Bioinformatics project
User Manual

Lidia Fantauzzo, s273117
Polytechnic University of Turin, July 4th, 2021

Introduction

In this project, I had the purpose of dealing with the problem of segmentation in biomedical applications. To do so, I implemented three different architectures to deal with the detection of the tumor area in patients having brain gliomas. The public available dataset that I used was BraTS2020. I focused on the properties of each network and I analysed the results obtained with each of them. At the end, I also evaluated the use of MC dropout for segmentation networks and its effect in outlining segmentation zones with different levels of uncertainty. In this paper, I will report all the technical information needed to reproduce the workflow.

Running the code

The complete code is available HERE. To reproduce the code, it can be possible to launch it with the bash file *starter/brats2020.sh* in which specify the parameters of the running to be given to argument parser implemented inside the code. Otherwise, the code can be launched directly with Python by running the file *src/main.py*, specifying the parameters immediately after. The specification of how to load and divide the dataset is in the *data/README.txt* file.

1 Pre-processing Pipeline

The whole BraTS2020 dataset [6, 2, 1] containing 369 images was divided in training, validation and test sets by using 50 images for both the validation and test parts and using the remaining images for training. Original images were given in a .nii format, so they were images of brain scanned in a 3 dimensional way. Since I wanted to use 2D networks, able to recognize tumors also in simple 2D medical scans, I took only one of the central level of each image, the 65th, in order to have a 2D image. Moreover, each image was provided in four different modalities: flair, T1, post-contrast T1 and T2. Since each modalities better highlighted one of the three tumor regions (Figure 1), I stacked the four images along the 0 axis to have an input image with 4 channels. In this way, each modality could brought its contribution to the feature extraction.
Since I was interested in detecting the whole tumor regardless the type of the region, I segmented the images with two labels, tumor and non-tumor pixel. So I mapped each pixel with a value different from 0 with 1.
Padding. The original size of the images was 240x240. The network that required the

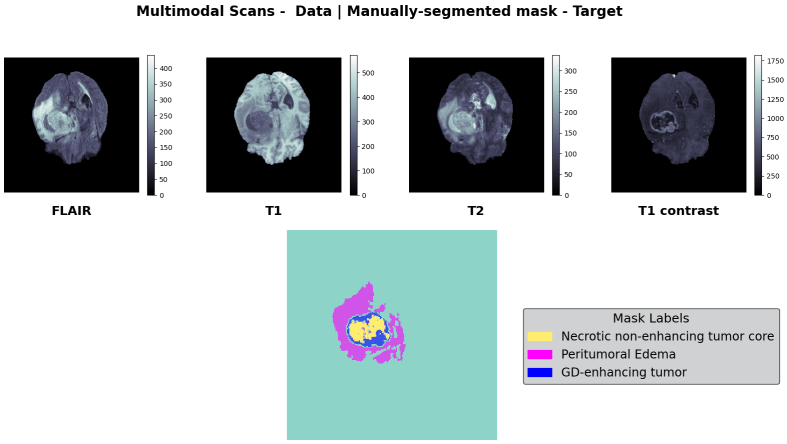

Figure 1: The four modalities of scans are shown: as it can be seen, each of them highlight the tumor regions in different ways. The tumor regions are then colored in the target image on the bottom of the picture.

highest downsample of the images in input was BisenetV2, which reached a downsample ratio of 1/32, so I needed images with a size divisible by 32. To do so, I padded all the original images, in all the modalities, by repeating the outermost pixel lines n times, where n was the number of lines needed to upsample the image to a size of 256x256. I could do this operation because the outermost pixel lines contains only labels representing the background.

Augmentation. In order to virtually increment the dimension of the dataset, I used data augmentation. In particular I used the 90 degree rotation transformation with a probability of 0.5 on the image and on the target. This kind of further processing tried to prevent the network to overfit over the training set.

Normalization. Each image was then normalized by subtracting the mean of each channel from each pixel and then dividing the result by the standard deviation. This ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network.

