# Progetto di Medical Imaging: sviluppo di CNN per segmentazione semantica

**appello di settembre 2023**

Lorenzo Lecce (n° matr. 830881)
Andrea Lucini Paioni (n° matr. 826578)

# Indice

# Introduzione

Questo progetto si basa su un dataset, denominato BRATS: si tratta di un dataset utilizzato per la segmentazione di immagini ottenute tramite MRI (o risonanza magnetica) di pazienti che presentano tumori cerebrali.

L'MRI, o Magnetic Resonance Imaging, è una tecnica di imaging usata per l'acquisizione di immagini tridimensionali di parti interne al corpo umano: solitamente tramite mezzo di contrasto, permette di osservare muscoli, ossa, midollo spinale, cuore, organi interni e vasi sanguigni; è particolarmente utilizzata in diversi ambiti, quali la neurologia, la neurochirurgia, l'urologia, la traumatologia, l'ortopedia, la gastroenterologia, la cardiologia e l'oncologia.

Le immagini MRI presenti all'interno del dataset BRATS includono una serie di differenti modalità di contrasto, indicate come T1, T2, Flair e T1c. Inoltre, per ciascun paziente è disponibile una mappa di segmentazione che indica le regioni di tessuto sano, le regioni infette dal tumore cerebrale e le regioni in cui è presente della necrosi...

L'obiettivo del progetto è cercare di migliorare l'individuazione di fenomeni tumorali cerebrali tramite l'utilizzo di reti neurali convoluzionali (o anche CNN). Le CNN sono particolarmente utilizzate in questo ambito, poiché permettono di creare modelli, sempre più affidabili e precisi, che aiutano ad individuare l'eventuale presenza o assenza di tessuto tumorale all'interno di un organo. Nel caso delle immagini presenti nel dataset BRATS, la segmentazione semantica mira a identificare le regioni nelle immagini di MRI in cui sono presenti agglomerati di cellule tumorali. L'obiettivo è di sviluppare modelli che siano affidabili nelle previsioni/individuazioni delle regioni di interesse, ma contemporaneamente efficienti dal punto di vista informatico, poiché si tratta di algoritmi molto pesanti dal punto di vista computazionale.

# Creazione CNN

## Configurazione ambiente e import dati

Iniziamo importando le librerie e impostando i limiti di memoria della GPU (il codice è stato sviluppato su Visual Studio Code, in locale, e non tramite Google Colab):

```
import matplotlib
import glob
import os
import numpy as np
import nibabel as nib
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,Conv2D, Activation, BatchNormalization,
MaxPooling2D, concatenate, Conv2DTranspose, UpSampling2D, SeparableConv2D
from tensorflow.keras.layers import ZeroPadding2D,Add,ReLU,AveragePooling2D
from tensorflow.keras import metrics as mtt
from tensorflow.keras import optimizers
import math
import matplotlib.pyplot as plt
```

*Codice 1: librerie*

Per lavorare in locale tramite Visual Studio Code, viene modificata la quantità di memoria allocata alla GPU

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  try:
    tf.config.set_logical_device_configuration(
        gpus[0],
        [tf.config.LogicalDeviceConfiguration(memory_limit=8*1024)])
    logical_gpus = tf.config.list_logical_devices('GPU')
```

```
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    print(e)
```
*Codice 2: allocazione GPU e memoria*

Vengono poi importati i dati, utilizzando la funzione *glob.glob()* per importare tutti i files di interesse:

```
fileL = glob.glob('dati/*/*/truth.nii.gz')
print(len(fileL))
iHead = nib.load(fileL[0])
```
*Codice 3: import dei files*

Si è deciso di analizzare 60 pazienti, e per ogni soggetto ci sono 5 files: un file è del tipo "truth.nii.gz", e indica la segmentazione (classi 0, 1, 2 e 4), e le quattro immagini di ogni paziente raccolte con la funzione *glob.glob()*, come indicato dalle variabili *nChan* e *chanNames*:

```
nPatAn = 60   # n° di pazienti da analizzare
nChan=4
chanNames = {1:'t1.nii.gz',2:'t1ce.nii.gz',3:'flair.nii.gz',0:'t2.nii.gz'}
```
*Codice 4: definizione pazienti e canali da analizzare*

Le immagini vengono fornite con una risoluzione di 1mmx1mmx1mm (quindi 240x240x155), decido di eseguire un sotto-campionamento delle immagini ad una risoluzione molto più bassa (del tipo 2x2x2 mm), in modo da ottenere due array: *immV* è una matrice che definisce le immagini sotto-campionate (infatti vengono definite all'interno delle matrici il numero di pazienti, la dimensione dell'immagine e infine il numero di canali, ovvero nel nostro caso 4), *segV* definisce le segmentazioni sotto-campionate (come la matrice immV, ma cambia giusto la definizione della segmentazione).

```
immV = np.zeros((nPatAn,120,120,77,nChan),dtype=np.float32) # per le immagini
segV = np.zeros((nPatAn,120,120,77,1),dtype=np.float32) # per le segmentazioni
```
*Codice 5: definizione array*

Il codice viene poi riprodotto per ogni paziente, e per ognuno dei 4 canali di ogni paziente:

```
for pIdx,fN in enumerate(fileL[:nPatAn]): # per ogni paziente
    pDir = os.path.dirname(fN)
    for chanL in chanNames.keys(): # per ogni canale
        iHead = nib.load(os.path.join(pDir,chanNames[chanL]))
        iTemp = iHead.get_fdata(dtype=np.float32)
        iTemp = iTemp[:,:,1:].reshape((2,120,2,120,2,77),order='F')
        immV[pIdx,:,:,:,chanL]= np.mean(np.mean(np.mean(iTemp,axis=-2),axis=-3),axis=0)
    iHead = nib.load(fN)
    iTemp = iHead.get_fdata(dtype=np.float32)
    iTemp = iTemp[:,:,1:].reshape((2,120,2,120,2,77),order='F')
    segV[pIdx,:,:,:,0]= np.max(np.max(np.max(iTemp,axis=-2),axis=-3),axis=0)
```
*Codice 6: sottocampionamento*

La risonanza ha intensità che non ha significato fisico (uno stesso soggetto può portare a risultati differenti in due macchinari differenti per MRI), dunque abbiamo deciso di riscalare le immagini in modo che tutte abbiano la stessa intensità: le scaliamo in modo tale che stiano tra -1 e 1. Dunque, prendiamo il 50° percentile dell'immagine al di fuori del background (per ogni paziente, e per ognuno dei quattro canali). Inoltre, si preferisce fare un'encoding delle classi in maniera categorica (come dall'ultima riga del codice seguente)

