MRI Brain Segmentation

Tesina progetto Deep Learning

Matteo Foschi

535207

Sommario

Image segmentation

Image segmentation è un task affrontabile mediante reti deep che consiste nell'identificare e distinguere diversi oggetti all'interno di un'immagine.
Nel mio specifico caso sto andando a fare image segmentation di immagini relative a MRI ovvero (magnetic resonance imaging) radiografie cerebrali per l'identificazioni di tumori.
Si utilizzano reti deep con l'obiettivo di fornire dati più grezzi alla nostra architettura, in maniera da permettergli di identificare autonomamente le caratteristiche più rilevanti per il nostro task. Assegnare alla rete l'incarico di fare un processamento delle immagini fornite.
Nel nostro specifico caso ci stiamo limitando nell'identificazione di una determinata area in un'immagine e non di classificare anche l'immagine segmentata.

Dataset

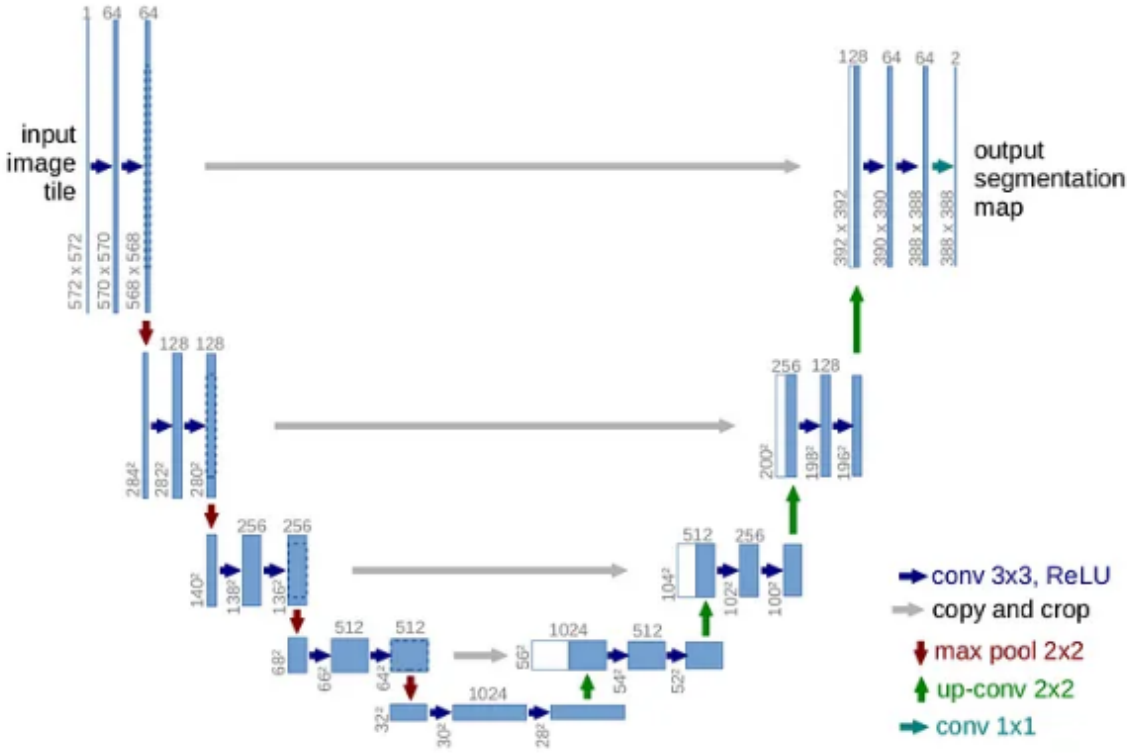
Il dataset è composto da immagini 256x256 con una profondità di 3 canali (RGB) e da maschere che servono ad identificare la zona dove è presente il tumore.

Data augmentation

Data augmentation viene utilizzato per addestrare una rete deep con pochi campioni.
Questo è possibile applicando delle trasformazioni ai miei esempi, "generandone" di nuovi (ruotando, facendo la trasposizione dell'immagine, alterare la scala di grigi, ecc..).
Questo fa sì che si abbiamo anche campioni tutti differenti l'uno dagli altri, in maniera da ridurre anche la specializzazione su una specifica feature, magari anche non rilevante ai fini del task, che la rete potrebbe andare ad identificare.
Nel nostro caso è stata fatta data augmentation del dataset di immagini, andando a fare un flip dell'immagine sia verticalmente, orizzontalmente e ruotando le immagini.

Unet

La Unet è una rete convoluzionale, sviluppata per fare image segmentation in ambito medico. Fa uso di data augmentation per ridurre il numero di campioni necessari, per addestrare la rete.
L'architettura prende delle immagini in input dall'encoder ed ha un decoder connesso ad esso mediante un bridge.
Sia l'encoder che il decoder sono composti da dei blocchi, sia connessi in serie, dove l'output di un blocco va in input al blocco successivo, sia in parallelo, dove ogni blocco dell'encoder è connesso mediante una skip connection al corrispettivo blocco del decoder.



Architettura Unet.

Quindi la rete è un'architettura encoder-decoder.

Encoder

Nella prima fase si hanno 4 blocchi che fanno uso di layer convoluzionali 3*3, sviluppati ed impiegati per l'identificazione di features map.
Questo perché vengono utilizzati per la combinazione ed identificazione di features, più ad alto livello. Viene utilizzata la batch normalization per la normalizzazione dei valori ed usata una relu come funzione di attivazione, per introdurre una non linearità.
Insieme ad essi si utilizzano dei max pooling layer 2*2 per ridurre la dimensionalità dell'informazione (ovvero l'immagine).
Ad ogni step di questa fase, vengono generate delle features map dai filtri dei layer convoluzionali.
Ogni blocco della prima fase genera un'output, che sarà fornito in input ad un blocco del decoder mediante una skip connection, oltre che a generare in output delle informazioni che andranno in input al blocco dell'encoder successivo.
Con la skip connection propago informazioni prima del max pooling, altrimenti ci sarebbe incompatibilità dimensionale con le immagini nei blocchi del decoder.