2 Architectures

2.1 U-Net

U-Net is a convolutional network, mainly used for biomedical image segmentation [7]. The general architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization, as shown in Figure 2.

The contracting path follows the typical architecture of a convolutional network and it consists of the repeated application of two Conv3x3, each followed by ReLU non linearity and a MaxPool2x2 for downsampling. At each downsampling step, the number of feature

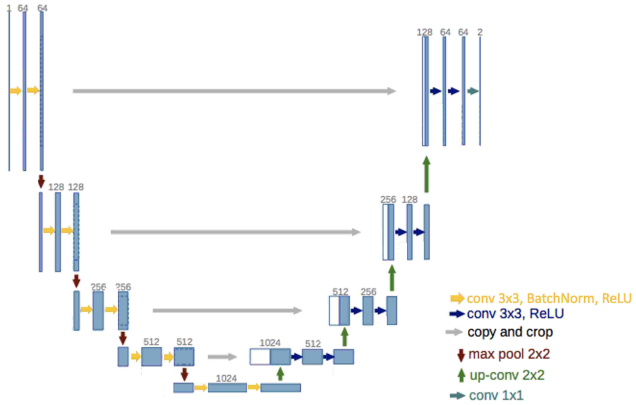

Figure 2: U-net architecture. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. White boxes represent copied feature maps. The arrows denote the different operations.

channels is doubled. Then in the expansive path there are four bilinear upsample of the feature map followed by a double Conv3x3 where the first convolution halves the number of feature channels. This is because each feature map obtained after the upsample is concatenated with the corresponding feature map of the contracting path, padded with 0 if needed. At the final layer a Conv1x1 is used to map each 64-component feature vector to the desired number of classes, in this case 2. This is the easiest implementation among the networks that I used, but this has the value of being fast and achieve good result with few images if you employ data augmentation.

2.2 DeeplabV3

Deep Convolutional Neural Networks have shown to be effective for the task of semantic segmentation however, the repeated combination of max-pooling and striding at consecutive layers of these networks significantly reduces the spatial resolution of the feature maps, resulting in loss of information. DeeplabV3 [3] use **Atrous Convolutions** which is a dilated convolution whose calculation is explained by the following formula

$$y(i) = \sum_k x(i + r * k)w(k).$$

For each location i on the output y and a filter w, atrous convolution is applied over the input feature map x where the atrous rate r corresponds to the stride with which the input signal is sampled. This is equivalent to convolving the input x with upsampled filters produced by inserting *r-1* zeros between two consecutive filter values along each spatial dimension. Atrous convolution allows to enlarge the field of view of filters to incorporate larger context. It thus offers an efficient mechanism to control the loss of information coming from downsample and finds the best trade-off between accurate localization (small field-of-view) and context assimilation (large field-of-view). Atrous convolution also allows to explicitly control how much to reduce the spatial resolution of

the final output by the *output stride*, namely the ratio of input image spatial resolution to final output resolution. DeeplabV3 uses this atrous convolution in the **Atrous Spatial Pyramid Pooling** (ASPP) where four parallel atrous convolutions with different atrous rates are applied on top of the feature map, plus a global average pooling is applied always in parallel, Figure 3.

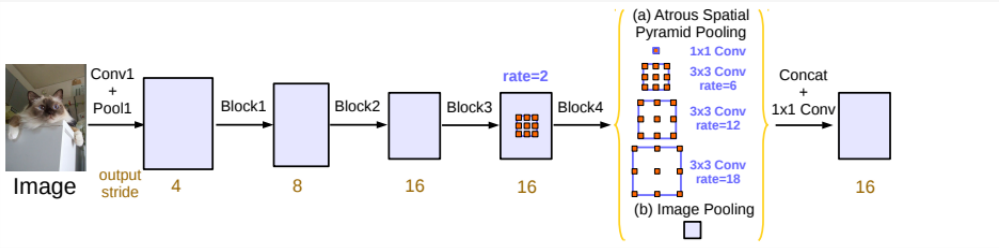

Figure 3: Backbone and DeeplabV3 with ASSP in parallel.

