Technical Report Series
Center for Data and Simulation Science

Giulia Baldini, Melanie Schmidt, Charlotte Zäske, Liliana L. Caldeira

Mapping Brain Clusterings to Reproduce Missing MRI Scans

Technical Report ID: CDS-2021-05
Available at https://kups.ub.uni-koeln.de/id/eprint/46376
Submitted on April 28, 2021

Mapping Brain Clusterings to Reproduce Missing MRI Scans

Giulia Baldini*,

Melanie Schmidt*, Charlotte Zäske†, Liliana L. Caldeira†

This report is Giulia Baldini's master's thesis, which was written in a cooperative project between the University of Cologne and the University Hospital of Cologne under the supervision of Melanie Schmidt and Liliana L. Caldeira and with manually segmented brain scans from Charlotte Zäske.

Abstract Machine learning has become an essential part of medical imaging research. For example, convolutional neural networks (CNNs) are used to perform brain tumor segmentation, which is the process of distinguishing between tumoral and healthy cells. This task is often carried out using four different magnetic resonance imaging (MRI) scans of the patient. Due to the cost and effort required to produce the scans, oftentimes one of the four scans is missing, making the segmentation process more tedious. To obviate this problem, we propose two MRI-to-MRI translation approaches that synthesize an approximation of the missing image from an existing one. In particular, we focus on creating the missing T2 Weighted sequence from a given T1 Weighted sequence. We investigate clustering as a solution to this problem and propose BrainClustering, a learning method that creates approximation tables that can be queried to retrieve the missing image. The images are clustered with hierarchical clustering methods to identify the main tissues of the brain, but also to capture the different signal intensities in local areas. We compare this method to the general image-to-image translation tool Pix2Pix, which we extend to fit our purposes. Finally, we assess the quality of the approximated solutions by evaluating the tumor segmentations that can be achieved using the synthesized outputs. Pix2Pix achieves the most realistic approximations, but the tumor areas are too generalized to compute optimal tumor segmentations. BrainClustering obtains transformations that deviate more from the original image but still provide better segmentations in terms of Hausdorff distance and Dice score. Surprisingly, using the complement of T1 Weighted (i.e. inverting the color of each pixel) also achieves good results. Our new methods make segmentation software more feasible in practice by allowing the software to utilize all four MRI scans, even if one of the scans is missing.

Keywords Clustering, hierarchical clustering, k-means, medical imaging, neural networks, Pix2Pix.

Acknowledgment We thank Petra Mutzel[1] and Jonas Charfreitag[1] for fruitful discussions, for providing access to their GPU workstation and for general support. We thank Thomas Lange[2] for the technical support with the Küppers cluster. We thank the Center for Data and Simulation Science[3] for creating the cooperation between the University of Cologne and the University Hospital of Cologne and for giving us access to their GPU machines.

*Department for Mathematics and Computer Science; University of Cologne, Weyertal 86-90, 50931, Cologne, Germany

†Radiology Department; University Hospital of Cologne, Kerpener Str. 62, 50937, Cologne, Germany

[1] https://ca.cs.uni-bonn.de/doku.php?id=start

[2] https://cs.uni-koeln.de/ls-juenger/people/lange

[3] https://cds.uni-koeln.de/en/

1 Introduction

Modern biomedical sciences have integrated the use of computer science to automate tasks. For example, machine learning can be used for brain tumor segmentation, which is the process of distinguishing between tumor and healthy brain cells. The annual BraTS Challenge[4] has the aim of awarding the best strategy for this purpose. Every year, the challenge focuses on a new data set, which is composed of four different three-dimensional MRI (*Magnetic Resonance Imaging*) scans of each patient's brain.

DeepMedic [23] is a brain tumor segmentation tool used by the radiology department of the University Hospital of Cologne, and it was trained with all four MRI images in order to produce robust results. However, it is uncommon to always have all four images; oftentimes only three are acquired because they are a time-consuming and costly resource.
This work aims to approximate missing MRI images; if only three images are given, we compute an approximation of the fourth one such that software like DeepMedic can be used. This obliterates the necessity of having all four images.
For this purpose we propose and evaluate two different methods: BrainClustering and Pix2Pix [20]. BrainClustering is a new approach and involves running clustering algorithms on the MRI scans we are interested in. Then, we compute tables that summarize the values that are present in the MRI images. After repeating this process with many images, we obtain extensive tables that can be queried to compute a new image. Pix2Pix is an already existing image translation method that is natively implemented for two-dimensional images. We modify this method to accept the three-dimensional structure of our data.

Finally, we evaluate the methods by running DeepMedic and HD-GLIO [19, 24] (another segmentation software) with four original images, and then swapping one of the images with the one produced by BrainClustering and Pix2Pix respectively. We use common measures such as the mean squared error, Dice score and Hausdorff distance to assess the results.

2 Data

In this section, we describe the MRI scans used throughout this work, which are taken from BraTS 13 and 19 [31, 8, 9, 7, 6]. Each MRI image is a three-dimensional image in NIfTI (*Neuroimaging Informatics Technology Initiative*) format. There are three points of view of the three-dimensional image: transverse, coronal and saggital (Figure 1), but for simplicity will only show the transverse slices.

There are four types of MRI scans that are relevant to this work: T1 Weighted, T2 Weighted, T1 Contrast Enhanced and T2 Fluid Attenuated Inversion Recovery (T2 FLAIR). They differ in *repetition time* and *time to echo*: Repetition Time (TR) is the time between subsequent excitation pulses of the MRI scanner. Time to Echo (TE) is the time between the center of the pulse and the reception of the echo signal. Additionally, T1 Contrast Enhanced requires a special substance, Gadolinium, to be injected into the patient before the scan. It enhances bright signals in the blood vessels and locations where the bloods vessels are disrupted, such as tumors. These differences result in some specific parts of the brain to be light in some sequences, and dark in others, as shown in Figure 2.
Moreover, the different tissues of the brain have the same general signal intensity across each scan, and we believe that there is a mapping that can be deducted between the scans.

MRI Mapping: Let t and s be any two of the four MRI sequences discussed above. An MRI mapping is a function f with $f(t) \approx s$, i.e. a function that is able to describe a mapping between sequence t and sequence s and can be used to transform one to a good approximation of the other.

[4] https://www.med.upenn.edu/cbica/brats-2019/

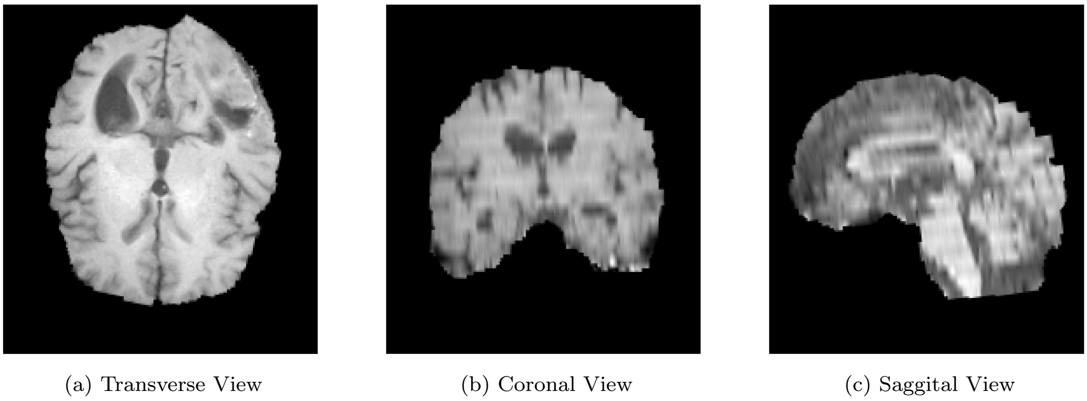

(a) Transverse View (b) Coronal View (c) Saggital View

FIGURE 1: Brain MRI scan of patient 5 (BraTS 13) from different points of view.

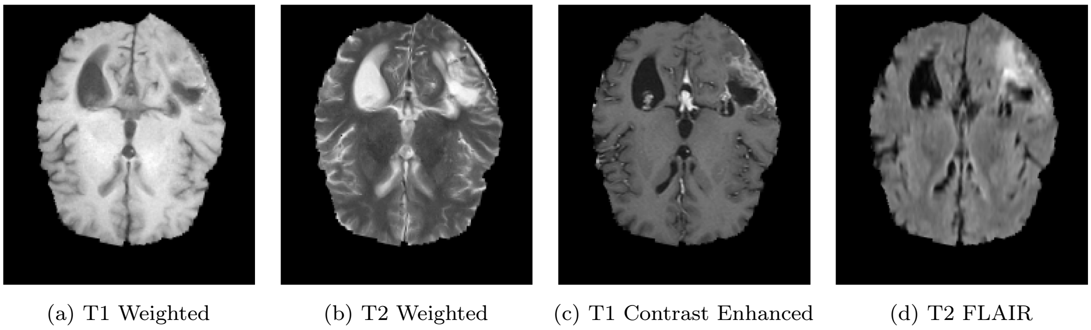

(a) T1 Weighted (b) T2 Weighted (c) T1 Contrast Enhanced (d) T2 FLAIR

FIGURE 2: Brain MRI scan of patient 5 (BraTS 13) with respect to the four discussed sequences.

Table 1 shows a general brightness mapping between T1 Weighted, T2 Weighted and T2 FLAIR with respect to different tissues, which can also be observed in Figure 2. The presence of a regional mapping means that, if we are able to divide the brain into smaller parts, then we can reproduce this mapping and use it to obtain one image given another.

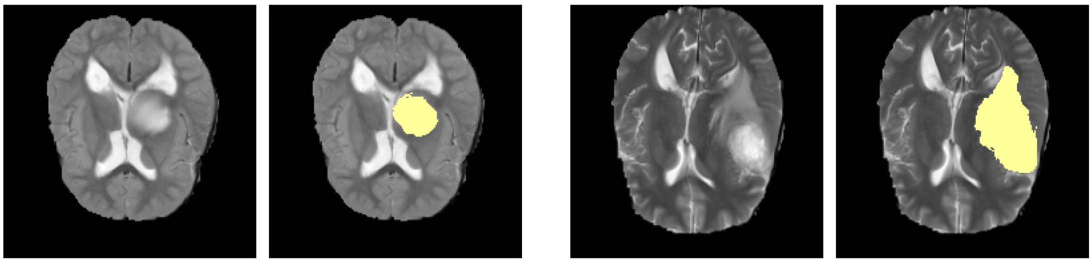
2.1 Tumor Segmentation

In addition to the four MRI scans, BraTS also provides the ground truth of the tumor for most of the images. Moreover, the data is divided into two subsets according to the tumor type: patients with *Low Grade Glioma* (LGG) and patients with *High Grade Glioma* (HGG).
Gliomas are a type of brain tumor that starts from non-neuronal cells. HGG are more invasive

Table 1: Color mapping of T1 Weighted, T2 Weighted and T2 FLAIR with respect to different tissues[5].

Tissue	T1 Weighted	T2 Weighted	T2 FLAIR
White Matter	Bright	Dark	Dark
Cerebrospinal Fluid	Black	White	Black
Inflammation	Dark	Bright	Bright

[5] Source: https://casemed.case.edu/clerkships/neurology/Web%20Neurorad/MRI%20Basics.htm, currently unavailable.



(a) Low Grade Glioma

(b) High Grade Glioma

Figure 3: [Best viewed in color] Example of patients with LGG (patient 0 from BraTS 19) and HGG (patient 10 from BraTS 19) together with their tumor segmentation (yellow).

and progress faster than LGG. The type of therapy that a patient undergoes depends on the type of glioma. Figure 3 provides an example of a low grade and a high grade glioma.

The BraTS 19 ground truth is annotated manually by one to four raters, and their labels have been approved by experienced neurologists. The segmentation is divided into four classes:

Class 1: The necrotic core of the tumor.

Class 2: An excess of fluid, also called cerebral edema, associated with the tumor.

Class 3: Non-enhanced part of the tumor.

Class 4: Contrast-enhanced part of the tumor, excluding the necrotic center.

The experts segmented the edema (Class 2) from T2 Weighted, and then reviewed its extension with T2 FLAIR. Initially this class contains all the other structures, which are relabeled in the following steps. Afterwards, the gross tumor core (Class 2 + 3 + 4) is found by investigating the hyper-intense lesions in T1 Contrast Enhanced (for the high grade glioma case) and in T2 Weighted and the hypo-intense lesions in T1 Weighted. The enhanced part of the tumor can be found by comparing the tumor core with T1 Contrast Enhanced, considering the T1 Weighted enhancing rim and excluding the necrotic core. The necrotic core (Class 1) is identified by the low intensity structures inside the enhancing rim of T1 Contrast Enhanced. Finally, the non-enhancing part (Class 3) is made up by the parts of the gross tumor core that neither belong to the enhancing core nor to the necrotic center.
However, the non-enhancing class is sometimes overestimated. Thus, from BraTS 17 on, it is combined with the necrotic class, producing the final three classes: Necrotic/Non-Enhancing Tumor, Edema and Contrast Enhanced Tumor.

Nonetheless, these are not the classes used for the BraTS evaluation. For the experiments, they propose three regions and compute their evaluation measures on each individually. The first class is Whole Tumor, which contains the entirety of the tumor. The second class is Tumor Core, which contains all parts but the edema. The last class contains only the active tumor, that is, the contrast enhanced part (without the necrotic area). The four original classes and the binary regions are presented with an example in Figure 4.

We also use these same regions for our evaluation, which we perform using two pieces of segmentation software. The first one, DeepMedic, was designed by the first place winner of BraTS 17. Given the four different brain MRI scans as input, DeepMedic uses a multi-scale three-dimensional Deep Convolutional Neural Network to achieve brain lesion segmentation. The radiology department of the University Hospital of Cologne used a clinical evaluation [36] on DeepMedic to train the neural network they currently use for this purpose. However, since the model was created 2017 and it was trained on the BraTS 2015 data, we decided to train it again using the BraTS 2019 data set. More information on how the model was trained can be found in Section 8.3.

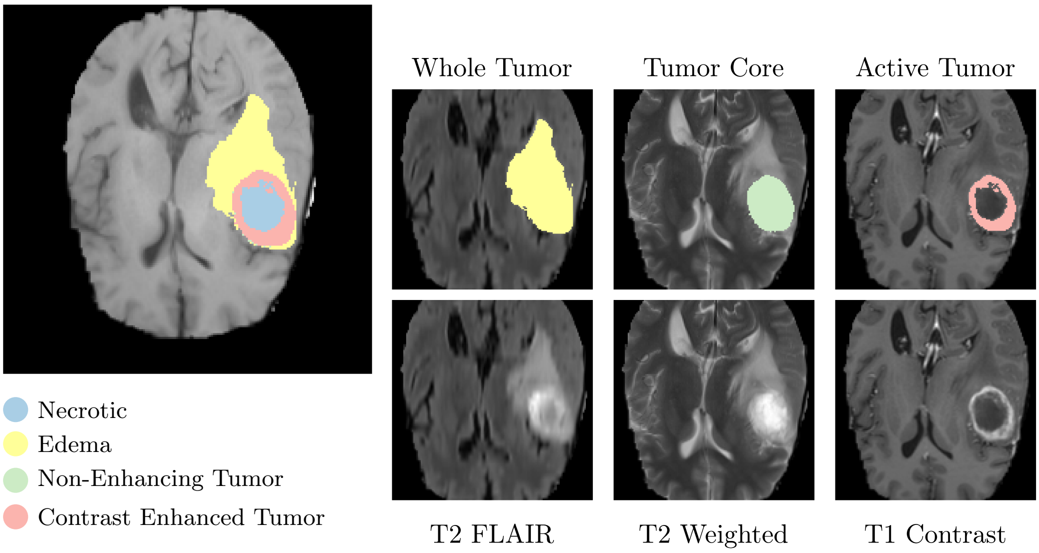


Figure 4: [Best viewed in color] Classes of tumor for patient 1 from BraTS 13. In the first picture all four classes are superimposed on T1 Weighted, even though the non-enhancing tumor is very small. On the right, the three regions for the evaluations of the tumor segmentation. The first region, Whole Tumor, can be mostly identified in T2 FLAIR, while Tumor Core and Active Tumor can be respectively determined in T2 Weighted and T1 Contrast Enhanced. This representation is inspired by Bakas et al. [9].

To exhaustively test our results we also evaluate our mappings using another tumor segmentation software called HD-GLIO, which is based on nnU-Net, a deep learning framework created by the second place winner of BraTS 18. The nnU-Net method can not only be used for finding tumors in the brain, but also in many other organs.

2.2 Preprocessing

In the field of tumor segmentation the images produced by an MRI scanner need to be preprocessed before being analyzed. Although the BraTS images used in this paper are already preprocessed, we would like to give a short summary of the process. A first important step is to make sure that all images have the same resolution, that is, they need to have the same voxel size. A voxel is a unit of three-dimensional graphic information. The images from BraTS are processed to have a voxel size of 1mm^3. Skull stripping is a very common procedure that removes the skull from the MRI scans and normalizes the image. This is necessary to ensure that the software is able to focus on the tissue. Commonly used software is BET from FSL [39, 21] and HD-BET [18]. This process creates a brain mask, which is a three-dimensional array that represents with zeros the background, and with ones the brain voxels. This is generally used as input for tumor segmentation software, or is applied to the input MRI such that the background is zero. Additionally, it is beneficial if the scans belonging to a patient are aligned such that it is possible to see common aspects and eventual differences. This alignment process is called registration. The BraTS images have been co-registered with respect to the same anatomical template.

3 Related Work

We are not the first to apply clustering on MRI images [33, 11, 29, 10]. In fact, clustering has been used as a resource to achieve brain tissue segmentation, which is the process of distinguishing the different tissues of the brain. In particular, Caldeira et al. [10] use k-means and Expectation-

Maximization to achieve this purpose. By contrast, in our work we focus on hierarchical clustering algorithms, because we believe that they more accurately capture the coloring structure of each tissue. However, our purpose extends beyond tissue segmentation and to the best of our knowledge we are the first to involve clustering for MRI translation.

Nonetheless, the general image-to-image translation problem for two-dimensional images has been extensively studied. In 2017, Pix2Pix [20] was proposed: A neural network model able to learn the mapping between any two related images (see Section 7.1). This paper sparked a lot of interest and many repositories with a similar purpose as ours appeared on GitHub[6,7,8,9]. Some, even with the same purpose[10]. In this particular project MRI translation is only stated as an objective, and the work on this repository has never been completed. Even though Pix2Pix (and similar networks) have been used in the field of medical imaging, we found no evidence of literature mentioning or conducting an evaluation on MRI translation.

4 Clustering

This chapter presents the clustering algorithms adopted in our work both in theory and in practice. An evaluation of our clustering methods follows in Section 5, and some illustrations for all algorithms discussed here are provided in Section 5 and Appendix A.

Conceptually speaking, clustering is the act of partitioning data points into groups such that similar points belong to the same group. If each three-dimensional pixel of an MRI scan represents a data point, then we can cluster these data points by their color in order to identify the different tissue types of the brain. This is due to the fact that different tissues assume distinct colors depending on the chosen sequence (see Table 1). Thus, clustering algorithms have the potential to detect brain tissues. Additionally, we also want to capture different shades of color inside each tissue, which can be achieved using hierarchical clustering algorithms. In this chapter, we present three different hierarchical clustering algorithms, two of which are based on k-means.

4.1 Data Dimensionality

Each brain scan is composed of $240 \times 240 \times 155 \approx 9$ million three-dimensional pixels. Around 80–90% of the scan consists of the background, while the brain matter accounts for roughly one to two million pixels. When clustering, we may either interpret each pixel as a four-dimensional coordinate (three spatial dimensions and the color), or as a (one-dimensional) color value. We believe that excluding the spatial information and considering only the color should be the correct approach, since similar tissues assume similar colors in the different scans, and tissues of the same type are not necessarily spatially connected. This means that the three-dimensional matrix of pixels can be transformed into a one-dimensional array without losing any relevant information.

The execution time of most clustering algorithms heavily depends on the dimensionality. Thus, an additional benefit of omitting the spatial information is a significant speedup compared to multi-dimensional clustering. Although we are not considering spatial data in the final version of our implementation, we did consider it during earlier stages of our research by using a weighted distance function, such that the spatial distances are less relevant than the color. The clustering results are worse than a pure color clustering, as can be seen in Section 5.3.

4.2 k-means Clustering

Lloyd's k-means clustering algorithm [27] is one of the most popular unsupervised learning methods, and is a heuristic for the k-means clustering problem.

[6] https://github.com/tychovdo/RevGAN

[7] https://github.com/imatge-upc/3D-GAN-superresolution

[8] https://github.com/arnab39/FewShot_GAN-Unet3D

[9] https://github.com/neoamos/3d-pix2pix-CycleGAN

[10] https://github.com/ravnoor/MRI-GAN

Definition 4.1 *k-means Clustering*
Let $\boldsymbol{x} \subseteq \mathbb{R}^d$ be a set of d-dimensional points, and let $S = \{C_1, C_2, \dots, C_k\}$ be a partition of $\boldsymbol{x}$. For each C_i, let $\mu_i = \frac{1}{|C_i|}\sum_{x_j \in C_i} x_j$ be its cluster center (i.e. μ_i is the mean of the points in C_i). The k-means problem consists in finding a partition S that minimizes the sum of within-cluster sum of squares (i.e. cluster variance), which is formally defined as

$$\sum_{i=1}^{k} \sum_{x_j \in C_i} \|\boldsymbol{x}_j - \mu_i\|_2^2. \tag{1}$$

The problem is NP-hard for $k \geq 2$ [3], and oftentimes efficient heuristic algorithms are used to obtain locally optimal solutions. There exist algorithms that achieve constant approximation rates, such as the primal-dual 6.357-approximation algorithm by Ahmadian et al. [2].
In the one-dimensional case, an optimal solution to the k-means problem can be found in polynomial time by exploiting the fact that the optimal clusters form consecutive intervals in the *sorted* list of points. This allows efficient dynamic programming solutions like k-means1d[11], which is based on work by Wu [43] and Grønlund et al. [15] and finds the optimal clustering in $\mathcal{O}(kn + n\lg n)$ time. We use this implementation whenever we deal with one-dimensional data. For multi-dimensional data we use k-means++ [5], which uses Lloyd's algorithm with a smart initialization strategy and computes a $\mathcal{O}(\log k)$-approximation of the optimal clustering.

4.2.1 k-means1d

As mentioned earlier, k-means1d uses dynamic programming to compute an optimal k-means clustering in the one-dimensional setting. Dynamic programming is the art of solving a problem by breaking it down into smaller subproblems that can stored for later use. This is many times done with the use of tables.
Let us consider $\boldsymbol{x} \subseteq \mathbb{R}$ sorted in non-descending order, and let k be the desired number of clusters. In this case the problem is divided into smaller tasks by computing the optimal clustering for each number $i = 1, \dots, k$ of clusters, and each prefix $x_1, \dots, x_m$ of $\boldsymbol{x}$, where $m = 1, \dots, n$.
We use a $k \times n$ matrix D that keeps track of the cluster costs, where the optimal cluster cost for the prefix $x_1, \dots, x_m$ and partition size i is saved in $D[i][m]$. We can fill the matrix by exploiting the function $\text{sumsq}(j, m)$, which returns the cost of clustering points $x_j, \dots, x_m$ into one cluster, i.e. the sum of squared distances for $x_j, \dots, x_m$ to their mean $\mu_{j,m} = \frac{x_j + \cdots + x_m}{m-j+1}$.

$$\begin{aligned}
\text{sumsq}(j, m) &= \sum_{\ell=j}^{m} (x_\ell - \mu_{j,m})^2 \\
&= \sum_{\ell=j}^{m} (x_\ell^2 + \mu_{j,m}^2 - 2x_\ell \mu_{j,m}) \\
&= \underbrace{\left(\sum_{\ell=j}^{m} x_\ell^2\right)}_{\beta_{j,m}} + (m - j + 1)\mu_{j,m}^2 - 2\mu_{j,m} \underbrace{\sum_{\ell=j}^{m}(x_\ell)}_{\alpha_{j,m}}
\end{aligned}$$

In order to answer $\text{sumsq}(j, m)$ for arbitrary j and m in constant time, we first precompute the values $\alpha_{1,q}$ and $\beta_{1,q}$ for $1 \leq q \leq n$. It is easy to see that the total precomputation time is $\mathcal{O}(n)$, since each value α_q (and β_q) can be computed in constant time as $\alpha_{q-1} + x_q$ (and $\beta_{q-1} + x_q^2$). Since we have $\alpha_{j,m} = \alpha_{1,m} - \alpha_{1,j-1}$ (analogously for $\beta_{j,m}$), and $\mu_{j,m} = \alpha_{j,m}/(m - j + 1)$, the precomputed values are sufficient to determine $\text{sumsq}(j, m)$ in constant time.

[11] https://github.com/dstein64/kmeans1d

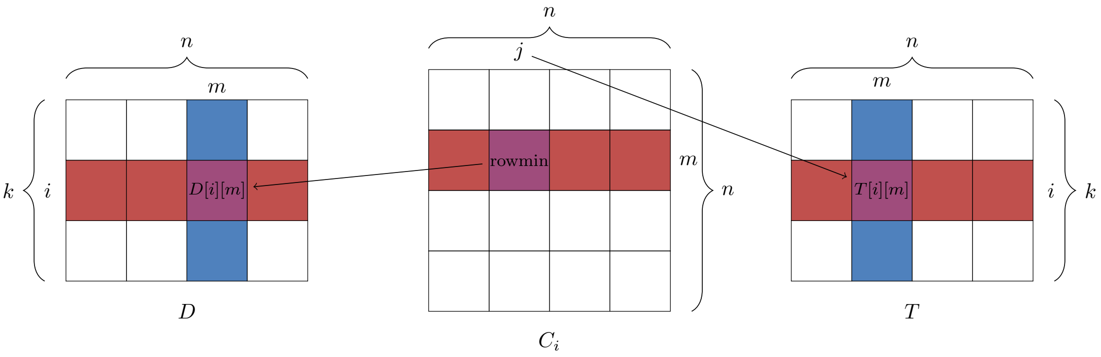


FIGURE 5: [Best viewed in color] Relationship between the C_i, D and T tables. Finding the minimum j in row m in table C_i is equivalent to computing $D[i][m]$ and finding $T[i][m]$.

Each entry $D[1][m]$ of the first row of this matrix, where there is only one cluster, can be computed in constant time as $D[1][m] = \text{sumsq}(1, m)$. The first row is the ground for the following recurrence relation for $i > 1$:

$$D[i][m] = \min_{j=1}^{m} D[i-1][j-1] + \text{sumsq}(j, m). \tag{2}$$

where $D[i-1][j-1]$ is the optimal cost for clustering the prefix $x_1, \dots, x_{j-1}$ into $i-1$ clusters. In this case x_j is the first point of the last cluster recorded until now. We save the index j that yields the minimum in Equation 2, with ties broken by choosing the smallest j, in an auxiliary table T, such that we store the start and end indices of each cluster for each partition i.

$$T[i][m] = \arg\min_{j=1}^{m} D[i-1][j-1] + \text{sumsq}(j, m)$$

Finally, once both tables are built, we find the cluster centers and the assignments for k by backtracking the table T.
The space occupied by the tables is $\mathcal{O}(k \cdot n)$, while the running time is $\mathcal{O}\left(k \cdot n^2\right)$ because for each cell of the matrix we take $\mathcal{O}(n)$ computation time to find the optimal j. However, both time and space complexity can be reduced. We focus on the time and refer to Grønlund et al. [15] for the reduction of space.

Now we reduce the time spent to compute each row of the table to linear time. Let C_i be a $n \times n$ matrix, where $C_i[m][j]$ contains the cost of the optimal clustering of $x_1, \dots, x_m$ into i clusters where the rightmost (highest mean) cluster contains exactly the points $x_j, \dots, x_m$. Using D and sumsq, we can define $C_i[m][j]$ as:

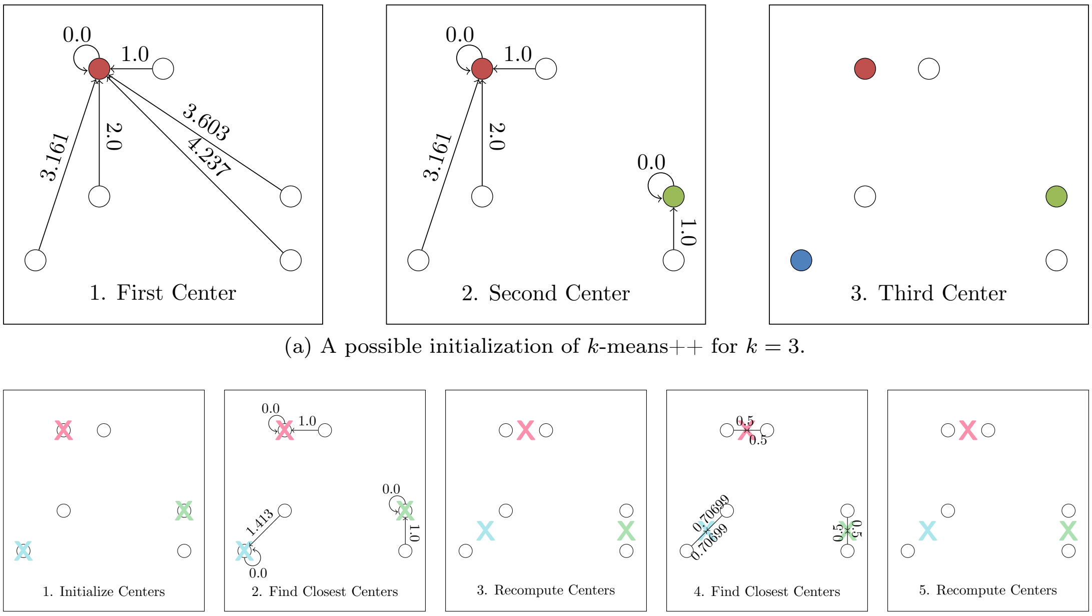
$$C_i[m][j] = D[i-1][j-1] + \text{sumsq}(j, m)$$

Recalling Equation 2, we may in turn define $D[i][m]$ by using the table C_i:

$$D[i][m] = \min_{j=1}^{m} D[i-1][j-1] + \text{sumsq}(j, m) = \min_{j=1}^{m} C_i[m][j].$$

This means that, instead of computing the recurrence relation in Equation 2, we can build $D[i][m]$ by performing computations on the C_i table. The relationship between the tables is shown in Figure 5: Each row of the C_i matrix represents the optimal cost up to a certain j, so if we find the minimum-cost j for every row, then we have computed $D[i][m]$ and $T[i][m]$ of every m with respect to i, i.e. row i of both matrices.

Now we are left with the problem of finding the minimum in each row of the C_i matrix. We use the *SMAWK* [1] algorithm, which efficiently solves exactly this problem for totally monotone matrices.



(a) A possible initialization of k-means++ for $k = 3$.

(b) Lloyd's algorithm.

Figure 6: [Best viewed in color] A full run of k-means++. In (a) we present the center initialization according to the k-means++ algorithm. The first center is chosen at random, then the distances of each point to the closest center are recomputed and a new center is chosen according to the squared distribution of the squared distances. Once all centers have been initialized, we apply Lloyd's method (b). First, we find the closest centers for each data point. The points assigned to one center become a cluster and the cluster centers are recomputed as the center of mass of their points. The process is repeated until the centers converge.

