U-Net for Brain tumor segmentation

Francesco Corti
Department of Computer Science
University of Pisa
f.corti3@studenti.unipi.it

September 9, 2021

Abstract

This report illustrates the work done for the project of Intelligent Systems for Pattern Recognition class. A U-Net Convolutional Network for Biomedical Image Segmentation was implemented [6] by using Pytorch as software framework [5]. The dataset was obtained from the Cancer Image Archive, a service that de-identifies and hosts a large archive of medical images of cancer accessible for public download.

1 Introduction

One in three people will be diagnosed with cancer in their lifetime, only in the United States 23,000 cases of brain cancer were diagnosed in 2015. They can be more aggressive with a life expectancy of at most two years or less aggressive with a life expectancy of several years. Magnetic Resonance Imaging (MRI), since is not harmful to the patient, is used to provide high-quality image by creating an electromagnetic field around the tissue that we want to observe. Since the shape of tumours varies it is very hard to train a model in order to perfectly perform segmentation of them.

Historically, most of the automatic brain tumour segmentation methods used hand-designed features. However, these methods were outperformed by deep learning models that can learn a hierarchy of increasingly complex features such as Convolutional Neural Networks (CNN). This breakthrough of CNN is happening since 2012 after the ImageNet competition where the AlexNet model won against all the other hand-crafted models obtaining more than the 10% of the accuracy score against all the opponents [4].

The report is structured as follows: in Section 2 a description of the data and the preprocess done on it is described, then a brief description about the Convolutional Networks is done. In Section 3 a description of the architecture of the model used is shown and commented. In Section 4 the result obtained are shown and briefly commented. In Section 5 a discussion of the results obtained and the possible further approaches is done. In the end, in Section 6 a wrap up on the project is given followed by some personal considerations.

2 Brain Tumor Segmentation

2.1 Cancer Genome Atlas Low Grade Glioma Dataset

The dataset contains MR images, stored in TIF format, from TCIA Lower-glade gliomas (LGG) collection with segmentation masks approved by a board-certified radiologist at Duke University. The whole dataset used for training and validate the model is publicly available on Kaggle. An example of the images contained in the dataset is provided in the Fig.1.

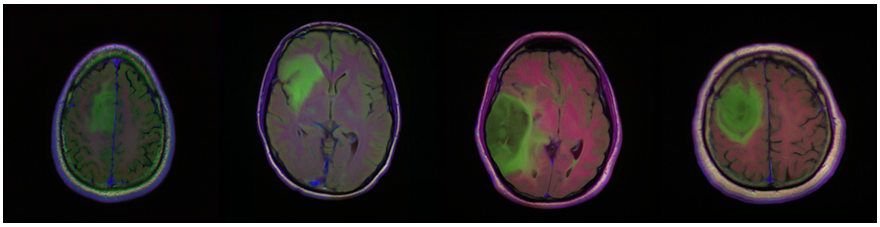

Figure 1: MR images from the LGG dataset.

The dataset is obtained from The Cancer Genome Atlas (TCGA), a public database containing more than 2.5 petabytes of data related to cancer. Out of all the patients, 110 of them were selected from 5 institutions. Every patient has a number of slices that varies from 20 to 88, for every slice a mask that contains the region of the tumour is present. These masks are used as label to train the model to correctly segment the cancer region in the slices provided as input. In order to obtain a better accuracy a preprocessing is applied on the images before the training procedure. This preprocessing phase is composed of the removal of the regions that don't concern the brain (skull stripping), normalization of the tissues and Z-score normalization of the entire dataset. Also, to obtain a model invariant to image deformation and angulation a data augmentation of the dataset can be applied before the training starts by performing rotation, scaling and horizontal flipping of the images.