ASPP with different atrous rates effectively captures multi-scale information. Going deeper in the implementation, the four feature maps resulting from the atrous convolutions are concatenated along the channel axis and then a BatchNorm and a Conv1x1 are applied. At the same time, after the global average pooling, implemented as an AdaptiveAvgPool2d 1x1 in torch, a Conv1x1, a BatchNorm and then a Conv1x1 again are applied. The resulting feature map that has a size [C,1,1] is repeated along height and length in order to be able to sum it with the result of the previous atrous convolution. As last operation there is a BatchNorm.
The input map given to the ASPP module is the result of the ResNet backbone. As backbone, I implemented ResNet18 [5] and I used the pretrained version on ImageNet, as required by [3], by copying the weights from the ResNet18 model from torchvision. ResNet18 is originally made by a first block composed by a Conv7x7 with stride 2, a BatchNorm, a ReLU and a MaxPool3x3 with stride 2. Then there are 4 main blocks containing each two **Residual Blocks**. A residual block is made by a Conv3x3, a BatchNorm, a ReLU and then again a Conv3x3 and a BatchNorm. The number of channel of convolutions inside each main blocks are respectively 64,128,256 and 512. After each residual block the output is summed with the output of the previous residual block, which is transformed with the right number of channel if needed, by a Conv1x1. In order to adapt this implementation as backbone, the last block has to be changed from the original one by inserting an atrous convolution with dilatation 2 that maintain the ration between the input and the output resolution to 16. Then also the global pooling and the Linear layer of ResNet have to be removed in order to link this backbone to DeeplabV3. After the last layer of DeeplabV3 there is a Conv1x1 with a number of output channel equal to the number of classes and a bilinear interpolation to retrieve the original size of the input.

2.3 BiSeNetV2

The Bilateral Segmentation Network [8] is an architecture that takes into consideration both low-level details and high-level semantics and which is a good trade-off between speed and accuracy; indeed it is a lightweight network due to reducing the channel capacity and a fast-downsampling strategy. This means that it has a really small number of weights (8.2 M) to be stored and trained compared to deeper networks. This allows to embed this architecture also in mobile devices which have processors with low processing capacity and not necessarily large memory spaces.
The principal structure of BiSeNetV2 is made by two main branches, Figure 4, a Detail Branch, with wide channels and shallow layers to capture low-level details and generate high-resolution feature representation and a Semantic Branch, with narrow channels and deep layers to obtain high-level semantic context. Then a Bilateral Guided Aggregation layer to merge the information coming from the two branches.

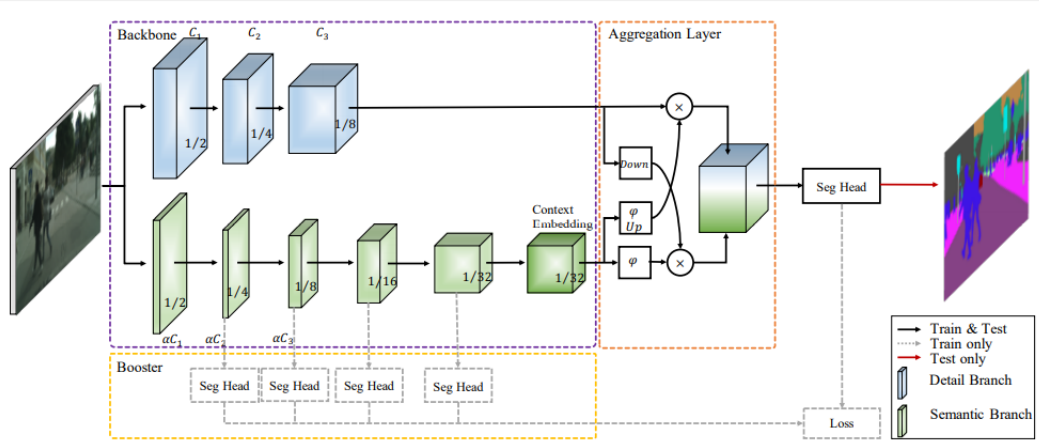

Figure 4: General structure of BiSeNetV2 with Detailed Branch, Semantic Branch, Bilateral Guided Aggregation layer and the Booster Strategy.