Definition 4.2 *Matrix Monotonicity*
A matrix is **monotone** if the minimum of any row always occurs at an index that is greater than or equal to the index of the minimum of the previous row. A matrix is said to be **totally monotone** if each 2×2 submatrix is monotone i.e. the minima of the rows are not in the bottom left and in the top right of the matrix.

Wu [43] shows that the C_i matrix is indeed totally monotone.
We will not explain the SMAWK algorithm in detail, but give a brief high-level overview of the algorithmic idea. First, the matrix is reduced by creating a stack of columns and progressively removing the ones that cannot have a minimum. Afterwards, it calls the algorithm recursively on the odd-indexed rows and once the minima for these rows are found it uses interpolation to find the minima of the even-indexed rows.
The algorithm takes $\mathcal{O}\left(c(1 + \log(\frac{r}{c}))\right)$ time to compute the minima of all rows of an $r \times c$ matrix, which equals $\mathcal{O}(n)$ time for our $n \times n$ matrix C_i. Note that we do not have to precompute or physically store the matrix C_i, since we can simulate constant time access to it by exploiting already computed entries of D and the function sumsq. Since we run the SMAWK algorithm on k matrices, the total computation time is bounded by $\mathcal{O}(k \cdot n)$. Since we required the data points to be sorted, the time bound of k-means1d is $\mathcal{O}(n \log n + k \cdot n)$. The working space with the optimization by Grønlund et al. [15] is bound by $\mathcal{O}(n)$.

4.2.2 k-means++

As mentioned earlier, one of the most popular heuristics for multi-dimensional k-means is Lloyd's algorithm [27]. However, it has the downside that the solution quality heavily depends on the initialization, and (in case of a poor initialization) it may produce results that are arbitrarily worse than the optimal solution. The algorithm k-means++ [5] obviates this problem by providing a smart initialization that chooses the initial cluster centers by selecting data points with a probability that is inversely proportional to the square of their distance to the already initialized centers. Thus, the first centers are likely to be points that have great distance from each other, yielding a better convergence. The steps of the k-means++ algorithm are described in Algorithm 1, and in Figure 6 we provide an example for three clusters and six points.

Algorithm 1 k-means++

Input: Data set $\boldsymbol{x}$ of length n and dimension d, a number of clusters k.
Output: A list of length n representing the cluster assignment with k different labels, a list of k centers.

1. Initialize the centers:
   (a) The first center is chosen uniformly at random among the data points.
   (b) For each point x_i in the data set, compute the smallest Euclidean distance between the point and the centers c_j that have already been chosen:
   $$D(x_i) = \min_{j=1} \|x_i - c_j\|_2.$$
   (c) Choose the next center from the existing data points, where each point x_i is sampled with probability:
   $$p(x_i) = \frac{D(x_i)^2}{\sum_{x_j \in \boldsymbol{x}} D(x_j)^2}.$$
   (d) If k centers have been assigned, then stop. Otherwise, go to Step 1b.
2. Assign each point of the data set to their closest center. All points assigned to one center build a cluster.
3. For each cluster, compute the centroids: Each centroid is given by the mean of the data points assigned to that clusters. The centroids become the new centers.
4. Compute the so called Frobenius norm, which is the sum of squared differences between the old centers and the new centroids.
5. Convergence check: If the Frobenius norm is smaller than a certain value (10^{-4}), then stop. Otherwise, go to Step 2 until a predefined number of iterations is reached (100).

In practice, we can draw a data point from the probability distribution in Step 1c by using a uniform random number generator and the prefix sums $\gamma_i = \sum_{\ell=1}^{i} D(x_\ell)^2$. We simply draw a value $q \in [0, \gamma_n)$ uniformly at random, and then find the smallest index i with $\gamma_i > q$. We return x_i as the drawn data point.
After initialization, this algorithm corresponds to Lloyd's method, which is depicted in Figure 6b. For completeness, we also provide implementations for two additional initialization methods: random center initialization and initialization with given centers.

4.3 Agglomerative Clustering

In this section, we discuss hierarchical agglomerative clustering, an algorithm to produce a tree that represents the distance relationship between the points. The algorithm proceeds in a bottom-up manner: It starts with all data points as leafs, and then creates new nodes by merging the two closest clusters in each iteration, until there is only one single cluster. The most common

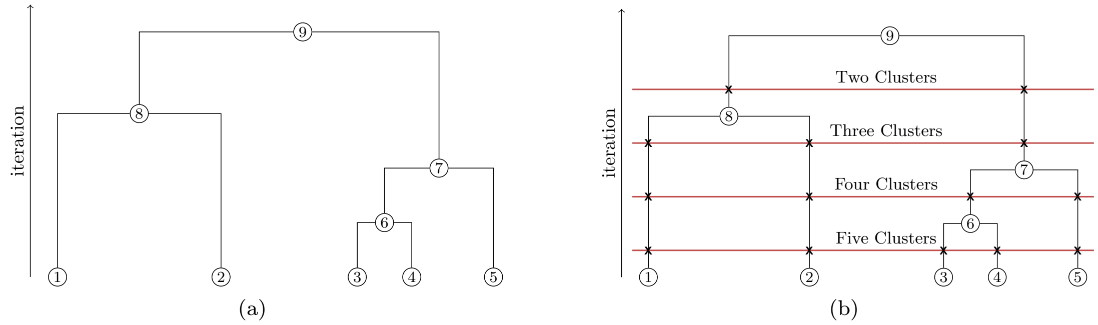


FIGURE 7: [Best viewed in color] A dendrogram for $n = 5$ and the possible cuts that can be performed to retrieve clusterings of size $1, \ldots, n$.

data structure used for this purpose is a *dendrogram*.

Definition 4.3 *Dendrogram*
A dendrogram is a binary tree where each node represents a cluster, i.e. the leafs underneath a node are exactly the points that belong to the corresponding cluster.

Note that we use a simplified definition of dendrogram in which we omit the distance labels on edges. Let us consider a binary dendrogram produced by hierarchically clustering n data points. If we draw the dendrogram in an n-level layout such that level n contains all leaves, and all other levels contain exactly one node (see Figure 7a), then we can find a clustering for any $k \leq n$ by cutting the tree horizontally, as shown in Figure 7b.

In practical applications, we oftentimes know a minimum number k_{min} of clusters that we might be interested in. In such cases, the topmost $k_{\text{min}} - 1$ levels of the dendrogram are irrelevant, since we would only horizontally cut the dendrogram below the $(k_{\text{min}} - 1)$-th level. Therefore, in this work we generalize dendrograms to forests of (not necessarily binary) trees. The trees are not necessarily binary because for some other clustering methods we allow merges of multiple clusters in each iteration, which yields non-binary trees.

So far, we only stated that in each step of the hierarchical agglomerative clustering we merge the two *closest* clusters, without specifying the distance measure we use to determine them. In fact, the distances between the clusters can be computed with a multitude of linkages and metrics. The metric is used to compute the initial distances between the actual data points. The linkage is a function that computes the distances between a new cluster (i.e. a node created by a merge) and each of the other clusters and data points. We believe that the most adequate linkage for this project is the `Ward` linkage, also known as Ward's method, which corresponds to the k-means objective function and aims at minimizing the within-cluster variance. Our choice is based on the fact that k-means is a very common choice for clustering problems, and we also use it for our other clustering methods, which means that choosing the corresponding linkage makes the clustering methods comparable. However, we also conduct an empirical evaluation of the linkage methods in Section 5.4, and according to our results `Ward` is actually the best linkage for our use case.

We present the complete process in more detail in Algorithm 2 and Figure 8 using Ward's method and the traditional squared Euclidean distance metric.

There are two different approaches of providing the input for hierarchical clustering [4]: the stored matrix approach and the stored data approach. The stored matrix approach requires the upper triangle of a symmetric matrix of pairwise distances between the points, while the stored data approach takes the data points as input.

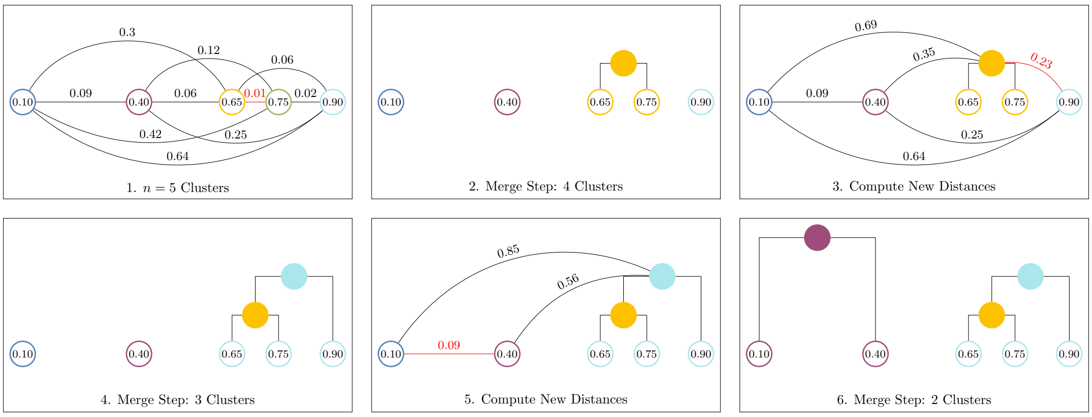
Algorithm 2 Agglomerative Clustering with Ward's Method

Input: n data points of dimension d, a number of clusters k.
Output: A dendrogram represented in an $(n-k) \times 2$ matrix, where each row denotes a merge and the columns denote the node indices that were joined at each step (example in Figure 8).

1. Create one cluster per data point, each numbered between 1 and n.
2. Compute the pairwise distances between the points with the squared Euclidean distance.
3. For each $i \in \{1, \ldots, n-k\}$:
    (a) Find the two closest clusters, I and J.
    (b) Merge I and J. The indices of the merging clusters I and J are stored in row i of the dendrogram matrix. The new node created by merging the clusters has index $i+n$.
    (c) Compute the distance between the newly merged cluster $I \cup J$ and every other cluster or data point K. For `Ward`, this is:

$$d(I \cup J, K) = \sqrt{\frac{(n_I + n_K)d(I,K) + (n_J + n_K)d(J,K) - n_K d(I,J)}{n_I + n_J + n_K}},$$

    where n_I, n_J, n_K are the number of elements in the respective clusters and $d(I,K), d(J,K), d(I,J)$ are the current distances.



1. $n = 5$ Clusters

2. Merge Step: 4 Clusters

3. Compute New Distances

4. Merge Step: 3 Clusters

5. Compute New Distances

6. Merge Step: 2 Clusters

```
3 4 // The points 0.65 (3) and 0.75 (4) are merged, creating node 6
5 6 // The point 0.9 (5) and 6 are merged, creating node 7
1 2 // The points 0.1 (1) and 0.4 (2) are merged, creating node 8
```

FIGURE 8: [Best viewed in color] Agglomerative procedure to obtain $k = 2$ clusters using Ward's method. At the beginning we compute the distances using the squared Euclidean metric. Then, we merge the two closest clusters and compute the distances to the new node. We repeat the process until we obtain $k = 2$ clusters. Below, the dendrogram obtained by the given example represented by an $(n-k) \times 2$ matrix. Each row represent a merge, and the two elements are the cluster indices of the merged elements.

Müllner [34] implemented specific methods for each approach, evaluated the algorithms with respect to different linkages, and built the package fastcluster[12], which applies the best algorithm according to the data approach and the linkage. He also improved the search step (Step 3a) by using a heap, which makes the repeated searches faster.

The asymptotic running time of this implementation with `Ward` linkage is $\Theta(n^2)$, while the space complexity depends on the input approach. The stored matrix approach requires $\Theta(n^2)$ of memory, while the stored data approach requires $\Theta(n \cdot d)$, where d is the number of dimensions. This is achieved by using a dynamic nearest neighbor approach for the search of the smallest distance that also handles the addition and removal of clusters to and from the dendrogram. However, the stored data approach is only provided for the linkage methods `single`, `centroid`, `median`, and `Ward`, and it only accepts the Euclidean distance as metric for the latter three.

4.4 Nesting Clustering

In this section, we describe a hierarchical clustering method that, similarly to agglomerative clustering, uses a bottom-up approach, but merges more than two clusters in each iteration. It was first introduced by Lin et al. [26].

Their paper provides a generalized approximation strategy for incremental algorithms, which can also be applied to hierarchical clustering algorithms. To better explain the nature of the problem, the authors propose the example of a company that has a set of n locations where they could build new facilities. Initially, the company wants to open $k \leq n$ facilities, with the plan to open additional facilities later. A good choice for the initial k facilities should not only be profitable immediately, but it should also remain profitable even when adding further facilities in the future. The output of an incremental algorithm for this problem is an ordering of the n facilities according to their suitability. Opening the first k facilities of the ordering *for any* k is then close to the optimal choice of k facilities.

In hierarchical clustering we have a similar situation. Initially, when $k = n$, there are n clusters and the k-clustering corresponds to the data points. In each iteration a selected pair of clusters is merged, obtaining k-clusterings of smaller k, until we reach one single cluster. An important property of incremental problems is that for any k, k' with $k < k'$ the solution obtained with k is *ordered* with respect to the k' solution, where the concept of ordering is specific to the problem. In the hierarchical clustering case a merge occurs in each iteration, and thus the k'-clustering is always a *refinement* of the k-clustering. Given an objective function for the k-clustering algorithm, we want to ensure that its cost for any k is close to the cost of the optimal k-clustering. Thus, we introduce the concept of *α-competitiveness*. An algorithm is said to be α-competitive if the cost of the k-clustering is no more than α times the cost of the optimal k-clustering for all values of k.

We first give a brief description of AltIncApprox (the algorithm by Lin et al., see [26, Algorithm 2]) applied to hierarchical k-means. First, we compute a clustering with an α-approximation algorithm for each $k \in \{1, \dots, n\}$, where n is the number of data points. For each clustering we compute its cost, and afterwards assign the different clusterings to intervals according to their costs (where the interval sizes grow exponentially with higher associated costs). For each interval, we choose the clustering with the smallest k. These steps correspond to AltIncApprox, Steps 1 to 4 [26] and produce a list of interesting choices of k. We describe the details of our practical implementation for this part of the algorithm in Algorithm 3 and with an example in Figure 9.

For the special case of k-means1d some steps of Algorithm 3 are superfluous. Since a table with all the cluster costs is created (Section 4.2.1), we run a clustering for the largest possible size (maximum number of clusters) and then retrieve the cluster costs from the table instead of performing Step 1. Additionally, Step 2 is redundant because we have an optimal clustering for each cluster size, and the cluster costs increase as the number of clusters decreases.

[12] http://danifold.net/fastcluster.html

Algorithm 3 Finding Interesting Cluster Amounts for Hierarchical Clustering

Input: n data points of dimension d, a maximum (e.g. $k_{\text{max}} = n$) and a minimum number of interesting clusters (e.g. $k_{\text{min}} = 1$).
Output: A list of interesting cluster amounts.

```
1. For each number of clusters $k \in \{k_{\text{max}}, \dots, k_{\text{min}}\}$:
    (a) Compute a clustering with $k$-means (using $k$-means++ or $k$-means1d, depending on $d$).
    (b) Compute the clustering cost according to the $k$-means objective function (Equation 1).
2. Sort the clustering costs in non-descending order.
3. According to their cost, group the different choices of $k$ into intervals $[0, 2^0), [2^0, 2^1)$ $[2^1, 2^2), \dots, [2^{t-1}, 2^t)$, where $t$ is chosen such that the highest present clustering cost $z$ satisfies $z \in [2^{t-1}, 2^t)$.
4. For each of these intervals, choose the smallest $k$ whose cost lies in the interval.
```

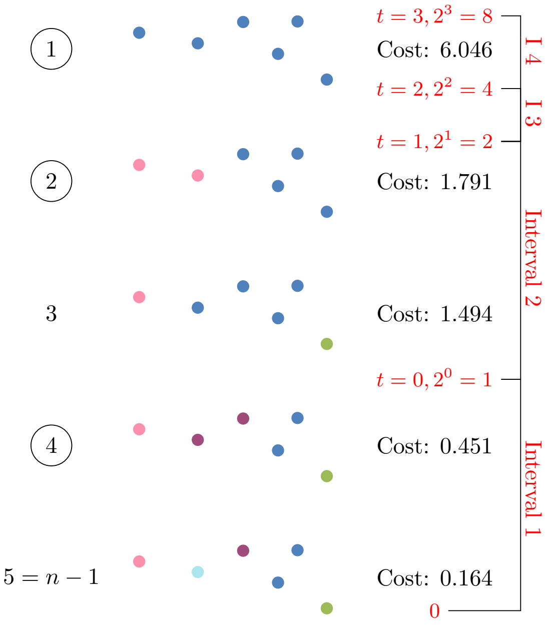


FIGURE 9: Algorithm to find interesting cluster amounts for six two-dimensional points. Let the number of maximum clusters be six and the minimum number be one. Starting from five, for each size we compute a clustering and the cluster cost (data points of equal color belong to the same cluster). Then, we divide the costs into intervals with cost in range $[0, 2^0), [2^0, 2^1), [2^1, 2^2), [2^2, 2^3)$. For each interval we pick the smallest cluster amount, which gives us the following numbers: $[4, 2, 1]$.

Afterwards, the list of cluster amounts is used as input for the nesting routine, which ensures compatibility among the chosen clusterings. We describe our nesting routine in Algorithm 4, which corresponds to AltIncApprox, Steps 5 to 6 [26]. The routine is also presented with an example in Figure 10, where we show that the result of this algorithm is also a dendrogram.

Algorithm 4 Nesting Routine for Hierarchical Clustering

Input: n data points of dimension d, a list of cluster amounts $[k_1, \dots, k_m]$.
Output: A dendrogram represented as a list of length equal to $k_1 + k_2 + \dots + k_m$, which contains variable length lists that represent the assignments (example in Figure 10).

1. Create a clustering S_1 with n clusters, each with the respective data point as center.
2. From the list of cluster amounts to compute, take the largest not yet computed number k_i and compute clustering S_2 (using k-means++ or k-means1d, depending on d).
3. Make the two clusterings compatible: Assign each center of the bigger sized clustering (S_1) to the closest center of the smaller sized clustering (S_2), which results in a clustering S_{new} of size $k \leq k_i$. The data points of this clustering are the centers of S_1, while the centers of S_{new} are (possibly a subset of) the centers of S_2.
4. Recompute the centers of S_{new} as the average of the centers assigned to each cluster.
5. If we have processed all possible k_i, then stop. Otherwise, let $S_1 \leftarrow S_{\text{new}}$ and continue with Step 2.

Lin et al. [26] show that if the problem satisfies the (γ, δ')-*nesting property* then the AltIncApprox algorithm computes a solution with a clearly defined competitive ratio.

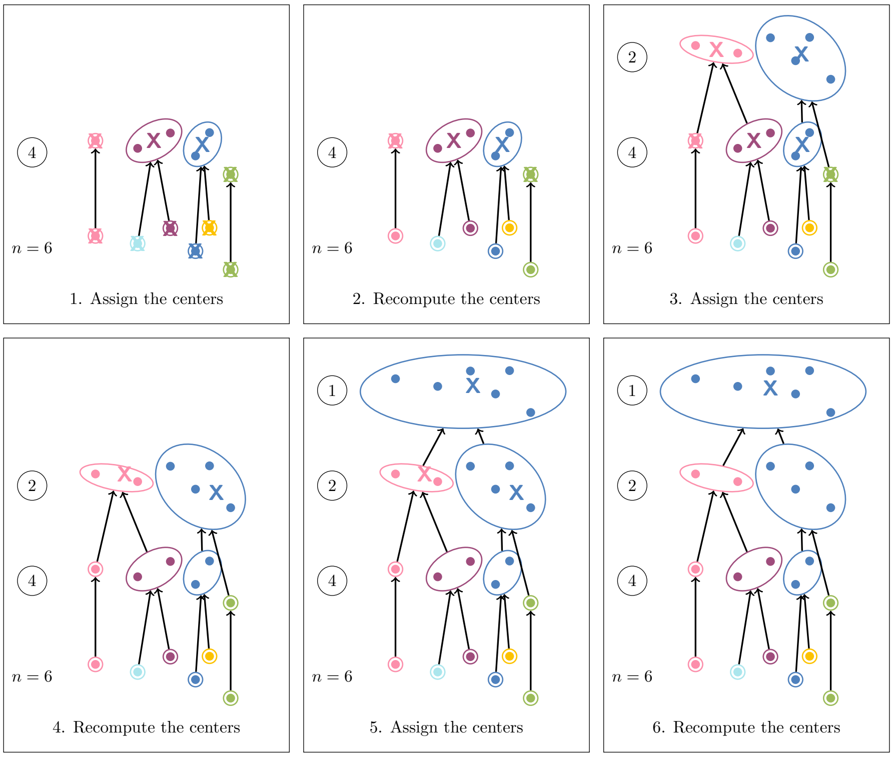
Definition 4.4 *(γ, δ')-nesting (Manuscript from Anna Arutyunova and Melanie Schmidt)*
Let P be a set of points, let S_1 and S_2 be two clusterings and let a_1 and a_2 be the functions that map the points to the respective cluster centers of S_1 and S_2.
A k-clustering problem satisfies the so called (γ, δ')-*nesting property* for reals $\gamma \geq 1, \delta' > 0$ if for any two solutions $Z_1 = (S_1, a_1)$ and $Z_2 = (S_2, a_2)$ with $|S_1| > |S_2|$, a solution $Z = (S, a)$ can be computed such that

- Z_1 and Z_2 are hierarchically compatible, i.e., for all $c \in S_2$ there exists a $c' \in S$ such that for all $x \in P$ with $a_2(x) = c$ it holds that $a(x) = c'$,
- $\text{cost}(P, Z) \leq \gamma \cdot \text{cost}(P, Z_1) + \delta' \cdot \text{cost}(P, Z_2)$, and
- $|S| \leq |S_2|$

In the manuscript by Anna Arutyunova and Melanie Schmidt it is also shown that the k-means cost function satisfies the $(3, 2)$-nesting property. In their analysis they choose the mapping $a(x)$ to be the one defined in Step 3 of Algorithm 4. In order to show the competitive ratio of hierarchical k-means we need the following theorem:

Theorem 4.1 *Competitive ratio for (γ, δ')-nesting (Theorem 2.5 [26])*
If (γ, δ')-nesting holds for reals $\gamma \geq 1, \delta' > 0$, and an α-approximation algorithm exists for the problem, then AltIncApprox $(\gamma, \delta', \alpha)$ computes an incremental solution with competitive ratio $4\gamma\delta'\alpha$.

Thus, using k-means1d in AltIncApprox results in a hierarchical clustering algorithm with a competitive ratio of 24, whereas using k-means++ results in a competitive ratio of $24 \cdot \mathcal{O}(\log n)$. Note that our implementation of Algorithm 3 and Algorithm 4 does not strictly correspond to the original AltIncApprox algorithm. Our modifications are that we allow arbitrary parameters $k_{\min}$ and $k_{\max}$ for Algorithm 3 (instead of fixed values $k_{\min} = 1$ and $k_{\max} = n$), and that we add Step 4 to Algorithm 4. We believe that these modifications produce better results in practice.



1. Assign the centers

2. Recompute the centers

3. Assign the centers

4. Recompute the centers

5. Assign the centers

6. Recompute the centers

```
// Depth 3: Matching from 6 to 4 Cluster
3 4 // Green (1) is matched with green (4) of the level below
3 5 // Pink (2) is matched with pink (5)
3 2 6 // Purple (3) is matched with purple (2) and light blue (6)
3 1 3 // Blue (4) is matched with blue (1) and yellow (3)
// Depth 2: Matching from 4 to 2 Cluster
2 1 4 // Blue (1) is matched with green (1) and blue (4)
2 2 3 // Pink (2) is matched with pink (2) and purple (3)
// Depth 1: Matching from 2 to 1 Cluster
1 1 2 // Blue (1) is matched with blue (1) and pink (2)
```

FIGURE 10: [Best viewed in color] Nesting routine for six points and the given cluster list: $[4, 2, 1]$. We create a six-center clustering by assigning each data point to a center and create another clustering of size four. We make them compatible by assigning each center of the larger clustering to a center of the smaller clustering, and recompute the centers as the average of centers for each cluster, which will be the input for the next step. We produce a clustering for the next largest size and proceed until all the clusterings in the list have been made compatible. Below, we provide the nesting dendrogram of length $4 + 2 + 1 = 7$ for the given example. Each row represents one cluster of one level. The first element of each row indicates the depth, while the other elements of the row represent the clusters of the level below that are being merged together.

Finally, we discuss the space consumption of the output dendrogram and the asymptotic worst-case execution time for this algorithm.
Let $[k_1, \dots, k_m]$ be the sorted clustering list, where k_1 is the largest number. The total size of the dendrogram is computed by considering the sum of the list elements (because of the depth indicator) plus the number of cluster indices for each level below. Thus, the total size of the dendrogram is

$$\sum_{i=1}^{m} k_i + n + \sum_{i=1}^{m-1} k_i = k_m + n + 2\sum_{i=1}^{m-1} k_i.$$

The time spent computing the nesting routine for one-dimensional data is $\mathcal{O}(n \log n)$ for the sorting and $\mathcal{O}(k_1 \cdot n)$ for the initial run of k-means1d, since we only compute it for the largest k_i. Then, for each number of clusters k_i: We retrieve the cluster centers by backtracking ($\mathcal{O}(n)$), we match the clusters, which takes $\mathcal{O}(k_{i-1} \cdot \log k_i)$ (where $k_{i-1} = n$ in case $i = 1$) and recompute the clusters $\mathcal{O}(k_i)$. Note that the $\mathcal{O}(n \cdot \log k_1)$ time needed for the first matching step is dominated by the initial sorting time. In practice, these steps take a relative small time with respect to the initial clustering.

$$\begin{aligned}\mathcal{O}(\text{Nesting1d}) &= \mathcal{O}(n \log n) + \mathcal{O}(k_1 \cdot n) + \mathcal{O}(m \cdot n) + \mathcal{O}\left(\sum\nolimits_{i=2}^{m} k_{i-1} \cdot \log k_i\right)\\ &= \mathcal{O}(n \log n) + \mathcal{O}(k_1 \cdot n) + \mathcal{O}(m \cdot n) + \mathcal{O}(m \cdot k_1 \cdot \log k_2)\end{aligned}$$

In the practical case where $m = \mathcal{O}(k_1)$ and $k_1 = \mathcal{O}(\text{polylog}(n))$ (k_1 is polynomial in $\log n$), the term becomes

$$\mathcal{O}(\text{Nesting1d}) = \mathcal{O}(n \log n) + \mathcal{O}(k_1 \cdot n)= \mathcal{O}(n \log n + n \cdot k_1),$$

which is the asymptotic complexity of k-means1d.

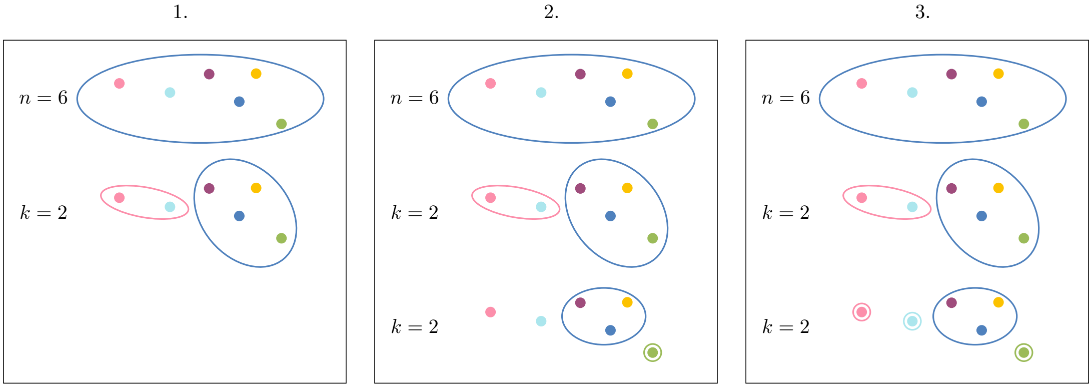
4.5 Leveled Clustering

In this section, we present another class of hierarchical clustering algorithms, where we can choose the number of levels and the number of clusters for each level. We call this type of clustering *leveled clustering*.
The previously described hierarchical clustering methods produce dendrograms with an arbitrary number of levels (or n levels). The aim of our clustering methods is to distinguish between different tissues in the brain. Additionally, we also want to produce different shadings inside each tissue. However, it is not necessary to have many levels to have some color distinction inside the main tissue clusters. Thus, given cluster amounts $[k_1, \dots, k_m]$ sorted in non-ascending order, we produce a (bottom-up) dendrogram with m levels, where the level at depth $m - i + 1$ has k_i clusters. The bottom-most level of the dendrogram contains the data points, which are then progressively grouped into fewer but larger clusters as we proceed to higher levels.

The dendrogram produced by agglomerative clustering has one level for each value q with $1 \leq q \leq n$. To produce a leveled clustering we cut the dendrogram above levels $k_1, \dots, k_m$, which creates the desired number of levels.

Conveniently, the nesting algorithm can already produce a dendrogram from a given list of cluster amounts. In fact, we can take the second part of the algorithm (Algorithm 4, Figure 10) and use our list of choice as input.

Additionally, we discuss a leveled clustering method that is based on k-means and uses a top-down strategy. Top-down hierarchical clustering algorithms are referred to as *divisive* clustering algorithms. An example is *bisecting k-means* [38]. This divisive clustering method produces a binary dendrogram similar to the one created by agglomerative clustering, but in a top-down manner. We first apply k-means with $k = 2$ on the full data to create two clusters, and then



```
// Depth 1: First part of the first division creates cluster 1
1 1 2 3 4 // Cluster with blue (1), purple (2), yellow (3) and green (4)
// Depth 2: Divide the cluster above into two parts
2 1 2 3 // Blue(1), purple (2) and yellow (3) are one cluster
2 4 // Green (4) is one cluster
// Depth 1: Second part of the first division creates cluster 2
1 5 6 // Cluster with pink (5) and light blue (6)
// Depth 2: Divide the cluster above into two parts
2 5 // Pink (5) is one cluster
2 6 // Light blue (6) is one cluster
```

FIGURE 11: [Best viewed in color] Leveled k-means routine for six points and recursive cluster amounts $[2, 2]$. At the beginning all points belong to the same cluster, then a clustering of the first size is created. For each portion of the data, we build a new clustering of the next size. Below, the k-means dendrogram of length $2 + 2 \cdot 2 = 6$ for the given example. Since the algorithm is recursive, the dendrogram levels are also printed in a recursive manner. The first element of each row identifies the depth, while the remaining elements of the row are the data point indices belonging to that cluster.

recursively refine each clusters by running k-means (again with $k = 2$) until each cluster consists of a single data point. In our version of this algorithm we may choose a parameter $k > 2$ for each refinement step, as outlined in Algorithm 5 and visualized in Figure 11. The input of this algorithm is also a list of cluster amounts $[r_1, \dots, r_m]$, but differently from the bottom-up lists used for agglomerative and nesting clustering, it does not need to be sorted as the division in clusters is applied recursively to each amount. In this case, the number of clusters at depth i is expressed as $\prod_{j=1}^{i} r_j$. As an example, let us consider the list $[2, 4, 2]$ of recursive clustering amounts. The data points are first divided into two large clusters, then each cluster is divided into four medium-sized clusters, and finally each medium-sized cluster is divided into two small clusters. In this case, at depth one we have 2 clusters, at depth two we have $2 \cdot 4 = 8$ clusters, and at depth three we have $2 \cdot 4 \cdot 2 = 16$ clusters, resulting in the bottom-up list $[16, 8, 2]$ of clustering amounts.