2.2 Convolutional Networks

Convolutional Networks (also known as convolutional neural networks, CNNs) are neural networks model used to process data that has a grid topology shape [2], these data most commonly are images. CNNs can learn the hierarchical representation of features by stacking multiple layers that are composed of three main parts. These parts are a Convolution layer, where the convolution operation on the input is performed, a Detector layer where each linear activation is run through a nonlinear activation function and a Pooling layer which is used to make the representation invariant to translations and to reduce the number of dimensions of the input. The output of these layers is sometimes called feature map. These layers learn a hierarchical representation of the data used as training by modifying the weights, moreover their purpose is to extract features of the input which are then used in the last part of the network that output the classes

probability by passing the input through a feedfoward neural network followed by a softmax function (if multiple classes want to be classified). Fig.2 shows the architecture of a CNN used for images classification that has two outputs units.

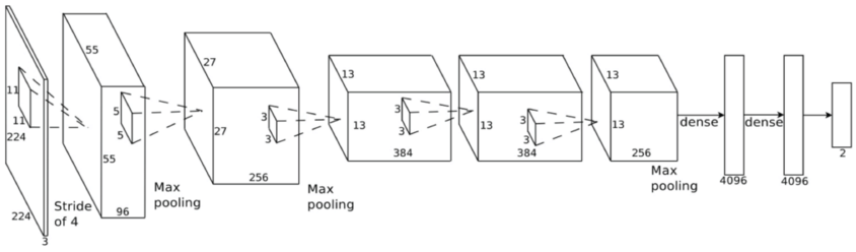

Figure 2: Convolutional Neural Network for classification.

However, the typical use of CNN is on classification tasks, but in biomedical image processing the expected output of the model should be a segmentation of the image (labelling of each pixel). If this task wants to be achieved a modification of the CNN architecture needs to be done.

3 U-Net

The CNNs segmentation models proposed in the past were mainly based on a sliding-window approach, every pixel of the image was classified based on its neighbour that was used as input to the network. However, this approach was slow, in fact for every pixel a patch based on its local context was built and there was a trade-off between patch dimension and accuracy of the classification. A bigger patch means a bigger context to be considered around a pixel but also a more expansive computation to be performed. U-Net is a neural network model used for biomedical image segmentation [6] that does not present a sliding window, instead the whole image is passed as input to the newtwork. It is composed by a contracting (downsampling) part followed by an expansive (upsampling) part. The contracting part contains multiple CNN layers each composed of two 3x3 convolutions followed by a non-linear activation function (rectified linear unit, ReLU) and 2x2 max pooling operation with stride 2 used for downsampling. The number of feature channels is doubled at each downsampling step. The expansive (upsampling) part has a two 3x3 transpose convolution part (up-convolution) that halves the number of feature channels each followed by a ReLU function. Every expansive block has a connection with the corresponding contracting part, these connections are called skip connections. These are used to avoid gradient vanishing when backpropagation is used and to pass features from the upsampling part to the downsampling part in order to recover spatial information when reconstructing the image. At the end, a final layer with 1x1 convolution is used to map the last features map to the number of classes that needs to be predicted. The Fig.3 shows the original U-Net architecture used for segmentation.

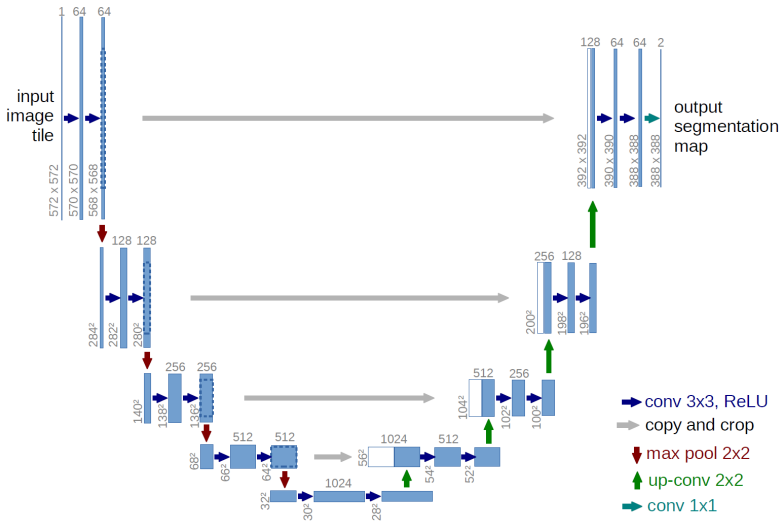

Figure 3: U-Net original architecture.