Detailed Branch. The basic unit of this branch is the Basic Block made by a Conv3x3 a BatchNorm and a ReLU layers. So we have three main blocks S1, S2 and S3. S1 contains 2 basic blocks and S2 and S3 contains 3 of them. The first convolution of each main block has stride 2 while the other have stride 1; while S1 and S2 have 64 neurons for each layer, S3 has 128 neurons. At the and of this branch the input image resolution is decreasing by 1/8.
Semantic Branch. This branch contains firstly a Stem Block whose structure is shown in Figure 5,c). it adopts a fast-downsampling strategy and has two branches with different manners to downsample the feature representation. Then both feature response of two branches is concatenated and the output has a resolution ratio of 1/4. Then there are three consecutive blocks: the first and the second contains each a Gather-and-Expansion Layer S2 and a Gather-and-Expansion Layer S1, while the third is made up by one

Gather-and-Expansion Layer S2 and three Gather-and-Expansion Layer S1. The structures of the Gather-and-Expansion layers are shown in Figure 5, a) and b). This layers take advantage of the benefit of depth-wise convolution, which is performed independently over each individual output channel. Deep-wise convolutions have been shown to yield similar performance to regular convolutions, while being much more efficient in terms of using much less parameters and less floating point operations. Indeed, while in regular convolution we have $C * K * K * O$ learnable parameters, where C is the number of input channels and K is the size of the kernel and O is the output channels, in depth-wise convolutions we only have $C * K * K * (O/I)$ learnable parameters where I is the input channels. This is due to the fact that, if we call O/I expansion ratio (ER), doing a depth-wise convolution is like having a number I of convolutional layers with ER output channels side by side, each seeing only one channel of the input map, and so producing ER channels of the output. All the channels are then stacked together producing an output channel of $I*ER = O$ like in the regular convolution, but with less parameters. The semantic branch is closed with the Context Embedding Block shown in Figure 5, d). This block uses the global average pooling and residual connection to embed the global contextual information efficiently. At the end of this branch the input image has a resolution of 1/32 compared to the original.

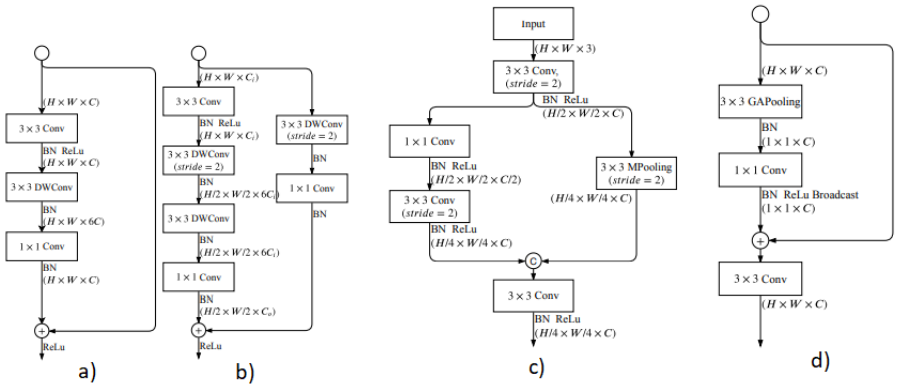

Figure 5: a) Gather-and-Expansion Layer S1. b) Gather-and-Expansion Layer S2. c) Stem Block. d) Context Embedding Block.

Bilateral Guided Aggregation Layer. The aim of this branch is to merge two types of feature response with different levels of feature representation, since simple combination ignores the diversity of both types of information, leading to worse performance. Bilateral Guided Aggregation Layer fuses the complementary information from both branches, as illustrated in Figure 6: features maps coming from both branches follow two different path each. Among other operations, on the output of the detail branch is used and Average Pooling with stride 2 to decrease the resolution ratio to 1/32 and multiply it element-wise with the output of the semantic branch. On the other hand, on the output of semantic branch is also applied an upsample to be able to multiply it with the output

of the detail branch and the final result is upsampled again to sum it with the other branch.