Lastly, we compute the space needed to store the dendrogram, and the worst-case asymptotic run time for this recursive k-means approach.
Let $[r_1, \dots, r_m]$ be the recursive cluster amounts. It is easy to see that in each level we have to redistribute all data points, i.e. the dendrogram contains each data point exactly once per level, resulting in $n \cdot m$ stored values. Additionally, at depth i, we have to store $\prod_{j=1}^{i} r_j$ list headers (i.e. the first entry of each row, which contains the current depth).

Algorithm 5 Leveled k-means

```
Input: $n$ data points of dimension $d$, a list of recursive cluster amounts $[r_1, \dots, r_m]$.
Output: A dendrogram represented as a list of length equal to the number of recursive calls, which contains variable length lists that represent the assignments (example in Figure 11).
  1. Compute a clustering with $r_1$ centers.
  2. For each cluster $l \in \{1, \dots, r_1\}$:
      (a) Append a row to the dendrogram that contains the current depth of the recursion, as well as exactly the identifiers of the data points that belong to cluster $l$.
      (b) If $m > 1$, then recursively run leveled $k$-means on the data points belonging to cluster $l$, with the cluster amounts $[r_2, \dots, r_m]$.
```

Thus, the total size of the dendrogram is:

$$m \cdot n + \sum_{i=1}^{m} \prod_{j=1}^{i} r_j$$

The worst-case asymptotic complexity of this algorithm for one-dimensional data depends on sorting ($\mathcal{O}\left(n \log n\right)$) and then applying the k-means1d algorithm recursively on each subportion ($\mathcal{O}\left(k \cdot n\right)$). Let us analyze the time spent for the first two levels:

$$\begin{aligned}\mathcal{O}\left(\text{Leveled } k\text{-means1d}\right) &= \mathcal{O}\left(n \log n\right) + \mathcal{O}\left(r_1 \cdot n\right) + r_1 \cdot \mathcal{O}\left(r_2 \cdot \frac{n}{r_1}\right) \\ &= \mathcal{O}\left(n \log n\right) + \mathcal{O}\left(r_1 \cdot n\right) + \mathcal{O}\left(r_2 \cdot n\right).\end{aligned}$$

Even though the resulting clusters may have different sizes, the total number of elements for which the clustering has to be computed on each level is always n. This results in:

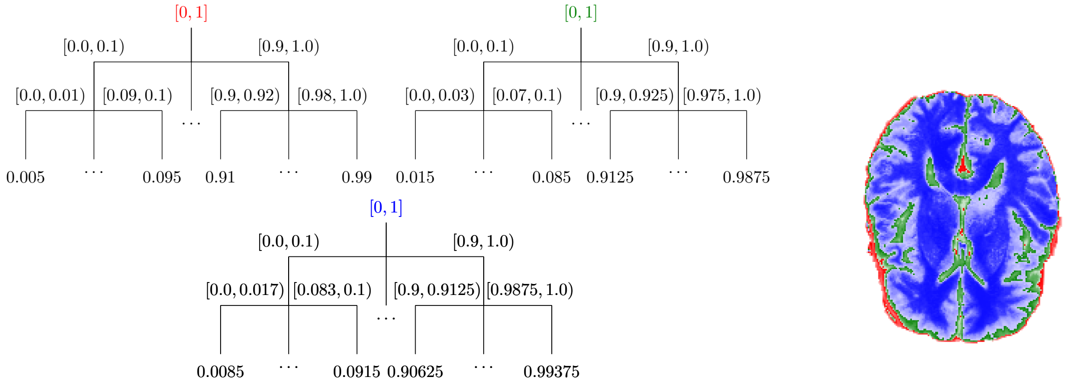
$$= \mathcal{O}\left(n \log n\right) + \sum_{i=1}^{m} \mathcal{O}\left(r_i \cdot n\right).$$

While Leveled k-means and Leveled Nesting have a similar time complexity, Leveled k-means is much faster in practice because it runs on smaller portions of the data, which significantly improves the cache behavior of the algorithm.

4.6 Dendrogram Output and Shaded Labels

As we discussed in Section 1 and at the beginning of this chapter, our aim is to cluster MRI scans to identify the different tissues of the brain and eventually capture a mapping between the different MRI sequences. We carry out this task using hierarchical clustering algorithms, which are able to identify the main tissues but also to distinguish different patches of color inside the main clusters. The hierarchical clustering algorithms presented here have a dendrogram as output. But how can we use a dendrogram to visualize the hierarchical structure of the clustering? One way to achieve this is to build labels that represent the shading of the tissues, as shown in Figure 12b. Let us assume that the dendrogram is a forest, where each tree represents one main tissue cluster. We assign to each of these main clusters the interval $[0, 1]$, which represents the intensity spectrum of one color. For example, for the color red zero means white, while one means bright red. Then, for each tree we recursively divide the interval and assign a portion to the children. Finally, when we reach the leaves, we take the middle of the interval. A representation of the process is shown in Figure 12a.

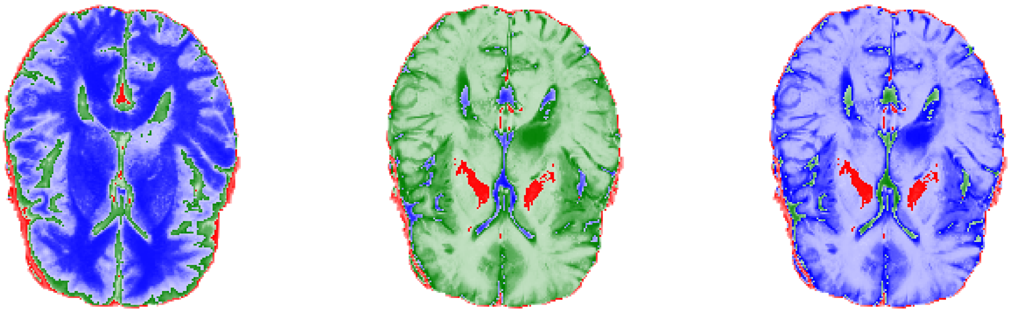
However, the dendrograms obtained with our hierarchical clustering methods have different structures (see Figure 8, Figure 10 and Figure 11). Thus, we select the nesting dendrogram to be our standard and convert the output of the other algorithms accordingly.



(a) Creation of hierarchical labels by recursively dividing the interval [0, 1].

(b) Hierarchical Clustering of T1 Weighted

Figure 12: [Best viewed in color] Procedure to create shaded labels and example result for nesting clustering of T1 Weighted of patient 2 (BraTS 19).



(a) T1 Weighted

(b) T2 Weighted

(c) Adjusted T2 Weighted

Figure 13: [Best viewed in color] Comparison of the clustering obtained with nesting clustering for T1 Weighted and T2 Weighted of patient 2 (BraTS 19).

4.7 Cluster Labels Matching

Finally, in this section we describe the strategy that we use to match clusterings computed from different MRI scans. Let us consider two scans (T1 Weighted and T2 Weighted) and a clustering method (nesting) as in Figure 13. It is easy to see that the blue cluster of Figure 13a corresponds to the green cluster of Figure 13b, and thus we wish to convert the clustering of T2 Weighted such that it is comparable to T1 Weighted (as in Figure 13c).
This problem can be transformed into a very common combinatorial optimization problem: the *maximum bipartite weighted matching.*

Definition 4.5 *Bipartite Graph*
A undirected bipartite graph is a tuple $G = (A, B, E)$ with *vertices* $A \cup B$ and *edges* $E \subseteq A \times B$. Let $w : E \to \mathbb{R}$ be a function that assigns a weight to each edge, then we call $G = (A, B, E, w)$ a weighted undirected bipartite graph.

Definition 4.6 *Weighted Matching*
Given a weighted undirected graph with edges E and weight function w, a matching $M \subseteq E$ is a set of edges such that no two edges in M share a vertex. The weight of M is the sum of its edge weights, i.e. $\text{weight}(M) = \sum_{e \in M} w(e)$. We say that M is a maximum weighted matching, if there exists no matching M' with $\text{weight}(M) < \text{weight}(M')$.

The maximum weighted matching problem involves finding a matching of maximum weight in a graph. For bipartite matching a very popular algorithm is the Hungarian algorithm [22], which has a $\mathcal{O}\left(V^3\right)$ worst-case asymptotic run time. This problem can be visualized as pairing people with jobs for which there is only one position, and thus is also called the assignment problem. This is exactly our aim; we would like to match the cluster labels from T1 Weighted to the cluster labels of T2 Weighted.
To reduce our problem to the maximum weighted matching problem we first need to compute a contingency matrix, a table that represents the frequency of labels of two clusterings. We compute it as described in Algorithm 6. Once we obtain a $k_1 \times k_2$ table, we pad it with zeros to obtain a square matrix. This matrix represents the weights of a bipartite graph.

Algorithm 6 Contingency Matrix

Input: A clustering S_1 of T1 Weighted and a clustering S_2 of T2 Weighted, both of length n.
Output: A two-dimensional matrix that represents the frequency of the labels in the clusterings.

```
1. Let $k_1$ and $k_2$ be respectively the number of clusters in the first clustering and in the second
   clustering.
2. Initialize a $k_1 \times k_2$ matrix $F$ with zeros.
3. For each $i \in \{1, \dots, n\}$:
   (a) Let $l_1$ be the cluster assignment of pixel $i$ in $S_1$.
   (b) Let $l_2$ be the cluster assignment of pixel $i$ in $S_2$.
   (c) Increase the value of $F[l_1][l_2]$ by one.
```

The Hungarian algorithm is based on the concepts of alternating and augmenting paths.

Definition 4.7 *Alternating Path and Augmenting Path*
Let $G = (V, E)$ be a graph with a set of nodes V, a set of edges E and a matching $M \subseteq E$ and let $P = v_1, \dots, v_k$ be a path from v_1 to v_k with $v_i \in V,$ for $1 \leq i \leq k$ and $(v_j, v_{j+1}) \in E,$ for $1 \leq j < k$. We call P an alternating path if it contains alternating edges in M and in $E \setminus M$. An alternating path P is an augmenting path if the first vertex v_1 and the last vertex v_k are not incident to an edge in M.

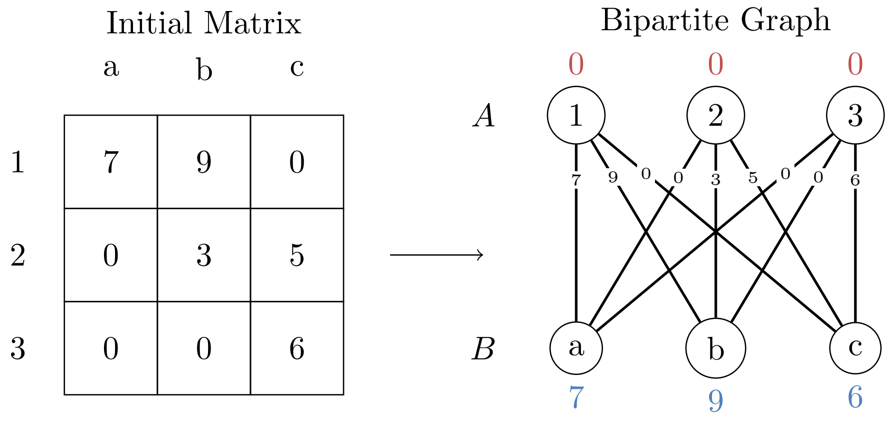
Note that if there exists an augmenting path P for a matching M, then this matching cannot be maximum, because by taking the symmetric difference $M \oplus P$ we obtain a matching of larger size.
We present the Hungarian algorithm for bipartite graphs in Algorithm 7[13] and an example computation in Figure 15. The idea of the algorithm is to start with an empty matching and search for augmenting paths that would give a better weight. In case one is found, then the matching is augmented using the symmetric difference.

[13] Adapted from lecture slides by Blum (http://tcs.cs.uni-bonn.de/doku.php?id=teaching:ws1920:vl-advalgo).

Algorithm 7 Hungarian Algorithm

Input: A square matrix representing the weights of a bipartite graph $G = (A, B, E, w)$.
Output: A list of tuples representing an assignment.

1. Start with an empty matching $M = \emptyset$.
2. Define node weights:
   (a) Let a_j be the node weights for the nodes in A and b_i the node weights for the nodes in B (Figure 14).
   (b) Feasibility rule: Let w_{ij} be the edge weight between node $i \in B$ and $j \in A$, then the sum of b_i and a_j should always be greater or equal than the edge weight ($b_i + a_j \geq w_{ij}$).
   (c) Assign node weights according to the feasibility rule. A common initialization is taking the largest weight of the incident edges for each node in B, and zero for each node in A.
3. Build a new graph, G^*, where there are only the edges for which $b_i + a_j = w_{ij}$.
4. If all nodes are matched, terminate. Otherwise, look for an augmenting path in G^* starting at a node that is not contained in the matching.
5. If a path P is found, build a new matching with the symmetric difference ($M = M \oplus P$) and go to Step 4. Otherwise, extend G^*:
   (a) For each edge that is incident to the currently explored nodes in B and that is incident to the **not** explored nodes in A compute the unbalance $\pi_{ij} = b_i + a_i - w_{ij}$.
   (b) Let δ be the minimum unbalance.
   (c) Adjust the reachable nodes of B with $b_i = b_i - \delta$, and the not reachable nodes of A with $a_j = a_j + \delta$. The other nodes are left unchanged.
   (d) Go to Step 3.



	a	b	c
1	7	9	0
2	0	3	5
3	0	0	6

FIGURE 14: [Best viewed in color] Transformation from square adjacency matrix to bipartite graph and creation of the node weights.

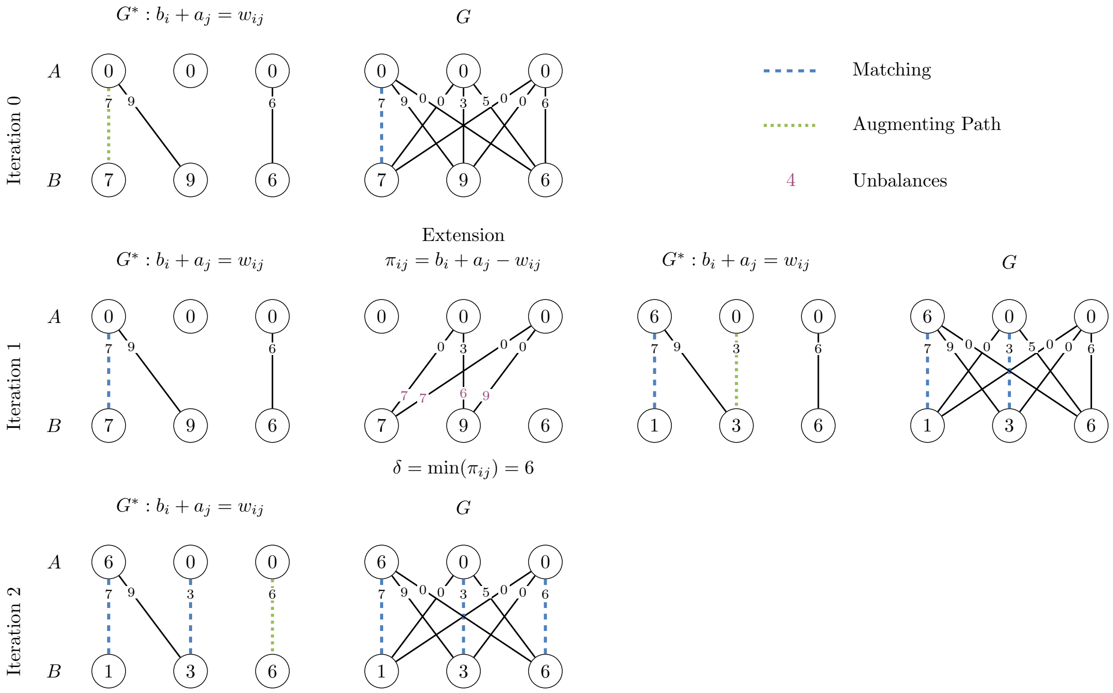

$G^* : b_i + a_j = w_{ij}$
G
Iteration 0
Iteration 1
Iteration 2
A
B
Matching
Augmenting Path
4
Unbalances
Extension
$\pi_{ij} = b_i + a_j - w_{ij}$
$\delta = \min(\pi_{ij}) = 6$

FIGURE 15: [Best viewed in color] Computation of an assignment using the Hungarian algorithm. The procedure involves selecting a new root and looking for an augmenting path. In case no path can be found, an extension step is performed to make more edges available.

5 Clustering Evaluation

In this section, we assess the performance of our clustering methods. First, we evaluate some k-means clustering methods and compare different agglomerative clustering linkage functions. Using our findings, we identify the respectively best k-means and agglomerative method, and finally assess their accuracy together with nesting. We compare each clustering with the real tissues of the brain, which have been identified manually by a medical doctor in a process called tissue segmentation. We have four segmented images (BraTS 13 Patients 1, 2, 3, 5), where each brain voxel has been classified into one of four tissues: Grey Matter, White Matter, Cerebrospinal Fluid and Other/Tumour. However, the segmentation is not perfect and there are some gaps between the tissues.

While the segmentation has been done on BraTS 13 Patients 1, 2, 3, 5, the same patients exist in BraTS 19 under different patient names. We have run our clustering algorithms on both sets, but we found some discrepancies in normalization in BraTS 19 and thus we decided to use the sequences from BraTS 13 for our clustering evaluation.

To have a fair comparison between the segmented and the clustered images, we only use the points that are not background in the segmented image as input for the clustering. Then, we run our clustering algorithm of choice. We do not take the shading and the levels described in the previous chapter into account, but we only create a plain main-tissues clustering. The algorithms Leveled k-means and Leveled Nesting are based on k-means, and thus we do not present a separate comparison for them. However, we attach an example of the results in Section A.1.

Although the number of classified tissues is four, we run the clustering methods with three clusters instead. This is because the class Other/Tumour is not really a color class and can contain different tissues and part of the tumor, which makes it hard for a clustering algorithm to detect it. However, we show a comparison of k-means1d with four and three clusters for completeness.

All evaluations but one (Section 5.4) are based on the clustering of the entire three-dimensional brain, but for simplicity only the middle transverse slice is shown in the figures below.

The custom clusterings have also been postprocessed by mapping them to the segmentation image with the method described in Section 4.7.

5.1 Evaluation Metrics

In this section, we describe the scores used for our assessment. For all of them (except Rand index) we use the implementation of scikit-learn[14]. For all measures described here, a higher value represents a better clustering.

Precision and Recall. Precision and recall [30] are common measures used in information retrieval and machine learning. They are computed for each class (in our case, tissue) according the concepts of true positives, false positives, false negatives, true negatives.
For completeness, we state their definitions using the Grey Matter class as an example.

- **True Positives**: pixels that were predicted as Grey Matter, and belong to the Grey Matter.
- **False Positives**: pixels that were predicted as Grey Matter, but belong to another class. This error is also called *false alarm*.
- **False Negatives**: pixels that were predicted as another class, but they belong to the Grey Matter. This error is also called *miss*.
- **True Negatives**: pixels that were predicted as another class, and actually belong to another class.

The precision and the recall score are computed according to the formulas below (where TP, FP, FN, and TN are the numbers of true positives, false positives, false negatives, and true negatives).

$$\texttt{precision} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FP}}, \ \texttt{recall} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}}$$

Thus, the precision tells us how many of the retrieved instances were actually relevant, and the recall informs us on which percentage of the relevant instances we managed to collect. Their values are bounded in range [0, 1].
These measures are computed for each class (in our case the classes are exactly the tissue type), but since we are in a multi-class environment, we use three different methods to combine them and create a single score:

- **Micro**: True positives, true negatives and false positives are computed as the sum of the values for each class and then the metrics are computed.
- **Macro**: The metrics are computed for each class and then they are averaged by the number of different classes.
- **Weighted**: The metrics are computed for each class and then the average weighted by the number of true instances for each class is computed.

For example, in our setting the Other/Tumour class is always smaller than the others, and if this class is not correctly detected, then the result will be a lower precision and recall score in the macro type than in the weighted type.

[14] https://scikit-learn.org/stable/modules/model_evaluation.html

Rand Index and Adjusted Rand Index. The simple Rand index takes all the pairs of labels from the two different clusterings into account and is computed as the number of pairs where the two clusterings agree over the total number of labels. We compute it as described by Manning et al. [30].
In more detail, we can build a contingency matrix that describes the relationship between the segmentation clustering (ground truth) and the current prediction we are examining.
In the contingency matrix we can observe four different situations:

- $\texttt{agree}_1$: pixels that have the same tissue in the truth and the same tissue in the predicted clustering.
- $\texttt{agree}_2$: pixels that have different tissues in the truth and have different tissues in the predicted clustering.
- $\texttt{disagree}_1$: pixels that have different tissues in the truth and have the same tissue in the predicted clustering.
- $\texttt{disagree}_2$: pixels that have the same tissue in the truth and different tissues in the predicted clustering.

The Rand index is computed as

$$\texttt{RI} = \frac{\texttt{agree}_1 + \texttt{agree}_2}{\texttt{agree}_1 + \texttt{agree}_2 + \texttt{disagree}_1 + \texttt{disagree}_2},$$

which has the same meaning as

$$\texttt{RI} = \frac{\texttt{TP} + \texttt{TN}}{\texttt{TP} + \texttt{TN} + \texttt{FP} + \texttt{FN}}.$$

However, the concepts of true/false positive and true/false negative are a bit different from the ones previously described. In fact, here we compute these values in a combinatorial way on the contingency matrix, and thus this index can account for permutations (up to one). The index has values in range $[0, 1]$.
We can also compute a new kind of precision and recall using these new values.

The Rand index can then be adjusted for chance for the case in which the two clusterings are picked at random, creating the adjusted Rand index [16].

$$\texttt{ARI} = \frac{\texttt{RI} - \mathbb{E}\left[\texttt{RI}\right]}{\max(\texttt{RI}) + \mathbb{E}\left[\texttt{RI}\right]}$$

Its value is bounded in range $[-1, 1]$, with random or independent clusterings having close to zero scores, and identical clusterings having value one.

Accuracy. This score represents the fraction of samples that have been classified correctly. Let `truth` be the manually segmented image and let `pred` be the predicted result. Then the accuracy score is defined as:

$$\texttt{ACC} = \frac{1}{\texttt{n\_samples}} \cdot \sum_{i}^{\texttt{n\_samples}} \mathbf{1}(\texttt{truth}_i = \texttt{pred}_i).$$

where **1** is the indicator function, which has value one if the labels are the same and zero otherwise. The value of the accuracy score is bounded in range [0, 1].

Adjusted Mutual Information Score. Similarly to the adjusted Rand index, the adjusted mutual information score measures the agreement of the predicted data and the truth, and it is also adjusted for chance in the same manner. Thus, this score is also bounded above by one and has value close to zero for random assignments.
The standard mutual information score quantifies the mutual dependence of the two sets and is related to the concept of entropy of random variables. We refer to the original paper [41] for a more detailed explanation.

Fowlkes-Mallows Index. The Fowles-Mallows index [13] also uses the concepts of true positives, false positives and false negatives and can in fact be seen as the geometric mean of the combinatorial precision and recall:

$$\mathtt{FM} = 2 \cdot \frac{\mathtt{TP}}{\sqrt{(\mathtt{TP} + \mathtt{FP}) * (\mathtt{TP} + \mathtt{FN})}}.$$

Its score is in range $[0, 1]$.

5.2 Implementation and Experimental Setup

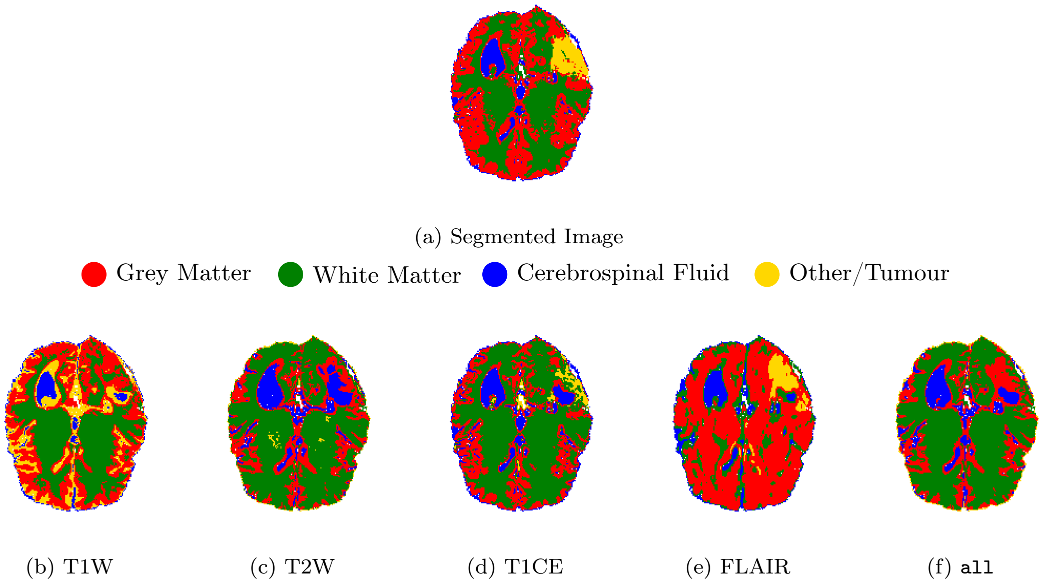
All clustering algorithms have been implemented in C++ 17 according to the specifications described in the previous chapter, and are ported to python with Boost.Python[15]. Even though we mostly use the clustering algorithms for one-dimensional data, the provided implementation supports up to four dimensions, and could easily be modified to support an arbitrary number of dimensions.
The following tests were run on the compute cluster of the University of Cologne. This high performance computing cluster has 36 nodes. Each node is equipped with two Intel Xeon CPU E5-2690 v2 CPUs[16] with 10 cores each and 128 GB of RAM. The cluster operates on the Debian GNU/Linux 10 distribution and jobs are scheduled with Oracle Grid Engine. All the experiments were run in parallel, using one core per process.

5.3 k-means Comparison

In this section, we compare the clusterings obtained with $k = 3$ and $k = 4$ for the k-means1d algorithm, and then assess the performance of three different k-means algorithms.
In the comparison between three and four clusters we also run one additional clustering method: `all`. It is a four-dimensional clustering on a matrix built by combining all four sequences. As can be seen in Figure 16, it is difficult for the clustering algorithms to detect the Other/Tumour class correctly. By comparison, in Figure 17 the results look more similar to the segmentation image. The sequences that are partially able to detect the Other/Tumour class are T1 Contrast Enhanced and T2 FLAIR. So, our motivation for the `all` approach is that it might be able to detect the Other/Tumour class because it takes into account both T1 Contrast Enhanced and T2 FLAIR. However, it is not particularly successful, especially in the case with three clusters. Thus, we will not show this clustering method for the following evaluations.
In Figure 19 we present the full evaluation metrics obtained with four clusters and three clusters. In T1 Weighted and T1 Contrast Enhanced most measures improve when switching from four to three clusters. In T2 Weighted and T2 FLAIR we have similar scores, but T2 Weighted has the majority of the scores being better for three clusters, while T2 FLAIR has the opposite. The behavior of `all` is similar to the one of T2 FLAIR and Figure 16f is a better representation of the manual segmentation, even though it does not detect the Other/Tumour class. In general precision and recall always have very close values, except for `all`. In this case the combinatorial

[15] https://www.boost.org/doc/libs/1_66_0/libs/python/doc/html/index.html

[16] https://ark.intel.com/content/www/us/en/ark/products/75279/intel-xeon-processor-e5-2690-v2-25m-cache-3-00-ghz.html



(a) Segmented Image

(b) T1W (c) T2W (d) T1CE (e) FLAIR (f) `all`

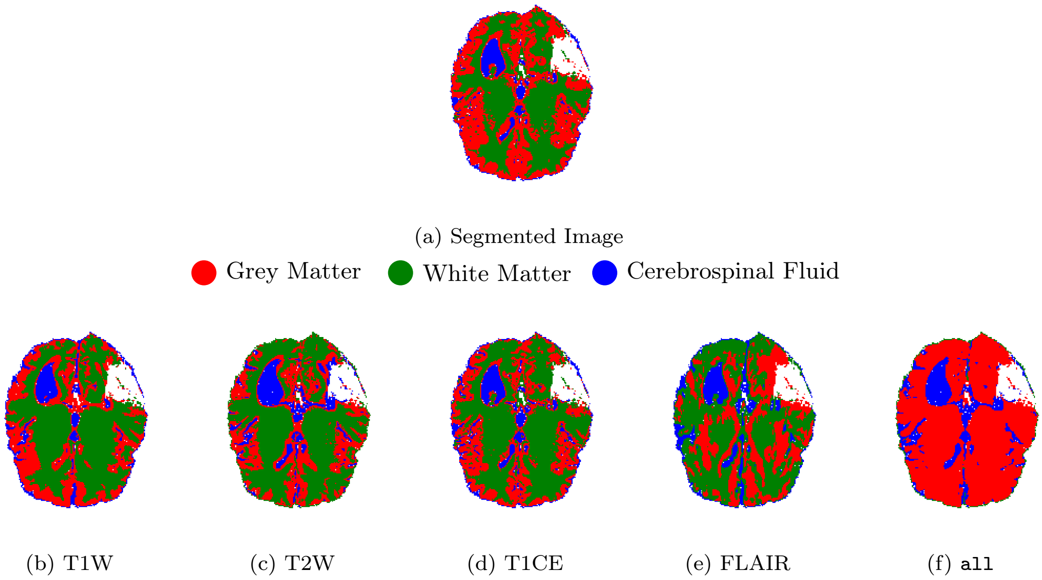
FIGURE 16: [Best viewed in color] Segmentation and k-means1d clustering with $k = 4$ for Patient 5 (BraTS 13).

recall is very high, which is due to the fact that most of the pixels are assigned to the red class (see Figure 17f), creating many false positives for this class but a low number of assignments for the other classes, which makes the overall number of false negatives small. The values for the adjusted measures (Rand score, mutual information) are always quite low, but not for the case of T1 Contrast Enhanced.