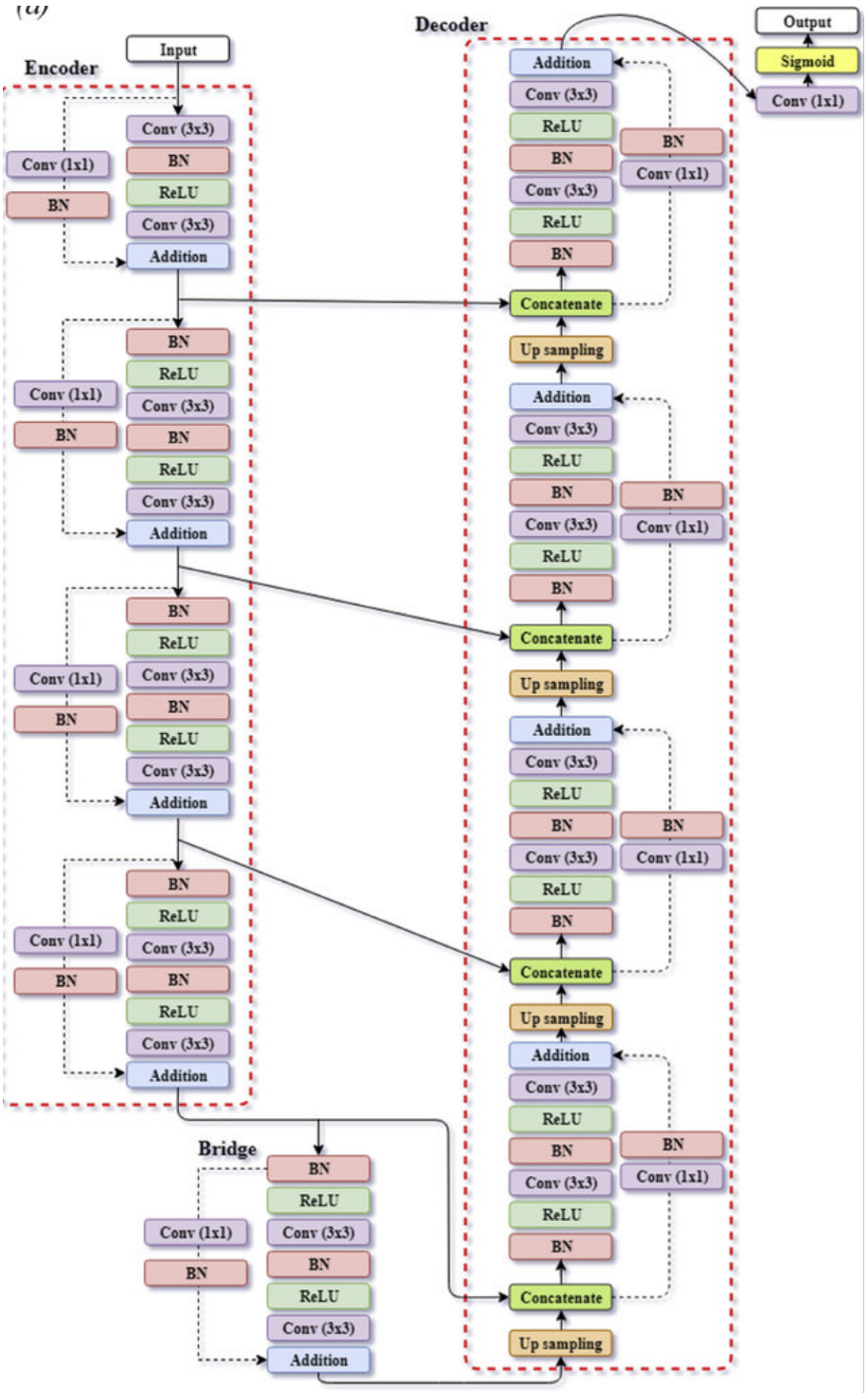
Decoder

Nella seconda fase invece, nel decoder, si hanno sempre 4 blocchi, dove ognuno di loro riceve due input, uno relativo al blocco dell'encoder con la skip connection ed uno dal blocco precedente.
I blocchi del decoder fanno mediante layer convoluzionali inversi 2*2 un'upsampling della dimensionalità di ciò che ricevono dal blocco precedente e poi concatenano i due input, avendo quindi più features map da elaborare.
L'input concatenato attraversa due layer convoluzionali sempre 3*3 con batch normalization e funzione di attivazione relu.
L'output dell'ultimo decoder passa attraverso un layer convolutivo 1*1, dove vengono ricompattate le features map ed il tutto passa per la funzione di attivazione, che è una sigmoide, che mi restituisce la maschera di segmentazione che mi classifica i pixel.

L'output generato, ovvero la maschera di segmentazione, dovrà avere dimensionalità pari alle immagini in input.
Il bridge è un blocco che esegue le stesse operazioni che esegue un blocco dell'encoder.
Nell'immagine infine possiamo vedere il numero di filtri applicati ad ogni layer convolutivo.

ResUnet

La ResUnet è un rete deep che consiste in un'evoluzione della Unet. E' sempre una rete encoder decoder connessa con un bridge connection.



Archtettura ResUnet.

L'architettura è un'encoder-decoder.

Encoder

L'encoder è diviso in 4 sottoblocchi.
Il primo prende le immagini in input e passano prima per un layer convoluzionale 3*3, poi viene fatta la batch normalization e poi applicata la funzione di attivazione relu.

L’output del blocco viene concatenato con l’input dello stesso blocco, che viene fatto passare per un layer convolutivo e applicata la batch normalization, mediante una pipeline parallela.
Mentre gli altri 3 blocchi dell’encoder, sono tutti uguali, ovvero fanno batch normalization, relu, layer convoluzionale 3*3, poi di nuovo batch normalization, relu e layer convolutivo 3*3. Dopo ogni blocco viene applicato un max pooling che ne dimezza la dimensione.
Alla fine di ogni blocco dell’encoder, l’output, viene concatenato con l'output generato in parallelo dall’atra pipeline, ovvero passandolo in un layer convoluzionale 1*1 e applicando la batch normalization allo stesso input ed i due output vengono messi insieme prima di passare nel pooling layer.

Decoder

Il decoder è anch'esso suddiviso in 3 sottoblocchi, ognuno prende in input l’output del corrispettivo sottoblocco dell'encoder mediante skip connection ed in più l'output del blocco precedente, concatenando l’informazione, ovvero le feature maps.
Inoltre viene fatto un upsampling dell’informazione, prima dell’elaborazione di ogni blocco nel decoder, mediante layer convolutivi inversi, raddoppiando la dimensionalità.
Nel decoder tutti i blocchi hanno la stessa struttura, ovvero due layer convolutivi con prima una batch normalization e poi una relu.
Anche qui troviamo una pipeline parallela, dove il suo output si va a sommare a quello del blocco del decoder prima di upsampling.

L’ultimo output passa per una sigmoide e restituisce una maschera di segmentazione.
Il bridge è un blocco che esegue le stesse operazioni che esegue un blocco dell’encoder, ma serve per poter eseguire un’elaborazione in più sulle immagini.
Nell’immagine sopra mostrata possiamo inoltre vedere il numero di filtri utilizzati ad ogni layer dell’architettura.

Skip connection

Le skip connection vengono utilizzate per passare informazioni aggiuntive al decoder, che sono ad uno stadio diverso di lavorazione.
Questo permette al decoder di avere più informazioni, dove alcune provengono dalla valle dell’encoder, mentre le altre non hanno subito tutto il processo di manipolazione.
Questo serve a migliorare le prestazioni e migliorare la rappresentazione.

Bridge

Il bridge mi permette di connettere in maniera sequenziale la rete, ovvero di connettere l’encoder al decoder.
E’ formato da due layer convoluzionali 3*3.

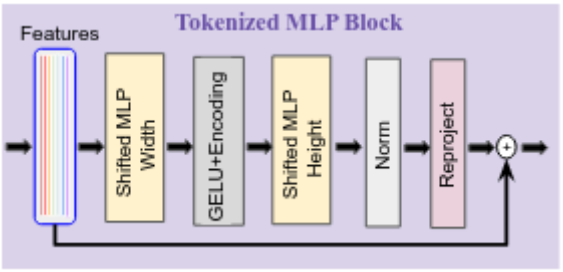
UNeXt

Questa architettura è basata sempre su una macro divisione encoder-decoder.

Encoder

L'encoder ha una prima parte (Convolutional stage) che viene utilizzata per dimezzare le informazioni in input, ad ognuno dei 3 step convolutivi.
Ognuno di essi implementa sia la batch normalization che layer convolutivi 3*3 con stride e strip pari ad 1 ed una ReLu activation function.
Dopo ogni layer convolutivo viene eseguito il max pooling con stride pari a 2 che mi dimezza la porzione dell'immagine.
Il primo layer convolutivi fa uso di 32 filtri, il secondo 64 ed il terzo 128.