```
intVal = np.zeros((immV.shape[0],nChan),dtype=np.float32)
for sub in range(immV.shape[0]):
  for chan in range(nChan):
    iTemp = np.squeeze(immV[sub,:,:,:,chan])
    lThr = np.percentile(iTemp,1)
```

```
    intVal[sub,chan]=np.percentile(iTemp[iTemp>lThr],50)
intVal = intVal.reshape((immV.shape[0],1,1,1,nChan))
immV = (immV-0.5*intVal)/intVal
segV[segV>3.5]=3
segV = tf.keras.utils.to_categorical(segV)
```
*Codice 7: immagini riscalate*

## Creazione training e test dataset

Le risonanze sono immagini 3D, dunque esiste la possibilità di creare una rete in 3 dimensioni, tuttavia questa struttura richiederebbe un utilizzo molta più elevato di memoria, oltre a più parametri da definire. Per questo motivo il nostro procedimento della rete 2D prevede operazioni su una fetta alla volta, e non su un'immagine 3D, in modo da avere 40*77 (dunque 3080) immagini di training.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(immV, segV, test_size=0.33,
random_state=42)
X_train = X_train.transpose((0,3,1,2,4))
X_train = X_train.reshape((-1,120,120,4))
y_train = y_train.transpose((0,3,1,2,4))
y_train = y_train.reshape((-1,120,120,4))

X_test = X_test.transpose((0,3,1,2,4))
X_test = X_test.reshape((-1,120,120,4))
y_test = y_test.transpose((0,3,1,2,4))
y_test = y_test.reshape((-1,120,120,4))
```
*Codice 8: creazione dataset di training e di test*

## Implementazione rete neurale

Ora implementiamo una rete neurale convoluzionale (o CNN), in 5 passaggi:

- Il layer di input: i dati inizialmente inseriti nella rete neurale (specifica le caratteristiche delle immagini, come dimensioni, numero di canali…);
- Una sezione di encoder (o rete di classificazione): con due filtri convoluzionali (con pochi filtri), e successivamente un sotto-campionamento col Maxpooling2D (raddoppiando il numero di filtri) fino ad arrivare a 15x15, e alla sezione Bridge; utile per catturare informazioni di base;
- Una sezione di Bridge, formata da tre filtri di seguito, e poi Upsamling2D da 15x15 a 30x30; utile per ottenere informazioni avanzate;
- Una sezione di rete Decoder: a partire da una serie di Concatenate, per concatenare all'encoder con la stessa risoluzione fino a tornare al 120x120 (per legare features molto astratte a features base)
- Il layer di output: con 4 layer (per le quattro classi possibili, quindi sfondo/tessuti normali del cervello e i tre tipi di tumore differente) e attivazione di tipo softmax (ideale per le classificazioni) per ogni pixel delle singole immagini.

```
inputL = Input([immV.shape[1],immV.shape[2],nChan])
## Encoder section
# 120x120
c1 = Conv2D(16,3,activation='relu',padding='same') (inputL)
c2 = Conv2D(16,3,activation='relu',padding='same') (c1)

c3 = MaxPooling2D() (c2) # 60x60
c4 = Conv2D(32,3,activation='relu',padding='same') (c3)
c5 = Conv2D(32,3,activation='relu',padding='same') (c4)

c6 = MaxPooling2D() (c5) #30x30
c7 = Conv2D(64,3,activation='relu',padding='same') (c6)
```

```
c8 = Conv2D(64,3,activation='relu',padding='same') (c7)

c9 = MaxPooling2D() (c8) #15x15
## Bridge
c10 = Conv2D(128,3,activation='relu',padding='same') (c9)
c11 = Conv2D(128,3,activation='relu',padding='same') (c10)
c12 = Conv2D(128,3,activation='relu',padding='same') (c11)
c13 = UpSampling2D() (c12) #30x30

## "Decoder" network
c14 = concatenate([c13,c8])
c15 = Conv2D(64,3,activation='relu',padding='same') (c14)
c16 = Conv2D(64,3,activation='relu',padding='same') (c15)

c17 = UpSampling2D() (c16) #60x60
c18 = concatenate([c17,c5])
c19 = Conv2D(32,3,activation='relu',padding='same') (c18)
c20 = Conv2D(32,3,activation='relu',padding='same') (c19)

c21 = UpSampling2D() (c20) #120x120
c22 = concatenate([c21,c2])
c23 = Conv2D(16,3,activation='relu',padding='same') (c22)
c24 = Conv2D(16,3,activation='relu',padding='same') (c23)

# Output layer. 4 classi come output (0: sfondo/cervello + 3 tipi di tumore)
outL = Conv2D(4,3,activation='softmax',padding='same')(c24)
model = Model(inputs=inputL,outputs=outL)
```

*Codice 9: implementazione rete neurale*

```
model.summary()
```

*Codice 10: summary modello ottenuto*

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 120, 120, 4)] | 0 | [] |
| conv2d (Conv2D) | (None, 120, 120, 16) | 592 | ['input_1[0][0]'] |
| conv2d_1 (Conv2D) | (None, 120, 120, 16) | 2320 | ['conv2d[0][0]'] |
| max_pooling2d (MaxPooling2D) | (None, 60, 60, 16) | 0 | ['conv2d_1[0][0]'] |
| conv2d_2 (Conv2D) | (None, 60, 60, 32) | 4640 | ['max_pooling2d[0][0]'] |
| conv2d_3 (Conv2D) | (None, 60, 60, 32) | 9248 | ['conv2d_2[0][0]'] |
| max_pooling2d_1 (MaxPooling2D) | (None, 30, 30, 32) | 0 | ['conv2d_3[0][0]'] |
| conv2d_4 (Conv2D) | (None, 30, 30, 64) | 18496 | ['max_pooling2d_1[0][0]'] |
| conv2d_5 (Conv2D) | (None, 30, 30, 64) | 36928 | ['conv2d_4[0][0]'] |
| max_pooling2d_2 (MaxPooling2D) | (None, 15, 15, 64) | 0 | ['conv2d_5[0][0]'] |
| conv2d_6 (Conv2D) | (None, 15, 15, 128) | 73856 | ['max_pooling2d_2[0][0]'] |
| conv2d_7 (Conv2D) | (None, 15, 15, 128) | 147584 | ['conv2d_6[0][0]'] |
| conv2d_8 (Conv2D) | (None, 15, 15, 128) | 147584 | ['conv2d_7[0][0]'] |