To validate our choice of choosing k-means1d as standard method for our hierarchical k-means clusterings (used in Leveled k-means, Nesting and Leveled Nesting) we present a time and effectiveness comparison with respect to the k-means1d algorithm, the k-means++ algorithm run in one dimension and the k-means++ algorithm run in four dimensions. In the latter each point has as first coordinate the color (the only variable considered in the previous clusterings), and as second, third and fourth positions in the three-dimensional grid of the image. We call this last method k-means Positional. Whenever we compute the squared Euclidean distance between two points during the algorithm, each squared distance is weighted: the squared distance of the color has a weight of 0.85, while the square distances of the indices positions have a weight of 0.05 each. We tested this method with different weights and the result is that the smaller the weights are, the better. However, even with very small weights, this method is hardly better than the other k-means method, as shown in Figure 18 and Figure 20.

The performance of k-means1d and k-means++ is very similar, but the average running time per sequence of k-means1d (≈ 1.55 seconds) is three times the running time of k-means++ (≈ 0.44 seconds). The running time for k-means Positional is almost four times (≈ 1.7 seconds) the running time of k-means++. Furthermore, it can produce clusterings that, for the most part, are too related to the spatial position of the pixels, independently of how small the weights are. In Figure 18d, only the blue cluster was correctly identified by k-means Positional (but also by k-means1d and k-means++). The other clusters are strongly misshaped in the k-means Positional clustering, clearly showing that considering the spatial coordinates negatively influences the results.

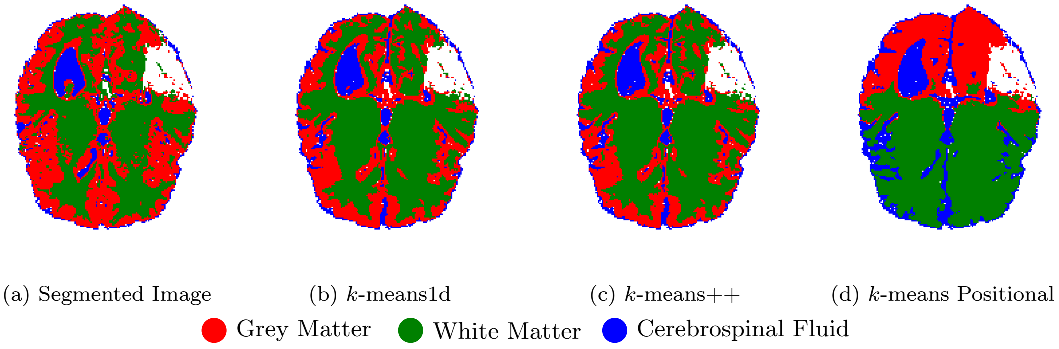
In Figure 20 we can see that the effectiveness of all methods for T1 Weighted and T2 Weighted is comparable, with slightly better values for k-means1d and k-means++. These two methods are clearly the best in T1 Contrast Enhanced. In more detail, k-means1d has better scores for T1 Weighted and T2 Weighted, while k-means++ has better scores for T1 Contrast Enhanced.



(a) Segmented Image

(b) T1W (c) T2W (d) T1CE (e) FLAIR (f) `all`

FIGURE 17: [Best viewed in color] Segmentation and k-means1d clustering with $k = 3$ for Patient 5 (BraTS 13).



(a) Segmented Image (b) k-means1d (c) k-means++ (d) k-means Positional

FIGURE 18: [Best viewed in color] Segmentation and different k-means clusterings for T1 Weighted of Patient 5 (BraTS 13).

However, the positional clustering performs better than the other two methods in the case of T2 FLAIR. Since we do not use T2 FLAIR in MRI mapping and we mostly focus on T1 Weighted and T2 Weighted, we choose k-means1d to be the algorithm used for Leveled k-means, Nesting and Leveled Nesting. Despite its slower running time, we believe that k-means1d is the better choice over k-means++, since it always produces an optimal k-means clustering. Additionally, due to the dynamic programming nature of the algorithm, whenever we compute a clustering for some value of k, it is possible to retrieve the clustering for all values $k' \leq k$. This is an advantage when running Nesting or Leveled Nesting.

5.4 Agglomerative Comparison

In this section, we compare the linkage functions provided by the fastcluster package. Although we believe that `Ward` should be our linkage of choice, we conduct an empirical evaluation on all the offered linkages.

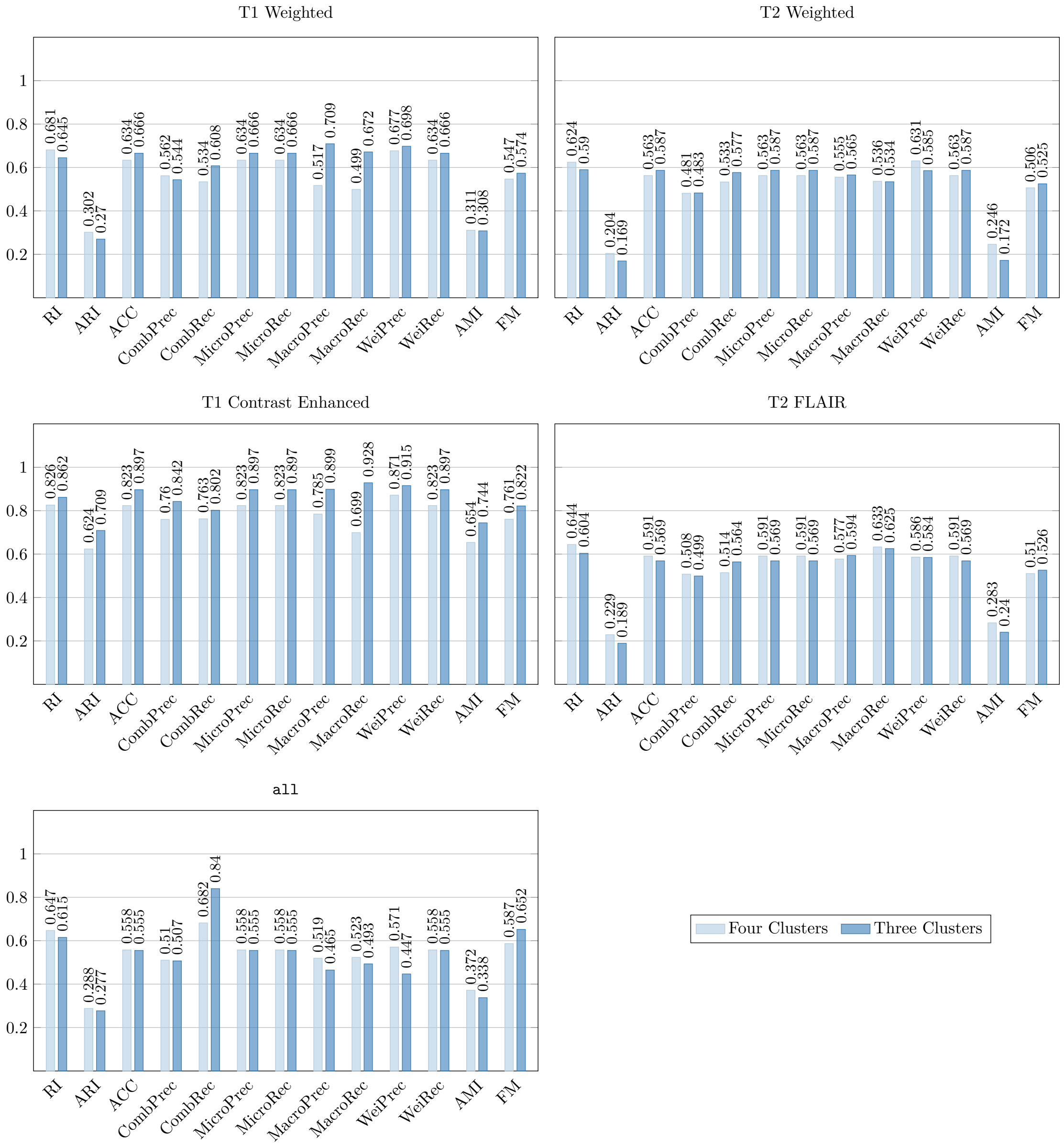


FIGURE 19: [Best viewed in color] Comparison between three clusters and four clusters for the clustering method k-means1d using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). The values are grouped by sequence and represent the average of the scores of the four segmented patients. For all measures a higher value indicates a better score.

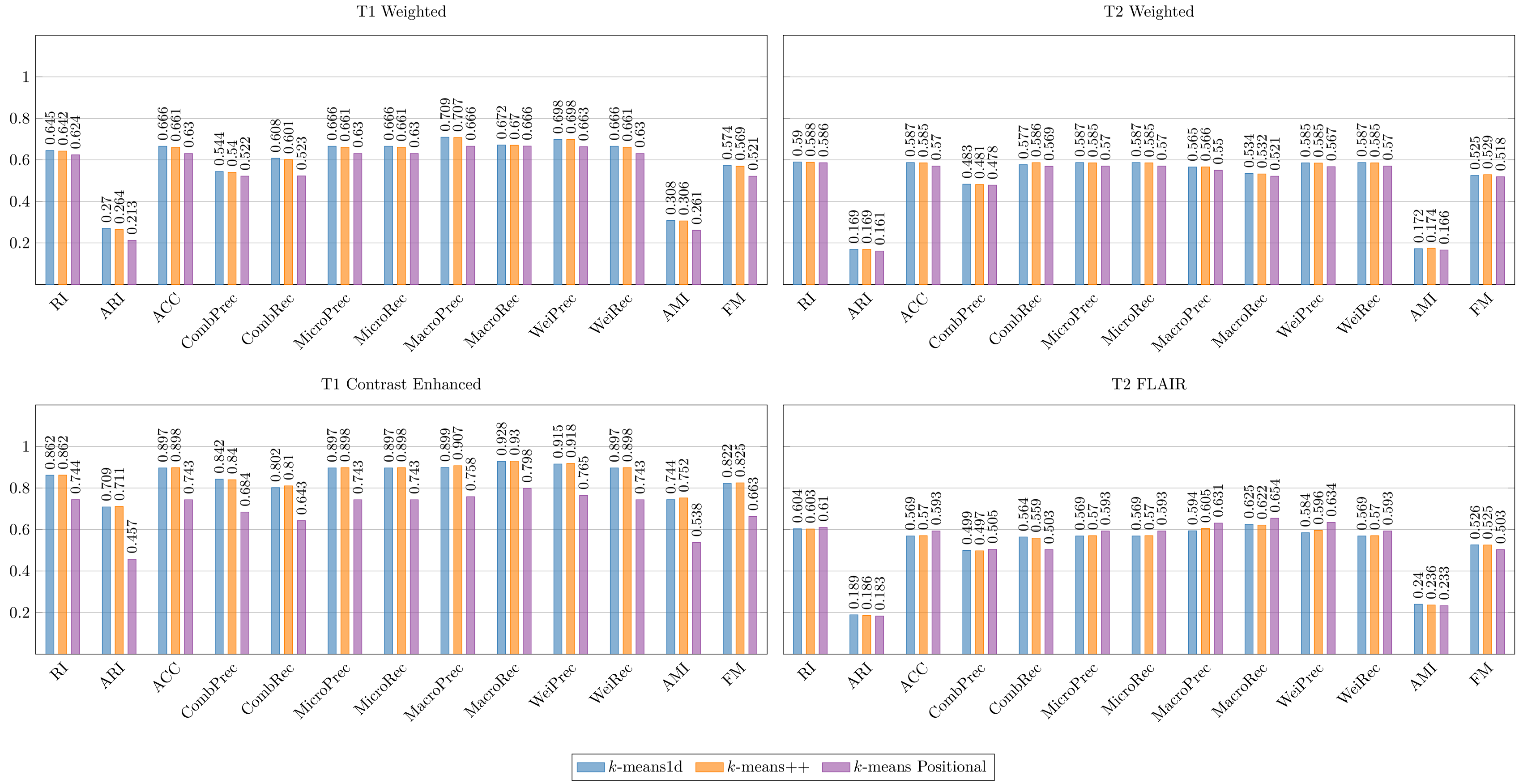


FIGURE 20: [Best viewed in color] Comparison between *k*-means1d, *k*-means++ and *k*-means Positional using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). The values are grouped by sequence and represent the average of the scores of the four segmented patients. For all measures a higher value indicates a better score.

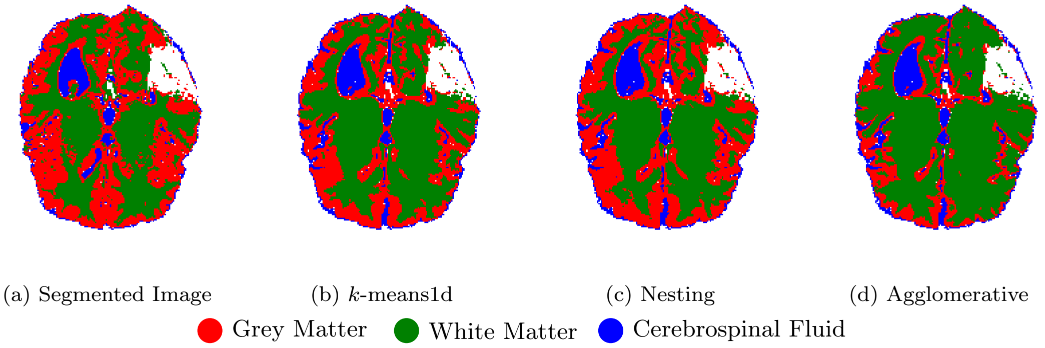

(a) Segmented Image (b) k-means1d (c) Nesting (d) Agglomerative

FIGURE 21: [Best viewed in color] Segmentation and k-means1d, Nesting and Agglomerative (Ward) clusterings for T1 Weighted of Patient 5 (BraTS 13).

Due to the size of our data, for single, centroid, median and Ward we use the more memory-efficient stored data input approach implemented by fastcluster. Additionally, in our use case this implementation is also around four times faster than the stored matrix approach. Since fastcluster requires that the metric for centroid, median and Ward is the Euclidean distance, we use the same metric for all the other methods to have a better comparison. It is very tedious to run the non-memory efficient metrics on the full brain, and thus we conduct this evaluation only on one slice of the brain (middle slice in transverse visualization) instead of running it for the entire volume. For this evaluation we leave out the combinatorial, micro and macro version of precision and recall to make the results in Figure 22 more readable. For completeness, we discuss the omitted results in Section A.2, where we also show some comparison pictures for the different linkages. In T1 Weighted and T1 Contrast Enhanced Ward performs best in most cases. In T2 Weighted and T2 FLAIR most linkages perform similarly. The linkage single does not perform well and its adjusted Rand index is in many cases close to zero. As usual, T1 Contrast Enhanced has the best scores, and the best is many times obtained by Ward.

5.5 Comparison of k-means1d, Nesting, Agglomerative

Finally, we discuss the performance of the best k-means method (k-means1d), together with Nesting (with k-means1d) and the best agglomerative linkage (Ward). In Figure 23 we present the results for these three clustering methods. We also add a complete per patient evaluation for this comparison in Section A.3. In T1 Weighted, Nesting obtains the best score in most measures while k-means1d and Agglomerative are better for T2 Weighted. As we have previously seen, T1 Contrast Enhanced produces the highest scores, and all clustering methods perform well, even though k-means1d and Nesting are better. T2 FLAIR presents similar scores to T2 Weighted, and all clustering methods have very similar scores. In Figure 21 we show a comparison of the three clustering methods on T1 Weighted. All methods are able to correctly identify the blue clusters. The green cluster created by Agglomerative is too big, while Nesting produces the most realistic segmentation.

To conclude, we achieve better results with the T1 sequences than with the T2s, which is expected because the segmentations are mostly done on the T1 images. For all sequences except T1 Contrast Enhanced we generally observe low adjusted Rand scores and adjusted mutual scores. However, these scores seem to be generally low for our use case. For our given data set, we cannot pinpoint a single clustering method and number of clusters that fits best for all patients, but from the average we can say that the methods based on k-means1d work better.

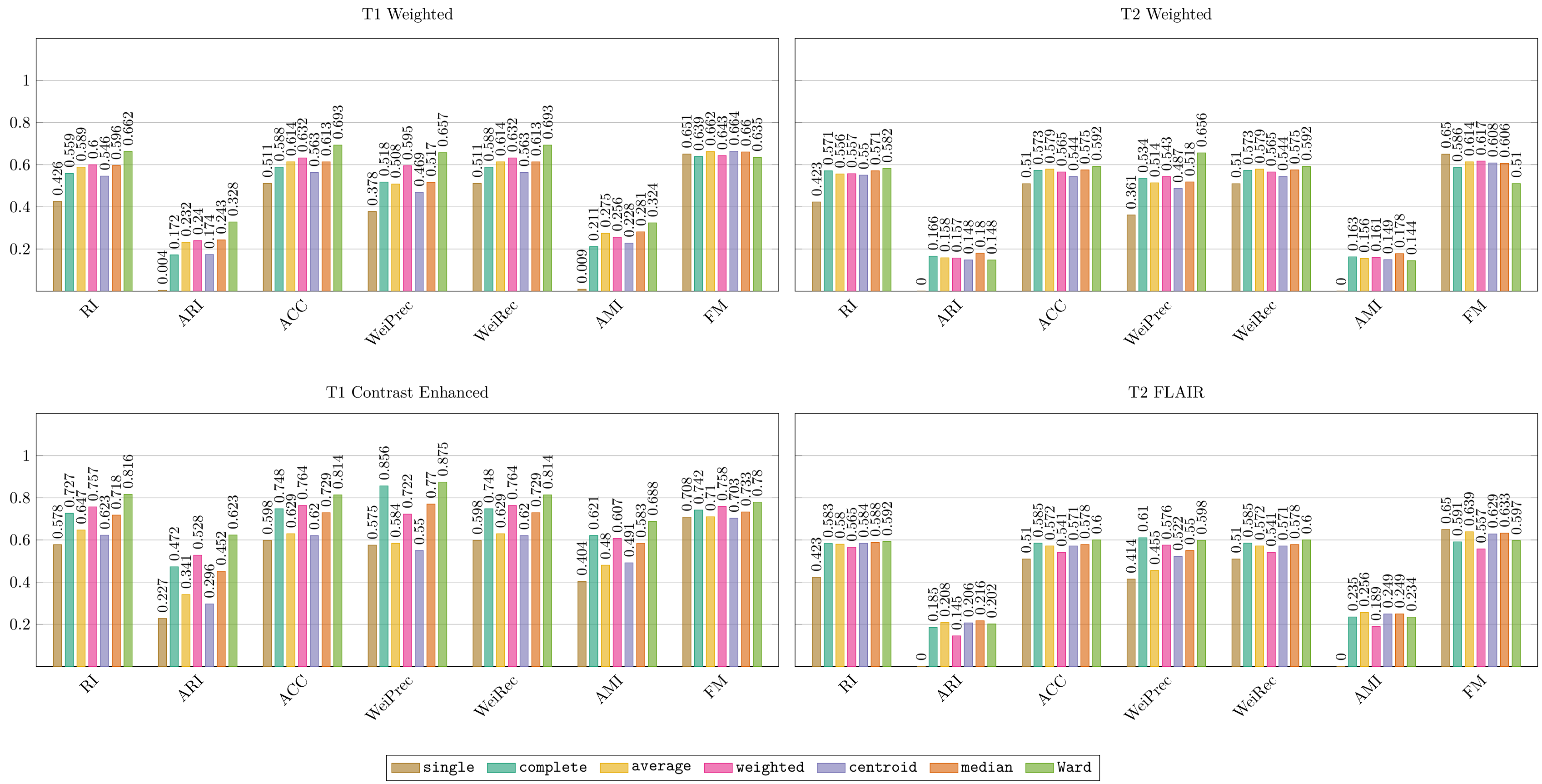


FIGURE 22: [Best viewed in color] Comparison between different hierarchical clustering linkages using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), weighted precision and recall (WeiPrec, WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). The values are grouped by sequence and represent the average of the scores of the four segmented patients. For all measures a higher value indicates a better score.

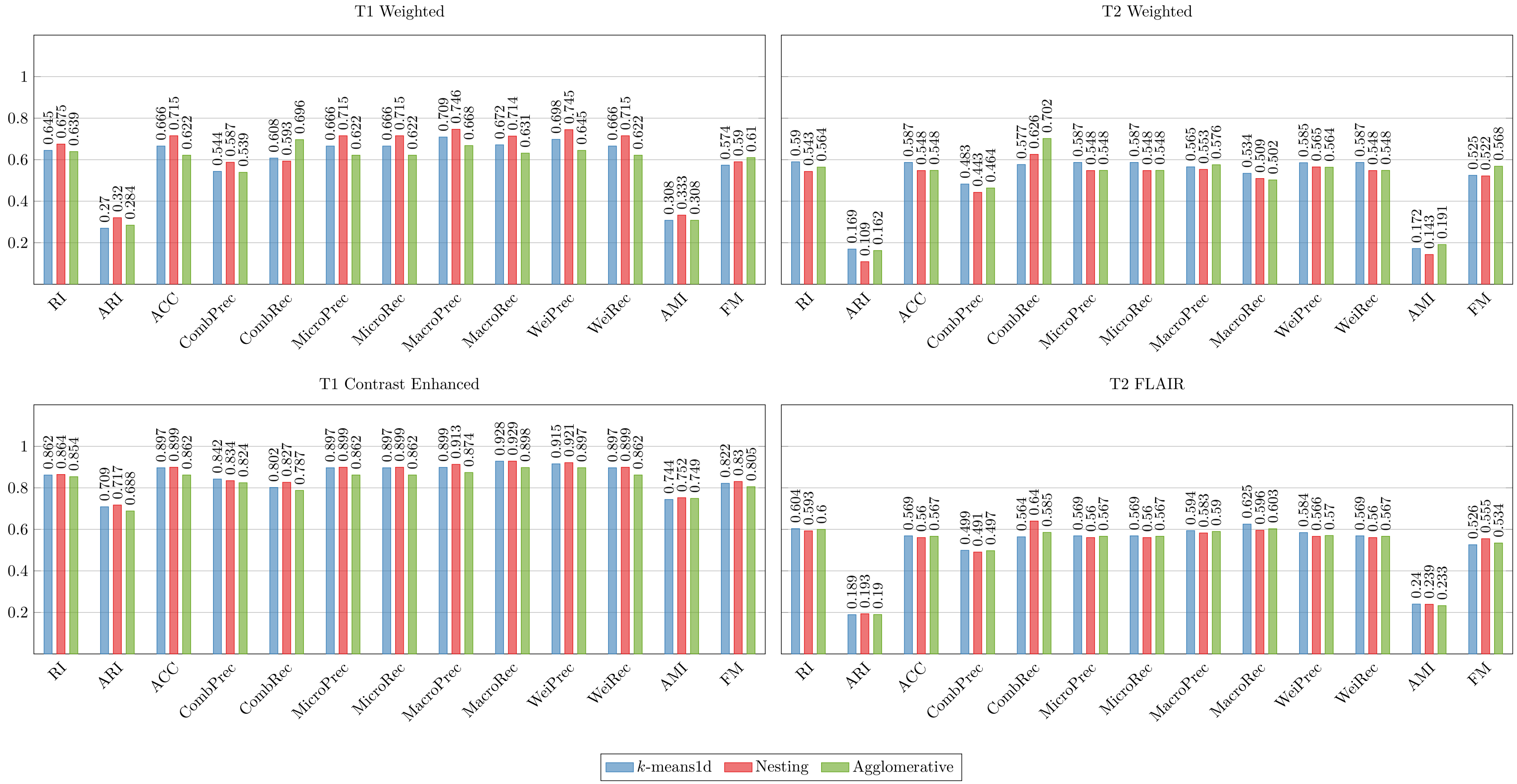


FIGURE 23: [Best viewed in color] Comparison between k-means1d, Nesting and Agglomerative (`Ward`) using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). The values are grouped by sequence and represent the average of the scores of the four segmented patients. For all measures a higher value indicates a better score.

6 MRI-to-MRI Translation: BrainClustering

In this section, we present our solution for MRI-to-MRI translation, called BrainClustering. Intuitively the MRI translation problem appears to be solvable with learning. Moreover, because of the amount of data provided by BraTS, there is the possibility of training. However, our solution does not create a model in the usual sense of the term, but it creates tables that can be queried.
In Section 1 we described a general color mapping between the different sequences (T1 Weighted, T2 Weighted, T2 FLAIR) with a table, which we present again in Figure 24. Even though the table is missing the Grey Matter class, it is easy to see that the mappings described in the table apply to the images in Figure 25. However, it is much harder to distinguish the White Matter in T2 FLAIR. Thus, since there is a visible translation between T1 Weighted and T2 Weighted, we mostly focus on mapping these two images. Furthermore, in practice it oftentimes occurs that only T1 Weighted, T1 Contrast Enhanced and T2 FLAIR are produced, while T2 Weighted is the missing image. Therefore, we focus on reproducing T2 Weighted from T1 Weighted.

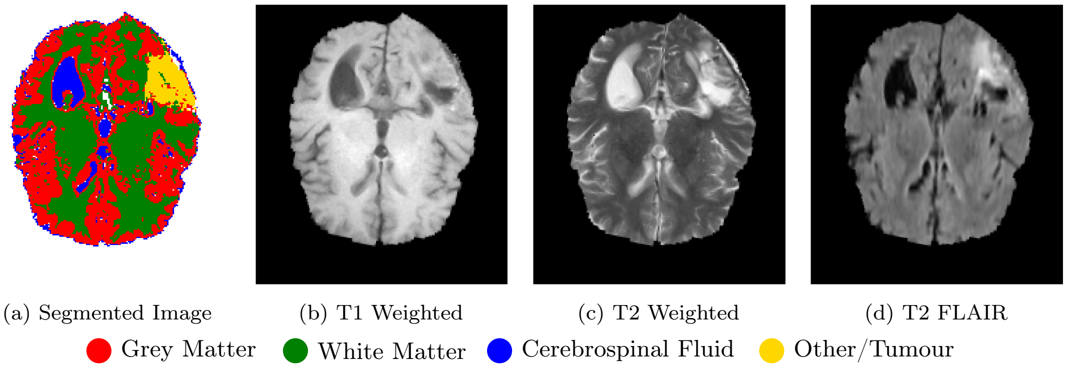

(a) Segmented Image (b) T1 Weighted (c) T2 Weighted (d) T2 FLAIR

FIGURE 24: [Best viewed in color] Segmented image and MRI scans for T1 Weighted, T2 Weighted and T2 FLAIR of patient 5 (BraTS 13).

Tissue	T1 Weighted	T2 Weighted	T2 FLAIR
White Matter	Bright	Dark	Dark
Cerebrospinal Fluid	Black	White	Black
Inflammation	Dark	Bright	Bright

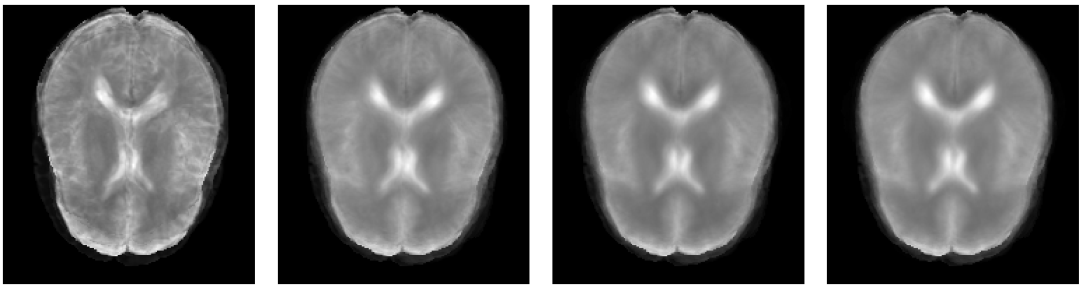
FIGURE 25: Color mapping of T1 Weighted, T2 Weighted and T2 FLAIR with respect to different tissues[17].

6.1 Simple Approaches

Before moving to our main approach we present two simple approaches that we considered in the early stages of our research.

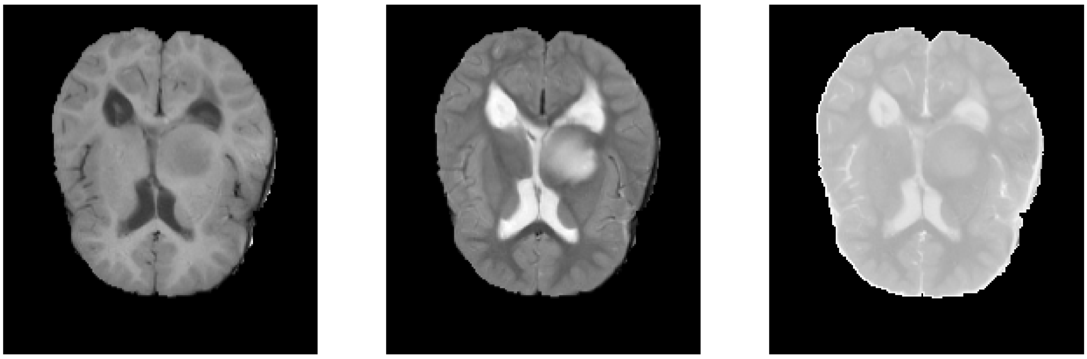
General Image Approximation. Our very first approach involves the generalization of the T2 Weighted images by taking the average of the points of different T2 Weighted images for each pixel. This gives us an approximation of the images, which is blurry and imprecise, but still has

[17] Source: https://casemed.case.edu/clerkships/neurology/Web%20Neurorad/MRI%20Basics.htm, currently unavailable.



(a) Average of ten images (b) Average of 50 images (c) Average of 100 images (d) Average of 300 images

FIGURE 26: Average of T2 Weighted with different amounts of images.



(a) T1 Weighted (b) T2 Weighted (c) Complementary of T1 Weighted

FIGURE 27: T1 Weighted, T2 Weighted and complementary T2 Weighted created by complementing T1 Weighted for patient 0 (BraTS 19).

structure, as shown in Figure 26. However, since this approach cannot be queried for a specific image and cannot reproduce the tumor information, we did not investigate it in more detail.

Complementary. T1 Weighted and T2 Weighted appear to be complementary of each other. Thus, given a T1 Weighted with points p_{T1}, we compute the complementary T2 Weighted with points p_{T2}:

$$p_{\mathrm{T2}} = \min(\mathrm{T1}) + \max(\mathrm{T1}) - p_{\mathrm{T1}}.$$

Figure 27 shows an example of the possible results. This approach produces images with poor coloring, but with a sharp and well-defined structure. The color of the Cerebrospinal Fluid area is bright enough, but the rest of the image is too light. We evaluate the performance of this method together with the other more complicated approaches in Section 8.

6.2 BrainClustering Learning

In this section, we explain our motivation and give an overview of our learning process. Assuming that we can find a function that expresses the relationship between T1 Weighted and T2 Weighted, then we can produce T2 Weighted from T1 Weighted. The most basic idea is to create a function $f(x, y, z, c_1) = c_2$ that, given the position x, y, z of the pixel and its color c_1 in T1 Weighted, outputs the corresponding color c_2 in T2 Weighted. This is clearly not feasible; the quantity of

Full Color Spectrum		Meaningful Subset	
Color T1	Color T2	Color T1	Color T2
0.1	0.60	0.1	0.60
0.2	0.50	—	—
0.3	0.42	0.3	0.45
0.4	0.40	—	—
0.5	0.38	—	—
0.6	0.30	0.6	0.30

(a) Toy example of the ideal function f_i (left) and the function g_i (right).

$$\begin{aligned}\tilde{f}_i(0.2) &= \frac{g_i(0.1)\cdot(0.3-0.2)+g_i(0.3)\cdot(0.2-0.1)}{0.3-0.1} = 0.525\\ \tilde{f}_i(0.3) &= g_i(0.3) = 0.45\\ \tilde{f}_i(0.5) &= \frac{g_i(0.3)\cdot(0.6-0.5)+g_i(0.6)\cdot(0.5-0.3)}{0.6-0.3} = 0.35\end{aligned}$$

(b) Computed values for the example above.

