A U-Net Implementation for Brain MRI Segmentation

Faidra Anastasia Patsatzi and Isabel Whiteley Tscherniak

Department of Electrical and Computer Engineering, Technical University of Munich

March 15, 2023

Abstract — In this project, a dataset containing MRI (magnetic resonance imaging) brain scans, coupled with masks of brain tumors is processed and used to train a U-Net architecture for image segmentation. This U-Net model can then determine the location of brain tumors by outputting a binary image, referred to as a segmentation mask. We aim to evaluate the performance of the created model on this dataset.

1 Introduction

Like in many aspects of our life, AI is playing an increasingly important role in our healthcare systems. Specifically, by using machine learning algorithms, to replace or mimic human cognition in the analysis and comprehension of often very complex medical data. The motivation is that this technology could produce more accurate diagnoses and enable more personalized treatment. Additionally, it could help medical professionals identify disease markers and trends that would otherwise be overlooked.

One of the most promising applications is in biomedical image segmentation. This specific application could free up time for overworked healthcare workers and improve diagnostic accuracy. In the following, we assess the application of the U-Net architecture on this problem.

2 Theoretical Foundations

2.1 Introduction to CNNs and U-Net

Convolutional Neural Networks (CNN) are a type of artificial neural network often applied in computer vision tasks. In the case of semantic image segmentation, a CNN is employed for pixel-level classification and outputs the desired segmentation map [1]. U-Net is a particular implementation of a CNN developed by the Computer Science Department of the University of Freiburg [2], specifically created for biomedical image segmentation. U-Net is a fully convolutional network, i.e. it does not include any fully connected layers in its architecture, as opposed to a CNN, which can include fully connected layers. This design has two main advantages [2]. First of all, it needs only few images to train on to achieve high accuracy on test images. Secondly, it is very fast, especially in comparison to previous techniques striving to produce highly accurate binary segmentation masks such as a sliding-window convolutional network.

2.2 U-Net Architecture

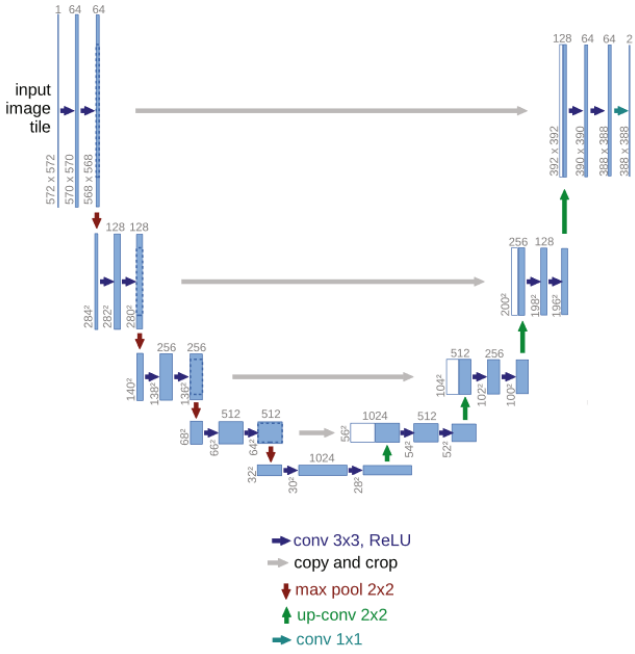

Figure 1 An example of a U-Net architecture for 32x32 pixels in the lowest resolution. Each of the blue rectangles represents a multi-channel feature map. Above the rectangle, the number of channels is indicated and at the lower left edge the x-y-size is listed. The white rectangles denote the copied feature maps and the arrows indicate the different operations [2].

U-Net was named after the 'U' shape of the schematic for the architecture (Figure 1). The input image first undergoes a contraction path (left side of the schematic) in which the input is downsampled and the features of the image are extracted as channels. The right side of the schematic shows the expansive path, where the image is upsampled and results in an output segmentation map. These two paths are roughly symmetrical. In between the contraction and expan-

sive path there are three skip connections. The input in the original U-Net is 572x572 and only has a single channel because the input is a grayscale image [2]. The output in that case is 388x388, and has two channels for two classes. The input size is not equal to the output size, due to the cropping of images after unpadded convolutions. Each step in the downsampling stream consists of two 3x3 valid convolutions and a max pooling layer. The number of channels is doubled in each step. In the upsampling path, the image is upsampled iteratively using a transposed convolution. The skip connections' outputs are concatenated along the channel's dimensions, and equivalent to the downsampling path, each step in the upsampling path has two 3x3 convolutions with valid padding. In the final step there is an additional 1x1 convolution, which does not change the input size of the image, but the number of channels is changed to the desired number of output channels.