The name of the network comes from the fact that the expansive part is symmetric to the contractive part and create this u-shape architecture.

3.1 Network architecture

In our case, the U-Net architecture is different compared to the original one. The first convolutional layer has a different size, due to the input images that have a 3D shape and different dimensions. The last layer of our network uses a Sigmoid function, since our goal is to predict a tumour mask on the image, instead of a Softmax function. I followed the network architecture implementation used inside Buda et al. [1] since good performance on the same dataset was achieved. The model also use Batch Normalization [3] after each layer to reduce the Internal Covariate Shift which is the change in the distribution of network activations due to the change in network parameters during training, however the results show that in our case removing the Batch Normalization does not worsen the performance of the network if the learning rate is not high.

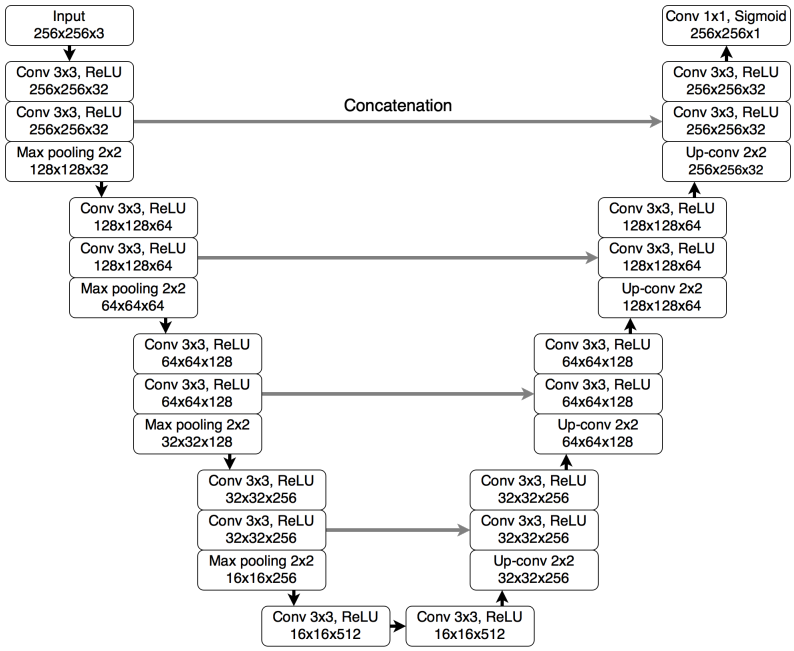

Figure 4: U-Net used to perform segmentation.

3.2 Network training

The produced segmentation maps and the input images are used to train the network with stochastic gradient descent, in our case to speedup the computation a batch size of 16 was chosen, however multiple batch sizes were tried and compared. The first U-Net implementation used a cross-entropy loss function over the final feature map produced by the network, however in our case, the Dice Similarity coefficient is used, this coefficient measures the overlap between the segmentation mask predicted by the model as output and the manually annotated mask provided as ground truth. The Dice coefficient is defined as follows:

$$DC(y, \hat{y}) = \frac{2 * |y \cap y|}{|y| + |\hat{y}|} \tag{1}$$

where y is the predicted matrix by the model and $\hat{y}$ is the ground truth mask provided as label. Which in our case is computed as:

$$DC(y, \hat{y}) = -\frac{2 \sum_i y_i \hat{y}_i}{\sum_i y_i + \sum_i \hat{y}} \tag{2}$$

so the loss function to be optimized is $1 - DC(y, \hat{y})$.