Figure 28: Example of the ideal mapping f_i and the partial mapping g_i and computed values according to the newly defined $\tilde{f}_i$ according to Equation 3.

the pixels would be hardly manageable, and it would lead to overfitting. A simple, but more general idea is to create a function f that, for any color c_1 of T1 Weighted, outputs the color c_2 of T2 Weighted, i.e. $f(c_1) = c_2$. However, as can be seen in Figure 25 and Figure 24, the color c_2 may assume different values according to the tissue it belongs to. Since there are differences in color changes with respect to the m tissues, we should create m functions $f_1, \ldots, f_m$. Each function is then used to predict the T2 Weighted colors of one tissue type, i.e. $f_i(c_1) = c_2$ means that a pixel of color c_1 in T1 Weighted that belongs to tissue type i has color c_2 in T2 Weighted. In practice, we may choose to store such functions as ordered tables or maps (e.g. search trees) that contain the (c_1, c_2) color pairs. However, there can be arbitrarily many values of c_1, such that it is not feasible to actually store the mapping of the entire color spectrum. Instead, we only consider a small but meaningful subset of the possible values of c_1. Let us consider an auxiliary function g_i which represents an approximation of the mapping between T1 Weighted and T2 Weighted for tissue i. This function is not defined on the full color spectrum C, but only on a smaller portion, for which it can output approximated values. In Figure 28a we present an example of an ideal function f_i and of a corresponding g_i function. This function is missing some values, which may be present in the T1 Weighted sequence that we are currently using as input. To obviate this problem we can design $\tilde{f}_i$ such that it returns the values of g_i whenever they exist, or computes an interpolation of the existing values otherwise. Thus, $\tilde{f}_i$ can be formally defined as:

$$\tilde{f}_i(c) = \begin{cases} g_i(c), & \text{if } c \in C \\ g_i(c_{\min}), & \text{if } c < c_{\min} \\ g_i(c_{\max}), & \text{if } c > c_{\max} \\ \dfrac{g_i(l)\cdot(u-c)+g_i(u)\cdot(c-l)}{u-l}. & \text{otherwise} \end{cases} \qquad \begin{aligned} c_{\max} &= \max\{e \mid e \in C\} \\ c_{\min} &= \min\{e \mid e \in C\} \\ l &= \max\{e \mid e \in C \land e < c\} \\ u &= \min\{e \mid e \in C \land e > c\} \end{aligned} \tag{3}$$

For clearer understanding, we compute the values of $\tilde{f}_i$ for the given example in Figure 28b.

We now are left with two problems: First, we need a method to identify the tissue types in each scan. Second, for each tissue we need to select a small but meaningful subset of colors to be stored. As discussed in Section 5, clustering can identify the main tissues of the brain, but can also be used to solve our second problem by creating hierarchical structures. For each main tissue, we also create a sub clustering of a certain size such that we can create smaller patches. Then, we compute the average of a patch in T1 Weighted and the average of the corresponding patch in T2 Weighted, and add only these values to the table, such that we obtain a generalization.

As in many other learning procedures, we divide the data set into training and testing. Each patient entry of the training data set is composed of two images, the input (T1 Weighted) and the real output image (T2 Weighted). During our process we iterate through the patients, compute the clusterings for both input and real output, and save the results into tables. After the training is completed, we retrieve the tables and compute a new image.

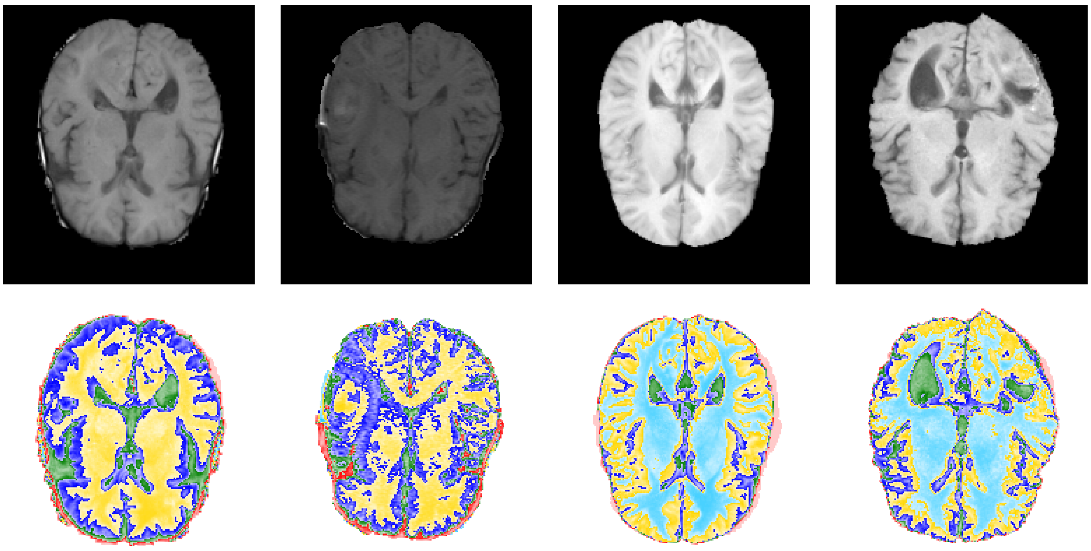
Below we summarize the parameters needed for our learning process:

- **Clustering algorithm**: Leveled k-means, Leveled Nesting or Agglomerative. However, this method has been implemented independently of the dimensionality and the specifications of the single clustering algorithms. Thus, it could be used with any clustering algorithm that produces a leveled structure.
- **Number of main clusters**: This number should represent the number of tissues of the image, i.e. the main color areas. Even if our segmented images have been divided into four colors, we allow any number because they might be able to classify the brain into color classes more appropriately.
- **Number of sub clusters**: After dividing the brain into some number of main clusters, we also want to divide each cluster into smaller clusters such that we can collect different color patches.
- **Reference image**: We ensure that the produced clusters are compatible between patients by performing a cluster label matching (see Section 4.7) with respect to the same reference image for the training and testing phase.

The reference image is used for the following reason: What if, when computing the clustering on one patient the White Matter is yellow, and on another patient it is light blue (as in Figure 29)? The values produced by these clusterings should theoretically be mapped by the same function, but are not. However, with a reference image, we can perform label matching and find the standardized label a cluster should obtain. There are two problems with this approach:

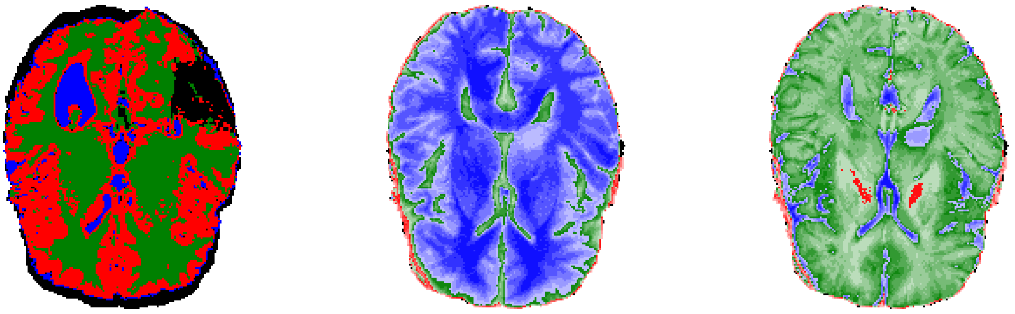
1. The matching is not ensured to be correct. Even though the Hungarian algorithm does obtain the optimal result, the matching is based on the contingency matrix counts, which may not accurately represent the different structures of the brains.
2. It is necessary that the labels represent the same pixels. This means that if a point is part of the background in T1 Weighted but a tissue point in T2 Weighted, we should either consider it in both images or in none. Label matching requires that the same pixels are taken from the images being investigated. Since our MRI scans have been registered to the same template, it is reasonable to assume that similar tissues will occur at similar pixel positions.

Theoretically, the reference image should represent a standardized image. Thus, we choose a manually segmented MRI scan from a doctor. As discussed in Section 5, these images are only segmented into four tissues, but we allow any number of main clusters. Additionally, as explained in the same section, the tissue Other/Tumour is not really a color class and is very hard to capture with a clustering algorithm. Thus, whenever the number of clusters is three, we use a segmented image as reference. Otherwise, we run k-means1d on the first training image and use this clustering as reference image for all others.



(a) Patient 5 (b) Patient TCIA01 150 (c) Patient 15 (d) Patient 19

FIGURE 29: [Best viewed in color] Leveled k-means clustering with $k = 5$ for T1 Weighted of four patients (BraTS 19). In (a) and (b) the lightest cluster (light blue) is very small. This is due to the fact that these images have very intense signals in these areas, which then form one cluster. The central yellow area of (a) and (b) corresponds to the light blue area of (c) and (d), and the blue area of (a) and (b) corresponds to the yellow area of (c) and (d).



(a) Segmented Image (b) T1 Weighted Clustering (c) T2 Weighted Clustering

FIGURE 30: [Best viewed in color] Union of pixel between segmented image and the current training patient's T1 Weighted and T2 Weighted. In black the pixels that were added by computing the union.

6.3 Training

In this section, we discuss the BrainClustering training process in more detail. First, we initialize empty maps $g_1, \ldots, g_m$, and then start the actual learning by considering the first patient. We compute a hierarchical clustering for both images T1 Weighted and T2 Weighted according to the number of main clusters and sub clusters that we have chosen. However, the clusterings should be computed on the union of the *non-background* points of T1 Weighted, T2 Weighted and the reference image because of Problem 2 of label matching. Figure 30 shows a representation of the added pixels for each image. Even though these pixels are used to compute the clustering, they are not considered when building the contingency matrix. Another approach would be to consider the intersection of the points from these images, but that might prevent some smaller clusters from being created. One more solution, however more cumbersome, would be to keep track of all the pixels that are in the intersection between T1 Weighted and T2 Weighted, and T1 Weighted and the reference image, and skip them when computing the contingency matrix.

Afterwards, we match the labels between T1 Weighted and the reference image, and between T1 Weighted and T2 Weighted. Let i be the label of one of the main clusters in the reference image and let $\mathrm{T1}_i$ and $\mathrm{T2}_i$ be respectively the corresponding clusters in T1 Weighted and T2 Weighted. The next step is to compute the correspondence between the smaller patches of T1 Weighted and T2 Weighted. For each main cluster i, we compute the intersection of pixels positions of cluster $\mathrm{T1}_i$ and $\mathrm{T2}_i$ (due to Problem 2). Next, we compute the label matching between the smaller patches of T1 Weighted and T2 Weighted inside cluster i. Once we obtain this second match, we compute the average of the points in each sub cluster of T1 Weighted, which we call a_1. Then, we compute the average for each corresponding sub cluster of T2 Weighted, which we call a_2. Finally, we add a new entry (a_1, a_2) to g_i, which is the map corresponding to the current main cluster. It may happen that there already is an entry with key a_1 stored in the map. In this case, we compute the average with the moving average formula:

$$g_i(a_1) \leftarrow g_i(a_1) + \frac{a_2 - g_i(a_1)}{\#_i(a_1)}, \tag{4}$$

where $\#_i(a_1)$ is the number of times we tried to add an entry with key a_1 to the map g_i. Note that consequently we have to keep track of $\#_i(a_1)$, i.e. we have to annotate each entry of the map with the corresponding cardinality. The entire process for one patient is outlined in Algorithm 8 and visualized in Figure 31.

The most time-consuming part of this process is the clustering. The label matching is done on small matrices (first on the main clusters, then on the sub clusters) and the loops to fill the tables just iterate over all points to compute the averages.

The process is repeated for all patients. Each time a patient has been processed, the tables are saved, such that, if desired, one could consider using the tables generated by any stage of the training process, or continue the training with new images. The table file, or model, is named after a combination of the clustering algorithm and the number of main and sub clusters, such that it is possible to choose which model to use.

6.4 Testing

In this section, we describe the querying process of BrainClustering. First, we retrieve the ordered tables, the reference image, the clustering algorithm and the numbers of clusters. For simplicity, the reference image, the clustering algorithm and the cluster amounts are saved with the model. The model is loaded and the ordered tables are given as input to the testing process for each queried patient. We cluster the T1 Weighted of the patient according to the clustering algorithm (with k-means1d if the algorithm is Leveled k-means or Leveled Nesting, with Agglomerative otherwise) with a plain color clustering of m clusters. Then, we compute a label matching between the retrieved reference image and the new clustered T1 Weighted. Let i be one of the main clusters in the reference image and let $\mathrm{T1}_i$ be respectively the corresponding cluster in T1 Weighted. Finally, we consider each point p of $\mathrm{T1}_i$ and query the respective function $\tilde{f}_i(p)$ using the mapping $g_i(p)$ as described in Equation 3 (i.e. if $g_i(p)$ exists then $\tilde{f}_i(p) = g_i(p)$, or otherwise $\tilde{f}_i(p)$ is computed by interpolating between the two closest values in g_i).
Once all the points have been computed, the images are postprocessed as described in the next paragraph. In Algorithm 9 we outline the full querying process. We also provide a simple example in Figure 32.

Algorithm 8 Training Procedure for One Patient

```
Input: The patient's T1 Weighted and T2 Weighted, a clustering algorithm, a number of main clusters $m$ and sub clusters $s$, a reference image.
Output: The tables $g_1, \dots, g_m$ containing a color mapping between T1 Weighted and T2 Weighted.
1. If the tables have not been initialized, create $m$ tables $g_1, \dots, g_m$.
2. Compute the union of the non-background points of the reference image, T1 Weighted and T2 Weighted.
3. Compute a hierarchical clustering with $m$ main clusters and $s$ sub clusters on both T1 Weighted and T2 Weighted.
4. Match the labels of the main clusters in T1 Weighted to the reference image labels.
5. Match the labels of the main clusters in T2 Weighted to the main clusters in T1 Weighted.
6. For each main cluster $i \in \{1, \dots, m\}$ of the reference image:
    (a) Let T1$_i$ be the corresponding cluster in T1 Weighted.
    (b) Let T2$_i$ be the corresponding cluster in T2 Weighted.
    (c) Compute the intersection of the points in cluster T1$_i$ and cluster T2$_i$.
    (d) Considering only the intersection, match the labels of the sub clusters of T1$_i$ to the labels of the sub clusters of T2$_i$.
    (e) Let $s_1$ and $s_2$ be the respective numbers of sub clusters in T1$_i$ and T2$_i$.
    (f) For each sub cluster $j \in \{1, \dots, \min(s_1, s_2)\}$:
        i. Let T1$_j$ and T2$_j$ be the corresponding sub clusters in T1$_i$ and T2$_i$.
        ii. Let $a_1$ be the average color of the points in T1$_j$.
        iii. Let $a_2$ be the average color of the points in T2$_j$.
        iv. Add $(a_1, a_2)$ to $g_i$. If the point already exists, compute the average (see Equation 4).
```

Algorithm 9 Testing Procedure for One Patient

```
Input: The patient's T1 Weighted, the color maps $g_1, \dots, g_m$.
Output: The approximated T2 Weighted image for the given patient.
1. Let $m$ be the number of main clusters for which the model was trained.
2. Compute a simple clustering of $m$ clusters on T1 Weighted. The clustering algorithm depends on the one used during training.
3. Match the clusters of T1 Weighted to the reference image.
4. For each main cluster $i \in \{1, \dots, m\}$ of the reference image:
    (a) Let T1$_i$ be the corresponding cluster in T1 Weighted.
    (b) For each point $p$ in T1$_i$, compute the color of $p$ in T2 Weighted with $\tilde{f}_i(p)$ as defined in Equation 3.
5. Remove noise from the image by applying a filter of choice.
```

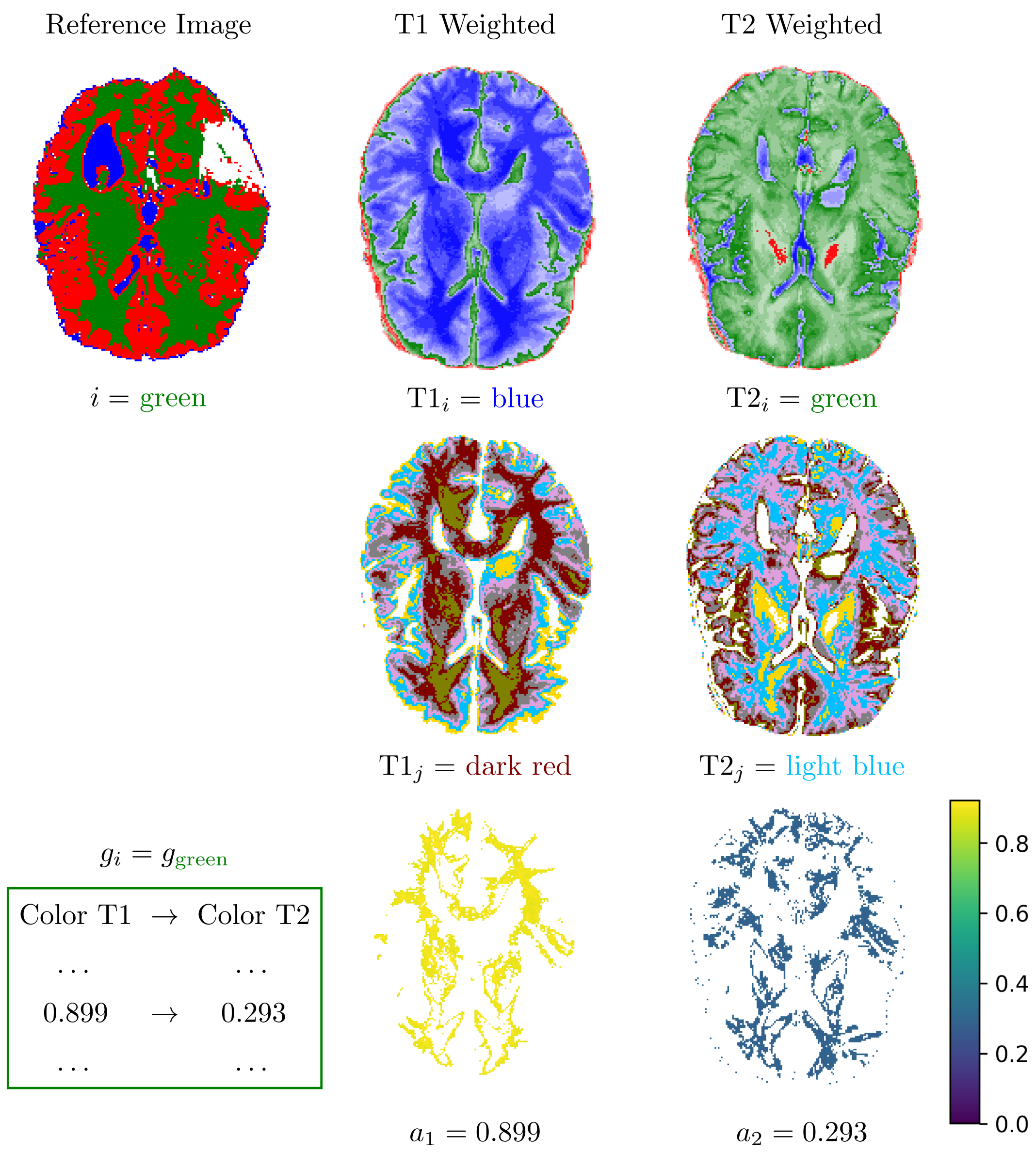


Figure 31: [Best viewed in color] Training procedure for patient 2 (BraTS 19) with Leveled k-means, three main clusters and 18 sub clusters. First, we select one main cluster i in the reference image. For example, let i be green. Then, we find the matching label for T1 Weighted (blue) and for T2 Weighted (green). We now focus only on this cluster and check the sub clusters (six per main cluster, a total of 18). We compute label matching again and choose one cluster, dark red in T1 Weighted, which corresponds to light blue in T2 Weighted. We now consider only the respective points in T1 Weighted and T2 Weighted and compute the average, which is then added to the mapping table of green, because green is the color of this main cluster in the reference image. The process is repeated for all sub clusters and main clusters. In this case the points are presented with a different coloring from the standard black and white images for a clearer distinction on the white background.

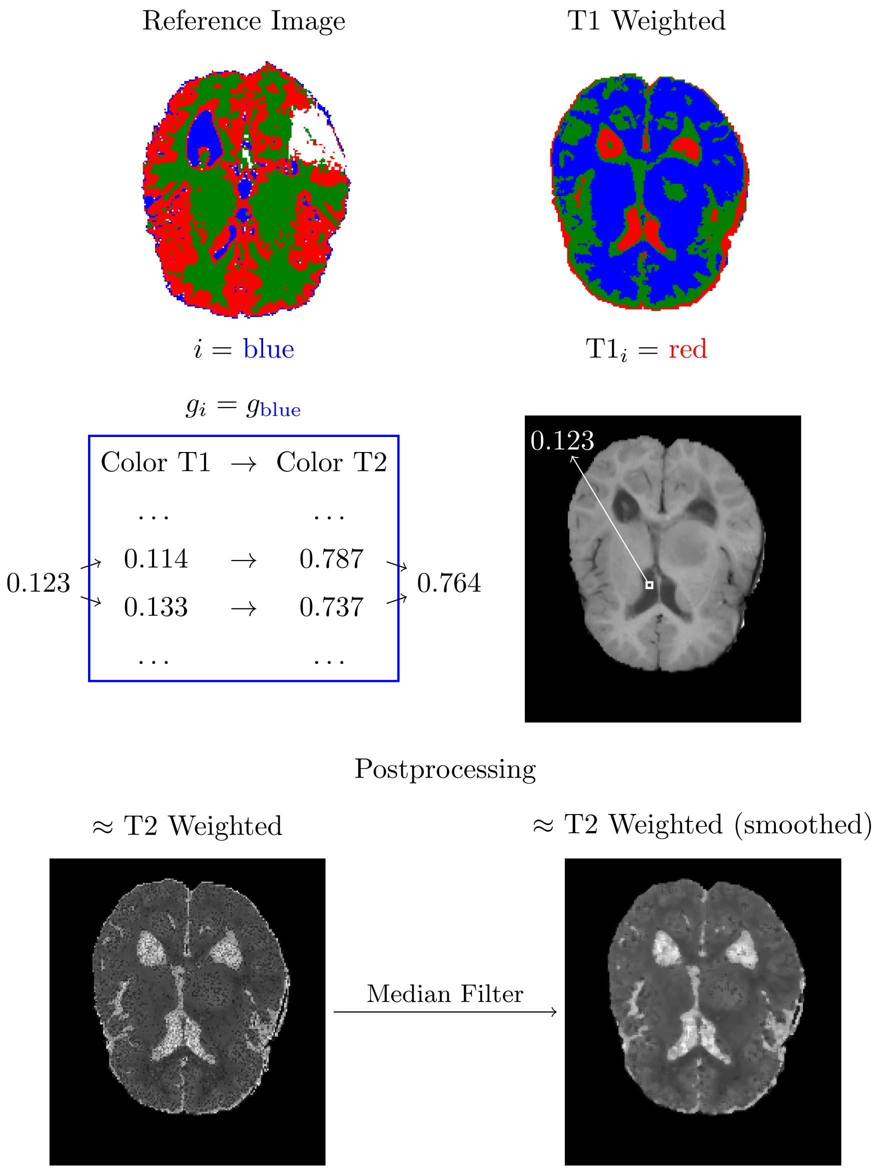


Figure 32: [Best viewed in color] Testing procedure for patient 0 (BraTS 19). The functions for this model are built with Leveled k-means, three main clusters and 18 sub clusters. First we retrieve the same reference image used for training. Then we compute a plain-color clustering, and choose a main cluster, for example red, and compute the label matching of red in the reference image, which is blue. We iterate through the points belonging to red, and for each of them we retrieve the color in the T1 Weighted image. We look for this color in the blue table, and interpolate if it does not exist. We repeat the same procedure for all points in each main cluster. Finally, we can see one resulting image and the smoothed image with the median filter. Since the number of sub clusters is very low, there is not much color variety in the image. In Figure 33 we provide the same image with three main clusters and 102 sub clusters for a better comparison.

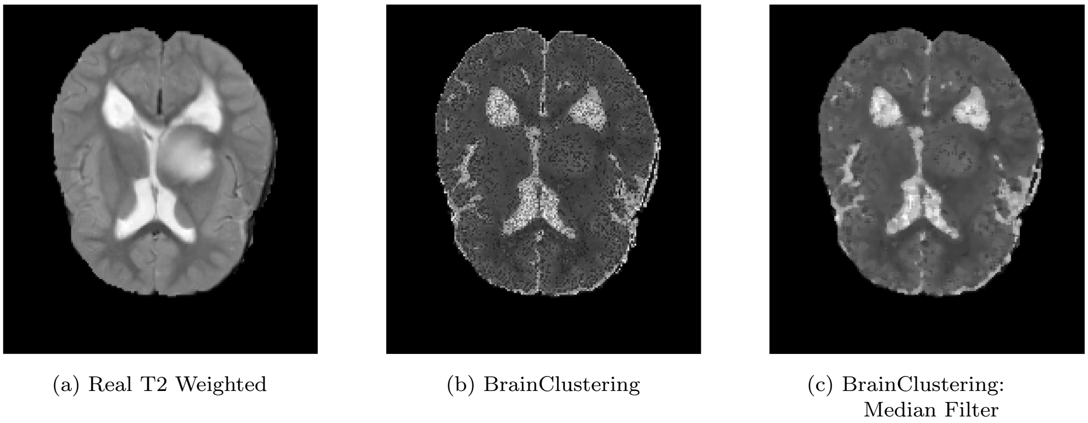

(a) Real T2 Weighted (b) BrainClustering (c) BrainClustering: Median Filter

FIGURE 33: Real T2 Weighted and T2 Weighted created with BrainClustering with three main clusters and 102 sub clusters (patient 0 from BraTS 19).

Filters and Smoothing. Filtering is a common process in image and signal processing, and it is used to eliminate some particular feature of an image. A very important concept in this field is the mathematical operation of convolution, which is used to combine two functions. In image processing the two functions are an image, where each pixel represents a value, and a convolution kernel, which depends on the type of filtering that will be applied. A common filter is the average filter, which has the following kernel structure:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

but the size of the kernel can be chosen freely, with the condition of changing the averaging value to the number of values in the matrix. This kernel is overlapped with each 3×3 submatrix of the image to process, and each image pixel is multiplied with the respective kernel pixel. Then, the sum of the modified pixels is computed. Finally, the image pixel corresponding to the middle of the matrix is replaced with the newly computed value. Note that to compute the pixels on the border some image padding is necessary, for example padding with a constant value.

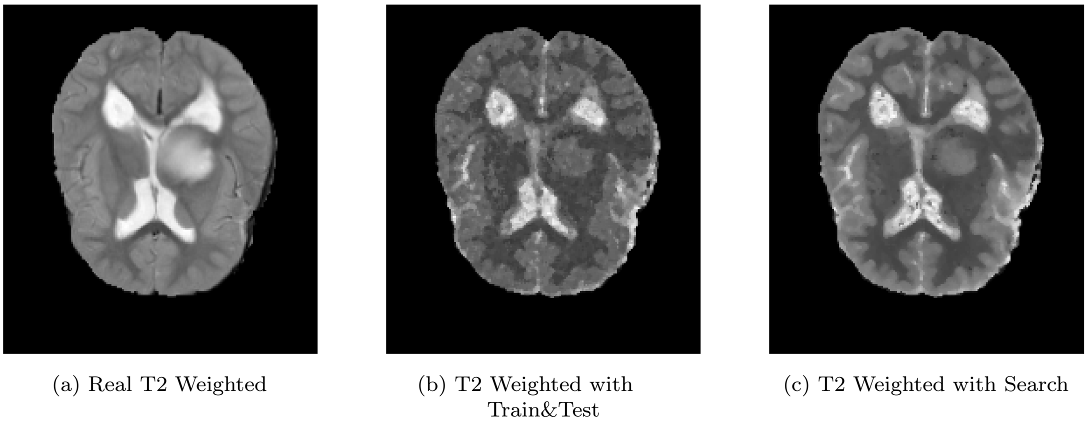
In our case, the images have a type of noise called *salt and pepper noise*, which is manifested by few pixels with a very different color with respect to the surrounding ones (see e.g. Figure 33b). The most effective smoothing method for this problem is the *median filter*, which takes the median of the pixels under the kernel area and substitutes the central pixel with the selected value. In this case, the pixel used for substitution is already present in the image. The result of the filtering process can be seen in Figure 33c.

For our purposes we use the efficient implementation of OpenCV[18] for the 3×3 median filter, but any arbitrary filter could be chosen instead. The border padding used for the median filter is implemented by padding with the closest value to the border.

6.5 Search Mode

In this section, we present an additional functionality of BrainClustering, a fusion of training and testing that produces more targeted results. According to our tests, having large tables produced by training on many images produces noise in the results. Thus, we propose an additional method

[18] https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html



(a) Real T2 Weighted

(b) T2 Weighted with Train&Test

(c) T2 Weighted with Search

FIGURE 34: Real T2 Weighted and T2 Weighted created with BrainClustering with Leveled k-means, five main clusters and 200 sub clusters with training and testing and with search (patient 0 from BraTS 19).

where we do not precompute a large model before querying, but instead compute a small model every time we want to answer a query. Given a T1 Weighted query image, we first search in the training data set for the w patients whose T1 Weighted has the smallest MSE to the query image (where w is a small constant, e.g. $w = 5$). Then, we create a small model by performing the training process only on these w patients. Afterwards, we run the testing on the created model. This approach has the downside that we have to create a new model for every single query, thus significantly increasing the query time. However, as can be seen in Figure 34, we obtain a less noisy output.