| | | | |
|---|---|---|---|
| up_sampling2d (UpSampling2D) | (None, 30, 30, 128) | 0 | ['conv2d_8[0][0]'] |
| concatenate (Concatenate) | (None, 30, 30, 192) | 0 | ['up_sampling2d[0][0]', 'conv2d_5[0][0]'] |
| conv2d_9 (Conv2D) | (None, 30, 30, 64) | 110656 | ['concatenate[0][0]'] |
| conv2d_10 (Conv2D) | (None, 30, 30, 64) | 36928 | ['conv2d_9[0][0]'] |
| up_sampling2d_1 (UpSampling2D) | (None, 60, 60, 64) | 0 | ['conv2d_10[0][0]'] |
| concatenate_1 (Concatenate) | (None, 60, 60, 96) | 0 | ['up_sampling2d_1[0][0]', 'conv2d_3[0][0]'] |
| conv2d_11 (Conv2D) | (None, 60, 60, 32) | 27680 | ['concatenate_1[0][0]'] |
| conv2d_12 (Conv2D) | (None, 60, 60, 32) | 9248 | ['conv2d_11[0][0]'] |
| up_sampling2d_2 (UpSampling2D) | (None, 120, 120, 32) | 0 | ['conv2d_12[0][0]'] |
| concatenate_2 (Concatenate) | (None, 120, 120, 48) | 0 | ['up_sampling2d_2[0][0]', 'conv2d_1[0][0]'] |
| conv2d_13 (Conv2D) | (None, 120, 120, 16) | 6928 | ['concatenate_2[0][0]'] |
| conv2d_14 (Conv2D) | (None, 120, 120, 16) | 2320 | ['conv2d_13[0][0]'] |
| conv2d_15 (Conv2D) | (None, 120, 120, 4) | 580 | ['conv2d_14[0][0]'] |

---

Total params: 635588 (2.42 MB)
Trainable params: 635588 (2.42 MB)
Non-trainable params: 0 (0.00 Byte)

Osserviamo dal riassunto i parametri della rete, i parametri di input di ogni layer, e il numero dei parametri che abbiamo (in questo caso 635588 parametri).

## Diagnostiche rete neurale

Per valutare l'accuratezza di una segmentazione semantica, si può imputare lo Score Dice (ovvero il rapporto tra l'intersezione e l'unione di due aree, pari a 1 se sono perfettamente sovrapposte e pari a 0 se sono perfettamente disgiunte):

```
from tensorflow.python.ops import math_ops
def dice_accuracy(y_true, y_pred):

  y_true = math_ops.cast(y_true[:,:,:,1:] > 0.5, y_pred.dtype)
  y_pred = math_ops.cast(y_pred[:,:,:,1:] > 0.5, y_pred.dtype)
  intersec = y_true*y_pred
  union = y_true+y_pred
  dicePat = (1+2*K.sum(intersec,axis=[1,2]))/(1+K.sum(union,axis=[1,2]))
  return K.mean(dicePat)
```

*Codice 11: funzione di definizione Dice Score*

```
optim = optimizers.Adam(learning_rate=1e-3)
model.compile(optimizer=optim,loss='categorical_crossentropy',metrics= [dice_accuracy])
```

*Codice 12: definizione learning_rate della CNN*

Il modello si sviluppa in 40 epoche (le fasi di apprendimento della rete neurale), con batch sizes (le dimensioni dei campioni che si propagano nella rete neurale) pari a 8, e learning rate (quanto il modello impara più o meno velocemente in algoritmo di apprendimento) pari a 0.001:

```
fitHist =
model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=8,epochs=40)
```

*Codice 13: esecuzione della CNN*

```
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.plot(fitHist.history['loss'])
plt.plot(fitHist.history['val_loss'])
plt.subplot(122)
plt.plot(fitHist.history['dice_accuracy'])
plt.plot(fitHist.history['val_dice_accuracy'])
```

*Codice 14: grafici dice loss e dice accuracy*

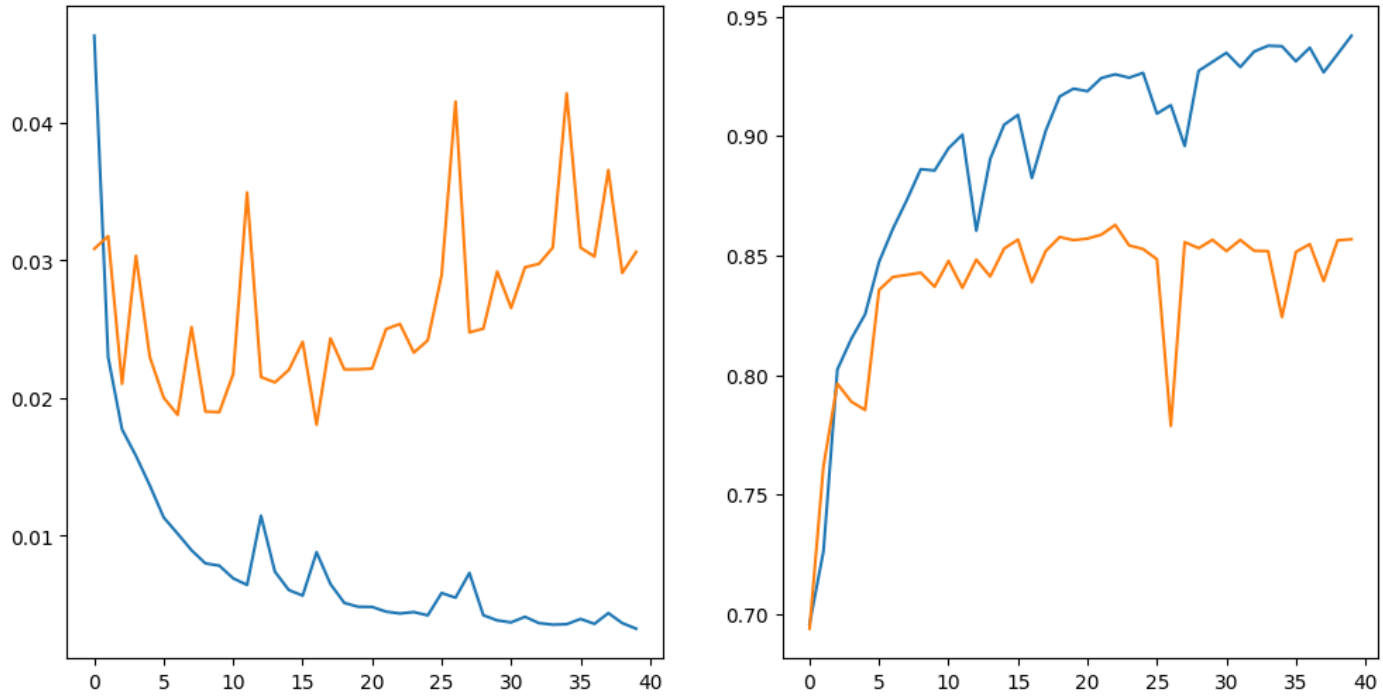


*Figure 1: grafici dice loss e dice accuracy*

È possibile osservare che la dice accuracy del dataset di training cresce col passare delle epoche fino al 95%, mentre la dice accuracy del dataset di validation rimane costante circa dalla 7°/8° epoca in poi.