The motivation for this architecture can be explained as follows; along the contraction path, the network extracts and learns features from the image, however it loses the spatial information necessary to determine the original location of learned features in the input image. The contraction path is then supposed to reintroduce this information, which is achieved by augmenting the data (through the skip connections) after each upsampling layer [2].

3 Fundamental Data Analysis

3.1 Data Set

The dataset used in this project contains brain MRI scans and manual FLAIR abnormality segmentation masks. These segmentation masks, consisting of white pixels, mark the area where a tumor was identified. The images originate from The Cancer Imaging Archive (TCIA) and include data from 5 hospitals in the US [4].

3.2 MRI Data Pre-preprocessing

To determine how many of the brain scans in the data set include a tumor, we define a diagnosis function. This function labels a scan as positive, if there is one or more non-black (mask) pixels in the scan's segmentation mask. According to this function there are 2556 negative diagnoses and 1373 positive diagnoses in the data set. Positive diagnosis samples are shown in Figure 2.

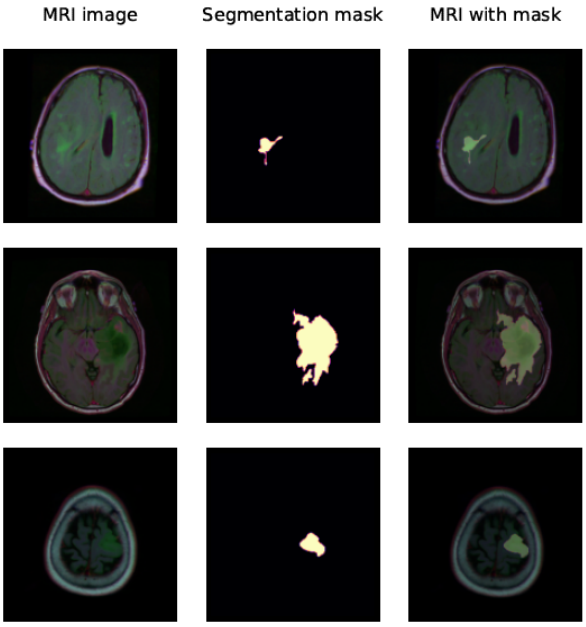

Figure 2 Example positive diagnosis MRI images from the data set and corresponding segmentation masks. The light coloured pixels in the mask denote where a tumor is located.

After determining the diagnoses of all the images in the data set, the next pre-processing step entailed using data augmentation techniques to combat overfitting and improve the model's generalization and robustness [5]. To achieve this, we chose to use the Albumentations Python package [5] to create an augmentation pipeline for the images and segmentation masks after import. All images were resized to 128x128 pixels and normalized to have zero mean and unit variance. The training set received additional augmentation through horizontal and vertical flips as well as a random rotation of 90 degrees. These augmentations were applied with a probability of 50% to each image.

As a next step we removed very dark brain scans. This was done by filtering out scans (and their corresponding masks) with a disproportionately low number of non-black pixels. Data cleaning was also achieved in this step, since completely black brain scan images were removed from the data set. A total of 50 samples, specifically 47 negative diagnoses and 3 positive diagnoses, were removed from the data set. Examples of such images can be seen in Figure 3.

Further augmentation, such as mirroring of areas around the border and padding, was not deemed necessary since the brain scans are round and printed on a black square background. Therefore, there are no tumors on the borders of any images and the model is not expected to produce segmentation maps which include border pixels.

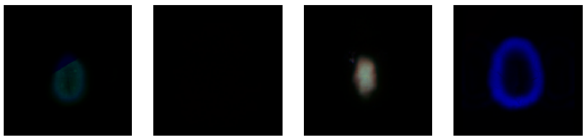

Figure 3 Four examples of images removed in data preprocessing.