Poi troviamo una parte, sempre nell' encoder, chiamata MLP stage divisa in più step:

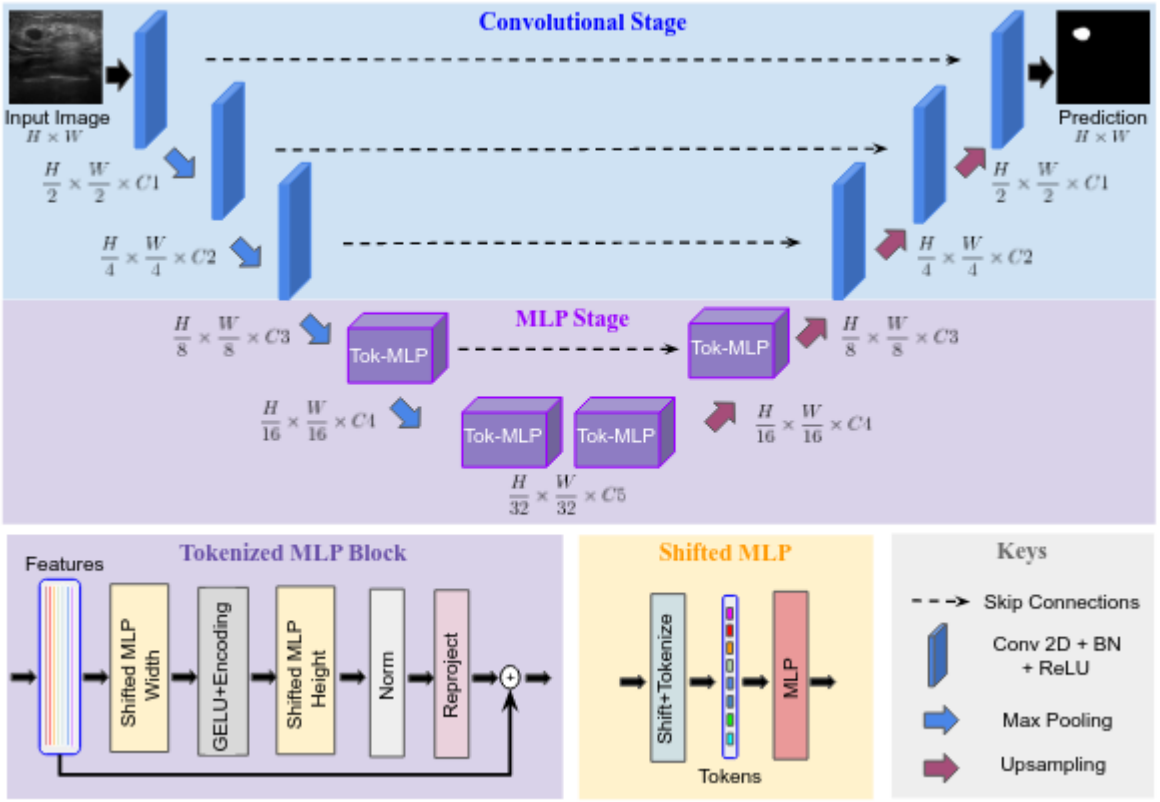


Tok-MLP della UNeXt.

prende in input le features map generate dai layer convolutivi precedenti ed esegue una Shifted MLP width, ovvero va a ridurre la dimensione dell'immagine, la larghezza, così da aumentare il focus su porzione di immagini che verranno "tokenizzate". Poi c'è un'altro step di Shifted MLP Height che fa le stesse operazioni ma riducendo l'altezza dell'immagine.
Dopo entrambi questi step, di riduzione, genera dei token mediante deep wise convolutional layer (DW-Conv) per aggiungere maggiore informazione posizionale alle features, ed ha molti meno parametri rispetto ad altre tecniche, risultando quindi più performante.
Vengono utilizzati per aumentare l'efficienza della rete quando le dimensioni delle immagini di test e di training non sono perfettamente identiche, quindi l'accuracy e l'errore non sono molto buoni.
In questo step come funzione di attivazione viene utilizzata la GELU.
A valle troviamo infine una normalizzazione.

Decoder

Mentre nell'ultima fase del decoder vengono implementati layer convolutivi inversi per aumentare l'informazione e riportare l'immagine alla dimensione di partenza per il calcolo dell'errore, ed ogni componente del decoder ha in input non solo l'elaborazione del layer precedente ma anche delle skip-connection al relativo layer di encoder.

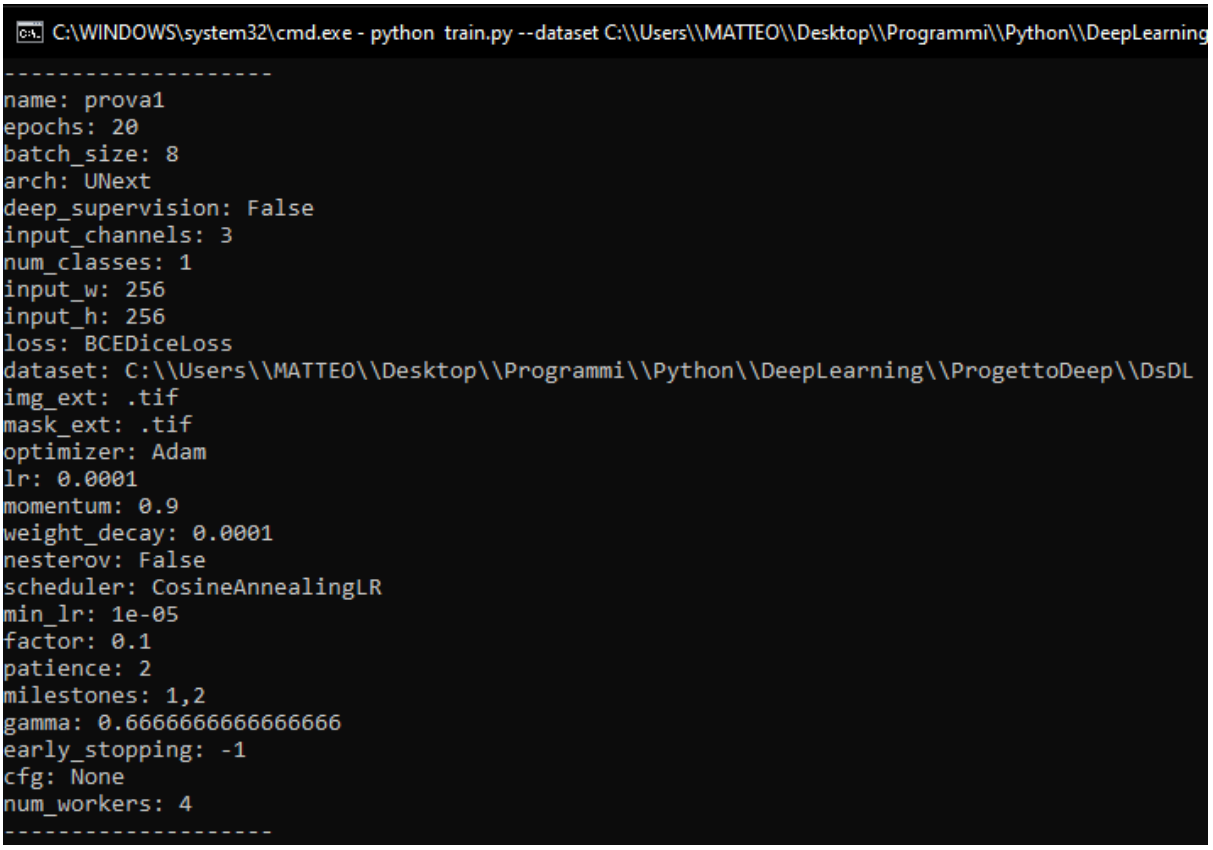


Architettura UNeXt.