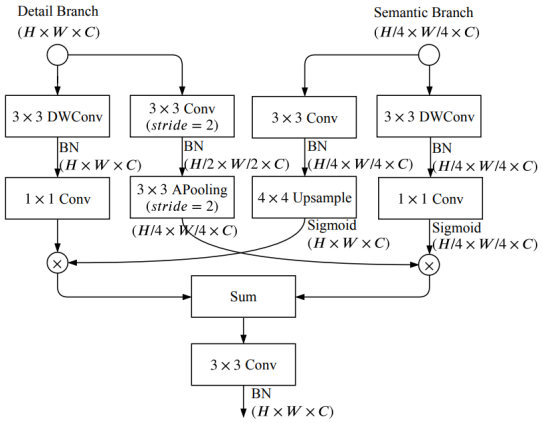

Figure 6: Bilateral Guided Aggregation Layer.

Segmentation Head. At the output coming from the aggregation layer is applied the Segmentation Head that consist in a Conv3x3, BatchNorm, ReLU then a Conv1x1 with an output channels equal to the number of classes and as last an upsample to restore the original size of the input image and compare it with the target image.
Booster Training Strategy. This strategy is proposed to improve the segmentation accuracy, enhancing the feature representation in the training phase and being discarded in the inference phase. This is done by inserting the auxiliary Segmentation Head to different positions of the Semantic Branch as shown in Figure 4. So at the end of the training we have 5 losses that are summed together and back-propagated.

3 Training and Testing pipeline

For the training and testing phase I created a specific class called Trainer, initialized with the model, the optimizer, the scheduler and the loss criterion, which I'm going to explain. As optimizer I used the Stochastic Gradient Descent Algorithm (SGD) with momentum. The equation that update the weights is

$$w_t = w_{t-1} - lrV_{dw_t} \quad \text{where} \quad V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta)\frac{\delta L}{\delta w_{t-1}}$$

where β is the momentum and lr is the learining rate. This means that the present gradient is dependent on its previous gradient and so on. This accelerates SGD to converge faster and reduce the oscillation. As scheduler I used the StepLR which decreases the value of the lr by multiplying it by a decay factor after a number of steps indicated by a

parameter. Finally, I used the Cross Entropy Loss per pixel given by

$$L = -\frac{1}{n * p} \sum_{i=1}^{n} \sum_{j=1}^{p} \sum_{k=1}^{K} y_k^{i,j} \log(p_k^{i,j})$$

where the sum over i goes over the n images in the batch, the sum over j goes over the p pixels in an image and the sum over k goes over the classes; $y^{i,j}$ denotes the one-hot encoded pixel class label and in image i of pixel j and $p^{i,j}$ is the predicted probabilities vector for the same pixel.
For what concern the metrics used to evaluate the predictions, I used the mean intersection over union, that for each class calculate the area of intersection between the grand truth and the prediction over their union. If we want to see it from the point of view of the confusion matrix, it is calculated as

$$\frac{\text{TP}}{\text{TP+FP+FN}} \tag{1}$$

To accumulate the results coming from each iteration and then getting the result every epoch, I created a class called 'mIoU' which was initialized with a 2x2 confusion matrix with all zeros and reset at every epoch. At every step, I accumulated in the confusion matrix the prediction for each single pixel of each image and then I got the IoU for class 1 (tumor) by using 1, and the IoU for class 0 by using 1 and replacing TP with TN. Then I also calculated the mean IoU by averaging the two results.
As said in the section 1, I divided the dataset in train, val and test sets. So during training every 10 epochs, I did a validation over the validation data loader to check the trend of learning of the network over images never seen during training. This allows to avoid to overfit on training data by choosing hyper-parameters that perform well only on the training set. Then at the end of the training phase, I used the test set to gain the final performance of the network.