Infine, calcoliamo una serie di dice score relativi ad alcune regioni specifiche (ovvero *l'intero tumore*, definito dalle classi 1+2+4, il *core tumor*, ovvero indicato dalle classi 2+4, e *l'active tumor*, definito dalla sola classe 4).

```
def diceFromMap(map1,map2):
    unionLab = np.sum(map1) +np.sum(map2)
    interLab = np.sum(map1 & map2, dtype=np.float32)
    return (2*interLab+.1)/(unionLab+.1)
def computeDices (yPred,yTrue):
    yPred = np.asarray(yPred)
    yTrue = np.asarray(yTrue)
    if np.sum(yTrue[:,:,:,1:])<1:
        return(np.nan,np.nan,np.nan)
```

```
    diceWhole = diceFromMap(np.sum(yPred[:,:,:,1:],axis=-1)>np.sum(yPred[:,:,:,:1],axis=-
1),
                            np.sum(yTrue[:,:,:,1:],axis=-1)>np.sum(yTrue[:,:,:,:1],axis=-
1))
    diceCore = diceFromMap(np.sum(yPred[:,:,:,2:],axis=-1)>np.sum(yPred[:,:,:,:2],axis=-
1),
                           np.sum(yTrue[:,:,:,2:],axis=-1)>np.sum(yTrue[:,:,:,:2],axis=-
1))
    diceEnh = diceFromMap(np.sum(yPred[:,:,:,3:],axis=-1)>np.sum(yPred[:,:,:,:3],axis=-1),
                          np.sum(yTrue[:,:,:,3:],axis=-1)>np.sum(yTrue[:,:,:,:3],axis=-1))
    return (diceWhole,diceCore,diceEnh)
```

*Codice 15: definizione ulteriori dice scores*

```
nTestPat = 20
diceV = np.zeros((20,3))
for idx in range(20):
    yPred = model(X_test[idx*77:(idx+1)*77,:,:,:])
    diceV[idx,:]=np.array(computeDices(yPred,y_test[idx*77:(idx+1)*77,:,:,:]))
plt.figure(figsize=(9,3))
titleList = ['Whole','Core','Active']
for i in range(3):
    plt.subplot(1,3,i+1)
    _=plt.hist(diceV[:,i])
    plt.title(titleList[i])
for i in range(3):
    outStr = titleList[i] + ' Dice: mean = {:.1f}% Median = {:.1f}%'
    print(outStr.format(np.mean(diceV[:,i]*100),np.median(diceV[:,i]*100)))
```

*Codice 16: definizione dice scores con grafici*

| | MEDIA | MEDIANA |
|---|---|---|
| **WHOLE DICE** | 83.6% | 90.4% |
| **CORE DICE** | 80.9% | 86.6% |
| **ACTIVE DICE** | 79.0% | 83.8% |

*Table 1: tabella dice scores modello iniziale*

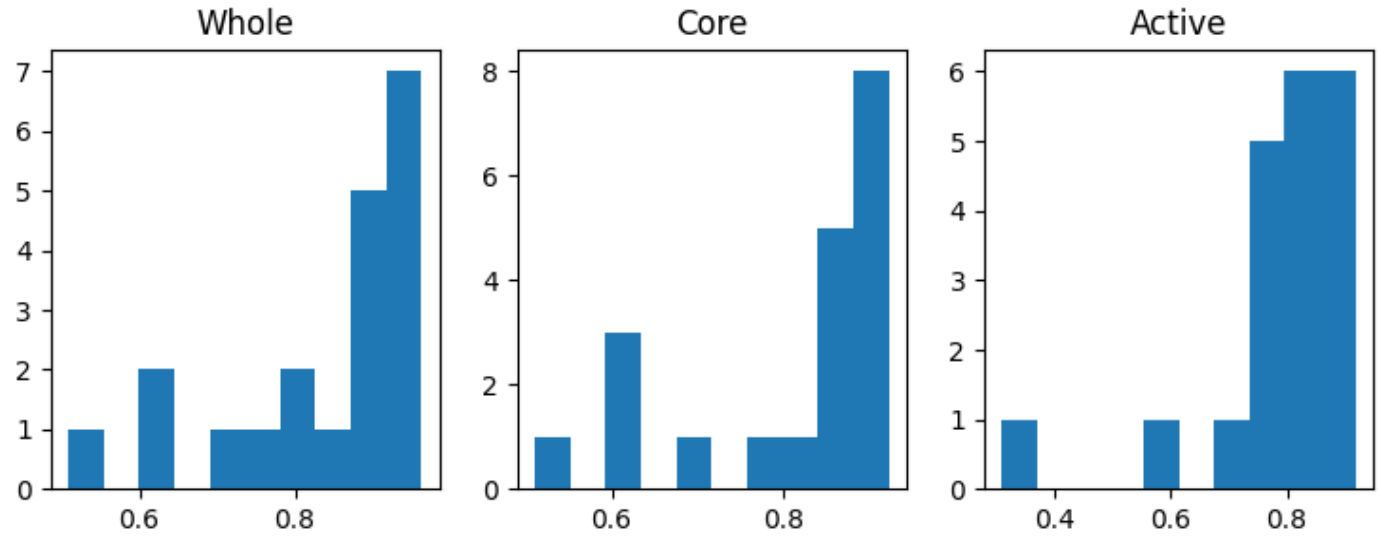


*Figure 2: grafici dice scores modello iniziale*

Otteniamo i valori di Dice Scores (calcolati sui pazienti, non sulle singole “fette”) per le tre regioni: la media e la mediana dell’intero tumore presentano valori molto elevati, ovvero l’83.6% della media e il 90.4% della mediana per la regione più grande (dunque è ragionevole avere dice buoni, perché i bordi delle regioni grosse e sferiche sono più

semplici da individuare); nonostante tutto, anche per la regione del tumore attivo i valori sono elevati, con media del 79% e mediana dell'83.8%.

Gli istogrammi definiscono le distribuzioni dei punteggi di Dice per le tre tipologie di tessuto tumorale: distribuzioni con valori elevati per valori del Dice Score elevati indicano previsioni efficienti delle tre tipologie di tumore.