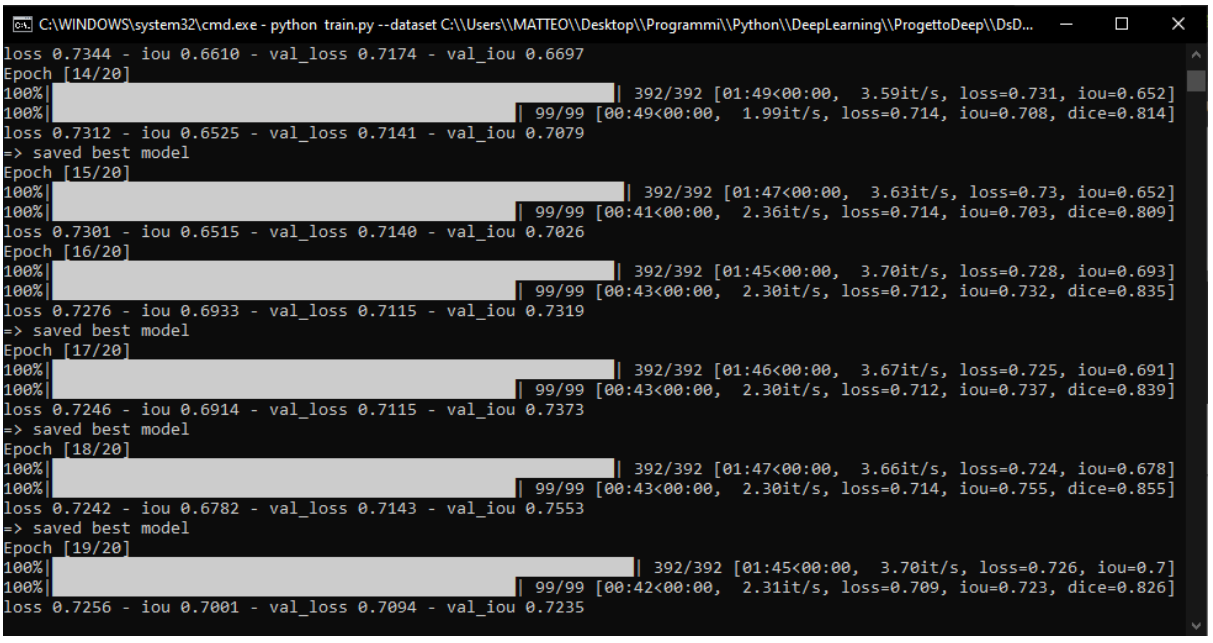
Risultati

1° Configurazione

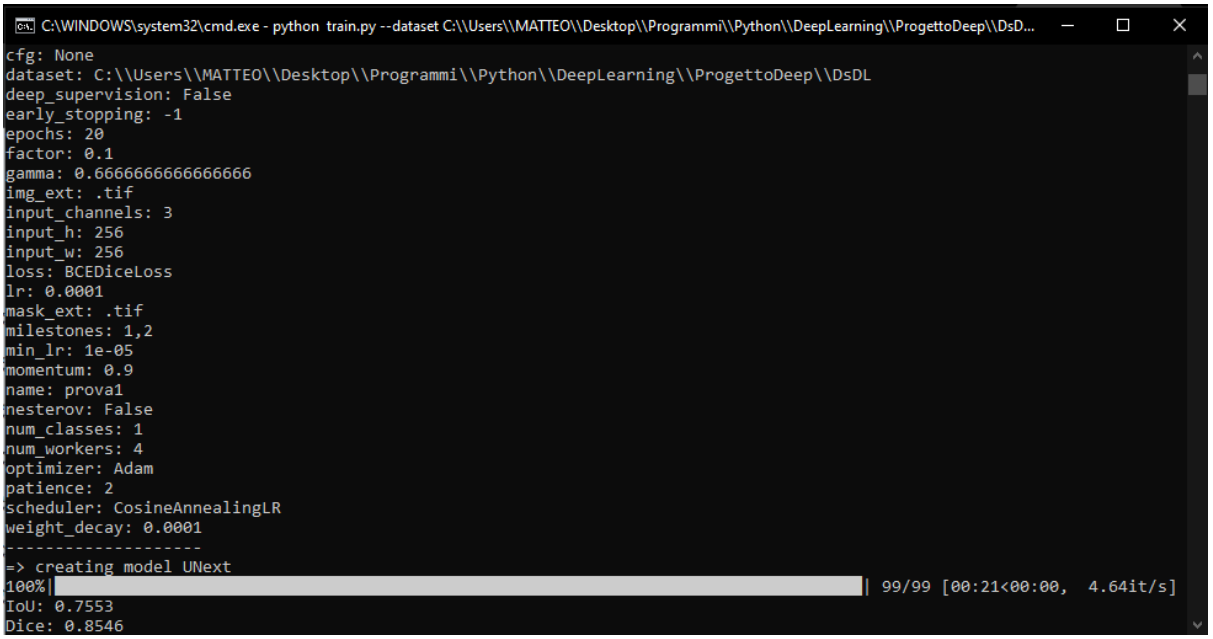
Nel primo screen possiamo vedere la configurazione del modello utilizzata per la prima fase di training, mentre nel secondo alcune epoche di training del modello. Come si può vedere era attivo l'early stopping, quindi il modello alla fine di ogni epoca andava a fare una validazione, su un set di campioni estratti dal training set, in modo da salvare il modello ogni volta che si ottenevano risultati migliori ed ignorare quelli invece dove il modello o inizia a regredire di performance, in modo da poter ristabilire la configurazione migliore ottenuta e salvare i pesi della rete.
L'ultimo modello salvato e migliore ottenuto non è detto che sia quello dell'ultima epoca.



Configurazione 1° run della UNeXt.

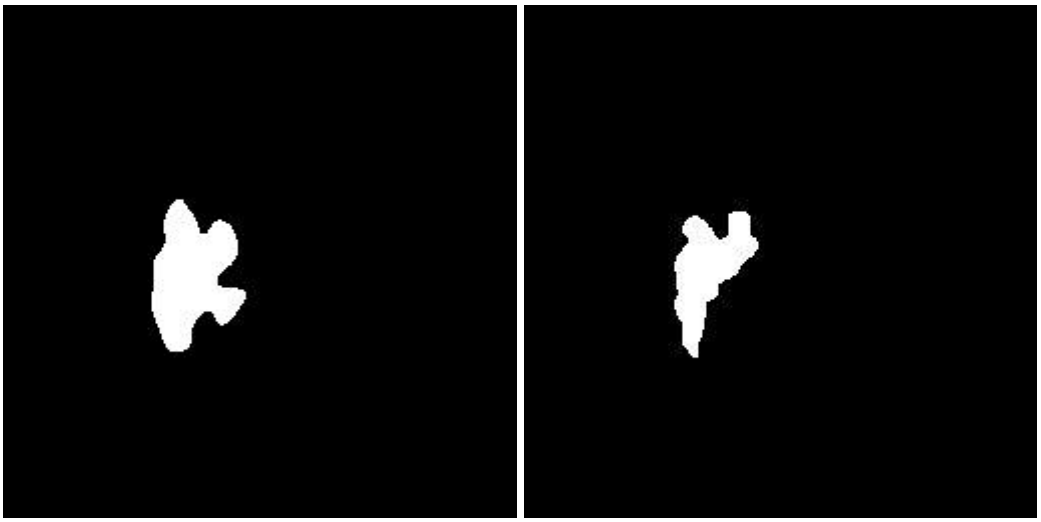


Training della UNeXt.