4 Model Structure and Implementation

4.1 Creating and fitting the model

The model structure closely follows that of the original 2015 paper. We chose to implement the model in Python using the PyTorch package [6]. We chose to make some changes to improve the original architecture, also based on methods that were developed after the U-Net paper was published in 2015. This includes using a batch normalization layer [7] after each convolution. In order to preserve the original image size in the output segmentation map, we deciced to use zero padding instead of valid padding (unpadded) in all convolutional layers. Additionally, we opted for a one-channel output, instead of a two-channel output as in the original architecture.

Adjusting the network's architecture to the one-channel output, we applied a sigmoid function after the final convolutional layer instead of a softmax function, to derive the class probabilities. A considerable class imbalance, resulting from the definition of classes on pixel-level and the existence of disproportionately many negative diagnoses with completely black segmentation masks, was compensated by adjusting the classification threshold for the sigmoid function's output values [8]. This threshold was set to 0.3.

4.2 Scoring the model

4.2.1 Implementation of a pixel-wise weighted binary cross entropy loss

For every segmentation mask with two or more separate (not connected) masked regions, a custom function was used in order to calculate a weight map, which assigns higher values to the border pixels of the masked areas. The weight maps are calculated using this formula from *Ronneberger et al.* [2]:

$$w(x) = w_c(x) + w_0 \cdot \exp\left(\frac{-(d_1(x) + d_2(x))^2}{(2\sigma)^2}\right) \quad (1)$$

where w_c are the class weights, d_1 the distance to the border of the nearest tumor and d_2 the distance to the border of the second nearest tumor, if there are two or more tumors identified. If there more than two separate tumors in the segmentation map, all distances are calculated and the shortest are used in the final calculation. In this use case, we choose to assign increased weights to the boundaries of the tumors, in order to emphasize the details in the shape of identified tumors in model training. In our implementation we used $\sigma = 5$ and $w_0 = 10$.

The weighted loss is calculated by elementwise multiplication of the pixel weights with the pixel cross entropy losses. For this we defined a custom loss function, which multiplies the calculated weight map with the matrix resulting from PyTorch's binary cross entropy loss function [6].

This approach was adopted in order to force the model to learn the border pixels in scans where two or more non-connected tumors are present. The final weight map results from adding the border-pixel weight map and the class weight map, which is used to incorporate the pixel-level class imbalance of the dataset into the model. Weight maps generated using samples from the data set can be seen in Figure 4.

For our weighted loss implementation and image processing (section 3) we used the scikit-learn [9] and scipy [10] Python libraries.

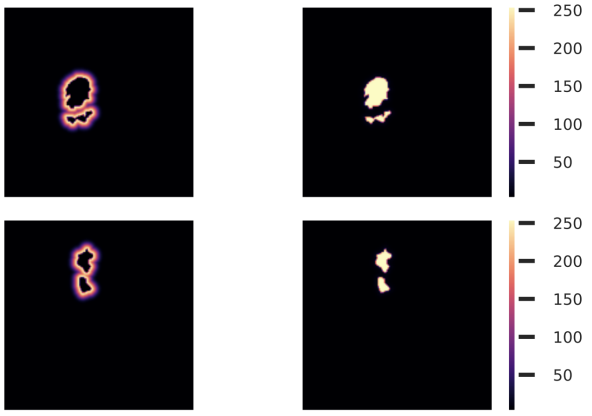

Figure 4 Two examples of weight maps and corresponding segmentation masks.

4.2.2 Dice coefficient

The dice coefficient, also called the Sørensen–Dice coefficient, is a statistic utilized to measure the similarity of two samples.

Given two sets, X and Y, it is defined as

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|} \tag{2}$$

where |X| and |Y| are the cardinalities of the two sets [11]. In our implementation, X is the predicted mask and Y is the true mask. The dice coefficient equals twice the number of elements common to both sets divided by the sum of the number of elements in each set. We used the dice coefficient both in a loss function and as an accuracy metric.

4.2.3 Loss in model training

The pixel-wise weighted binary cross entropy loss is one of the loss functions defined for model training. We also chose to use a dice loss, based on the dice coefficient as described in the section above. The third loss function we defined can be calculated by adding the dice loss to the weighted loss. This combination of the two loss functions is referred to as "combined loss" in the following sections.