4 Monte Carlo Dropout

Dropout is a regularization technique, that is, it helps prevent overfitting. Dropout layers switch-off some neurons at each training step, hence, each time the model's architecture is slightly different and we can think of the outcome as an averaging ensemble of many different neural networks, each trained on one batch of data only. Each neuron has some probability p of being ignored, called the dropout rate. Dropout is only used during training. At inference time, we want to use all the trained neurons and connections.
Here lies the novelty introduced by the Monte Carlo Dropout [4]. Monte Carlo refers to a class of computational algorithms that rely on repeated random sampling to obtain a distribution of some numerical quantity. Monte Carlo Dropout is a realization that the use of the regular dropout can be interpreted as an approximation of a probabilistic model: we can treat the many different networks (with different neurons dropped out) as Monte Carlo samples from the space of all available models. This can give an idea

about the model's uncertainty and often improves its performance. So to insert this concept inside the algorithm we simply apply dropout at test time. Then, instead of one prediction, we get many, one by each model. We can then average them to calculate a more accurate performance that converge to the expected value of the distribution of the results. I inserted the dropout layers in the three architectures after Convolutional layer, right after the BatchNorm and the ReLU if present. This is done because it was demonstrated by [4], that this is equivalent to have a Gaussian distribution for the weights.
To implement the MC Dropout inside the code and therefore enabling the dropout layers during testing time, I created a function that switched in training mode only the dropout layers before the test. So after this I did 100 forward test step and collected all the results in order to be able to calculate mean and standard deviation. All the results obtained are shown in next section.

5 Results

In this section I reported the results obtained with all the networks. To run this project I used a GPU 24x nVidia Tesla V100 SXM2 - 32 GB. In Table 1, I reported the results obtained in a cross-validation fashion by evaluating on the validation set, to fine tune the hyper-parameters. Then the best combination of parameters for each network where also tested on the test dataset and the corresponding results are shown in Table 2. For what concern the validation process, with U-Net architecture, I started with low lr and higher deacay rate, but looking at the decreasing of the loss I decided to increase the lr. The best result was reached with lr equal to 0.08 and a batch size of 32 and a mIoU of 0.8234. Otherwise, with DeepLabV3, I tried again similar hyper-parameters but this time a learning rate with an higher decreasing frequency compared with the one of U-Net gave me the best results. In fact, every 50 step the scheduler decreased the lr by 0.6 and the final mIoU was 0.8527. It is important to underline that this is also the setting that among all reached the best IoU with respect to class 1.
With the last network, BiSeNetV2 I discovered that the best value of lr was 0.05, because using 0.08 at a certain point the loss began to rise again after the descent, while using 0.01 the decreasing of the loss was not enough steep. Then the best result was obtained with batch size 16.
Comparing the results of all the network together, we can say that DeepLabV3 achieved the best accuracy in predicting the correct region of the tumor but since the network is really deep the time of training and prediction was very long, maybe not so feasible for an online use of this network. In fact from Table 1, we can see that DeepLab took almost six hours to train and also for this reason I could not do many experiments with hyper-parameters with this network. U-Net was faster, but the short distance that separated its mIoU test from that of BiSeNetV2 (Table 2) was not worth the longer times taken by U-Net compared to BiSeNetV2. In fact BiSeNetV2 was the fastest and its performance was a good trade-off between accuracy and speed. For an online purpose like this one it would be perfect. In Figure 7, the ground truth label is shown at the top

and at the bottom side there are the segmentation predicted on an image of the test set by U-Net, DeepLabV3 and BiSeNetV2 in order.

