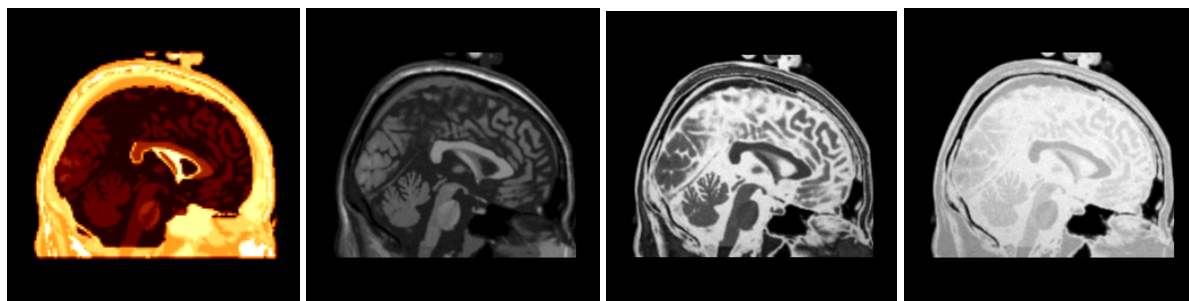


ESERCITAZIONE 10 25/05/2022 (CNN segmentazione)

Scopo dell'esercitazione è utilizzare una convolutional neural network (CNN) per la segmentazione di immagini biomediche. In particolare verrà utilizzata una U-NET 2D per segmentare la materia cerebrale.

I dati provengono da un simulatore MR, in particolare il simulatore MR cerebrale brainweb (<https://brainweb.bic.mni.mcgill.ca/>). Il simulatore parte da un atlante anatomico (a) e permette di definire il tipo di sequenza (T1, T2, PD), il thickness di fetta, il rumore, e la disomogeneità del campo RF. Nel nostro caso utilizziamo i dati tratti da un database di 20 cervelli normali. https://brainweb.bic.mni.mcgill.ca/anatomic_normal_20.html