4.2.4 Pixel-wise accuracy

In addition to dice accuracy, we also score our model using pixel accuracy. This accuracy measure indicates the number of pixels that are classified correctly in the generated segmentation mask. According to *Hurtado* and *Valada* [12], "this metric calculates the ratio between the amount of adequately classified pixels and the total number of pixels in the image as:"

$$PA = \frac{\sum_{j=1}^{k} n_{jj}}{\sum_{j=1}^{k} t_j} \tag{3}$$

where n_{jj} is the total number of pixels both classified and labeled as class j. In other words, n_{jj} corresponds to the total number of true positives for class j. t_j is the total number of pixels labeled as class j.

5 Performance and Results

The data set was first split into a training, validation and test set in a 72:8:20 percent ratio. The default parameters chosen for model training are a batch size of 10 due to GPU memory limits, 50 epochs and an initial learning rate of 10^{-4}. PyTorch's Adam was used as an optimizer [6].

Dice accuracy was chosen as the most appropriate accuracy metric, since our model achieved a pixel accuracy of over 97% from the first few epochs. This high score can be justified by the fact that a correctly classified background pixel, in both negative and positive diagnoses, is treated by the pixel accuracy metric equally to a correctly classified mask pixel. Meanwhile, the dice coefficient focuses on the overlap of predicted and true mask pixels and is thus considered a more appropriate accuracy metric for image segmentation [11]. In order to measure the performance of the model with different loss functions, we trained it using the configurations documented in Table 1.

Config.	Loss function	Training dice accuracy	Test dice accuracy
1	Weight BCE	0.80	0.77
2	Dice	0.75	0.73
3	Combined	0.83	0.85

Table 1 Performance using different loss functions.

We then trained the model using the combined loss, since configuration 3 yielded the best results, with different learning rates as shown in Table 2. The number of epochs was increased for the lowest learning rate.

Learning rate	Epochs	Training dice accuracy	Test dice accuracy
$0.5 \cdot 10^{-4}$	70	0.83	0.84
10^{-4}	50	0.83	0.85
$0.5 \cdot 10^{-3}$	50	0.82	0.79

Table 2 Performance using the combined loss function with different learning rates.

As can be seen in the learning curve in Figure 5, our model using the combined loss function achieves a significant reduction in both training and validation loss in the first few epochs, and continues to effectively minimize both losses until epoch 50. The learning curve does, however, indicate learning instability, considering the frequent minor peaks and drops especially in the validation loss curve. This behaviour can be attributed to the fact that we used a custom loss function combining two losses (multi-task learning), and the resulting challenge of finding the optimal trade-off between the two losses.

The highest test accuracy achieved overall as seen in Table 1 amounts to 85%. The predicted masks shown in Figure 6 have been generated by running the best model on the test set and sampling the results.

6 Conclusion

This project demonstrates how a U-Net deep learning architecture can be employed in healthcare, by train-

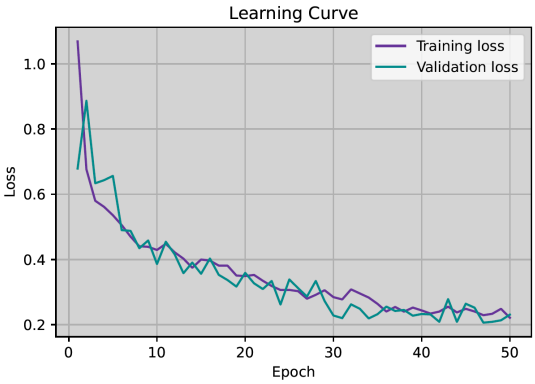

Figure 5 Learning curve of model training with parameter configuration 1.

ing it to identify brain tumors in MRI scans. Our approach utilizes the U-Net architecture detailed in *Ronneberger et al.* [2]. In our implementation, we make minor adjustments to the original architecture, such as using only one output channel and zero padding in all convolutions. Additionally, the appropriate preprocessing steps are taken to categorize and clean the data as well as to augment the images to improve the model's robustness and generalization capability. This model was trained using a custom loss function based on the weighted binary cross-entropy and dice losses, ultimately reaching a maximum of 85% dice accuracy on the test set.