U-Net								
lr	epochs	batch_size	step_decay	deacy	time	mIoU	0 IoU	1 IoU
0.01	500	16	50	0.6	162	0.7832	0.9748	0.5916
0.01	500	16	100	0.7	157	0.7965	0.9897	0.6033
0.05	700	16	100	0.7	129	0.8162	0.9900	0.6424
0.05	700	32	100	0.7	145	0.8186	0.9900	0.6472
0.08	700	32	100	0.7	131	**0.8234**	0.9909	0.6559
DeepLabV3								
lr	epochs	batch_size	step_decay	deacy	time	mIoU	0 IoU	1 IoU
0.01	500	16	200	0.5	356	0.8168	0.9747	0.6589
0.01	500	32	200	0.5	341	0.8234	0.9856	0.6612
0.05	700	32	100	0.7	347	0.8290	0.9904	0.6676
0.08	700	32	100	0.7	328	0.8356	0.9958	0.6754
0.08	700	32	50	0.6	338	**0.8527**	0.9942	0.7112
BiSeNetV2								
lr	epochs	batch_size	step_decay	deacy	time	mIoU	0 IoU	1 IoU
0.01	500	16	100	0.7	62	0.7652	0.9875	0.5429
0.05	700	16	100	0.7	58	**0.7976**	0.9900	0.6052
0.05	700	32	50	0.7	47	0.7723	0.9559	0.5887
0.08	700	16	100	0.7	55	0.7117	0.9838	0.4396
0.08	700	32	100	0.7	46	0.7521	0.9865	0.5178

Table 1: Results obtained in the validation experiments done with the three architectures. lr represents the learning rate and mIoU is the mean intersection over union. 0 IoU is the metrics for class 0 and 1 IoU for class 1 (tumor). Time is expressed in minutes.

network	lr	batch_size	step_decay	deacy	mIoU	0 IoU	1 IoU
U-Net	0.08	32	100	0.7	0.7880	0.9674	0.6086
DeepLabV3	0.08	64	50	0.6	0.8313	0.9946	0.6680
BiSeNetV2	0.05	16	100	0.7	0.7798	0.9708	0.5888

Table 2: Best models evaluated on the test set.

5.1 MC Dropout results

For each network, I trained it with the best hyper-parameters found in the previous experiments, and then I tested them 100 times enabling the dropout even during the evaluation phase. As said before, averaging the results I obtained a more precise and

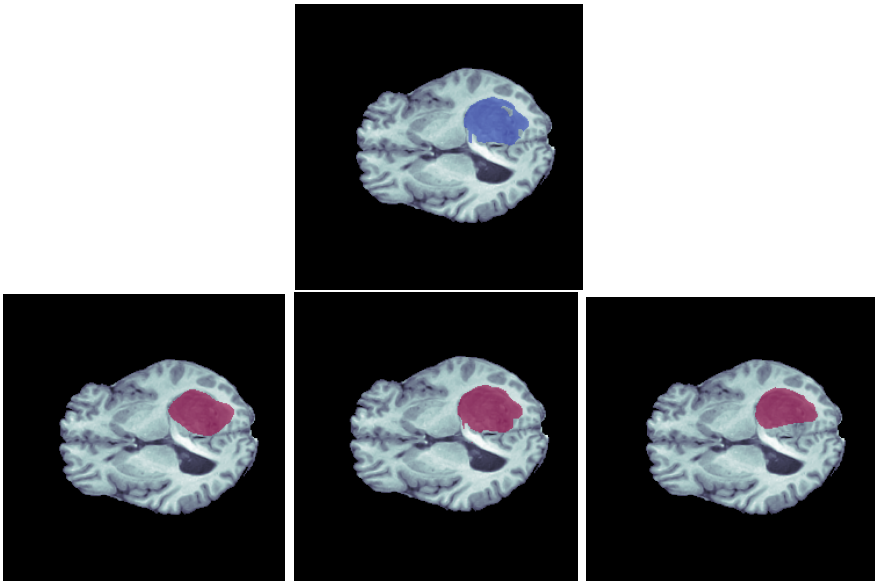

Figure 7: Ground Truth Label at the top and segmentation prediction of U-Net, DeepLabV3 and BiSeNetV2 in order.

reliable results on the metric. By calculating the standard deviation I was also able to estimate the uncertainty of the network over all predictions. In Table 3 , I reported the comparison between the performance obtain with each network with and without MC dropout and the corresponding standard deviation for the Bayesian results. The general performances with the MC Dropout were lower then the one without, but this was not so strange since introducing dropout at testing time, we are not using all the neurons trained but at every forward pass only the one which are not switched off by dropout. This explained also why BiSeNetV2 had a drop in performance, because it contains more convolutional layers than other networks. Although the lower results, these are results that are more reliable and for this reason even those who should use these devices would be more confident in their functionality.