7 Generalized Image-to-Image Translation: Pix2Pix

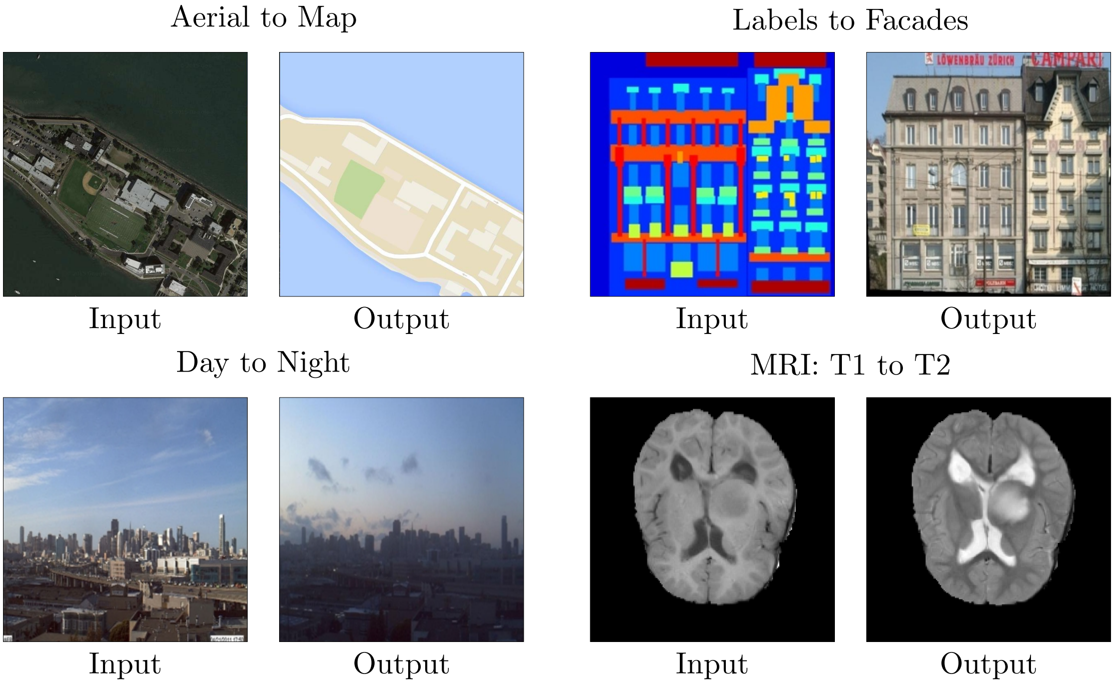
In this section, we discuss the Pix2Pix [20] framework and in Section 8.2.3 we describe the more technical details of our three-dimensional adaptation. Pix2Pix aims at solving the two-dimensional image translation problem using cGANs (*Conditional Generative Adversarial Networks*), which are able to learn the mapping between a set of input and output images. Some common use cases (and our objective) can be observed in Figure 35.

7.1 Generative Adversarial Nets

In recent years, CNNs (*Convolutional Neural Networks*) have popularly been used for solving image-related tasks. However, they require the specification of a loss function that the network should train on. GANs are a solution to this problem. In fact, they do not need a specific loss function, but are trained to create results that are indistinguishable from reality, and they can implicitly specify a loss function with this purpose.

A GAN [14] is a neural network framework that simultaneously trains two models, a *generator* G and a *discriminator* D, which work with each other in an adversarial process. In fact, G tries to capture the data distribution and reproduce the images given in the training, while D estimates whether the input it receives was given by the training data or by the generator. In other words, the objective of the generator is to trick the discriminator, while the discriminator wishes to output a correct guess. The process has the aim of generalizing the input data and generating a new output, which is believably similar to the training set.

Since we want to create a completely new output, the generator's input is random noise. The mapping from random noise to desired output is improved with backpropagation, which is the


Aerial to Map

Labels to Facades

Day to Night

MRI: T1 to T2

Figure 35: [Best viewed in color] Some default Pix2Pix examples (aerial shot to maps[19], labels to building facades [40], day to night panorama [25]) and our use case.

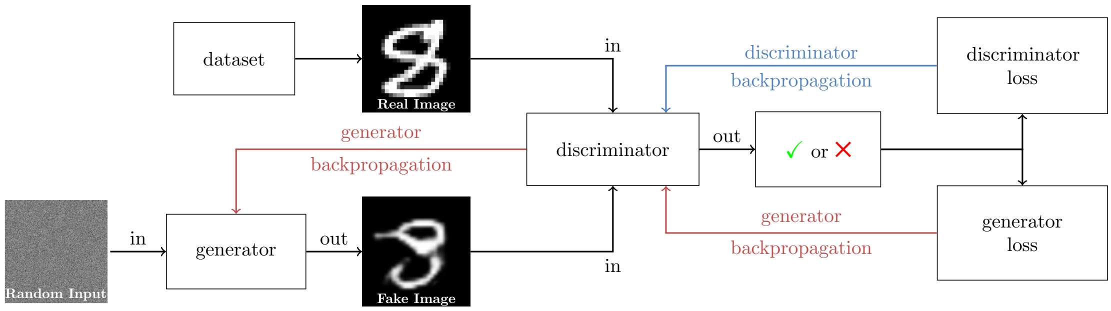


Figure 36: [Best viewed in color] Example of GAN network with images from MNIST[20], a data set of handwritten digits. The generated image is taken from Towards Data Science[21].

most common method for gradient descent on neural networks. In more detail, the generator creates a sample, which is then passed to the generator, to finally output a loss. These values are backpropagated through the discriminator and the generator to obtain gradients, which are used to change the generator's weights.

The discriminator also learns by backpropagation. The loss of the discriminator is a penalty for sample misclassification (fake samples as real, or real samples as fake), and its weights are updated by backpropagation of the loss on the discriminator network. In Figure 36 we present a schematic view of the GAN learning process.

Let G and D be any non-linear mapping functions. The generator learns the distribution p_g of the data $\boldsymbol{y}$ from noise $p_z(\boldsymbol{z})$ and creates a mapping function $G(\boldsymbol{z})$, while the discriminator

[19] Scraped from https://www.google.com/maps.

[20] http://yann.lecun.com/exdb/mnist/

[21] https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0

with mapping $D(\boldsymbol{y})$ outputs the probability that $\boldsymbol{y}$ comes from the training data. Because of the relationship between the two networks, the objective function as be described as:

$$\mathbb{E}_{\boldsymbol{y}}\left[\log D(\boldsymbol{y})\right] + \mathbb{E}_{\boldsymbol{z}}\left[\log(1 - D(G(\boldsymbol{z})))\right]. \tag{5}$$

In fact, we train D to maximize the probability that the real data $\boldsymbol{y}$ is identified as real, while we train G to maximize the probability that the data generated from noise $G(\boldsymbol{z})$ is detected as real by the discriminator. The latter objective corresponds to minimizing $1 - D(G(\boldsymbol{z}))$. The final form of Equation 5 comes from formalizing these high-level objectives with the log-loss principle of multi-class classification problems.
However, in practice it is indeed better to train G to maximize $\log D(G(\boldsymbol{z}))$, because early in the learning the discriminator is easily able to detect the generated samples, which may cause the GAN to saturate, and maximizing $\log D(G(\boldsymbol{z}))$ prevents this effect.

Conditional GANs. In our use case we do not want to create a generalization, but an output based on a controlled input. This can be achieved with *conditional* GANs [32].
cGANs introduce a condition to the objective function in Equation 5, which results in asking for a believable result that represents a mapping from a given input. Thus, we also have an additional parameter $\boldsymbol{x}$ for our mapping function G, which yields the following objective function:

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}\left[\log D(\boldsymbol{x}, \boldsymbol{y})\right] + \mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}\left[\log(1 - D(G(\boldsymbol{x}, \boldsymbol{z})))\right].$$

7.2 Pix2Pix

Pix2Pix [20] is presented as a solution to the common pixel-to-pixel translation problem, which is commonly handled by creating ad hoc software. This framework is based on cGANs, but uses a slightly different objective function, which according to their experiments improves the quality of the results. In Figure 37 we present a sketch of the behavior of the generator and of the discriminator in Pix2Pix.
A simple method to perform image translation is to pass a function through a network and compute the loss as the absolute (L1 loss) or squared (L2 loss) differences. The authors fuse this concept together with the cGAN loss, which means that the objective of the discriminator is unchanged, while the generator is asked to trick the discriminator while maintaining a small loss. According to the authors, the L2 loss produces more blurred images, and thus L1 is chosen instead.
The resulting objective function is:

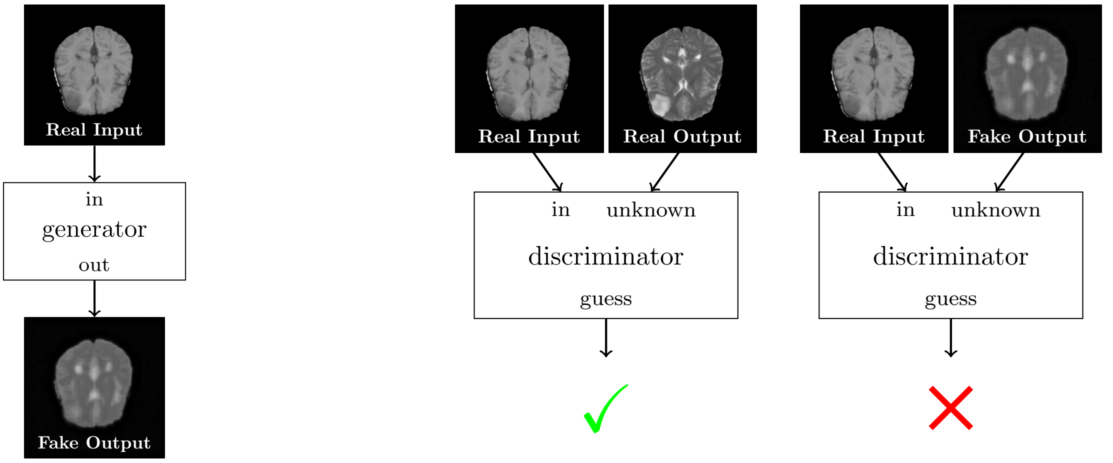
$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}\left[\log D(\boldsymbol{x}, \boldsymbol{y})\right] + \mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}\left[\log(1 - D(\boldsymbol{x}, G(\boldsymbol{x}, \boldsymbol{z})))\right] + \lambda \cdot \mathbb{E}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}\left[\|\boldsymbol{y} - G(\boldsymbol{x}, \boldsymbol{z})\|_1\right],$$

where λ can be any weight. The authors report that on their experimental setup using the cGAN alone ($\lambda = 0$) produces artifacts in the images, which can be reduced by using $\lambda = 100$.

Another difference from the previous two models is that the noise is not used as input anymore, but rather implemented in the form of dropout in some layers of the generator to prevent a deterministic output. Dropout is a common strategy to prevent neural networks from overfitting, and involves dropping a randomly selected set of neurons. However, in this case it is only used to provide some noise, which nonetheless only yields minor stochasticity to the output.

Next, we discuss in more detail the structure of the generator and of the discriminator for this particular framework.

[22] https://affinelayer.com/pix2pix/.



(a) Generator

(b) Discriminator

FIGURE 37: Generator and discriminator representation in Pix2Pix. Image inspiration from AffineLayer[22].

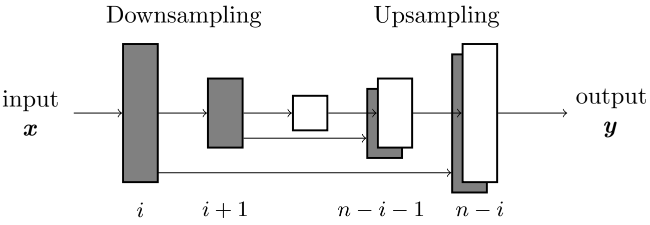


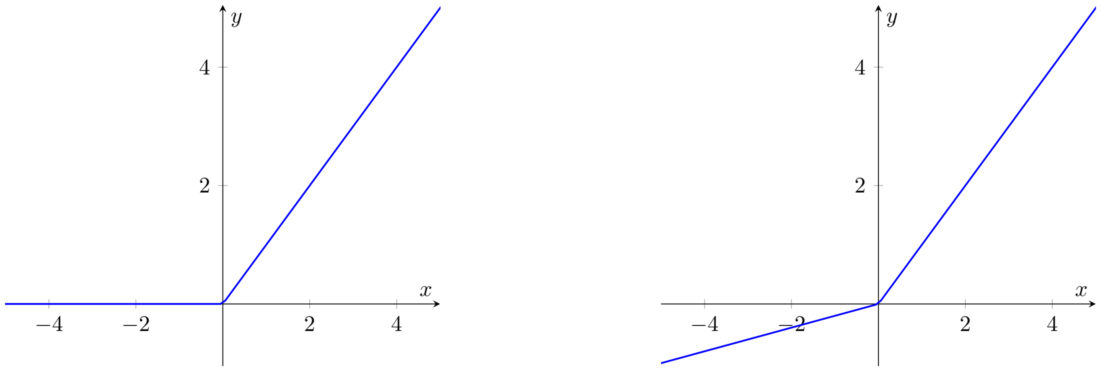
FIGURE 38: The structure of the U-Net network used in Pix2Pix.

7.2.1 Generator: U-Net

The feature detection process in CNNs is implemented by transforming the input into smaller and smaller interpretations, such that the essential features of the data can be understood. In image translation problems the desired mapping is between a high resolution input and a high resolution output, and thus we need a network architecture that is able to capture, with a sequence of downsampling and upsampling operations, the mapping between the images. Many solutions to similar problems employ an encoder-decoder network, which compresses the images into smaller and smaller representations using convolution and then reverses the process and obtains an image of the original size. In Pix2Pix they use a slightly different network, called U-Net, that additionally has skip connections, which give it the shape of a "U" (Figure 38). The skip connections add concatenations between each dowsampling layer i and each upsampling layer $n - i$, where n is the total number of layers.

However, convolution and deconvolution are not the only operations performed in the encoding/decoding process. In fact, the convolution result is batch normalized and passed through a non-linear activation function. *Batch norm* is the process of normalizing the current batch by subtracting the mean and dividing by the standard deviation, which improves accuracy and training speed [17]. The activation function of choice is the ReLU (*Rectified Linear Unit*) (Figure 39a) for the upsampling operation and the Leaky ReLU (Figure 39b) for the downsampling.

When designing the filters used in the convolution, it is important to know the size of the image. In fact, they created two types of U-Net (U-Net128 and U-Net256) which work respectively with images with sizes that are multiples of 128 and 256. However, thanks to data augmentation, any



(a) The ReLU function.

(b) The Leaky ReLU function with negative slope 0.2.

size can be processed to fulfill either of these requirements. In fact, if an image is too small, it can be padded, while if it is too big, it can either be scaled or cropped. In the latter case each image is randomly cropped such that the network always processes a different part of the image, and will eventually be able to capture the entire structure. Furthermore, the images may also be randomly flipped, such that the network does not get used to the position of an image and obtains a better generalization. In Figure 39 we present the U-Net128 network structure for a two-dimensional 128 × 128 input.

7.2.2 Discriminator: PatchGAN

The L2 and L1 losses can accurately capture low frequencies, but fail to produce sharp images, which yields blurred results. Thus, it is sufficient to focus the discriminator on high frequencies, letting the L1 loss handle the low ones. Therefore, the authors of Pix2Pix implement a discriminator called PatchGAN, that does not compare entire images, but rather smaller patches of a given size.

The current PatchGAN implementation takes the real input and the unknown output (either the real output or the output of the discriminator) and classifies each 70 × 70 patch of the unknown image into real or fake. The two images are concatenated and a binary image is created with a sequence of convolution operations. The size of the binary image depends on the input size and each pixel represents the believability of a patch of the unknown image. In Figure 40 we show the PatchGAN network structure for a two-dimensional 128 × 128 input.

8 Mapping Evaluation

In this chapter, we evaluate the quality of the produced images. Our evaluation is based on errors with respect to the real image and on the comparison between the real tumor segmentation and segmentations that can be achieved with our transformed images. The former is performed manually by a doctor, while the latter is produced by tumor segmentation software (DeepMedic and HD-GLIO) with three original MRI scans and one transformed image as input. As discussed previously, the tumor segmentation software has been trained with the four sequence types (T1 Weighted, T2 Weighted, T1 Contrast Enhanced, T2 FLAIR). We test our transformation functions by substituting the original T2 Weighted with a T2 Weighted produced with one of the previously discussed mapping functions (BrainClustering and Pix2Pix). All of our transformations produce T2 Weighted from T1 Weighted.

Furthermore, we also present a small preliminary evaluation of the results that can be obtained with DeepMedic for reproducing T1 Contrast Enhanced from T1 Weighted.

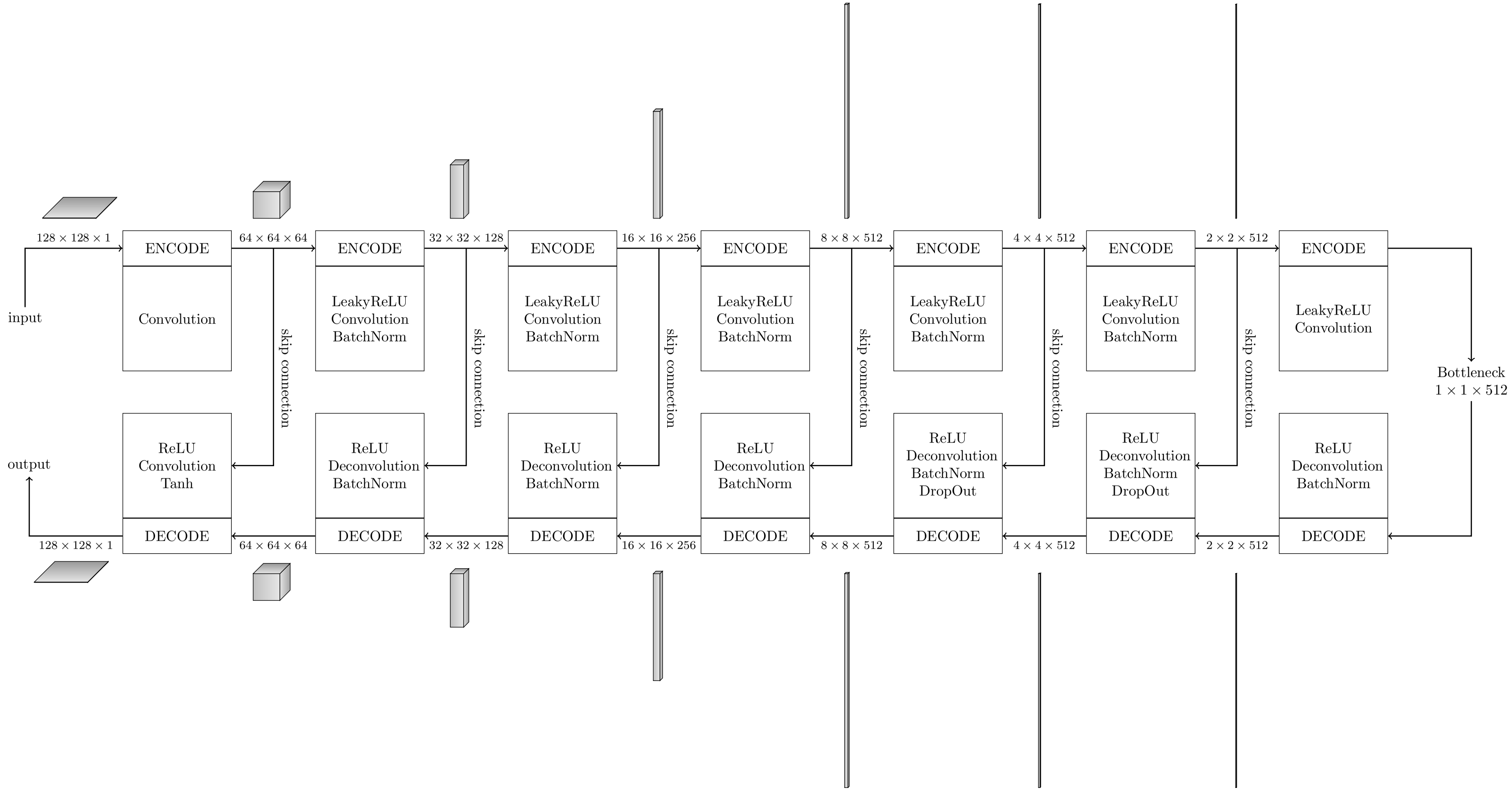


FIGURE 39: Generator structure: U-Net128 network for a 128 × 128 grayscale input. The U-Net128 network is designed to have seven encoding/decoding steps. Of these, all layers except the first and last (in encoding) have a batch norm layer, and only a few layers have drop out units. Each encoding layer performs convolution, downsampling the image until a bottleneck, where the process is reversed. Image inspiration from AffineLayer[22].

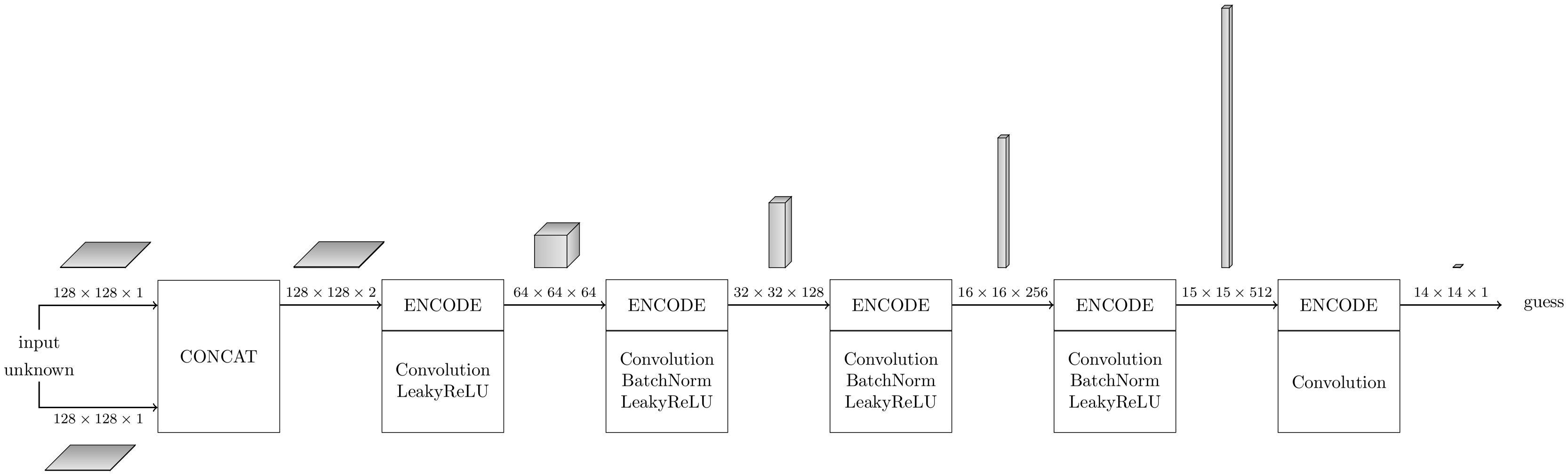


FIGURE 40: Discriminator structure: 70 × 70 PatchGAN for a 128 × 128 grayscale input. The number of intermediate layer is three by default, and in these layers the result of the convolution is also batch normalized and passed through a Leaky ReLU. Image inspiration from AffineLayer[22].

8.1 Evaluation Metrics

In this section, we describe the different evaluation metrics that we use: mean squared error (MSE), sensitivity, specificity, Dice similarity coefficient (DSC) and Hausdorff distance. We use the MSE to analyze how similar the approximated images are compared to the original ones. In contrast to that, the other measures are used to compute the quality of the tumor segmentation that can be achieved with our transformed images.
Additionally to the listed measures, we also compute the accuracy, which has been previously described in Section 5.1. The accuracy is computed between the ground truth of the segmentation and a segmentation produced with DeepMedic or HD-GLIO by checking if the labels correspond pixel-wise. However, this measure is exclusively computed on the area of the tumor to prevent a too high score given by correctly pinpointing the general position of the tumor. We discard the other previously used measures because they are not commonly used for this type of evaluation.
Sensitivity, specificity, DSC and Hausdorff distance are computed on binary data created by classifying the tumor segmentation files into three classes (Section 2.1): Whole Tumor, Tumor Core, Active Tumor. In this interpretation the pixels that have value one correspond to the membership to the class, and are zero otherwise. Thus, we obtain one score for each of the three classes. We compute these measures with the plastimatch package[23], which is equipped with efficient algorithms for medical imaging.

Mean Squared Error. Let `truth` be the real T2 Weighted image and let `pred` be the predicted result. The mean squared error measures the estimated difference between the truth and the prediction:

$$\texttt{MSE} = \frac{1}{\texttt{n\_samples}} \cdot \sum_{i=1}^{\texttt{n\_samples}} (\texttt{truth}_i - \texttt{pred}_i)^2.$$

Sensitivity, Specificity and Dice Similarity Coefficient or F1 Score. Sensitivity and specificity are the medical counterpart of precision and recall. In fact, sensitivity corresponds to recall, while specificity corresponds to the precision with respect to the true negatives:

$$\texttt{sensitivity} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}}, \ \texttt{specificity} = \frac{\texttt{TN}}{\texttt{TN} + \texttt{FP}}.$$

The measures are traditionally computed on binary data. Since the positive condition of our use case is the presence of a tumor, it is more relevant to consider how many brain voxels identified as healthy do not actually have a tumor, then to consider the percentage of selected the brain voxels that really contain a tumor.
However, we do consider the precision in an additional measure, called Dice Similarity Coefficient or F1 score, which is the harmonic mean of the precision and recall.

$$\texttt{F1 score} = 2 \cdot \frac{\texttt{precision} \cdot \texttt{recall}}{\texttt{precision} + \texttt{recall}} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}}$$

All values are bounded in range $[0, 1]$.

Hausdorff Distance. The Hausdorff distance [37] is commonly used in computer vision to detect the distances between shapes or volumes. Given regions T of the true tumor and P of the predicted tumor, the directed Hausdorff distance from T and P is the maximum distance of any point in T to their closest point in P:

$$\vec{d}_H(T, P) = \max_{t \in T} \min_{p \in P} d(t, p),$$

[23] https://plastimatch.org/

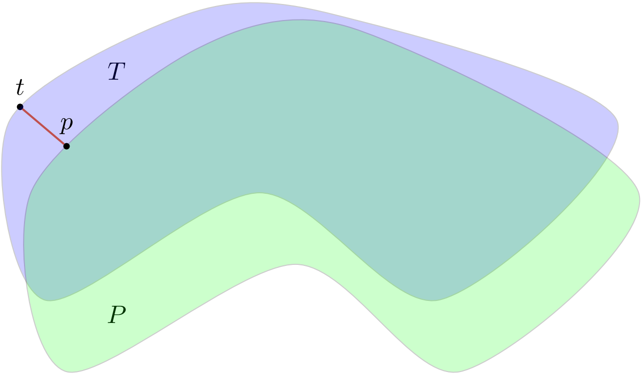


Figure 41: [Best viewed in color] Example of two possible segmentations in two dimensions, where T is the real tumor and P is our predicted version. The directed Hausdorff distance $\vec{d}_H(T,P)$ is indicated by the red line.

where $d(t,p)$ is the Euclidean distance between point t of the true segmentation and p of the predicted one. An example representation of this distance can be found in Figure 41.

The Hausdorff distance can also be undirected, and in such case the maximum of the two possible directed distances is taken:

$$d_H(T,P) = d_H(P,T) = \max(\vec{d}_H(T,P), \vec{d}_H(P,T)).$$

There are many variations to the Hausdorff distance [12]. The one used by the BraTS challenge is the undirected percent Hausdorff distance. The directed percent Hausdorff distance does not take the point t with the maximum distance, but instead takes the point whose distance is greater or equal than 95% of the distances of the other points in T (i.e. it considers the 95th percentile). The reason behind this is that a single point might by an outlier that has a significantly larger distance than usual. Let r be any percentage and let P_r be the function returning the r-th percentile of a multiset, then the directed r% Hausdorff distance is defined as:

$$\vec{d}_{H,r}(T,P) = P_r\left[\min_{p\in P} d(t,p) \mid t \in T\right]$$

The undirected percent Hausdorff distance is computed as the average of the two directed distances:

$$d_{H,r}(T,P) = d_{H,r}(P,T) = \frac{\vec{d}_{H,r}(T,P) + \vec{d}_{H,r}(P,T)}{2}.$$

8.1.1 Ranking

Due to the quantity of evaluation measures, which additionally are computed on different regions, it is sometimes complex to establish which approach performs best. To obviate this problem, we apply the same ranking scheme used by BraTS [9], which has been proven successful for other challenges [28, 42]. The ranking is based on two measures: the Dice score and the undirected 95% Hausdorff distance. The Hausdorff distance is bounded below (zero, the best value) but not bounded above, and thus it is converted to the bounded range $[0,1]$ by scaling and inverting the value to ensure comparability between the metrics.

We first compute the average over the two measures and the three regions for each patient and approach. Afterwards, we average over the patients to compute the final ranking.

8.2 Implementation Details and Experimental Setup

In this section, we discuss the data sets used for training and testing and we describe the more technical details of our implementation.

8.2.1 Data Sets

The original BraTS 19 data set is composed of 334 patients with tumor segmentation ground truth and 125 patients without ground truth. All patients have four MRI scans: T1 Weighted, T2 Weighted, T1 Contrast Enhanced, T2 FLAIR.
We split the BraTS data set with ground truth into 300 training patients and 34 validation patients. We choose the data set without ground truth to be our test set. All sets contain patients with high and low grade gliomas. A portion of 14 patients from our test set has been manually segmented by us, only for patients with high grade gliomas.

To summarize, we have the following data sets:

- `train`: 300 patients from the BraTS 19 data set, with tumor ground truth from BraTS.
- `val`: 34 patients from the BraTS 19 data set, with tumor ground truth from BraTS.
- `testUK`: 14 patients from the BraTS 19 data set, with tumor ground truth from the University Hospital of Cologne. The segmentation performed by the doctor has been converted to the BraTS 19 labels.
- `testNGT`: 111 patients from the BraTS 19 data set without tumor ground truth.

8.2.2 BrainClustering

The BrainClustering transformation functions are written in C++ 17 and ported to python with Boost.Python[24]. The main handling of the program, such as data collection, plotting and general operations are performed in python. However, the main algorithms are implemented in C++ using the specifications described in Section 6.
The color maps are implemented with the **`std::map`** data structure, which has logarithmic complexity for insertion and search. They are implemented as red black trees, which are binary search trees with the additional self-organizing property that their depth is $\mathcal{O}(\log n)$ at all times. The maps are saved with Boost.Serialization[25], which performs a reversible deconstruction of C++ data structures.

We use BrainClustering both in Train&Test and in search mode. In Train&Test mode the tables are built by training on 300 patient images (`train`). For all approaches we use patient 1 from BraTS 13 as the reference image whenever we have three main clusters, and we use a clustering of the first training image for any other number of clusters.

8.2.3 Pix2Pix

We extend the original Pix2Pix framework[26], which is implemented in PyTorch[27], a popular machine learning framework for python. However, there exist implementations using other frameworks, which produce different results: Tensorflow[28,29], Chainer[30] and Keras[31].
Thanks to the many useful operations already present in PyTorch, it is rather easy to modify the networks to take three-dimensional instead of two-dimensional inputs. However, we modified the

[24] https://www.boost.org/doc/libs/1_66_0/libs/python/doc/html/index.html
[25] https://www.boost.org/doc/libs/1_72_0/libs/serialization/doc/index.html
[26] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix
[27] https://pytorch.org/
[28] https://github.com/affinelayer/pix2pix-tensorflow
[29] https://github.com/yenchenlin
[30] https://github.com/pfnet-research
[31] https://github.com/tdeboissiere/DeepLearningImplementations/tree/master/pix2pix

data augmentation with an additional library, TorchIO [35], which implements useful preprocessing and data augmentation routines for medical imaging.