Test della UNeXt su test set.

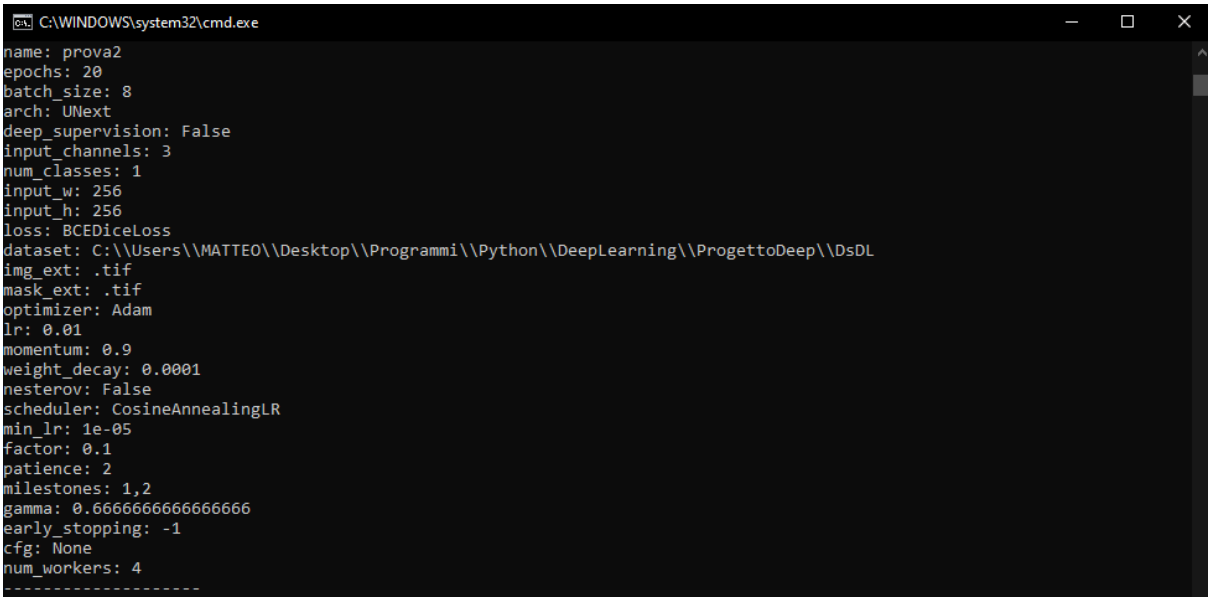
Immagine UNeXt Maschera reale



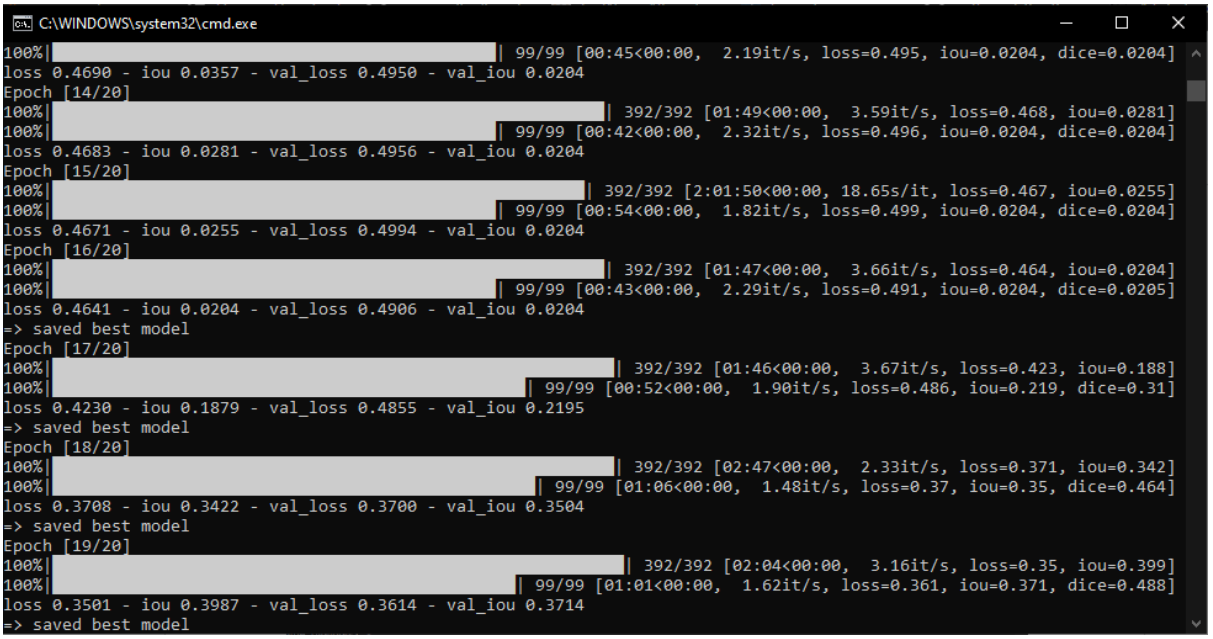
Confronto tra la maschera reale e quella generata dalla UNeXt.

2° Configurazione

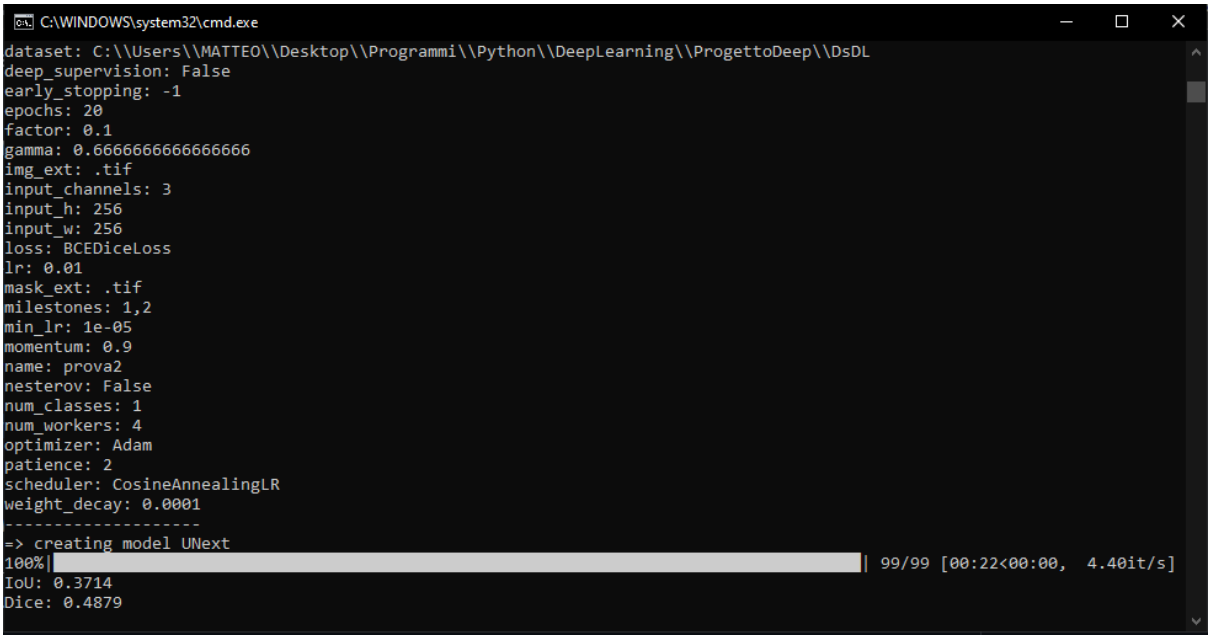
Rispetto alla 1° abbiamo aumentato il learning rate di un fattore 100, e si può vedere come le performance rispetto al primo run siano già diminuite, per via di uno spostamento troppo repentino durante il training.