## Caso singolo paziente

```
patIdx = 1   #  id del paziente scelto
slStart = 3
slStop = 70
slStep = 5

yPred = np.array(model(X_test[(patIdx*77):((patIdx+1)*77),:,:,:]))
yPred = np.argmax(yPred,axis=-1)
yTrue = y_test[(patIdx*77):((patIdx+1)*77),:,:,:]
yTrue = np.argmax(yTrue,axis=-1)
for slIdx in range(slStart,slStop,slStep):
    plt.figure()
    plt.subplot(131)
    plt.imshow(np.flip(X_test[patIdx*77+slIdx,:,:,1]),vmin=-0.3,vmax=1.0,cmap='gray')
    plt.contour(yTrue[slIdx,:,:],[0.5,1.5,2.5])
    plt.axis('off')
    plt.subplot(132)
    plt.imshow(X_test[patIdx*77+slIdx,:,:,2],vmin=-0.3,vmax=1.0,cmap='gray')
    plt.axis('off')
    plt.subplot(133)
    plt.imshow(yPred[slIdx,:,:],vmin=0,vmax=3)
    plt.contour(yTrue[slIdx,:,:],[0.5,1.5,2.5],cmap='jet')
    plt.axis('off')
```

*Figure 3: definizione immagini e risultato CNN paziente n°1*

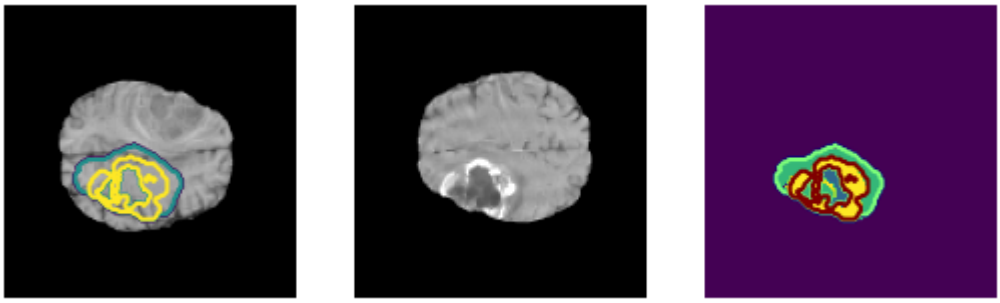

*Figure 4: immagine d'esempio di una slice paziente n°1*

Infine, osserviamo la segmentazione su un singolo paziente, eseguita su una fra le "slices" presenti nel dataset: per ogni fetta, infatti, vengono riportate un'immagine in cui si evidenzia il tumore individuato dalla rete neurale convoluzionale, un'immagine normale col tumore predefinito e il confronto tra quanto trovato dalla CNN e il quadro clinico veramente esistente per quella sezione dell'MRI.

# Miglioramenti rispetto al modello semplice

Quella individuata si tratta comunque di una rete neurale convoluzionale abbastanza semplice:

- prima di tutto è eseguita su immagini 2D ottenute a partire da rappresentazioni tridimensionali;
- inoltre, non sono state effettuate trasformazioni alle immagini, come rotazione/zoom/riscalature;
- infine, le quattro classi non sono ugualmente rappresentate;
- possono essere usate anche altre architetture

Per il nostro problema, abbiamo provato ad apportare dei miglioramenti sia per quanto riguarda le trasformazioni delle singole immagini, sia per quanto riguarda l'equidistribuzione delle quattro classi. Per questioni di prestazioni dei dispositivi, non è stato possibile approfondire il discorso delle rappresentazioni 3D, ma può essere uno spunto interessante per ulteriori ricerche.

## Data augmentation con ImageDataGenerator

La data augmentation consiste in un insieme di tecniche che permettono l'ampliamento del dataset a disposizione in un determinato progetto utilizzando i dati già presenti, senza la necessità di includere nuovi dati. Infatti, sono tecniche applicate a dati già esistenti, con lo scopo di modificarli leggermente mantenendoli utilizzabili e utili, e in questo modo ampliando i dati a disposizione per il dataset di training così da creare reti neurali maggiormente precise.

Tra i principali task di data augmentation applicabili all'analisi di immagini abbiamo la rotazione, gli shift (o spostamenti per decentrare un'immagine), i capovolgimenti (o flips, sia orizzontali che verticali), gli zoom, i cambiamenti di luminosità nell'immagine… In questo caso, ad esclusione dei cambiamenti di luminosità, vengono effettuate le altre operazioni con la funzione *ImageDataGenerator* della libreria *Keras*:

```
data_augmentation = ImageDataGenerator(rotation_range=0.2,
                                       rescale=1./255,
                                       zoom_range=0.3,
                                       width_shift_range=0.1,
                                       height_shift_range=0.1,
                                       horizontal_flip=True,
                                       vertical_flip=True)
augmented_train_data = data_augmentation.flow(X_train, y_train, batch_size=8)
```

*Codice 17: data augmentation*

```
fitHist_aug = model.fit(augmented_train_data, validation_data=(X_test, y_test), epochs=40)
```

*Codice 18: esecuzione della CNN dopo data augmentation*

```
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.plot(fitHist_aug.history['loss'])
plt.plot(fitHist_aug.history['val_loss'])
plt.subplot(122)
plt.plot(fitHist_aug.history['dice_accuracy'])
plt.plot(fitHist_aug.history['val_dice_accuracy'])
```

