	0,768

Table 2.1: Risultati Loss Function In Leteratura

3 IMPLEMENTAZIONE NUOVE LOSS

Dopo aver implementato le Loss Functions presenti in letteratura ed averne studiato le relative prestazioni, sono state sviluppate e testate altre loss ex-novo.
In particolare, le seguenti:

- LogCosh Tversky
- Exponential Logarithmic Dice
- Exponential Logarithmic Loss
- Ensemble di Loss
  - Exponential Logarithmic Dice + Focal Tverski
  - LogCosh Dice + Focal Tversky
  - LogCoshDice + Exponential Logarithmic Dice + LogCosh Tverski
  - Exponential Logarithmic Dice + Focal Tversky + Focal CrossEntropy
  - Exponential Logarithmic Dice + Focal Tversky + binary focalLoss
  - Focal Tversky + binary focalLoss
  - TopK Loss + Exponential Logarithmic Dice

3.1 LogCosh Tversky

LogCosh_Tversky è una variante della Tversky Loss [1.12]. In aggiunta alla precedente, è stata calcolata la loss function applicandole la funzione LogCosh come segue:

$$L_{LCTversky}(Y,T) = LogCosh(L_{Tversky}(Y,T)) \tag{3.1}$$

3.2 Exponential Logarithmic Dice

Exponential Logarithmic Dice è una variante della funzione di loss Dice [1.10] che, a differenza della funzione standard, vengono applicati un coefficiente ad esponente e la funzione logaritmo come segue:

$$L_{ELDice}(Y,T) = log(L_D(Y,T))^{\gamma} \tag{3.2}$$

3.3 Exponential Logarithmic Loss

Viene fatta una media pesata di Exponential Logarithmic Dice [3.2] e di Cross Entropy, come proposto da [5] [1.16] con $w_{Dice}, w_{CE} \in [0,1]$ e $w_{Dice} + w_{CE} = 1$, ottenendo in questo modo:

$$L_{EL}(Y,T) = w_{Dice} L_{ELDice} + w_{CE} L_{CE} \tag{3.3}$$

3.4 Ensemble di Loss

Un altro approccio utilizzato per la creazione di nuove loss function è quello di creare degli ensemble (= insiemi) di loss function presenti in letteratura e/o create ex novo, facendo poi una media pesata dei valori ottenuti.
L'idea di aggregare più loss function è data dall'esigenza di tenere conto di diverse caratteristiche, rilevate da diverse funzioni.
Verranno ora elencati gli ensemble sviluppati, i coefficienti α, β, γ, ove presenti, sono tali per cui $\alpha, \beta, \gamma \in [0,1]$

Exponential Logarithmic Dice - Focal Tverski: Media pesata delle due loss [3.2] e [1.19]

$$L_{ELDiceFT} = \alpha L_{ELDice} + (1-\alpha) L_{FT} \tag{3.4}$$

LogCosh Dice - Focal Tversky: Media pesata delle due loss [1.20] e [1.19]

$$L_{ELFT} = \alpha L_{LC_{Dice}} + (1-\alpha) L_{FT} \tag{3.5}$$

LogCosh Dice - Exponential Logarithmic Dice - LogCosh Tverski: Media pesata delle tre loss [1.20], [3.2] e [3.1]

$$L_{LCDiceELDiceLCTversky} = \alpha L_{LCDice} + \beta L_{ELDice} + \gamma L_{LCTversky} \tag{3.6}$$

Exponential Logarithmic Dice - Focal Tversky - LogCosh Tversky: Media pesata delle tre loss [3.2], [1.19] e [3.1]

$$L_{ELDiceFTLCTversky} = \alpha L_{ELDice} + \beta L_{ELDice} + \gamma L_{LCTversky} \tag{3.7}$$

Exponential Logarithmic Dice - Focal Tversky - Binary FocalLoss: Media pesata delle tre loss [3.2], [1.19], [1.4.5]

$$L_{ELDiceFTBFL} = \alpha L_{ELDice} + \beta L_{FT} + \gamma L_{BFL} \tag{3.8}$$

Focal Tversky - Binary FocalLoss: Media pesata delle due loss [1.19] e [1.4.5]

$$L_{FTBFL} = \alpha L_{FT} + (1-\alpha) L_{BFL} \tag{3.9}$$

TopK - Exponential Logarithmic Dice: Media pesata delle due loss [1.21] e [3.2]

$$L_{TKELDice} = \alpha L_{TK} + (1-\alpha) L_{ELDice} \tag{3.10}$$

3.5 Risultati

Applicando le loss functions presentate nel capitoli [3.1], [3.2], [3.3], [3.4] al modello presentato al capitolo [2.6] e facendo il training con il dataset presentato al [2.7], i risultatati ottenuti sono i seguenti:

Loss Function utilizzata	**Score mDice**
LogCosh Tversky	0,7629
Exponential Logarithmic Dice	0,8343
Exponential Logarithmic Loss	0,7690
Exponential Logarithmic Dice + Focal Tverski	0,8471
LogCosh Dice + Focal Tversky	0,8445
LogCosh Dice + Exponential Logarithmic Dice + LogCosh Tverski	**0,8550**
Exponential Logarithmic Dice + Focal Tversky + Focal CrossEntropy	0,8507
Exponential Logarithmic Dice + Focal Tversky + Binary FocalLoss	0,8445
Focal Tversky + Binary FocalLoss	0,8296
TopK Loss + Exponential Logarithmic Dice	0,8211