Configurazione 2° run della UNeXt.

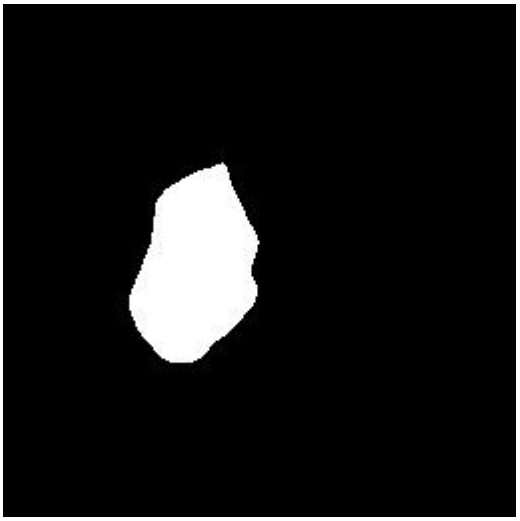


Training della UNeXt.

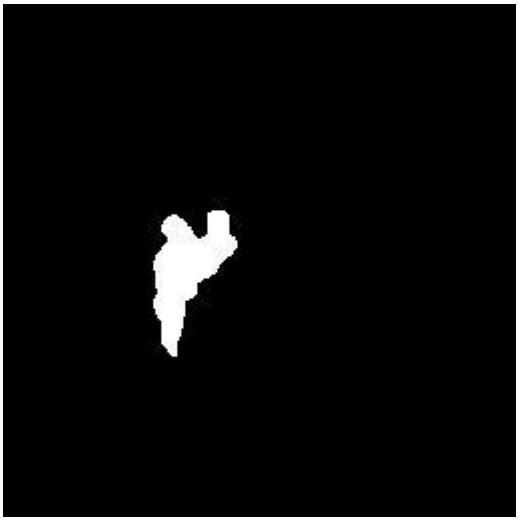


Test della UNeXt su test set

Immagine UNeXt



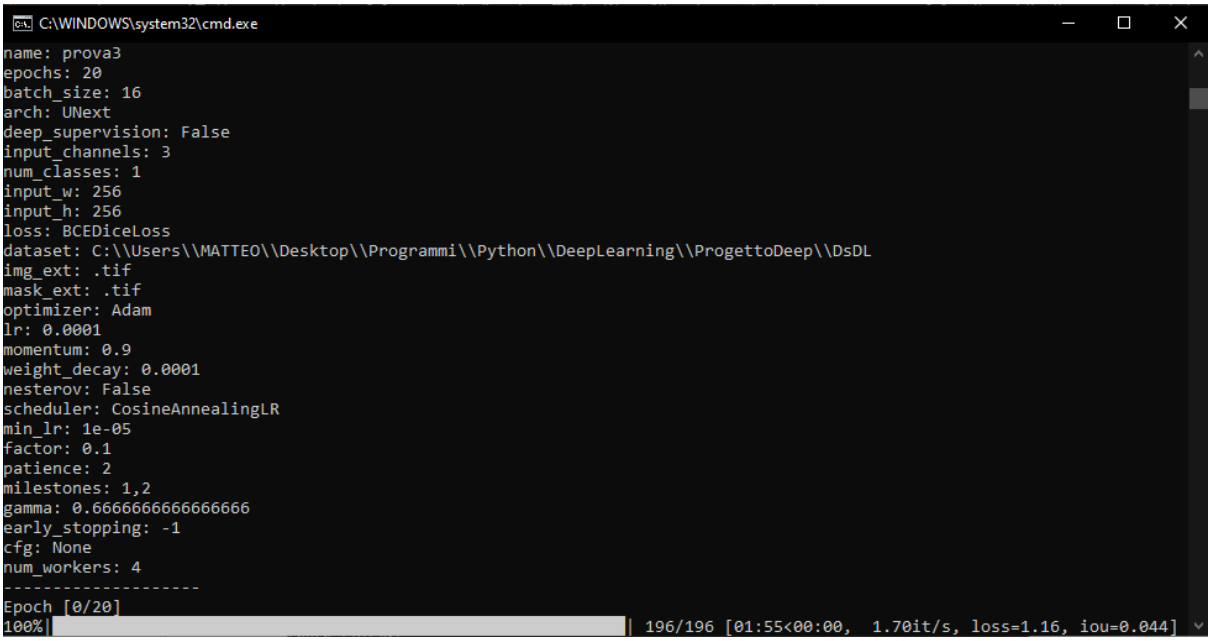
Maschera reale



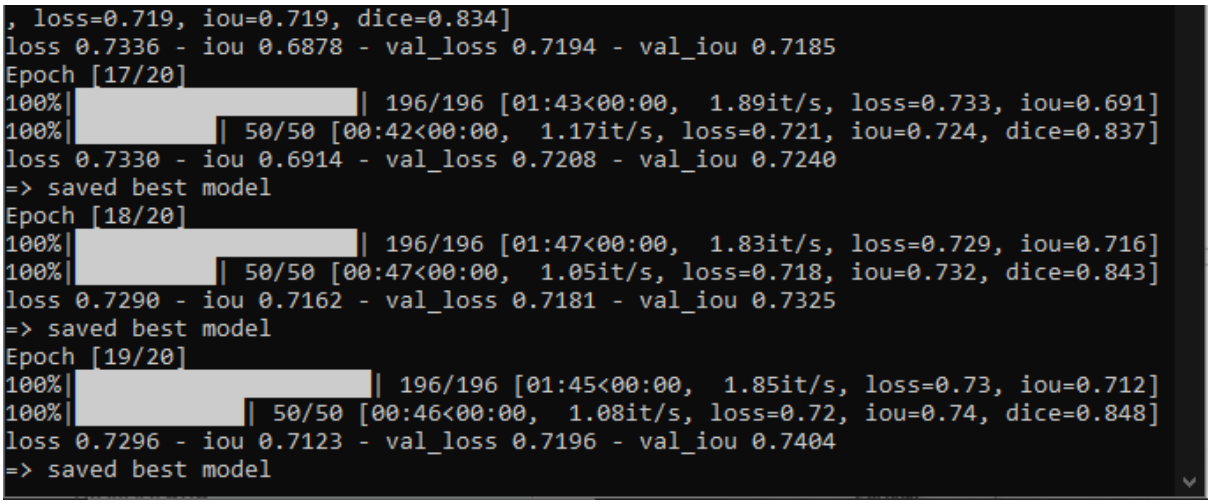
Confronto tra la maschera reale e quella generata dalla UNeXt.