The model requires around forty minutes of CPU for each training epoch with a batch size of 16 and more than 16Gb of RAM to store the dataset as torch tensors that are then forwarded inside the network. To speedup the computation, the training

of the model was performed on the Kaggle platform, this was done mainly due to the free computational power that the Kaggle platform offers. By setting up Pytorch correctly the model can be trained by using CUDA that exploits GPUs which speed up the computation by an order of magnitude. In fact, the model requires 4 minutes for each epoch if a GPU is used. Also, all the models were trained by using Adam Pytorch as optimizer.

4 Results

In Fig. 5 two output segmentation masks obtained by the model on two different inputs are shown.

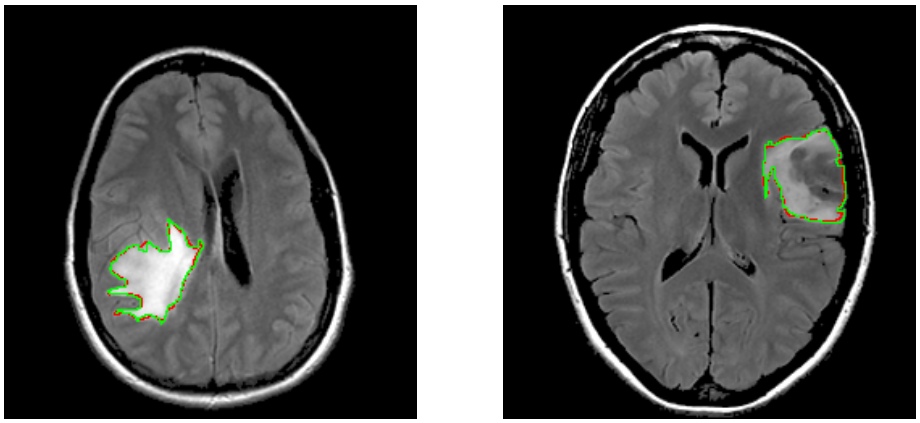

Figure 5: Masks generated by the model (green) compared to ground truth (red).

Table 1 and Table 2 show the results obtained by training the model by using and not using the Batch Normalization technique. After, Fig.6, Fig.7 and Fig.8 show the loss curves trend with and without the Batch Normalization.

#Epochs	**Learning rate**	**Loss train**	**Loss validation**
100	0.1	0.038945	0.358600
100	0.01	0.022355	0.297494
100	0.0001	0.006013	0.300957

Table 1: Values obtained without Batch Normalization.

#Epochs	Learning rate	Loss train	Loss validation
100	0.1	0.047595	0.257974
100	0.01	0.022803	0.349134
100	0.0001	0.004725	0.260191

Table 2: Values obtained with Batch Normalization.