		MC Dropout	
Network	mIoU	Averaged mIoU	standard deviation
U-Net	0.7880	0.7611	0.00916
DeepLabV3	0.8313	0.8031	0.00449
BiSeNetV2	0.7798	0.7446	0.0174

Table 3: Comparison of the results obtained with non bayesian network and bayesian network with MC Dropout along with the mean and standard deviation.

As said, MC Dropout is able to give the uncertainty about the prediction given by the network. In classification or regression cases, in which we obtain a single value as results it could be easier because we can focus on that, but here we have a single prediction

over all the pixels of an image. For this reason we do not have a unique estimation, but the uncertainty of the network can change areas by areas, depending also on the characteristic of the images. So in this case, to evaluate the certainty of the network in predicting the correct zone of the brain in which we can find the tumor, I decided to follow the above reasoning. Since I had only two class, each pixel could have value 0 or 1, so calculating the standard deviation on the single probability of prediction for the pixel would have given us the same information that counting the amount of 1 or 0 had for that specific pixel. In details, given a specific pixel of an image, if the standard deviation of the a pixel was high, it meant that the the probability would fluctuate a lot and the network gave us sometimes 1 and sometimes 0. On the other hand, if the standard deviation was low, the network would give us with an higher frequency 1 or 0. So, I summed pixel-wise all the prediction map obtained by the network and then plotted them with a color map in Figure 8. Regions with higher values (in blue) were the regions in which the network was more certain about the prediction of the cancer, while the zero values was not plotted for better understanding of the tumor area. For what concern the intermediate values, the network was more uncertain in the prediction. In fact, in the Figure 8 we can see that the network was strongly certain of prediction in the central area, while instead it lost security continuing towards the edges of the tumor. This was a predictable behavior since even the human eye can easily recognize the cancer core but he is more uncertain on the peripheral zones, due to the merging of healthy and diseased tissue that characterizes them.

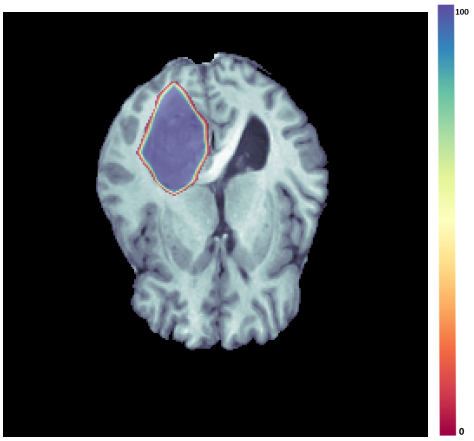

Figure 8: Uncertainty regions in predictions with Monte Carlo Dropout.

6 Conclusions

As we seen, DeepLabV3 was the network with the best results but the time taken for training was really long. BiSeNetV2 instead turned out to be the best trade off between performance and speed. These characteristics are necessary for devices that does not have huge memory storage and high performing processors. Then MC Dropout can also

be a good tools to increase the confidence of medical staff that use them, because it give an idea of the certainty of the prediction and so the prediction becomes less "black block".

References

[1] Spyridon Bakas, Mauricio Reyes, Andras Jakab, Stefan Bauer, and Markus Rempfler et al. Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the brats challenge, 2019.

[2] Sotiras A. et al Bakas S., Akbari H. Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features. *Sci Data 4*, (170117), 2017.

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[4] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[6] Bjoern H. Menze, Andras Jakab, and et al Bauer. The multimodal brain tumor image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, 2015.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[8] Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation, 2020.