Atlante

T1

T2

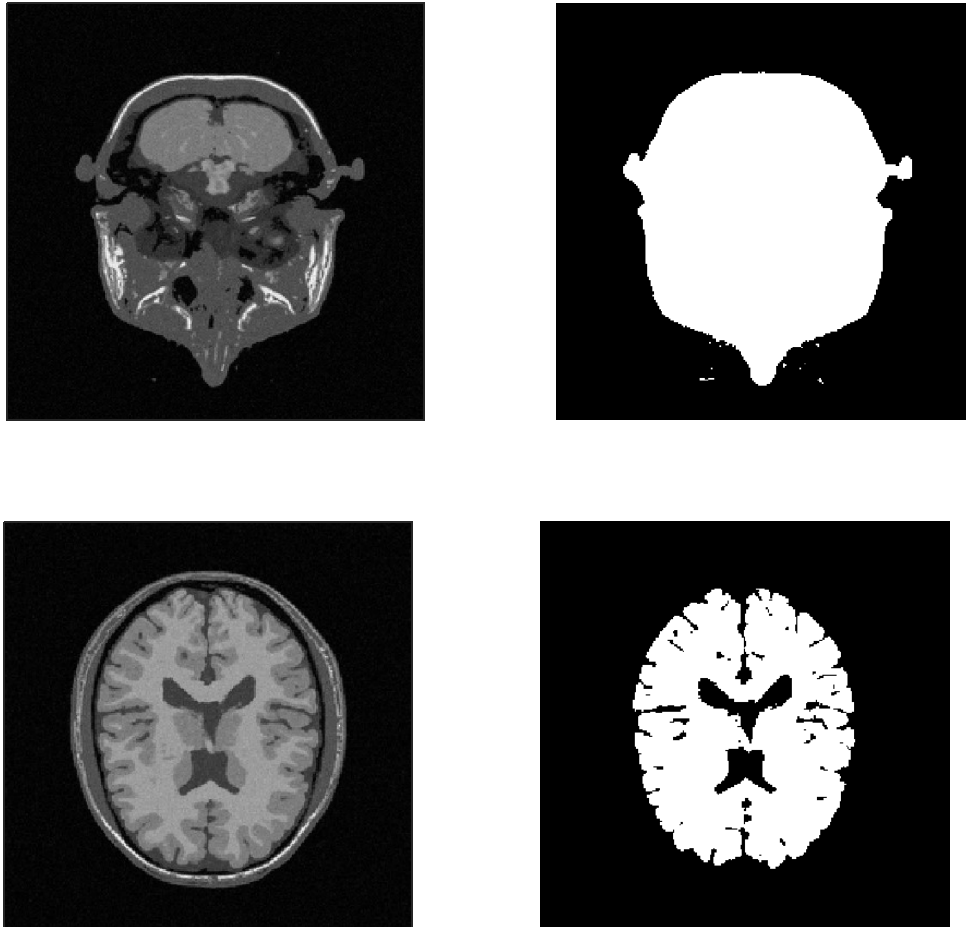
PD

Per ogni soggetto la simulazione è stata realizzata partendo da un set di dati fuzzy a 12 classi (0=Background, 1=CSF, 2=Gray Matter, 3=White Matter, 4=Fat, 5=Muscle, 6=Muscle/Skin, 7=Skull, 8=vessels, 9=around fat, 10 =dura matter, 11=bone marrow) per le quali abbiamo le 12 mappe di appartenenza. Questo tipo di data set tiene conto del PVE nel processo di formazione dell'immagine. Inoltre abbiamo un modello anatomico discreto che si ottiene assegnando ad ogni voxel l'indice della classe con il massimo valore di appartenenza. Il modello ha dimensioni 362x434x362. Infine il set di dati comprende per ogni paziente una immagine T1 simulata 3D di dimensioni 256x256x181. Essendo i FOV uguali, è necessaria una interpolazione per avere la stessa risoluzione spaziale sulle immagini e sulle maschere. Avendo solo 10 volumi non è possibile utilizzare un approccio 3D, quindi consideriamo tutte le slice dei vari volumi come immagini indipendenti che hanno in corrispondenza la loro maschera. Chiaramente è un approccio non ottimale in quanto si perde la continuità del modello 3D.

Nella directory DATA abbiamo i dati già interpolati.

- Nella cartella MR ci sono 1810 immagini 2D in formato png di dimensioni 256x256
- Nella cartella GRAY_MASK_C ci sono le corrispondenti maschere della materia cerebrale, cioè i pixel di classe 2 e 3 (materia grigia e materia bianca).

- Nella cartella GRAY_MASK_B ci sono le corrispondenti maschere di tutto il cranio, cioè i pixel di classe diversa da 0 (Background).



In figura sono riportati due esempi di corrispondenza tra immagini MR e maschere GT (sopra GRAY_MASK_B, sotto GRAY_MASK_C).

L'idea è di addestrare la U-Net a compiere un compito "facile", cioè segmentare l'intero cranio, e poi usare la rete addestrata come rete di partenza (transfert learning) per l'addestramento del compito "difficile", cioè segmentare la materia cerebrale.

Il dataset viene caricato con le funzioni ***imagedatastore*** che abbiamo visto in precedenza e ***pixellabelDatastore*** che carica le maschere. Se vogliamo eseguire un pre-processing sulle immagini dobbiamo definire una nuova funzione ReadFcn come visto in precedenza. Qui il cropping non è necessario perché le immagini del BrainWeb hanno già il FOV minimo. Il tempo di calcolo per

l'addestramento della rete senza utilizzare GPU può essere molto alto, quindi può essere opportuno ridimensionare le immagini per ridurre i tempi di calcolo, ad esempio passando da immagini 256x256 a immagini 128x128 o 64x64. Ovviamente vanno ridimensionate nello stesso modo anche le maschere.