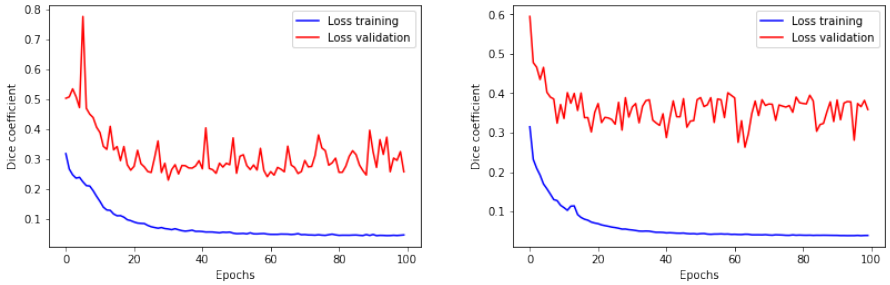

Figure 6: Learning curves with 0.1 with and without Batch Normalization

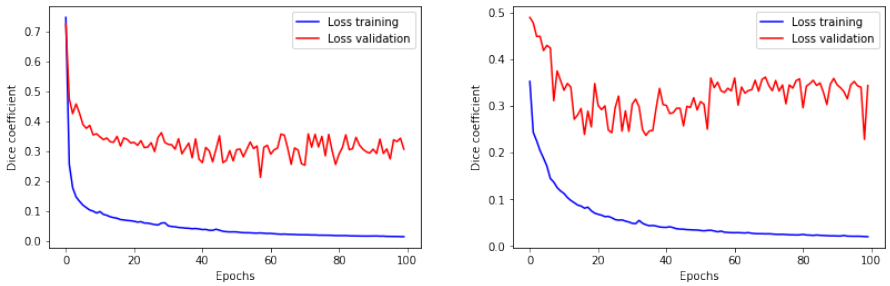

Figure 7: Learning curves with 0.01 with and without Batch Normalization

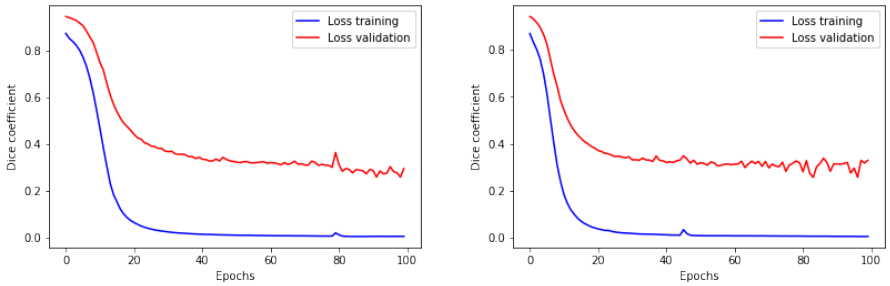

Figure 8: Learning curves with 0.0001 with and without Batch Normalization

In Table 3 the results obtained by varying the batch size are shown. The Table 4 shows the time taken to complete the training and validation phase as the batch size varies. After, Fig.9, Fig.10 and Fig.11 show the loss curves trend.

Batch size	Learning rate	Loss train	Loss validation
16	0.0001	0.018764	0.341420
32	0.0001	0.097907	0.343338
64	0.0001	0.445822	0.742853
16	0.1	0.065286	0.262480
32	0.1	0.053285	0.203653
64	0.1	0.074282	0.175922

Table 3: Values obtained by varying the batch size with 30 epochs.

Batch size	Total time	Epoch time
16	979.595	32.651
32	963,594	32.119
64	902.708	30.090

Table 4: Training time taken in seconds for 30 epochs by varying the batch size.