3° Configurazione

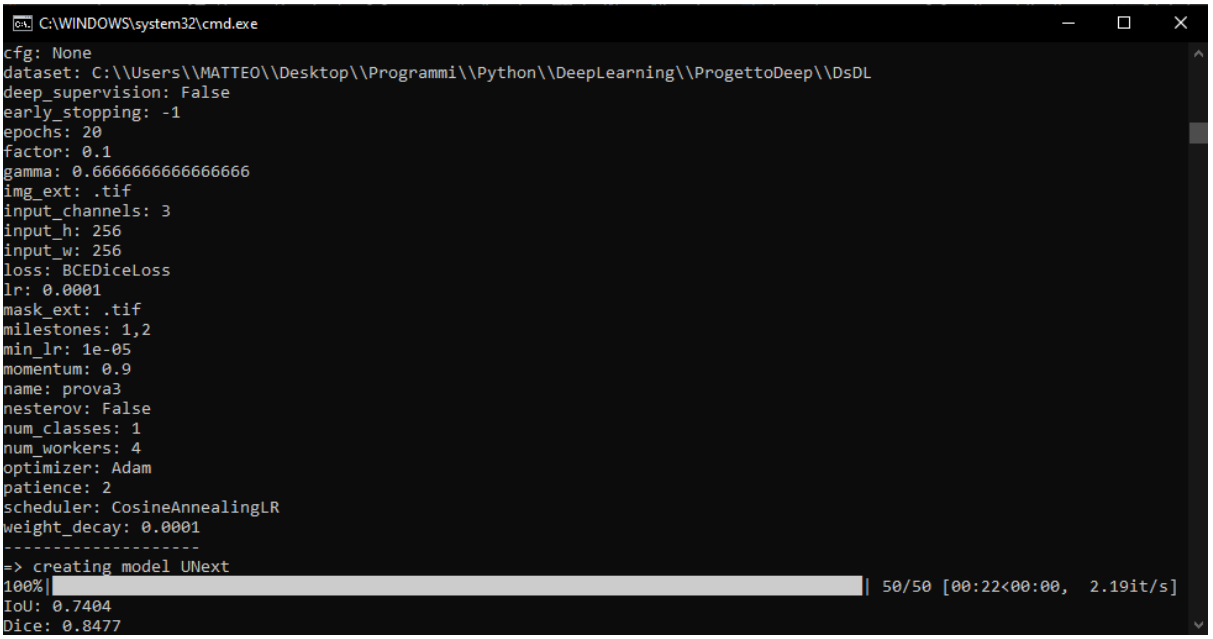
Rispetto alla 1° abbiamo aumentato la dimensione dei batch size a 16, ovvero dopo ogni quanti campioni fare l'aggiornamento dei parametri. Rispetto alla 1° configurazione, perchè la 2° ci ha mostrato un peggioramento.



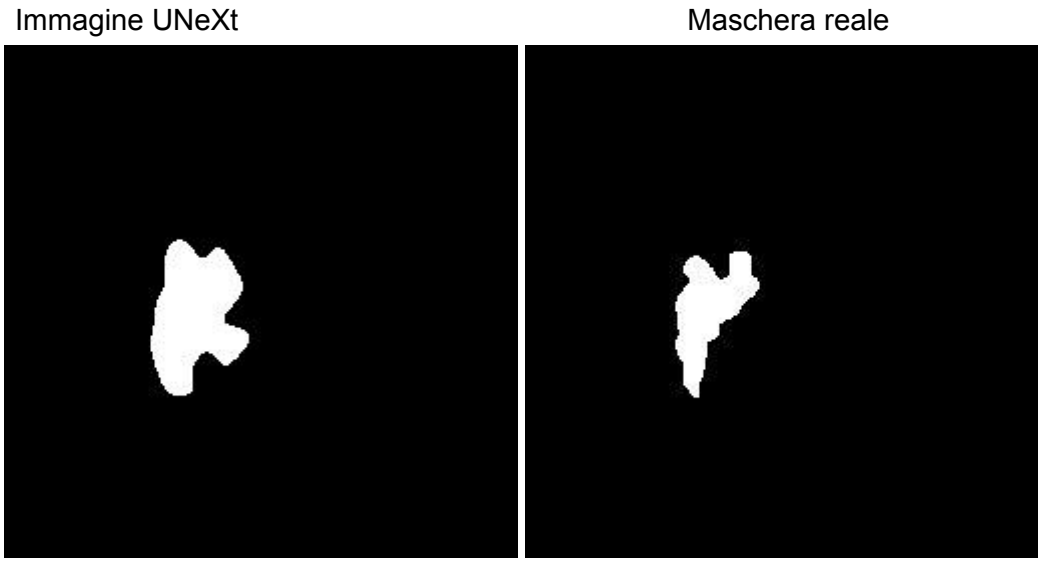
Configurazione 3° run della UNeXt.



Training della UNeXt.



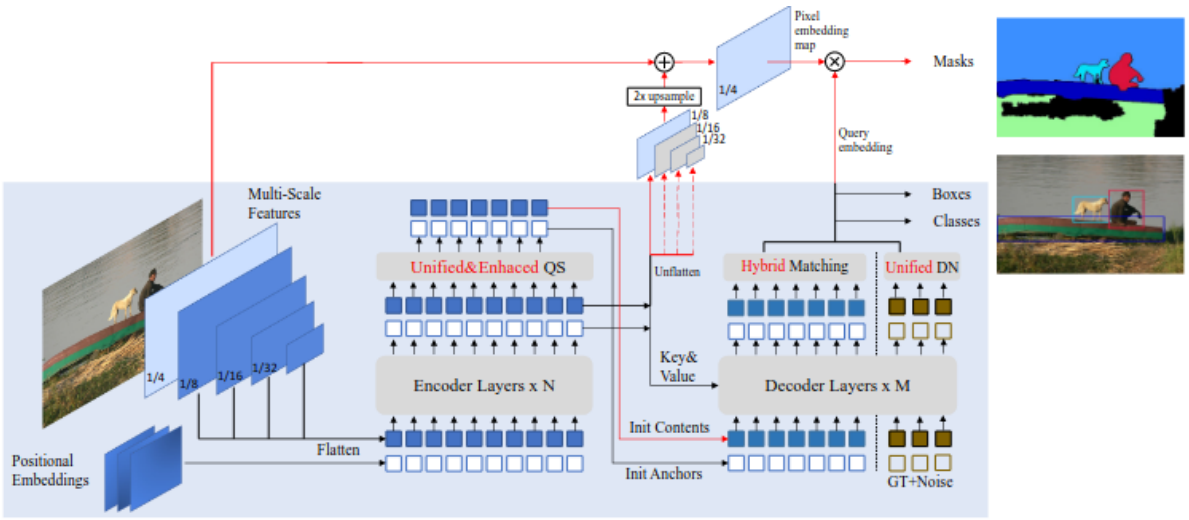
Test della UNeXt su test set



Confronto tra la maschera reale e quella generata dalla UNeXt.

Mask-Dino

Mask-Dino è un'evoluzione dell'architettura Dino, la quale si basa sulla struttura dei Transformer (con un encoder ed un decoder), ma con l'aggiunta di caratteristiche specifiche per il task di segmentazione. L'architettura dino non nasce per image segmentation ma per object detection e classification.



Architettura Mask-Dino.