*Codice 19: grafici dice loss e dice accuracy modello con data augmentation*

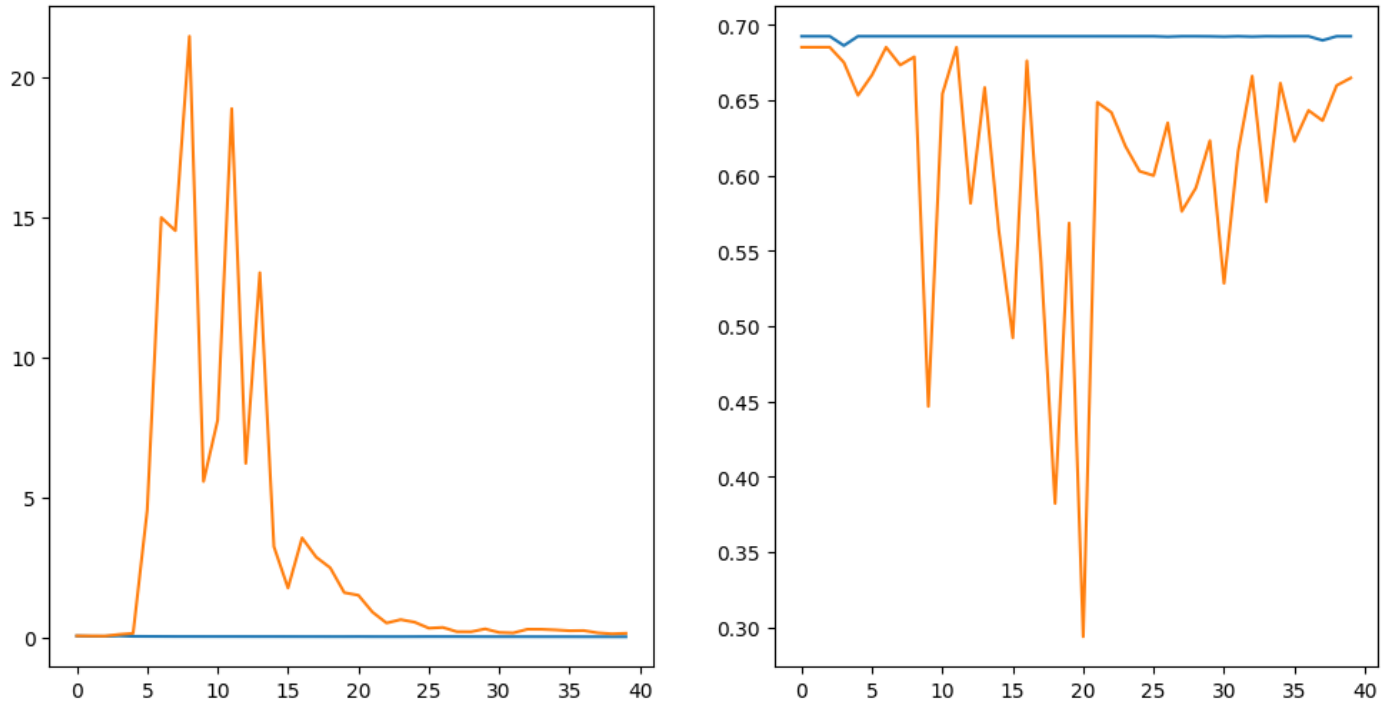


*Figure 5: grafici dice loss e dice accuracy modello con data augmentation*

```
nTestPat = 20
diceV = np.zeros((20,3))
for idx in range(20):
    yPred = model(X_test[idx*77:(idx+1)*77,:,:,:])
    diceV[idx,:]=np.array(computeDices(yPred,y_test[idx*77:(idx+1)*77,:,:,:]))
plt.figure(figsize=(9,3))
titleList = ['Whole','Core','Active']
for i in range(3):
    plt.subplot(1,3,i+1)
    _=plt.hist(diceV[:,i])
    plt.title(titleList[i])
for i in range(3):
    outStr = titleList[i] + ' Dice: mean = {:.1f}% Median = {:.1f}%'
    print(outStr.format(np.mean(diceV[:,i]*100),np.median(diceV[:,i]*100)))
```

*Codice 20: definizione dice scores con grafici modello data augmentation*

| | MEDIA | MEDIANA |
|---|---|---|
| **WHOLE DICE** | 0.1% | 0.0% |
| **CORE DICE** | 0.1% | 0.0% |
| **ACTIVE DICE** | 0.0% | 0.0% |

*Table 2: tabella dice scores modello data augmentation*

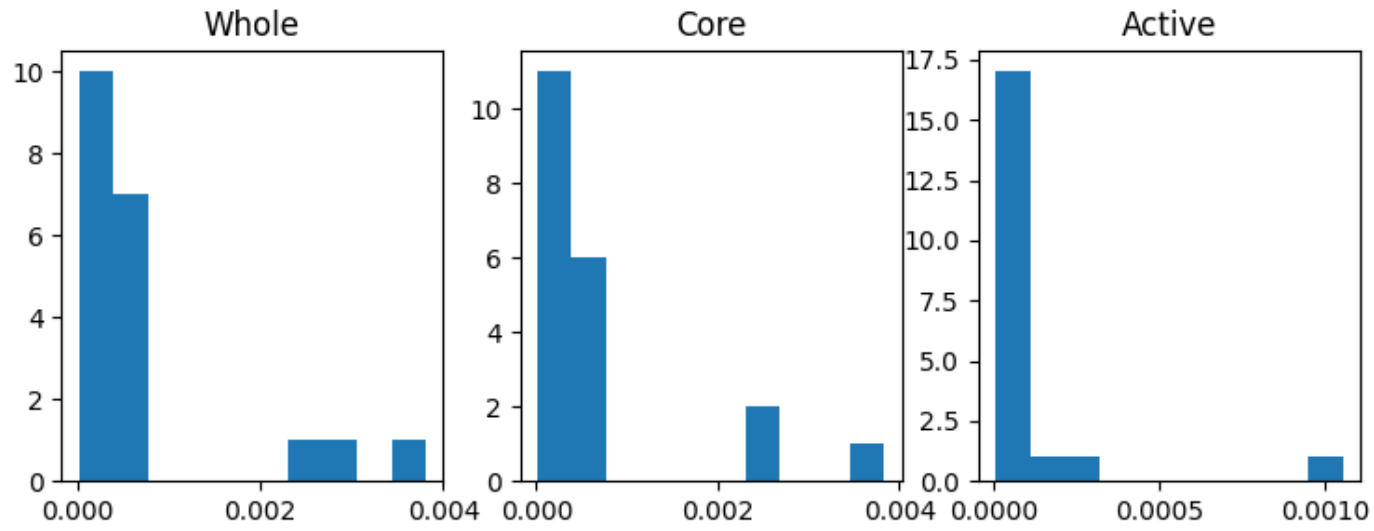


*Figure 6: grafici dice scores modello data augmentation*

Osserviamo che il modello creato in questo modo risulta peggio rispetto al modello originale creato in precedenza: in particolare, i Dice Scores trovati presentano valori veramente bassi (medie tra lo 0.1% e lo 0.01%, mediane tutte minori del 0.1%), e all'aumentare delle epoche non migliora la dice accuracy del dataset di training, com'era invece successo nel caso del primo modello creato.

## Rappresentazione delle classi

Un'ulteriore miglioramento può essere sviluppato a partire dalla rappresentazione delle quattro classi iniziali del dataset BRATS, ovvero:

- la classe 0: tessuto normale presente all'interno del cervello, o regioni al di fuori del cervello;
- la classe 1: la parte di tessuto necrotico meno visibile.
- la classe 2: l'edema, ovvero le fuoriuscite di liquidi causate dalla presenza del tumore all'interno del cervello
- la classe 3: la parte di tessuto tumorale ben visibile e ancora in fase di sviluppo e di crescita.