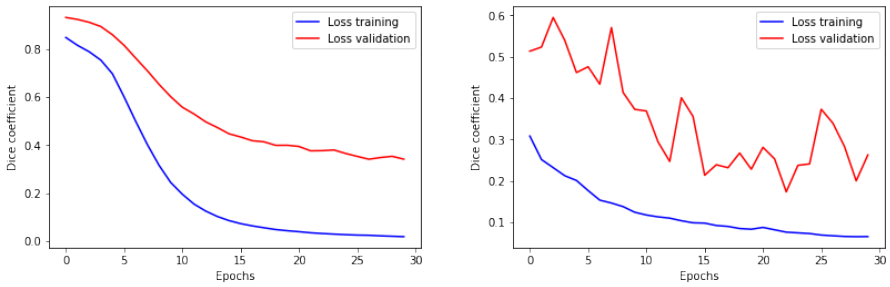

Figure 9: Loss curves with batch size of 16 and learning rate of 0.0001 and 0.1

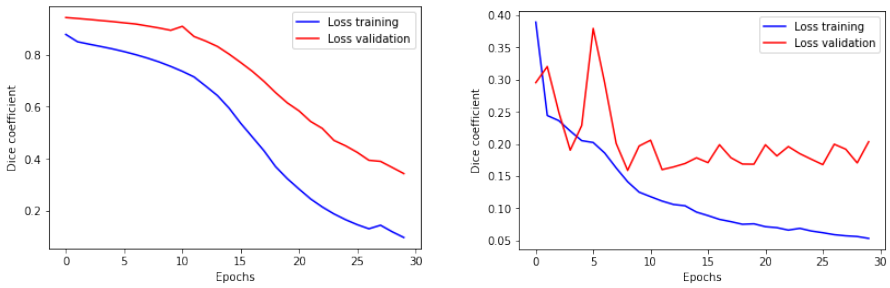

Figure 10: Loss curves with batch size of 32 and learning rate of 0.0001 and 0.1

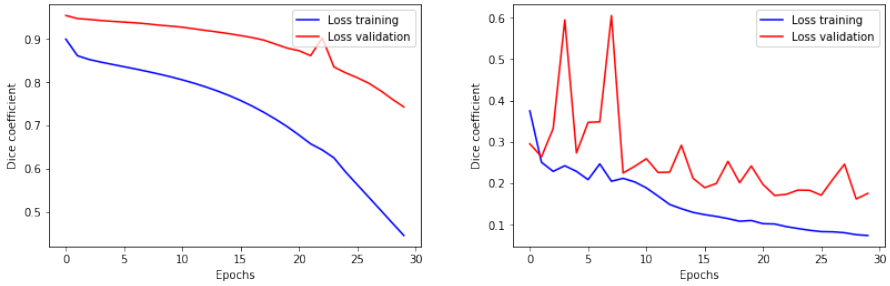

Figure 11: Loss curves with batch size of 64 and learning rate of 0.0001 and 0.1

In Fig.12, Fig.13 and Fig.14 the images obtained after applying the Total Variation denoising algorithm [7] are shown.

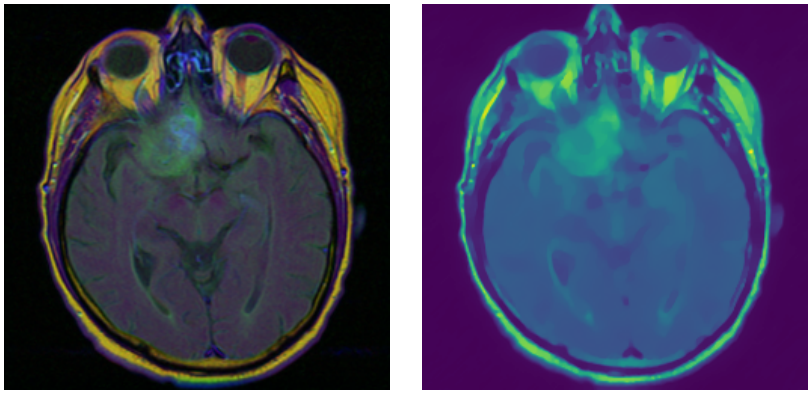

Figure 12: Brain scan with a Tumor obtained by applying Total Variation denoising algorithm

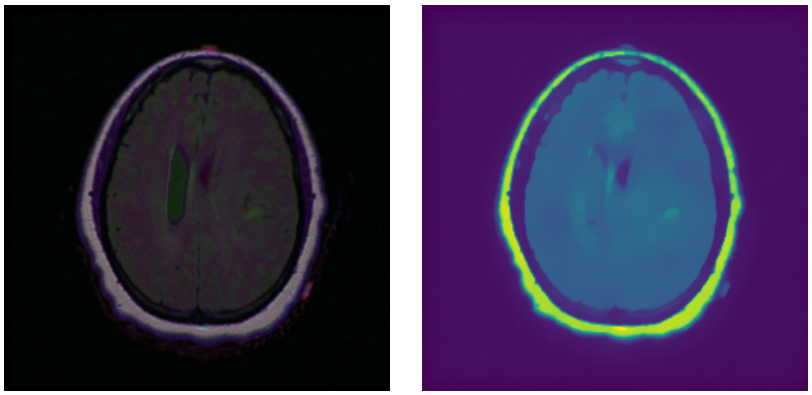

Figure 13: Brain scan with a Tumor obtained by applying Total Variation denoising algorithm