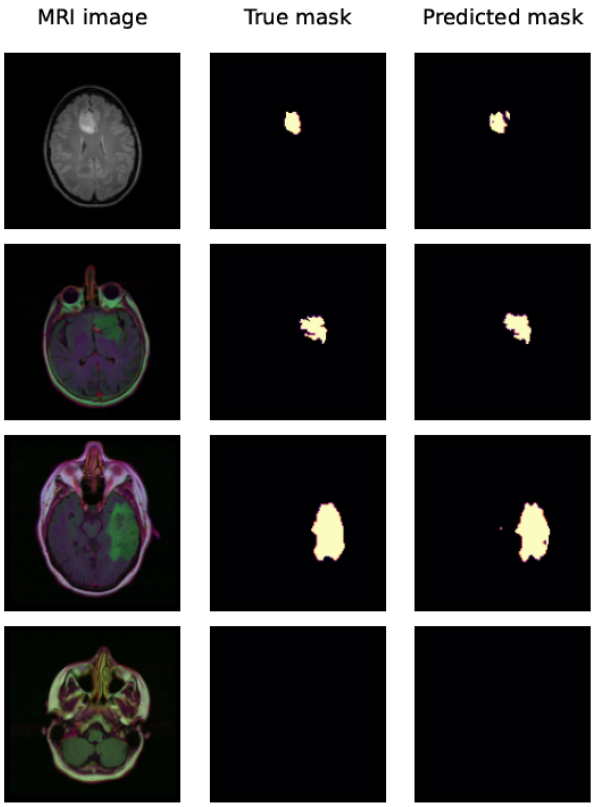

Figure 6 MRI samples, true segmentation masks and predicted segmentation masks.

References

[1] Liu, X., et al.. "Recent Progress in Semantic Image Segmentation". Artificial Intelligence Review, vol. 52, no. 2, Artificial Intelligence Review, 2019, pp. 1089–106, doi:10.1007/s10462-018-9641-3.

[2] Ronneberger, O., et al.. "U-net: Convolutional Networks for Biomedical Image Segmentation". STACS 98, STACS 98, 2015, pp. 234–41, doi:10.1007/978-3-319-24574-4_28.

[3] Du, Get, et al. "Medical Image Segmentation Based on U-Net: A Review." Research Gate, Journal of Imaging Science and Technology, 2020, https://www.researchgate.net/publication/339929534_Medical_Image_Segmentation_based_on_U-Net_A_Review.

[4] "The Cancer Genome Atlas Low Grade Glioma Collection (TCGA-LGG)." The Cancer Genome Atlas Low Grade Glioma Collection (TCGA-LGG) - The Cancer Imaging Archive (TCIA) Public Access - Cancer Imaging Archive Wiki, doi.org/10.7937/K9/TCIA.2016.L4LTD3TK.

[5] Buslaev, A., et al.. "Albumentations: Fast and Flexible Image Augmentations". Information, vol. 11, no. 2, Information, 2020, p. 125, doi:10.3390/info11020125.

[6] Paszke, Adam, et al. "Pytorch: An Imperative Style, High-Performance Deep Learning Library." ArXiv.org, 3 Dec. 2019, arxiv.org/abs/1912.01703.

[7] Ioffe, Sergey, and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." ArXiv.org, 2 Mar. 2015, arxiv.org/abs/1502.03167.

[8] Esposito, C., et al.. "GHOST: Adjusting the Decision Threshold to Handle Imbalanced Data in Machine Learning". Journal of Chemical Information and Modeling, vol. 61, no. 6, Journal of Chemical Information and Modeling, 2021, pp. 2623–40, doi:10.1021/acs.jcim.1c00160.

[9] Pedregosa, Fabian, et al. "Scikit-Learn: Machine Learning in Python." Journal of Machine Learning Research, 10 Nov. 2011,

[10] Virtanen, Pauli, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." Nature News, Nature Publishing Group, 3 Feb. 2020, www.nature.com/articles/s41592-019-0686-2. jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

[11] Zou, K. H., et al.. "Statistical Validation of Image Segmentation Quality Based on a Spatial Overlap Index1". Academic Radiology, vol. 11, no. 2, Academic Radiology, 2004, pp. 178–89, doi:10.1016/s1076-6332(03)00671-8.

[12] Hurtado, Juana Valeria, and Abhinav Valada. "Semantic Scene Segmentation for Robotics." Deep Learning for Robot Perception and Cognition, Academic Press, 11 Feb. 2022, https://www.sciencedirect.com/science/article/pii/B9780323857871000178.