In precedenza, nella fase di scalatura, ogni valore superiore a 3.5 era stato posto pari a 3, potenzialmente modificando il numero di pixel di classe 3 rispetto alle altre classi. Ora viene imposta una rappresentazione alla pari delle differenti classi. In questo modo, i valori negativi vengono posti tutti pari a 0, e i valori superiori a 3 vengono posti pari a 3. Il resto del codice è simile al modello iniziale.

```
intVal = np.zeros((immV.shape[0],nChan),dtype=np.float32)
for sub in range(immV.shape[0]):
  for chan in range(nChan):
    iTemp = np.squeeze(immV[sub,:,:,:,chan])
    lThr = np.percentile(iTemp,1)
    intVal[sub,chan]=np.percentile(iTemp[iTemp>lThr],50)
intVal = intVal.reshape((immV.shape[0],1,1,1,nChan))
immV = (immV-0.5*intVal)/intVal
segV_2 = np.clip(segV_2, 0, 3)  # per limitare i valori tra 0 e 3
segV_2 = tf.keras.utils.to_categorical(segV_2, num_classes=4)
```

*Codice 21: immagini riscalate CNN 3*

```
from sklearn.model_selection import train_test_split
X_train_cr, X_test_cr, y_train_cr, y_test_cr = train_test_split(immV, segV_2,
test_size=0.33, random_state=42)
X_train_cr = X_train_cr.transpose((0,3,1,2,4))
X_train_cr = X_train_cr.reshape((-1,120,120,4))
y_train_cr = y_train_cr.transpose((0,3,1,2,4))
y_train_cr = y_train_cr.reshape((-1,120,120,4))
```

```
X_test_cr = X_test_cr.transpose((0,3,1,2,4))
X_test_cr = X_test_cr.reshape((-1,120,120,4))
y_test_cr = y_test_cr.transpose((0,3,1,2,4))
y_test_cr = y_test_cr.reshape((-1,120,120,4))
```

*Codice 22: creazione dataset di training e di test 3*

```
inputL = Input([immV.shape[1],immV.shape[2],nChan])
## Encoder section
# 120x120
c1 = Conv2D(16,3,activation='relu',padding='same') (inputL)
c2 = Conv2D(16,3,activation='relu',padding='same') (c1)

c3 = MaxPooling2D() (c2) # 60x60
c4 = Conv2D(32,3,activation='relu',padding='same') (c3)
c5 = Conv2D(32,3,activation='relu',padding='same') (c4)

c6 = MaxPooling2D() (c5) #30x30
c7 = Conv2D(64,3,activation='relu',padding='same') (c6)
c8 = Conv2D(64,3,activation='relu',padding='same') (c7)

c9 = MaxPooling2D() (c8) #15x15
## Bridge
c10 = Conv2D(128,3,activation='relu',padding='same') (c9)
c11 = Conv2D(128,3,activation='relu',padding='same') (c10)
c12 = Conv2D(128,3,activation='relu',padding='same') (c11)
c13 = UpSampling2D() (c12) #30x30

## "Decoder" network
c14 = concatenate([c13,c8])
c15 = Conv2D(64,3,activation='relu',padding='same') (c14)
c16 = Conv2D(64,3,activation='relu',padding='same') (c15)

c17 = UpSampling2D() (c16) #60x60
c18 = concatenate([c17,c5])
c19 = Conv2D(32,3,activation='relu',padding='same') (c18)
c20 = Conv2D(32,3,activation='relu',padding='same') (c19)

c21 = UpSampling2D() (c20) #120x120
c22 = concatenate([c21,c2])
c23 = Conv2D(16,3,activation='relu',padding='same') (c22)
c24 = Conv2D(16,3,activation='relu',padding='same') (c23)

# Output layer. Abbiamo 4 classi di output (0: sfondo/cervello + 3 tipi di tumore)
outL = Conv2D(4,3,activation='softmax',padding='same')(c24)
model = Model(inputs=inputL,outputs=outL)
```

*Codice 23: implementazione rete neurale 3*

```
optim = optimizers.Adam(learning_rate=1e-3)
model.compile(optimizer=optim,loss='categorical_crossentropy',metrics= [dice_accuracy])
fitHist_cr =
model.fit(X_train_cr,y_train_cr,validation_data=(X_test_cr,y_test_cr),batch_size=8,epochs=
40)
```

*Codice 24: definizione learning_rate della CNN 3 ed esecuzione della CNN 3*

```
plt.figure(figsize=(12,6))
```

```
plt.subplot(121)
plt.plot(fitHist_cr.history['loss'])
plt.plot(fitHist_cr.history['val_loss'])
plt.subplot(122)
plt.plot(fitHist_cr.history['dice_accuracy'])
plt.plot(fitHist_cr.history['val_dice_accuracy'])
```