Table 3.1: Risultati Loss Function Implementate

3.6 Validazione

La seguente immagine [3.1] rappresenta l'output della rete neurale avendo in input due immagini random di polipi, non usate nel training set.
Le immagini sono state realizzate sovrapponendo l'immagine di input con la maschera di segmentazione in output e fornendo ad essa una trasparenza del 50%.
L'immagine a sinistra fa riferimento alla maschera "ideale".
L'immagine in centro fa riferimento alla maschera in output avendo come loss function l'ensemble di LogCosh Dice + Exponential Logarithmic Dice + LogCosh Tverski.
L'immagine a destra fa riferimento alla maschera in output avendo come loss function Focal Tversky.

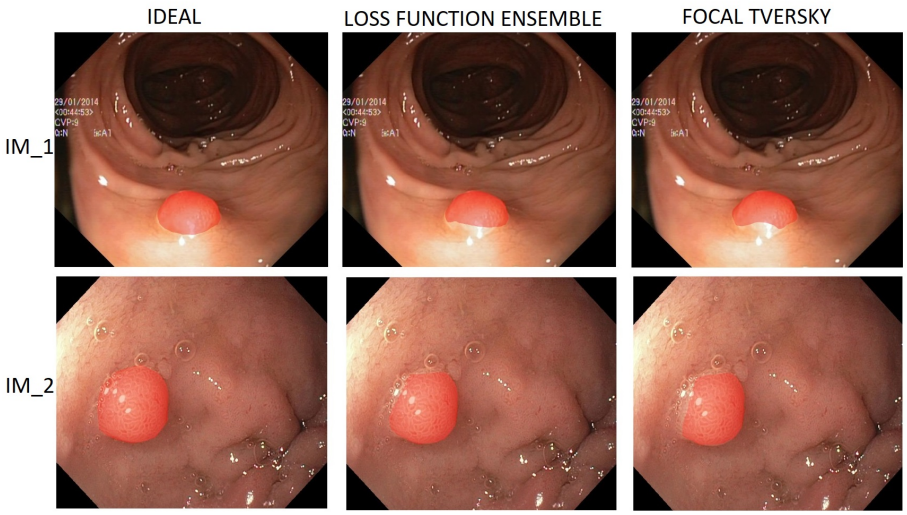

Figure 3.1: Output Rete

Si può notare che entrambi i modelli hanno avuto difficoltà nel segmentare i polipi nelle parti che presentano luminosità, questo può essere dato da un training set che non generalizza bene il problema oppure dall'errore intrinseco della rete, tuttavia l'ensemble di loss in questione è riuscito, sfruttando le caratteristiche delle loss usate, a generalizzare meglio il problema nel dato training set.

4 CONCLUSIONI

La semantic segmentation è uno strumento fondamentale per l'analisi di immagini in campo medico. Identificare anomalie sul corpo umano e delinearne la forma è di estrema importanza al fine di intervenire tempestivamente sulle malattie alla base e poter così curare il paziente.
In questa tesi, è stato studiato il comportamento di ResNet-18, al variare della propria loss function in un problema di segmentazione di polipi al colon.
È stata proposta un'analisi analitica dei risultati ottenuti dalla quale si evince che lo stato dell'arte non è preciso abbastanza da sostituire un umano sul rilevamento e sulla segmentazione dell'anomalia presente nell'immagine fornita dal colonscopio. Il modello può essere tuttavia integrato in un apparecchio per colonscopie come ausilio medico di analisi in tempo reale.
I lavori futuri legati a questa tesi sono relativi all'implementazione di nuove loss functions ex-novo, cercando correlazioni spaziali fra i pixel come nel caso di Neighbor Loss.

Bibliography

[1] Nabila Abraham and Naimul Mefraz Khan. *A Novel Focal Tversky loss function with improved Attention U-Net for lesion segmentation*. 2018. arXiv: 1810.07842 [cs.CV].

[2] Liang-Chieh Chen et al. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. arXiv: 1802.02611 [cs.CV].

[3] Shruti Jadon. "A survey of loss functions for semantic segmentation". In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)* (Oct. 2020). DOI: 10.1109/cibcb48159.2020.9277638. URL: http://dx.doi.org/10.1109/CIBCB48159.2020.9277638.

[4] Davood Karimi and Septimiu E. Salcudean. *Reducing the Hausdorff Distance in Medical Image Segmentation with Convolutional Neural Networks*. 2019. arXiv: 1904.10030 [eess.IV].

[5] Ken C. L. Wong et al. "3D Segmentation with Exponential Logarithmic Loss for Highly Unbalanced Object Sizes". In: *Lecture Notes in Computer Science* (2018), pp. 612–619. ISSN: 1611-3349. DOI: 10.1007/978-3-030-00931-1_70. URL: http://dx.doi.org/10.1007/978-3-030-00931-1_70.

[6] Jong Chul Ye and Woon Kyoung Sung. *Understanding Geometry of Encoder-Decoder CNNs*. 2019. arXiv: 1901.07647 [cs.LG].