Una volta caricato il data set con *imagedatastore* e *pixelLabelDatastore* lo dividiamo in training, validation, e test e definiamo la U-NET con *unetLayers*. Nel nostro caso l'ingresso della rete sarà una immagine 2D di dimensioni uguali a quelle che abbiamo definito nella creazione dei dati ed una uscita a due classi, quindi una maschera. Notiamo che aumentando il numero di classi potremmo segmentare più regioni contemporaneamente. La funzione *unetLayers* permette anche di personalizzare i parametri della rete, in particolare il numero di layer (*EncoderDepth*), la profondità dei layer (*NumFirstEncoderFilters*), la grandezza del kernel dei filtri convolutivi (*FilterSize*), etc. La U-NET "standard" ha 4 layer, profondità 32 e un filtro 3x3.

Il training della rete viene effettuato al solito attraverso la funzione

```
trainedUnet = trainNetwork(TrainingData,unet,options)
```

le opzioni di training vengono definite con la funzione `options = trainingOptions()`

alcune opzioni di interesse sono:

`optimizer`: tipo di ottimizzatore

`'InitialLearnRate'`: è il passo con cui viene esplorato il search space, se è troppo piccolo la convergenza sarà lenta, se è troppo grande si andrà in un minimo locale

`'MaxEpochs'`: è il numero massimo di iterazioni

`'ValidationPatience'`: è il numero di passi dopo cui il training si ferma se la loss non diminuisce. Se si usa questa opzione bisogna porre `'OutputNetwork'` uguale a `"best-validation-loss"` in modo che la rete risultante dal training sia quella con le prestazioni migliori e non quella corrispondente all'ultima iterazione.

`'Plots','training-progress'`: visualizza la curva di apprendimento

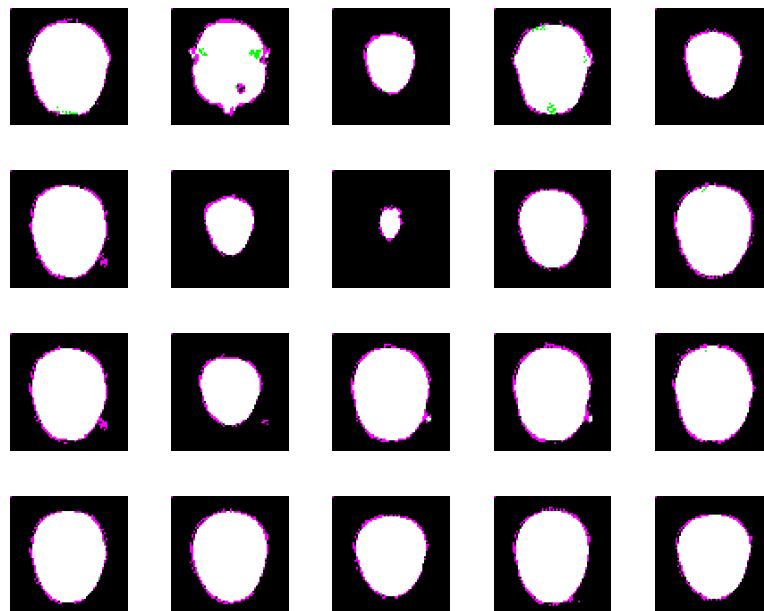
`'MiniBatchSize'`: è il numero minimo di immagini usato dal gradient descent.

`'Shuffle'`: nel training i dati vengono divisi in training e validation data in modo automatico secondo l'opzione shuffle (once solo una volta, `'every-epoch'` ad ogni epoca).

'ExecutionEnvironment','parallel' da usare se si ha un computer multi-core

Una volta completato il training possiamo testare la rete sul test set con ***semanticseg***. Con ***evaluateSemanticSegmentation*** possiamo valutare l'accuratezza e dalla confusion matrix possiamo valutare il DICE index della segmentazione. Possiamo anche confrontare i risultati con un algoritmo di image fusion tra le maschere GT e quelle ottenute. La rete addestrata può essere salvata come una variabile Matlab.