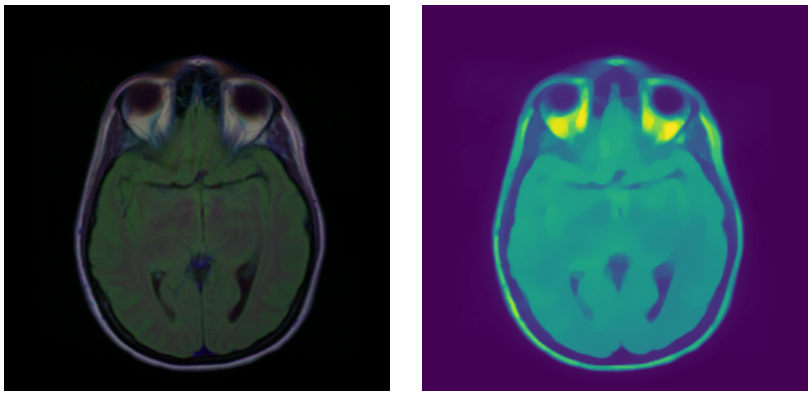

Figure 14: Brain scan **without** a Tumor obtained by applying Total Variation denoising algorithm

5 Discussion

As it can be seen from the table 1 and the table 2 the Batch Normalization technique with a learning rate of 0.0001 does not affect the learning procedure so much. However as expected from the theory, Batch Normalization let the network be trained with a high learning rate. This can be seen when a learning rate of 0.1 is used, in fact without the batch normalization the training error achieved is lower than the one obtained with Batch Normalization. But this result can be misleading, in fact, the validation error is higher than the one obtained when the technique is used. So the model could have been overfitting on the training data and won't be able to generalize properly on unseen data.

Another interesting analysis can be done on the result obtained by varying the batch size used. By looking at the Table 3 what it can be seen is that for a batch size of 16 the

losses discrepancy is 0.32, 0.24 for a batch size of 32, and 0.29 for a batch size of 64. This can lead us to the conclusion that the batch size hyperparameter needs to be tuned to found the best size that suits our problem. In our case for example the best score is achieved with the batch size equal to 32. Also, as it can be seen from the Table 4 on the GPU the time does not increase as the batch size increases, however the memory does increase. In fact, in my case, the GPU was not able to allocate a tensor generated with a batch size of 128 input.

One possible further approach would be to use as input to the CNN multiple images of the same slice obtained by applying some denoising algorithm on the original image. Moreover, an image u can be considered to be a random variable, drawn from a probability distribution. The goal of a denoising algorithm is to find u', the maximally probably hypothesis u solving the inverse problem given an observed image. We can view this procedure as an optimization problem involving a probabilistic model that leads us to a solution which is a version of the original image denoised. Some images obtained after this process are Fig.12, Fig.13 and Fig.14. It is interesting to notice that the first two figures obtained Fig.12, Fig.13 both present a tumour which can be easier to visualize after the denoising algorithm is applied (the tumour region is more yellow) and the Fig.14 which it is easy to view that it does not contain any tumour. The algorithm which I used is called Nonlinear total variation [7].

6 Conclusion and further approaches

As I stated in the Introduction 1, one person out of three will be diagnosed with cancer in his/her lifetime. It is crucial to find cancer as soon as possible to start the therapy. With this project, I had the possibility to explore a bit of the research that has been done in the last 10 years to speedup cancer detection. Currently, the academic research is focused on CNNs models that proved to be more accurate and efficient compared to trained radiology experts. However, the journey to find the perfect solution is not over yet; as we briefly view in this report neural networks are not a deterministic model due to the non-convex nature of the optimization problem of training, also the research of the best hyperparameters is always done by trying a brute force approach to find the best ones in terms of accuracy.

References

[1] M. Buda, A. Saha, and M. A. Mazurowski. Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm. *Computers in Biology and Medicine*, 109:218–225, Jun 2019. ISSN 0010-4825. doi: 10.1016/j.compbiomed.2019.05.002. URL `http://dx.doi.org/10.1016/j.compbiomed.2019.05.002`.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training

by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL `http://arxiv.org/abs/1912.01703`.

[6] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

[7] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.