Come possiamo vedere nell'immagine della mask-dino, fa uso dei positional embeddings, ovvero embeddings relativi ad informazioni posizionali delle varie porzioni dell'immagine rispetto alle altre, questo perché per l'identificazione degli oggetti è importante conoscere la loro posizione spaziale.
Mentre l'immagine in input viene passata all'interno di layer convolutivi, nell'intento di identificare features più ad alto livello da essa.
Gli output di queste due elaborazioni vanno in input al pre-annunciato encoder formato da più layers, ed il suo output finirà in input al decoder.
Fin qui l'architettura Dino e Mask-Dino sono uguali, ma ora troviamo un branch specifico per la segmentazione.
Questo branch si ispira all'architettura di mask2former, dove si va a generare una mappa di embeddings relativi ai pixel fondendo l'output dell'encoder del transformer dopo aver fatto un upsampling dell'immagine, da ⅛ ad ¼ (Ce); le immagini scalate ad ¼ (Cb) dai layer convolutivi iniziali ed infine viene aggiunto l'output del decoder del transformer, dopo averlo fatto passare in una funzione di interpolazione.
Li fondiamo ed otteniamo la nostra pixel embedding map (M()).

Per ricavare la nostra maschera di segmentazione andiamo ad eseguire il prodotto scalare tra la pixel embedding map e l'output del decoder del nostro transformer, ovvero la content query embedding (qc).

$$m = q_c \otimes \mathcal{M}(\mathcal{T}(C_b) + \mathcal{F}(C_e)),$$

Cb: features map ottenute dalla backbone scalando l'immagine ad ¼
Ce: output del decoder
m: output mask
qc: content query embedding
M(..): pixel embedding map
F(..): interpolation function
T(..): layer convolutivi per fare down scale dell'immagine

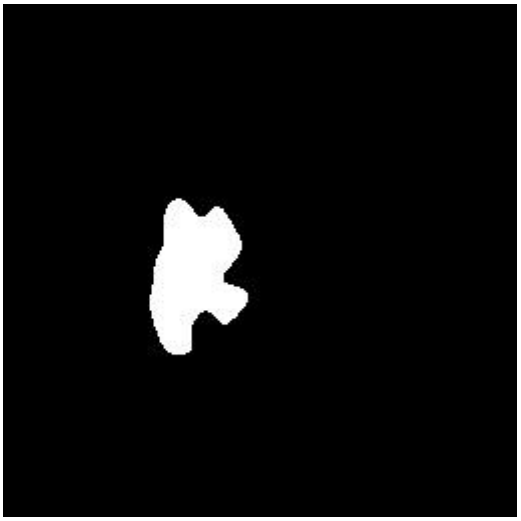
qc è l’output del nostro decoder, il quale prende in input non solo l’output dell'encoder ma anche altre informazioni come l'output dello Unified Enhanced QS.
Le unified enhanced sono utilizzate per selezionare solo le regioni di un'immagine che sono rilevanti per l'elaborazione dell'oggetto di interesse, per ridurre il numero di operazioni, ma in questo caso ci servono per ulteriori estrazioni di features a partire non da immagini, ma features maps.
infine nel decoder troviamo la unified denoising che viene addestrata/utilizzata per la rimozione del rumore all’interno dell’immagine per migliorare la segmentazione.
Come possiamo vedere dall’immagine l’architettura non va solo ad eseguire la segmentazione di immagine, ma è in grado di fare anche classificazione ed object detection.

Conclusioni

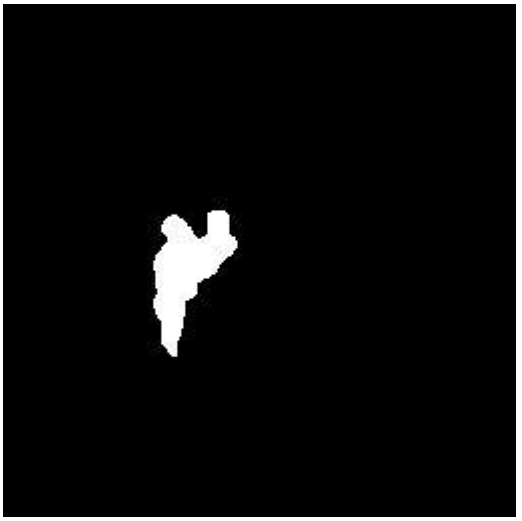
In conclusione possiamo vedere il confronto tra l'accuracy tra i vari modelli.
Ovvero la Unet e la ResUnet, hanno performance ancora migliori rispetto ad altre architetture anche più recenti come UNeXt.
Questo è possibile osservarlo anche nella comparazione delle maschere sotto riportate (quella generata e quella originale).
Le diverse accuracy sono date dalla precisione di margini identificati.
Tutti e tre i modelli sono stati in grado di riconoscere la presenza o meno dell’oggetto, ma la con una precisione differente dei margini.

Architettura	**Acc%**	**Dataset**
Unet	97%	mri segmentation
ResUnet	90%	mri segmentation
UNeXt	85%	mri segmentation
Mask-Dino	61%	generale

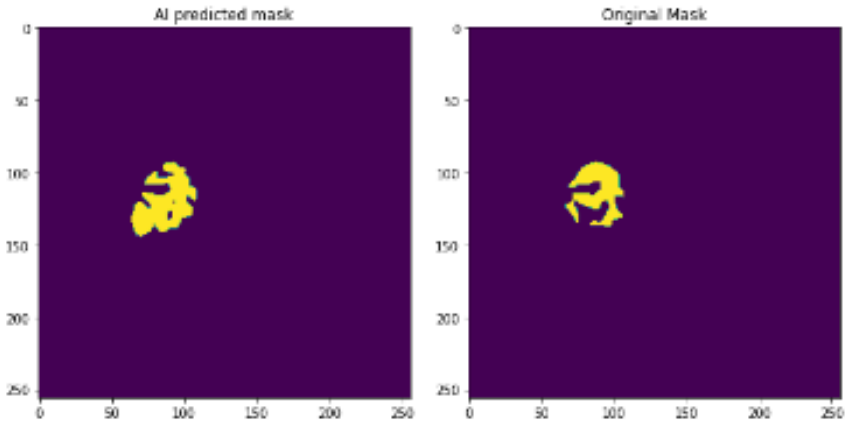
Immagine UNeXt



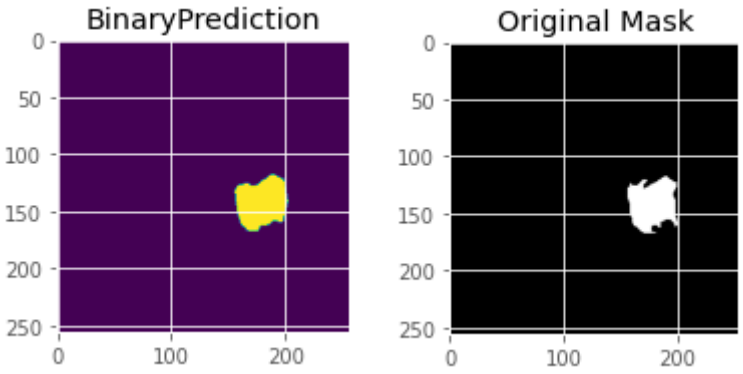
Maschera reale



Confronto tra la maschera reale e quella generata dalla UNeXt.



Confronto tra la maschera generata dalla ResUnet e quella reale.



Confronto tra la maschera della Unet e quella originale.

Bibliografia

Mask DINO: https://arxiv.org/pdf/2206.02777.pdf

UNetXt: https://arxiv.org/pdf/2203.04967.pdf

UnexXt gitHub: https://github.com/jeya-maria-jose/UNeXt-pytorch

Dataset MRI Brain: https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation

Unet: https://www.mdpi.com/1424-8220/19/18/3859

Res-Unet: https://www.mdpi.com/2306-5354/9/8/368