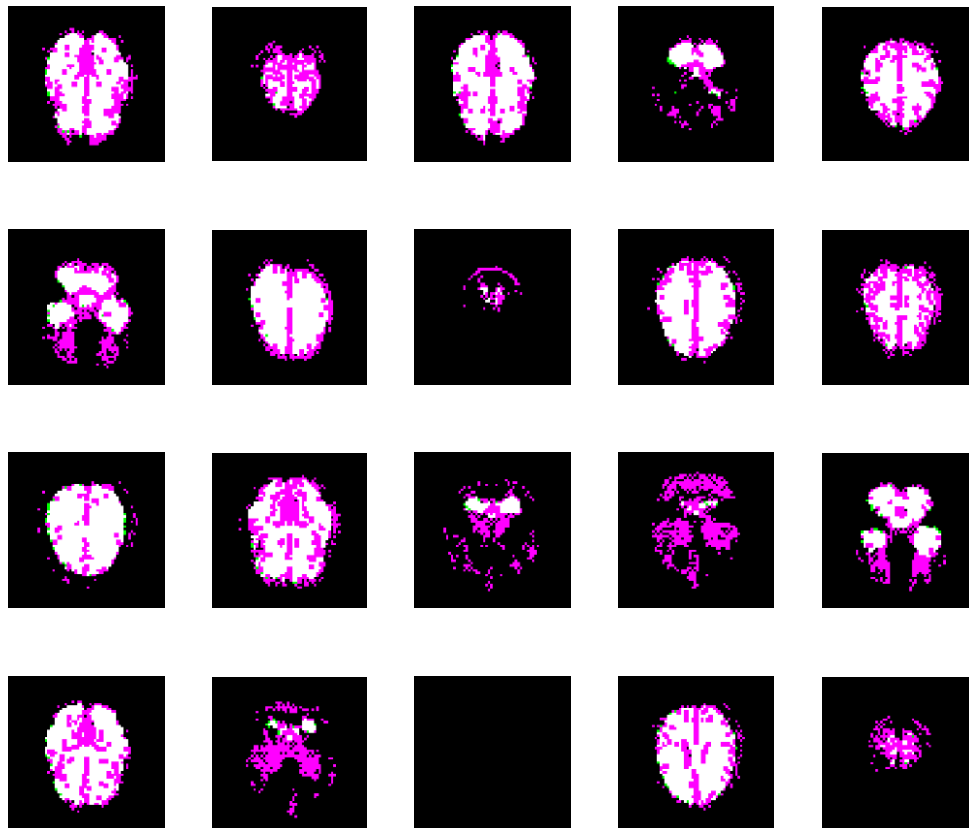
Per la segmentazione del cranio si riescono ad ottenere facilmente risultati molto buoni, con un DICE > 0.99 e Accuracy > 0.99.



Nella figura il confronto con imshowpair tra le maschere GT e quelle ottenute.

Nella rete per la segmentazione della materia cerebrale, può essere conveniente utilizzare la tecnica del transfert learning, cioè invece di partire da una U-Net con pesi random partire dalla U-Net addestrata in precedenza per un problema simile, cioè la segmentazione dell'intero cervello. Semplicemente invece di definire la rete con unetlayer caricheremo la rete ottenuta in precedenza nella funzione trainNetwork convertendola prima con ***layerGraph***.

La qualità della segmentazione dipenderà fortemente dal numero di iterazioni che a sua volta dipende dalla capacità computazionale. Si dovrebbero comunque ottenere valori di DICE > 0.85 e Accuracy > 0.97 .



Come si osserva dall'esempio di segmentazione riportato in figura anche con DICE e accuratezza apparentemente elevati la qualità della segmentazione appare limitata. Questo è dovuto al fatto che l'accuratezza pesa allo stesso modo i pixel di background e quelli della struttura di interesse, mentre la valutazione visiva si concentra su quest'ultima. Il DICE pesa maggiormente la struttura e quindi il suo valore è più realistico.