The default generator architecture for Pix2Pix is U-Net256. According to our preliminary tests on a two-dimensional slice of a brain, the results produced lower MSEs with this default architecture instead of a U-Net128 network. Thus, our first approach was to adopt this same network for three-dimensional data, which requires an input three-dimensional image with side length that is a multiple of 256. However, this amount of data could not be processed by our initial GPU setting, a 8 GB NVIDIA GeForce GTX 1070, because of the large amount of memory needed. Thus, the chosen architecture for the generator is U-Net128. During the training phase, the images are preprocessed to have side length 128 by performing random crops, while during the testing phase they are padded to have side length 256.

The network is trained on 300 training patients (`train`) and 200 epochs. All other network parameters are the default ones given by the authors of the paper.

8.2.4 DeepMedic

DeepMedic has been trained on one single GPU with 300 training patients (`train`) from the BraTS 19 data set, each with six NIfTI files: the four sequences, the true segmentation of the tumor, and the brain mask, which is a three-dimensional image that marks the brain voxels. Additionally, DeepMedic also takes as input a set of validation data (`val`) to further optimize the parameters. The network has been trained on ground truth coming exclusively from BraTS 19.

8.2.5 HD-GLIO

HD-GLIO has a pretrained model and thus requires the data to have a certain shape and positioning: The images have to be skull-stripped, co-registered and oriented to the standard template images (MNI152). In the BraTS images the first two steps are already performed, but for completeness we repeated the registration using `flirt`[32]. Before the registration, we also reoriented the images with `fslreorient2std`[33]. Since HD-GLIO has not been trained on our data, the tumor segmentation labels are in a different order and it is necessary to map them to the BraTS ordering before performing any evaluation.

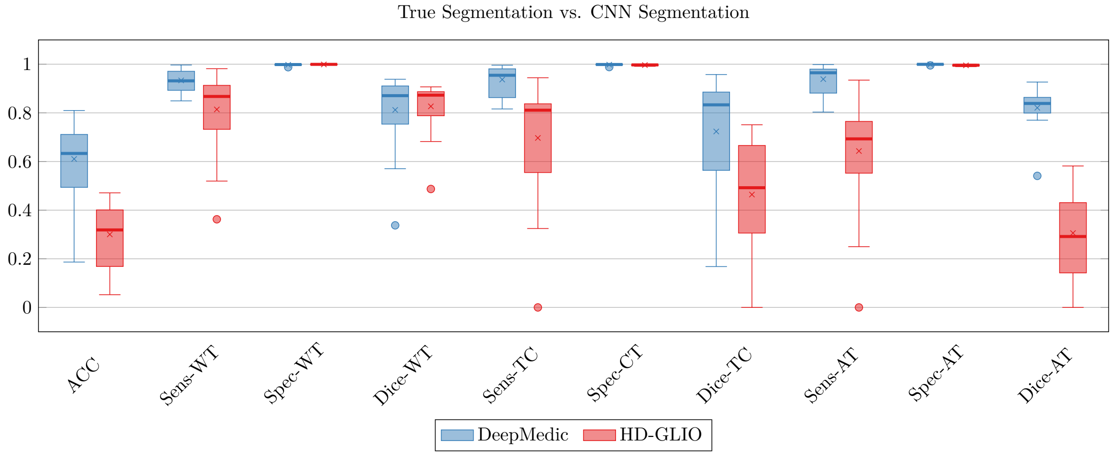
8.2.6 Experimental Setup

We use a workstation that was kindly provided by the chair of Computational Analytics[34] of the University of Bonn. The workstation has an AMD Ryzen 3960X 24-core processor[35] with 2.2 GHz clock speed and 128 GB of RAM and is equipped with two 11 GB NVIDIA GeForce RTX 2080Ti GPUs[36] with CUDA 11. The operating system is Ubuntu 20.04 Desktop.

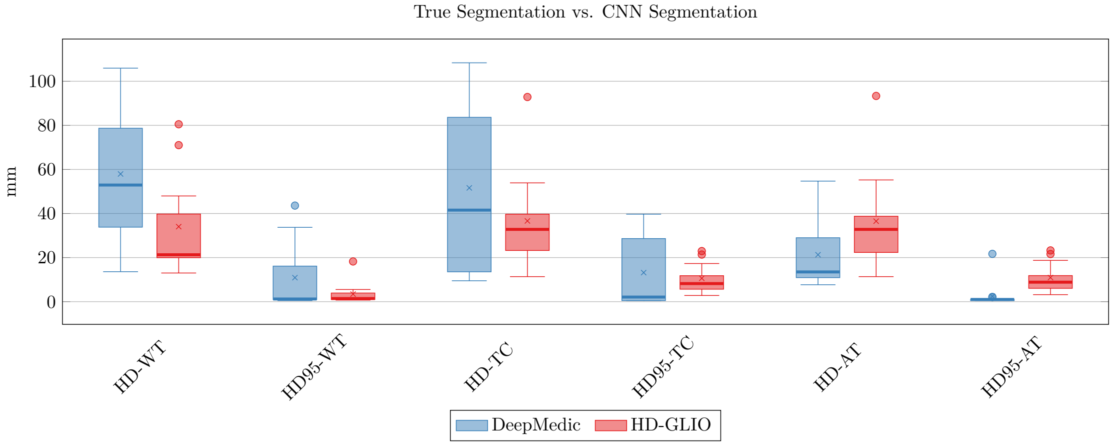
8.3 Tumor Segmentation with Original Images

In this section, we compare the performance of DeepMedic and HD-GLIO. These CNNs are tumor segmentation tools that, given T1 Weighted, T2 Weighted, T1 Contrast Enhanced and T2 FLAIR are able to predict the location of the tumor. This comparison is performed on a small set of patients (14 patients) from the test data set for which we have the ground truth (`testUK`). For this evaluation we can neither use the training nor the validation data, since they have been used to train DeepMedic.

[32] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FLIRT
[33] https://fsl.fmrib.ox.ac.uk/fsl/
[34] https://ca.cs.uni-bonn.de/doku.php?id=start
[35] https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3960x
[36] https://www.nvidia.com/de-de/geforce/graphics-cards/rtx-2080-ti/



(a) [Best viewed in color] The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor).



(b) [Best viewed in color] The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor).

Figure 42: Comparison between DeepMedic and HD-GLIO with respect to the true tumor segmentation. The scores represent the distribution of 14 patients from `testUK`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

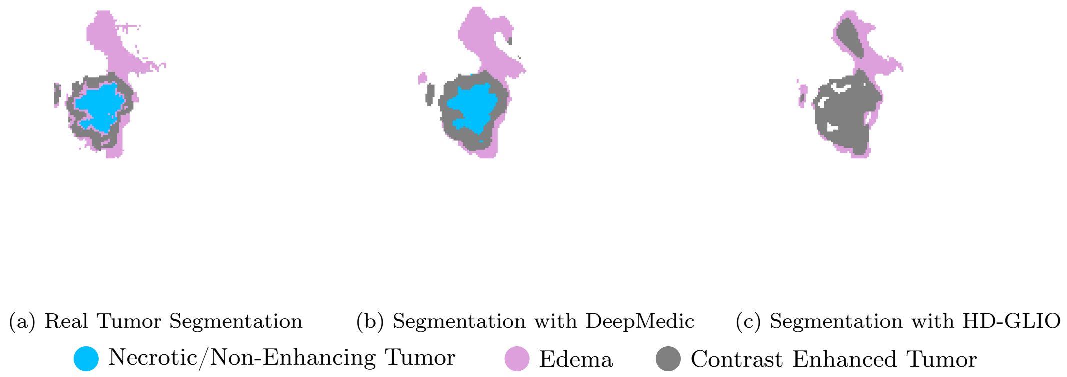

(a) Real Tumor Segmentation (b) Segmentation with DeepMedic (c) Segmentation with HD-GLIO

FIGURE 43: [Best viewed in color] Comparison between the real tumor segmentation and the ones produced by DeepMedic and HD-GLIO for patient CBICA BHN (`testUK`).

The tumor files of BraTS 19 indicate three different parts of the tumor: Necrotic/Non-Enhancing Tumor, Edema and Contrast Enhanced Tumor, previously described in Section 8.3, and represented with an example in Figure 43a. In Figure 43 we show an example of a real segmentation compared to the ones produced by DeepMedic and HD-GLIO. As it is possible to see from this example, HD-GLIO is not able to detect the Necrotic/Non-Enhancing Tumor class according to our tests.
Additionally, in Figure 42a and Figure 42b we analyze the performance of DeepMedic and HD-GLIO with respect to the truth by computing our evaluation measures. These results are not surprising: Since the training and the testing data set of DeepMedic come from the same source, they present more similarity and we expect it to have an advantage with respect to HD-GLIO. Both methods have high specificity on all three regions, which is a minimum requirement for good tumor segmentation software. However, DeepMedic has higher accuracy and sensitivity. The Dice score is better for DeepMedic in the Active Tumor and Tumor Core regions, and both approaches perform well in the Whole Tumor region. With respect to the Hausdorff distances HD-GLIO has an advantage, but not in the Active Tumor because HD-GLIO does not detect Necrotic/Non-Enhancing Tumor class, but the entire contrast enhanced area, which causes its distances to be higher.

8.4 Choice of Main and Sub Clusters

In this section, we empirically choose the number of main clusters and the number of sub clusters that we use to evaluate BrainClustering. To choose the most fitting values we first have a small evaluation of the mean squared errors of BrainClustering k-means with different values for our parameters, and then we choose a number of sub clusters and compare the results obtained with the segmentations. To avoid training many different models on the full training data, we run the evaluation in search mode by training on the five images that have the smallest mean squared error with respect to T1 Weighted (see Section 6.5).
The data set chosen for the first part of this experiment is composed by the patients for which we have the ground truth, but that have not been used for training. Thus, this evaluation is run on `val` and `testUK`. We present the mean squared errors obtained in both data sets in a single graph.

We choose three, four, five and six number of main clusters and 100, 200 and 500 number of sub clusters. Note that because of the way Leveled k-means is implemented, the number of sub

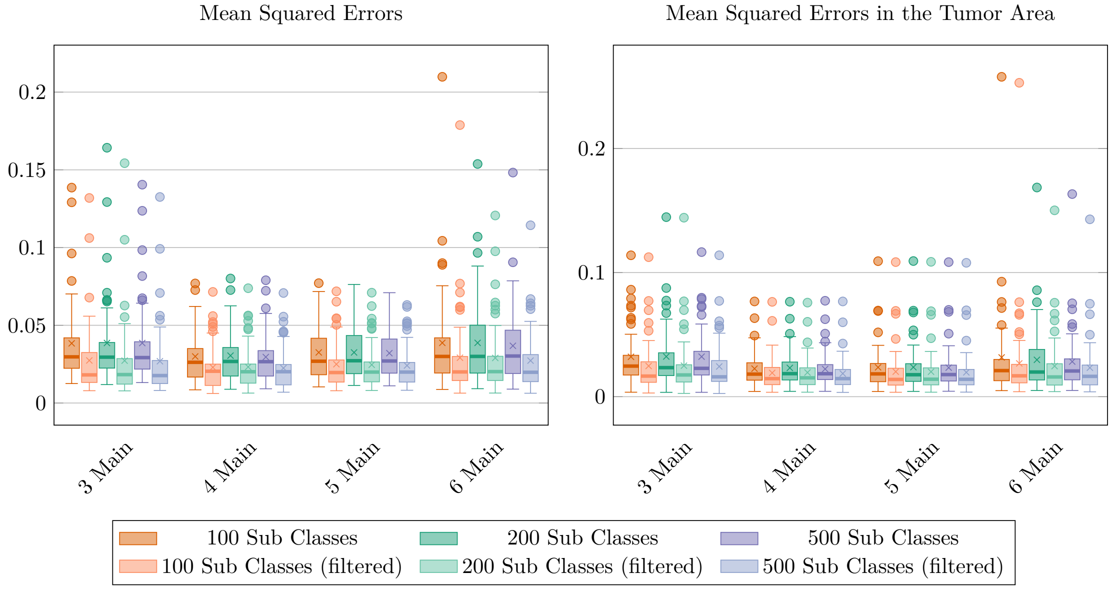


Figure 44: [Best viewed in color] Comparison between different numbers of main clusters and sub clusters for BrainClustering k-means in search mode using mean squared errors in the brain mask area and in the tumor area. The scores represent the distribution of 48 patients, 14 from `testUK` and 34 from `val`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

clusters may vary. Since the number of sub clusters needs to be divisible by the number of main clusters, for each number of sub clusters we take the next number that is divisible. For example, for three number of main clusters 100 is not valid, and thus we take 102.

In Figure 44 are the results of this experiment. The number of main clusters for which we achieve the best mean squared error in the brain area (not considering the background) and in the tumor area (only for the points belonging to the tumor) is four clusters. We expect higher values for any number of clusters greater than six, and we believe it would not be meaningful to run these tests with less than three clusters. In all cases, applying the median filter improves the result.

The number of sub clusters does not particularly influence the scores if the number of main clusters is less than six. However, the number of sub clusters makes a difference in the computation time. In Table 2 we have the average computation time for each pair of main clusters and sub clusters. Considering the same number of sub clusters, whenever we use a larger number of main clusters, the computation decreases. This is because Leveled k-means divides the number of sub clusters over the number of main clusters, so it is indeed faster to compute this kind of clustering with

Table 2: Average number of minutes spent running BrainClustering k-means in search mode (with five images) for different numbers of main and sub clusters.

Main Clusters	Average Runtime with 100 Sub Clusters	Average Runtime with 200 Sub Clusters	Average Runtime with 500 Sub Clusters
3	10.28 minutes	13.58 minutes	23.32 minutes
4	9.31 minutes	11.74 minutes	18.85 minutes
5	9.01 minutes	10.87 minutes	16.45 minutes
6	8.81 minutes	10.35 minutes	15.09 minutes

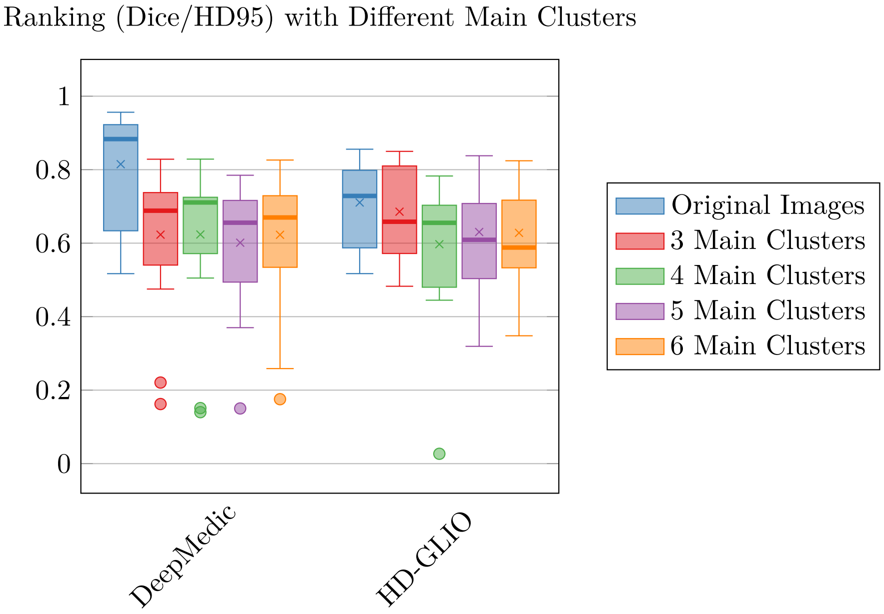
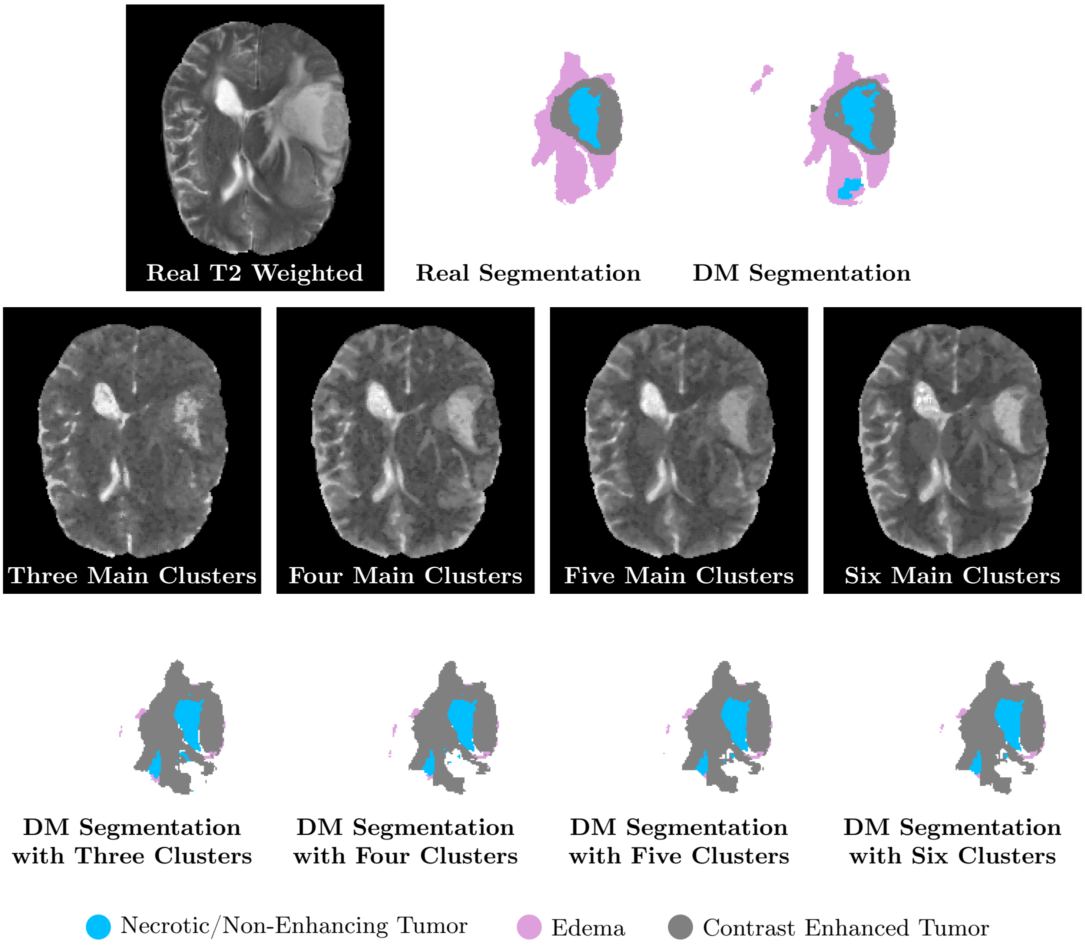


Figure 45: [Best viewed in color] Comparison of BrainClustering k-means (search) with different numbers of main clusters with respect to the true tumor segmentation. The used metrics are Dice score and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 14 patients from `testUK`. For all measures a higher value indicates a better rank. The values are computed from Figure 66 and Figure 67. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

smaller partitions. In most cases we obtain better results with 500 sub clusters. However, the time consumption doubles, so we choose 100 sub clusters to be the value used for our next experiments.

To motivate even further our choice for the main clusters, we build one data set for each value of main clusters composed by the original T1 Weighted, T1 Contrast Enhanced and T2 FLAIR and our generated T2 Weighted for that value and use it as input for DeepMedic and HD-GLIO. Note that, for the same reason as before, we do not use the images from the validation data set, because otherwise DeepMedic would have an unfair advantage. The number of main clusters that performs best is then chosen for the final evaluation with BrainClustering k-means, Nesting and agglomerative.
In Figure 46 we show the predicted output for T2 Weighted of patient TCIA01 401 of BraTS 19 for different numbers of clusters. The segmentations produced with our transformed images are not able to detect the edema class, which is an expected result because this tissue is commonly segmented from T2 Weighted.
In Figure 45 we present the computed ranking for the original segmentation (done with the original T1 Weighted, T2 Weighted, T1 Contrast Enhanced and T2 FLAIR) and the ones executed by swapping one image with the one produced with a certain number of clusters. Additional evaluation metrics are shown in Section B.1. As expected, the segmentation with the original images ranks first for both DeepMedic and HD-GLIO, but HD-GLIO is more robust to changes, and the differences between the segmentation produced with the original images and the ones produced with BrainClustering is smaller.



Real T2 Weighted

Real Segmentation

DM Segmentation

Three Main Clusters

Four Main Clusters

Five Main Clusters

Six Main Clusters

DM Segmentation with Three Clusters

DM Segmentation with Four Clusters

DM Segmentation with Five Clusters

DM Segmentation with Six Clusters

Figure 46: [Best viewed in color] Learned T2 Weighted for different numbers of main clusters for patient TCIA01 401 (`val`) and with the respective tumor segmentations created with DeepMedic.

Overall, DeepMedic produces segmentations of higher quality with the original images, but the segmentations produced with one transformed image yield similar rankings in DeepMedic and HD-GLIO (except three main clusters for HD-GLIO). Different numbers of main clusters are better in each method: For DeepMedic, four clusters performs best, while for HD-GLIO three is the best value. Thus, we will run the next evaluations with four main clusters for DeepMedic and three main clusters for HD-GLIO.

8.5 Transformed T2 Weighted Evaluation

In this section, we evaluate the quality of the T2 Weighted produced with different approaches. Our main objective is to investigate whether the transformed images produced by our approaches are able to achieve effective results in tumor segmentation. To do so, we run the segmentation software with the original T1 Weighted, T1 Contrast Enhanced, T2 FLAIR and we substitute the original T2 Weighted image with a generated image.
In the following, a list of all the methods that are compared in our benchmarks:

- **Original**: The original T2 Weighted is used.
- **Complementary**: T2 Weighted is created by complementing T1 Weighted.
- **Pix2Pix**: T2 Weighted is produced by using the corresponding T1 Weighted as input for the cGAN.
- **BC *k*-means**: BrainClustering is run in Train&Test mode, clustering 300 patients with Leveled *k*-means. T2 Weighted is produced by querying the tables with the corresponding T1 Weighted.
- **BC Nesting**: BrainClustering is run in Train&Test mode, clustering 300 patients with Leveled Nesting. T2 Weighted is produced by querying the tables with the corresponding T1 Weighted.
- **BC-S *k*-means**: BrainClustering is run in search mode, clustering the five patients with the closest MSE with Leveled *k*-means. T2 Weighted is produced by querying the tables with the query image.
- **BC-S Nesting**: BrainClustering is run in search mode, clustering the five patients with the closest MSE with Leveled Nesting. T2 Weighted is produced by querying the tables with the query image.
- **BC-S Agglomerative**: BrainClustering is run in search mode, clustering one patient with the closest MSE with Agglomerative. T2 Weighted is produced by querying the tables with the query image.

We do not run Agglomerative in Train&Test mode, but only in search mode because it is computationally expensive and has a large memory footprint. It takes around seven hours to compute the agglomerative clustering for one single image, compared to the ten minutes that it takes to compute the *k*-means clustering for five images. Nesting is also marginally more computationally expensive than *k*-means, and it takes approximately 15 minutes to process five images.
In Table 3 we show the training and testing times for BrainClustering and Pix2Pix. The number of hours spent for training BrainClustering *k*-means and BrainClustering Nesting are computed as the average of the time taken to compute the training for three main clusters and for four main clusters. As expected, the time taken by BrainClustering Nesting is more than twice the time taken by BrainClustering *k*-means. BrainClustering *k*-means is much faster than Pix2Pix, and the time taken for testing is very similar for all methods.
For all methods we produce both a plain T2 Weighted output and T2 Weighted filtered with the median filter. Since the median filter often yields a lower MSE, we choose the filtered images for the MSE computation and as input for the segmentation.

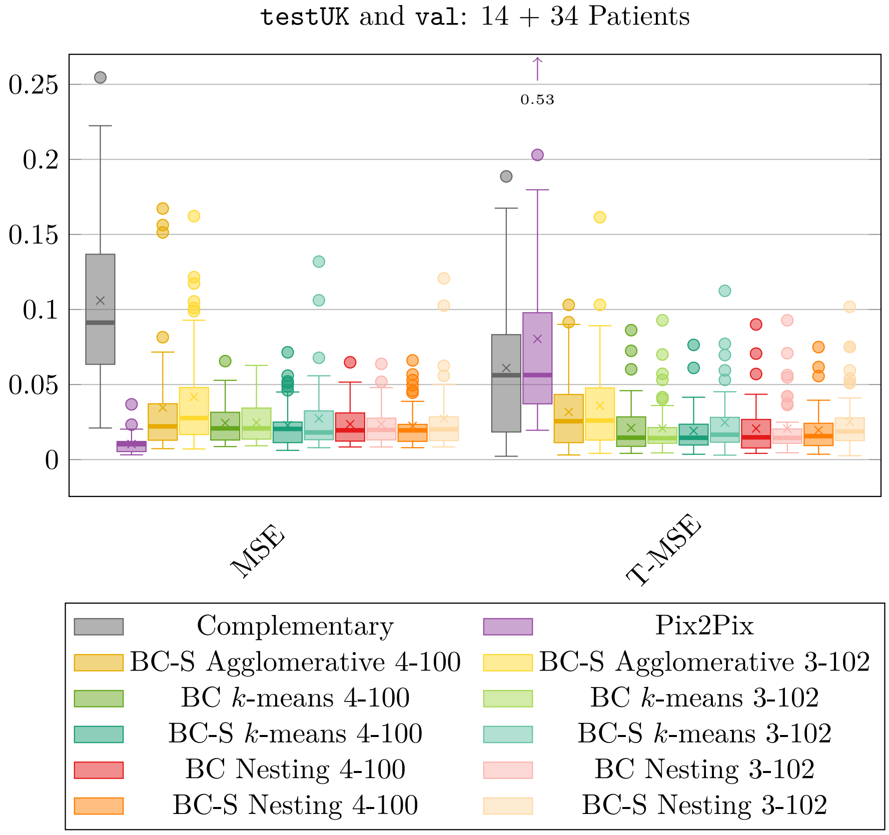


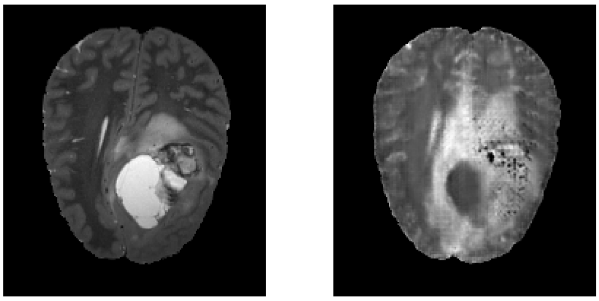
Figure 47: [Best viewed in color] Comparison of different approaches using mean squared errors in the brain mask area and in the tumor area. The scores represent the distribution of 48 patients, 14 from `testUK` and 34 from `val`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

8.5.1 Data with Ground Truth

In this section, we conduct our evaluation on data for which we have the ground truth, which is formed by the 34 patients from `val` and 14 patients from `testUK`. Since we had some discrepancies on which value of main clusters is better for DeepMedic and HD-GLIO, their input is different. In Figure 47 we present the mean squared errors for the data with ground truth. There are two values for each BrainClustering entry, because one is used for DeepMedic (four main clusters) and one HD-GLIO (three main clusters). Pix2Pix obtains the best MSE in the brain voxels, but the worst one in the tumor voxels. Figure 48 shows an example of a result with a good structure, but a poor coloring for the tumor area (white area in Figure 48a). Since the aim of the network is to produce a generalization, it is not able to capture the structure of the tumor at the current state of the project. However, it obtains the most realistic results because it aims at minimizing the absolute error between the input and the output image. Complementary has the worst MSE

Table 3: Approximate number of hours spent training and testing our training algorithms.

	Pix2Pix	BC k-means	BC Nesting
Training (300 Patients)	10.05	3.26	10.48
Testing (111 Patients)	0.32	0.29	0.28



(a) Real T2 Weighted (b) Pix2Pix T2 Weighted

Figure 48: Real T2 Weighted and synthesized T2 Weighted generated by Pix2Pix for patient CBICA ALT (testUK).

because it produces a poor coloring of the image. All the BrainClustering algorithms behave similarly, except for Agglomerative which has a worse MSE both in the brain and in the tumor voxels.
In Figure 49 and Figure 50 we show the results obtained with the different approaches for patient CBICA ARR. This patient has an average ranking for DeepMedic (0.69 out of a 0.63 average for all approaches) and a below average ranking for HD-GLIO (0.56 out of a 0.67 average for all approaches). Once more, the segmentation from HD-GLIO does not detect the Necrotic/Non-Enhancing Tumor class. In the resulting segmentations DeepMedic is not able to detect the edema, and in this particular case everything is classified as Contrast Enhanced Tumor. On the other hand HD-GLIO does find the edema and in some approaches it produces a result that is very similar to the one with four original images.
Figure 51a and Figure 51b show the final ranking of the chosen approaches for the two different data sets. For completeness, we present the detailed results of the individual scores in Section B.2. DeepMedic should perform better on the validation data set, since its parameters have been optimized with it. In the testUK data set the best results are obtained with Complementary for DeepMedic and with BrainClustering k-means in search mode for HD-GLIO. HD-GLIO is more robust, and the introduction of a transformed MRI does not decrease the quality significantly. It performs worst with Pix2Pix and BrainClustering Nesting in search mode. On the other hand, DeepMedic has a large difference between the segmentation with the original images and the other approaches. Overall, Agglomerative does not perform well, while the other BrainClustering approaches are comparable. In the validation data set we also have similar results.

To conclude, k-means is the best strategy of BrainClustering: It is fast, and produces the best results. However, Complementary produces equally good results, with the additional advantage of being simple and producing sharp images. Pix2Pix produces very realistic images, which however are not able to outperform the Complementary approach.