*Codice 25: grafici dice loss e dice accuracy modello 3*

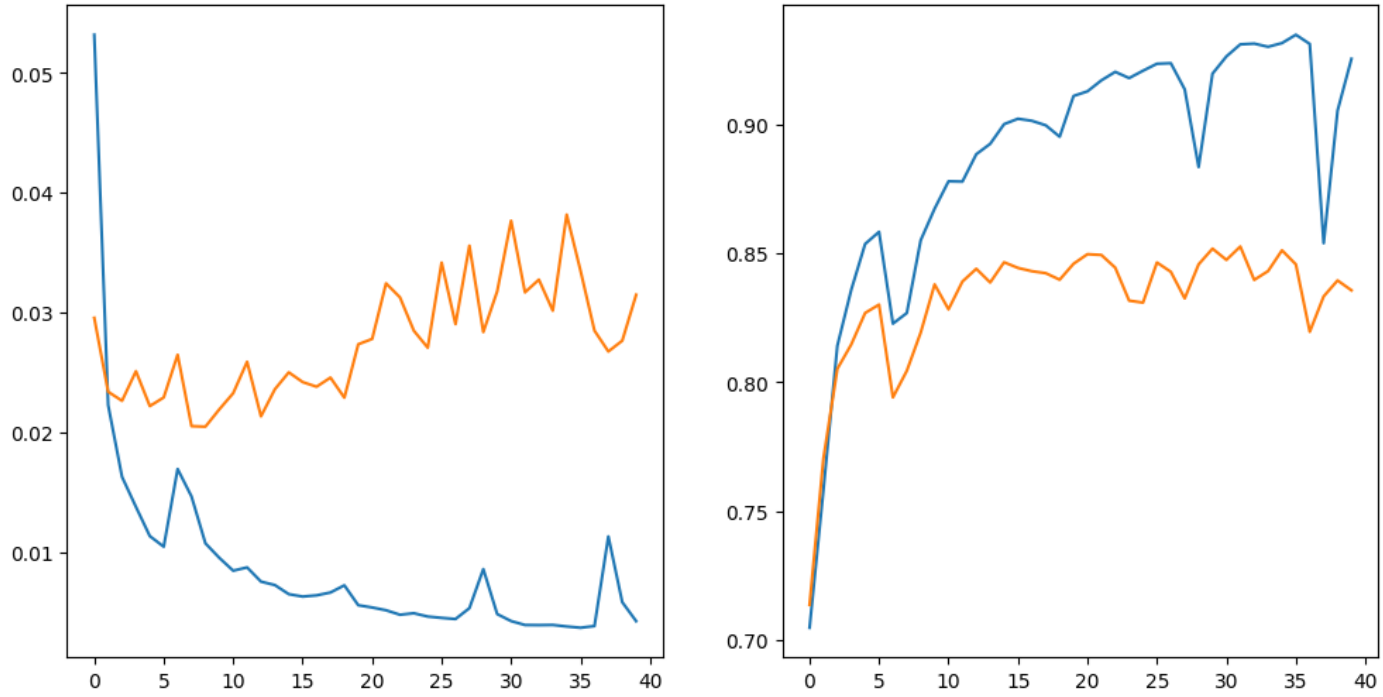


*Figure 7: grafici dice loss e dice accuracy modello 3*

```
nTestPat = 20
diceV = np.zeros((20,3))
for idx in range(20):
    yPred = model(X_test[idx*77:(idx+1)*77,:,:,:])
    diceV[idx,:]=np.array(computeDices(yPred,y_test[idx*77:(idx+1)*77,:,:,:]))
plt.figure(figsize=(9,3))
titleList = ['Whole','Core','Active']
for i in range(3):
    plt.subplot(1,3,i+1)
    _=plt.hist(diceV[:,i])
    plt.title(titleList[i])
for i in range(3):
    outStr = titleList[i] + ' Dice: mean = {:.1f}% Median = {:.1f}%'
    print(outStr.format(np.mean(diceV[:,i]*100),np.median(diceV[:,i]*100)))
```

*Codice 26: definizione dice scores con grafici modello 3*

| | MEDIA | MEDIANA |
|---|---|---|
| **WHOLE DICE** | 78.8% | 86.5% |
| **CORE DICE** | 75.4% | 83.3% |
| **ACTIVE DICE** | 72.7% | 79.4% |

*Table 3: tabella dice scores modello 3*

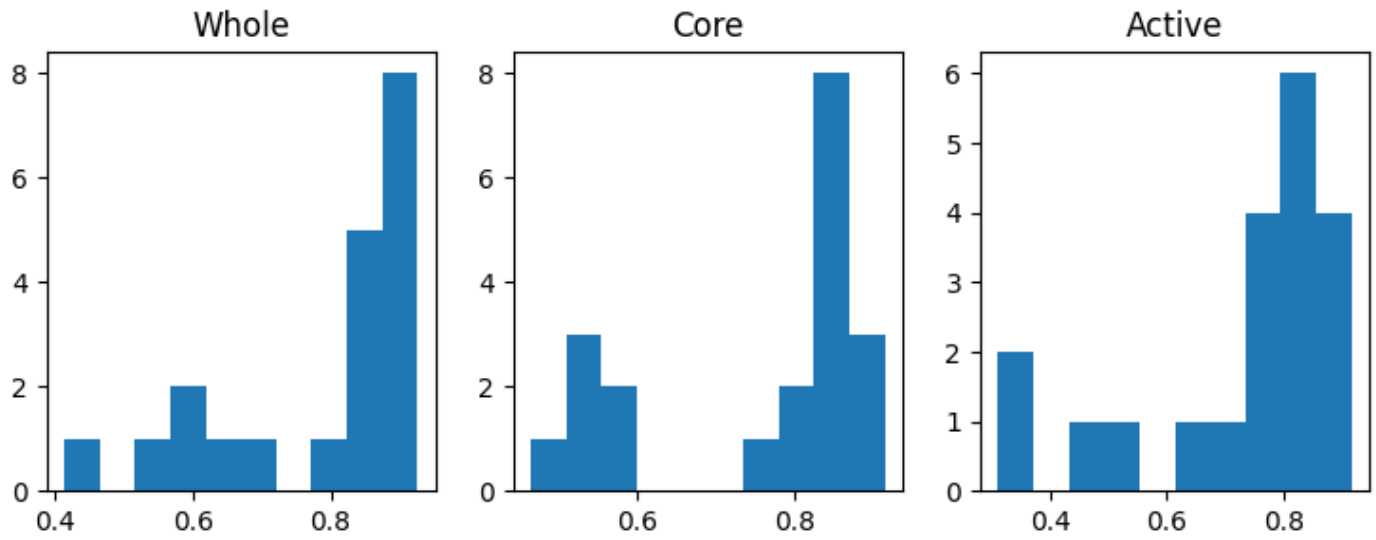


*Figure 8: grafici dice scores modello 3*

In questo caso osserviamo dai grafici e dalla tabella ottenuta relativa ai dice scores che il modello si comporta molto meglio del modello con data augmentation (medie tutte comprese tra il 78.8% e il 72.7%, mediane comprese tra l'86.5% e il 79.4%); tuttavia, il primo modello presenta comunque valori migliori. Come nel caso del primo modello, la dice accuracy del dataset di training cresce col passare delle epoche fino al 95% circa, mentre la dice accuracy del dataset di validation rimane costante circa dalla 10° epoca in poi con percentuali intorno all'85%.

# Conclusioni e sviluppi futuri

Abbiamo dunque osservato tre possibili reti neurali convoluzionali sviluppate sul dataset BRATS, con l'obiettivo di andare ad individuare, in modo consistente, la presenza/assenza di cellule tumorali in imaging ottenuto da macchinari per la risonanza magnetica.

Il primo modello sviluppato è risultato essere il migliore, dal punto di vista sia della dice accuracy per i dati di training, sia per i Dice Scores individuati nelle tre situazioni di tumore completo, core tumor e tumore attivo. Il secondo modello, quello in cui sono state applicate tecniche di data augmentation, ha avuto performance poco soddisfacenti, con valori molto bassi per tutti i dice scores e le dice accuracies. Il terzo modello, invece, quello su cui abbiamo applicato una differente rappresentazione delle classi, ha portato a risultati più che soddisfacenti, ma comunque meno precisi a livello di individuazione delle cellule tumorali rispetto al primo modello implementato.

Chiaramente altre tecniche (sia utilizzate singolarmente, che combinate tra di loro), come una CNN 3D, possono essere utilizzate per trovare una rete neurale convoluzionale con performance migliori, sia dal punto di vista della previsione e dell'individuazione delle cellule tumorali, sia dal punto di vista dello sforzo computazionale richiesto al nostro computer attraverso il codice sviluppato.

# Indice delle figure



# Indice dei codici



# Indice delle tabelle