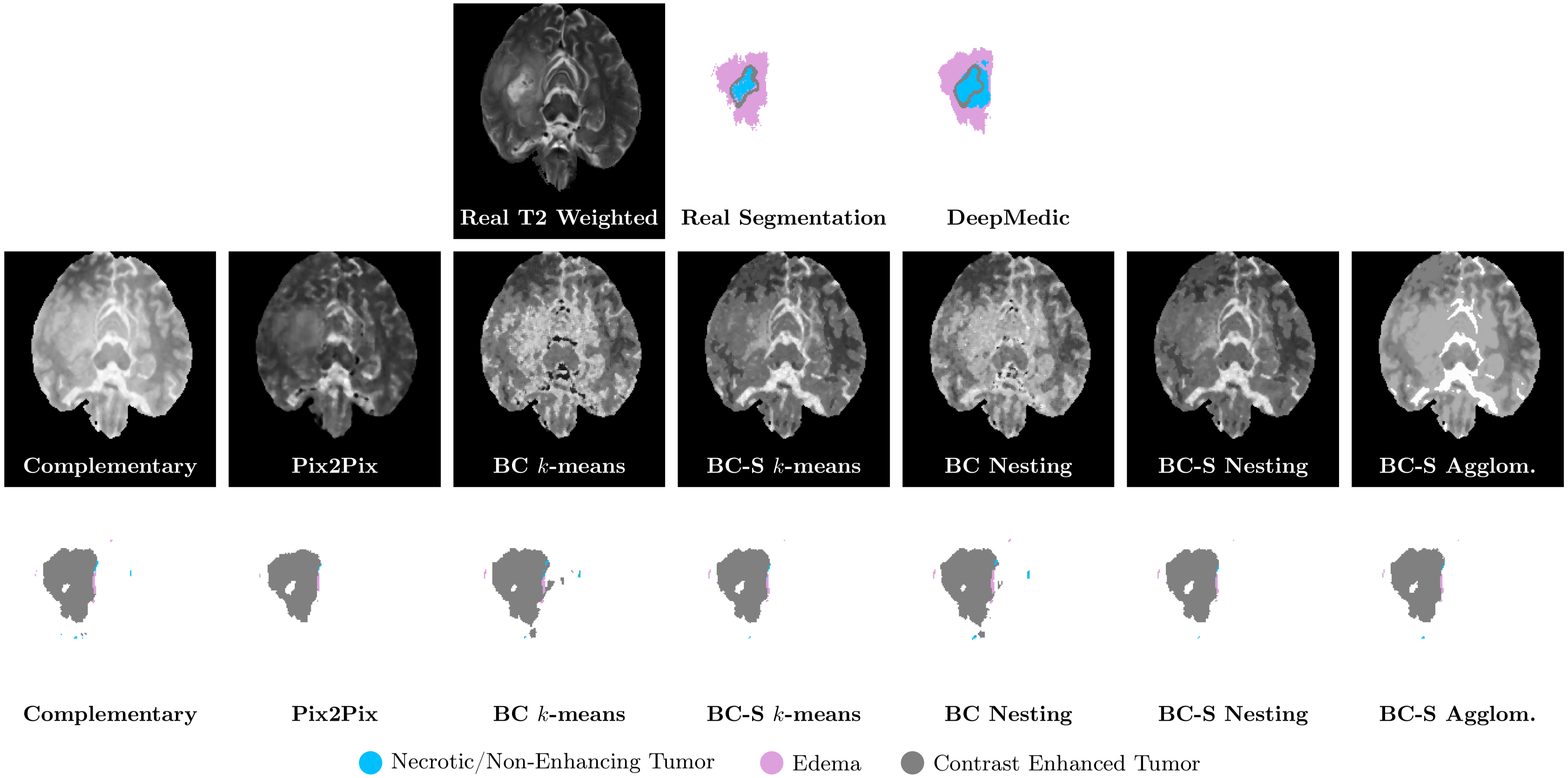


FIGURE 49: [Best viewed in color] Learned T2 Weighted and respective DeepMedic tumor segmentation of patient CBICA ARR (`testUK`) for different approaches.

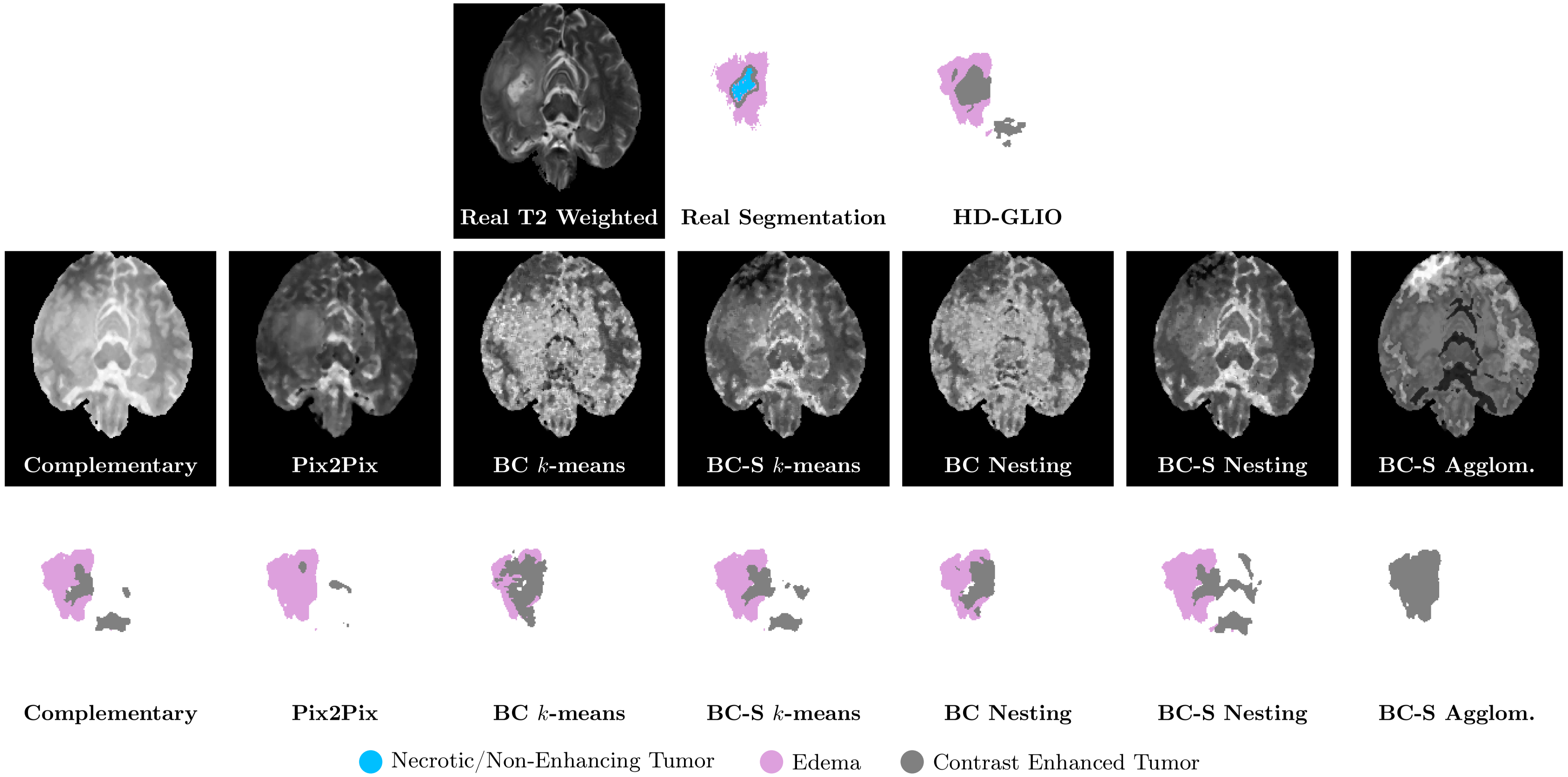
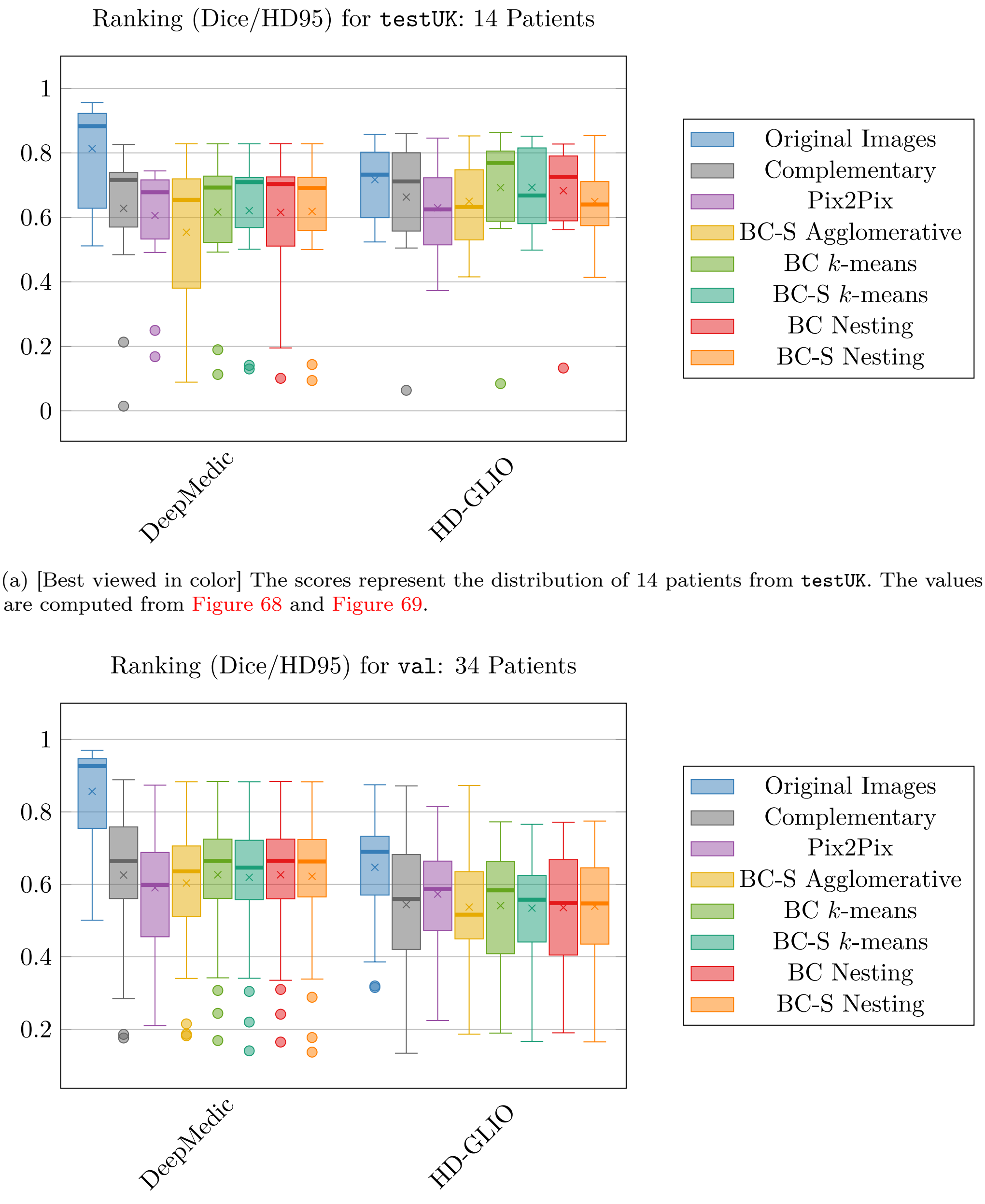


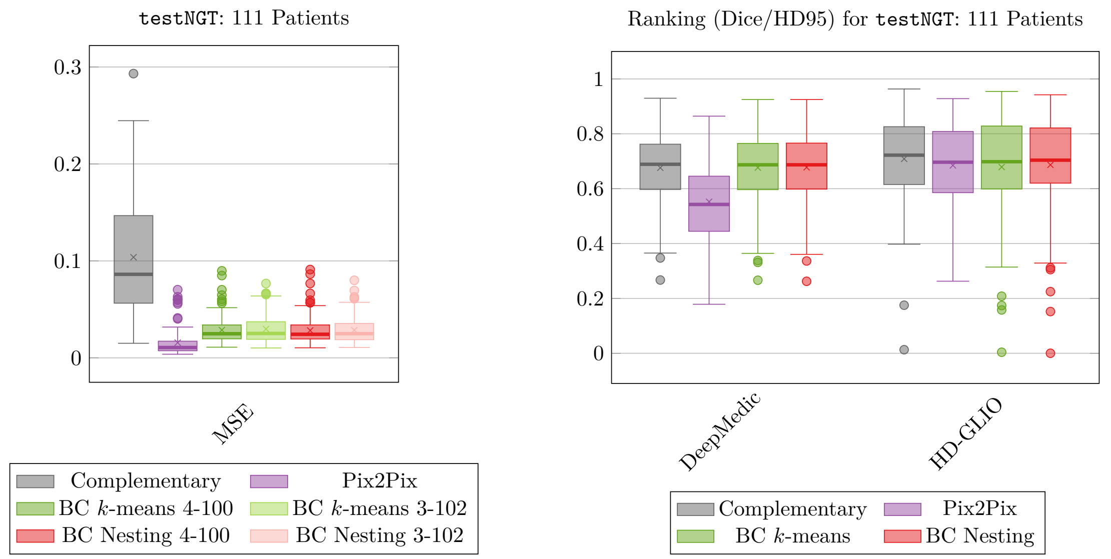
FIGURE 50: [Best viewed in color] Learned T2 Weighted and respective HD-GLIO tumor segmentation of patient CBICA ARR (`testUK`) for different approaches.



(a) [Best viewed in color] The scores represent the distribution of 14 patients from `testUK`. The values are computed from Figure 68 and Figure 69.

(b) [Best viewed in color] The scores represent the distribution of 34 patients from `val`. The values are computed from Figure 70 and Figure 71.

Figure 51: Comparison of different approaches with respect to the true tumor segmentation using Dice score and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor). For all measures a higher value indicates a better rank. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.



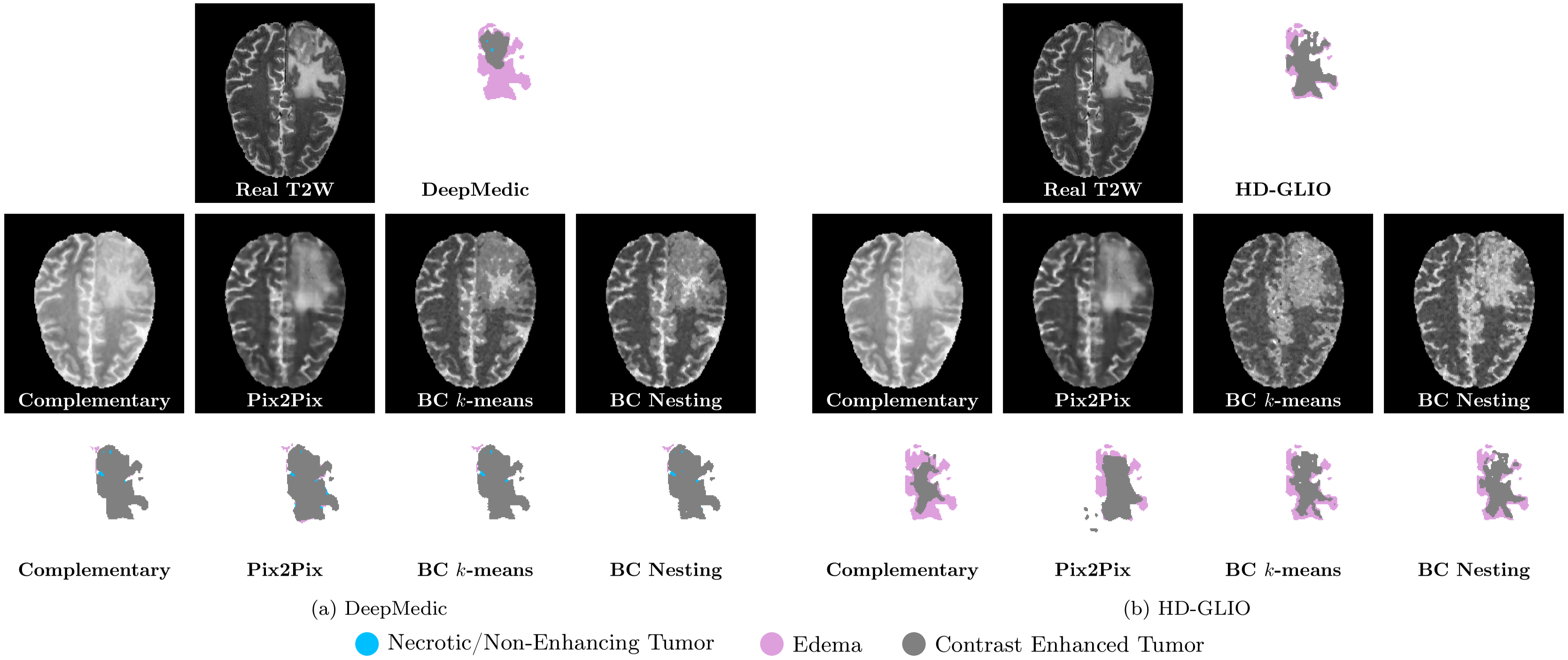
(a) Comparison of different approaches using mean squared errors in the brain mask area.

(b) Comparison of different approaches with respect to the segmentation with four original MRI scans using Dice score and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor).

Figure 52: [Best viewed in color] The scores represent the distribution of 111 patients from `testNGT`. For the values of Figure 52a a lower value indicates a better score, while it is the opposite for the rankings in Figure 52b. The values of the latter are computed from Figure 72 and Figure 73. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

8.5.2 Data without Ground Truth

Additionally, we also have a set of 111 patients for which we have no ground truth. Thus, we compare the results with respect to the original segmentation instead of comparing to the ground truth. In this case we only present the evaluation for Complementary, Pix2Pix, BrainClustering k-means and Nesting. We display the MSE values in Figure 52a, the full evaluation metrics in Section B.2 and the rankings in Figure 52b. The resulting MSE values match the performance of the previous data sets, so Pix2Pix is indeed the best at reproducing the images in terms of error. In Figure 53a and Figure 53b we show the result for the example patient CBICA AAM. This patient has an average ranking for DeepMedic (0.64 out of a 0.65 average for all approaches) and an above average ranking for HD-GLIO (0.83 out of a 0.69 average for all approaches). In the DeepMedic segmentation the BrainClustering methods are able to reproduce the segmentation of the original images better, while in HD-GLIO all methods perform similarly, but with Complementary being the best performing one. Also, HD-GLIO has higher ranks, meaning that the approaches with a substituted image have a similar segmentation to the one created with the original images, which is consistent with the results we obtained in Figure 51a and in Figure 51b.



(a) DeepMedic

(b) HD-GLIO

● Necrotic/Non-Enhancing Tumor ● Edema ● Contrast Enhanced Tumor

FIGURE 53: [Best viewed in color] Learned T2 Weighted and respective tumor segmentation of patient CBICA AAM (`testNGT`) for different approaches.

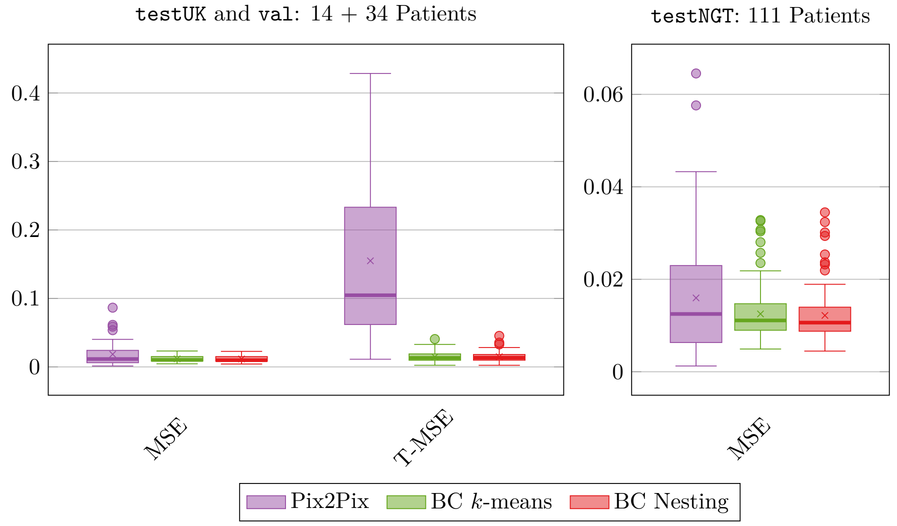


FIGURE 54: [Best viewed in color] Comparison of different approaches using mean squared errors in the brain mask area. The scores represent the distribution of 48 patients (14 from `testUK` and 34 from `val`) and 111 patients (`testNGT`). A lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

8.6 Additional Mappings: Transformed T1 Contrast Enhanced Evaluation

In this section, we present a small, preliminary evaluation of the T1 Contrast Enhanced images that can be created from the T1 Weighted images. In the previous evaluations we obtained the best results with DeepMedic, and thus we only use this segmentation software for this evaluation. For simplicity, we only use BrainClustering methods that use Train&Test, and do not consider the search mode approaches. Note that theoretically one should perform the initial main cluster evaluation again for the T1 Contrast Enhanced sequence, but since the nature of this evaluation is only preliminary, we simply use four main clusters. In this case, naive approaches like Complementary cannot be used because there is no obvious signal relationship between T1 Contrast Enhanced and T1 Weighted.

Figure 54 shows the mean squared errors for all data sets. The results of BrainClustering k-means and BrainClustering Nesting are similar, while Pix2Pix performs worse than it did with T2 Weighted. In general, all methods have a lower performance, as can be seen in Figure 56a and in Figure 56b. Patient TCIA10 266 (Figure 56a) has an average ranking (0.70 out of a 0.65 average for all approaches), while patient TCIA02 457 (Figure 56b) has a below average ranking (0.47 out of a 0.60 average for all approaches). As we noticed before, the images produced by Pix2Pix present more structure and are more realistic, but in this case they have more black areas, which are contoured by lighter areas. Unfortunately, the enhanced rim is what distinguishes a tumor area in T1 Contrast Enhanced, and thus these areas are recognized as tumors by DeepMedic. The images produced by BrainClustering do not have a lot of structure and look noisy. However, the tumor segmentations produced with them is better than the ones produced with Pix2Pix.
In Figure 55 we show the rankings for `testUK`, `val` and `testNGT`, while the full evaluation results are provided in Section B.3. Pix2Pix obtains better results in the `testUK` data set, while the BrainClustering algorithms are better in the other sets. The results produced with BrainClustering k-means are the best among the BrainClustering methods. However, the general performance is worse than for the T2 Weighted images, which shows that this transformation is harder to capture.

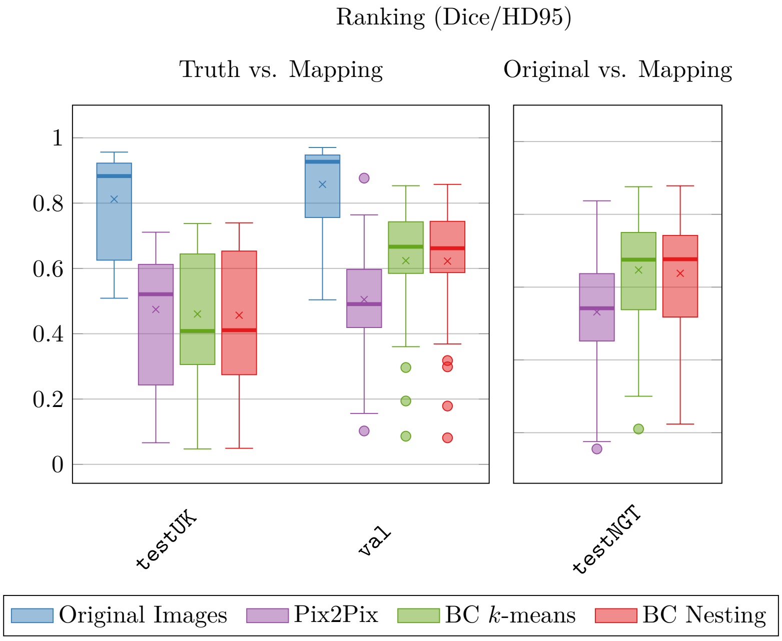
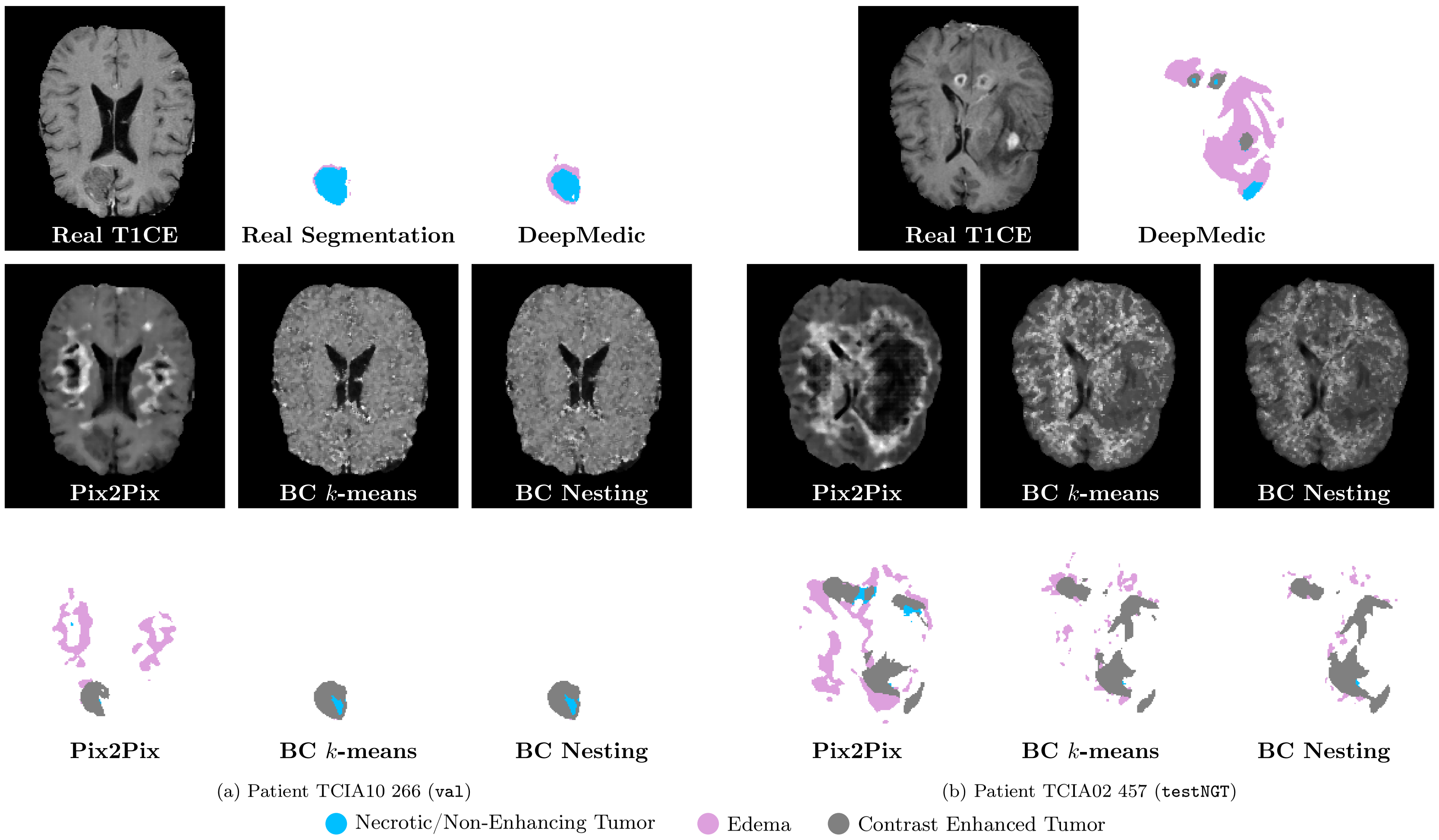


FIGURE 55: [Best viewed in color] Comparison of different approaches with respect to the segmentation with four original MRI scans using Dice score and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 14 patients (testUK), 34 patients (val) and 111 patients (testNGT). For all measures a higher value indicates a better rank. The values are computed from Figure 74a and Figure 74b for testUK, Figure 75a and Figure 75b for val and Figure 76a and Figure 76b for testNGT. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.



(a) Patient TCIA10 266 (`val`)

(b) Patient TCIA02 457 (`testNGT`)

FIGURE 56: [Best viewed in color] Learned T2 Weighted and respective DeepMedic tumor segmentation for different approaches.

9 Conclusion and Future Work

In this project, we proposed and evaluated two solutions for the MRI translation problem. None of them is able to produce satisfactory results for all use cases and with every tumor segmentation software. However, with a more targeted optimization of the parameters with respect to the used segmentation software and the needed sequences, it is possible to obtain satisfactory results that can correctly identify the tumor. In most cases, the segmentations produced with all the approaches were able to correctly pinpoint the location of the tumor, as shown by the high specificity. However, the tumor classes produced by DeepMedic with a transformed image did not have the same quality as the ones produced with four original MRI scans. HD-GLIO is more robust, and thus many times the same quality could be achieved. Surprisingly, a simple method like taking the complement of T1 Weighted in order to produce T2 Weighted results in good segmentations, even if the obtained sequence is very different from the actual T2 Weighted image. Pix2Pix produces realistic results with high structure but with poor colorings in the tumor area, which causes the segmentation to be worse than the other methods. BrainClustering generates less realistic results, often very noisy and with a poor local structure, but that can reproduce the general anatomy of the brain and of the tumors. The best results of BrainClustering can be achieved with BrainClustering k-means, which is also able to produce results in a very short amount of time. By contrast, the worst results are obtained with BrainClustering Agglomerative, which is also the slowest method.

T2 Weighted Transformation. Pix2Pix obtains the worst or second-to-worst rankings for the segmentations with DeepMedic and the worst or second-to-worst rankings in HD-GLIO for `testUK` and `testNGT`, but the best rank for `val`. In the DeepMedic ground truth rankings most methods achieve very similar results, except for BrainClustering Agglomerative, which performs very poorly. The segmentations from HD-GLIO produce more distributed rankings. Complementary always reaches high ranks.

T1 Contrast Enhanced Transformation. Since our BrainClustering methods are not specifically implemented for this task, we achieve lower quality results for this transformation. Moreover, even Pix2Pix struggles at reproducing this mapping, especially because it creates too light areas that are mistaken for tumors by the segmentation software. In general, we achieve lower scores than for T2 Weighted in the `testUK` data set, while the BrainClustering scores for `val` are similar to the ones obtained for T2 Weighted. The scores obtained by comparing the original segmentation of the `testNGT` data set are also similar to the ones obtained for T2 Weighted.

9.1 Future Work

The evaluation revealed the strengths and the weaknesses of our approaches. The fact that the results produced with Complementary obtain good segmentations can hint that a clear and distinct structure is sometimes more important than a perfect coloring of the image. Thus, an improvement could be to pass our resulting images in some resolution-enhancing software that reduces the present noise.

Moreover, we could investigate whether removing the reference image mapping from BrainClustering can produce better results. The BrainClustering mapping method accepts any clustering algorithm for data of any dimension and any number of clusters. However, we empirically showed that one-dimensional clustering can produce satisfactory results. Particularly, the hierarchical algorithms that use optimal solutions (Leveled k-means and Leveled Nesting) produce main clusters that are always sorted in the same order: The first cluster is always the one containing the smallest values, the second cluster contains larger values, and so on. This entails that the tissues with lower signals are always in the first cluster, and tissues with the highest signal are always in the last cluster. However, sometimes images contain points with too high signals, that can disrupt this logic and create clusterings that are not compatible over the full set of images (as in Figure 29). We have solved this problem using a reference image, but we think it would be beneficial to remove this mapping from our implementation, since it is not always optimal. This could be achieved by removing high signals from the brain scans, or by applying a better normalization to our data. An additional (and more practical) improvement to BrainClustering would be to parallelize the training, since the clusterings of different brain scans can easily be computed at the same time. This would significantly decrease the training time.

One way to improve Pix2Pix would be to perform an evaluation and optimization over its parameters for our specific use case. Moreover, since Pix2Pix is trained to minimize the absolute loss with respect to the target image, we could change this objective function such that it minimizes the loss of the tumor area, which would likely yield better results.

Finally, it would be interesting to evaluate all the possible mappings between MRI images and even run segmentations using two transformed images and two original scans to fully understand the potential of this research.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber. "Geometric Applications of a Matrix Searching Algorithm." In: *Proceedings of the Second Annual Symposium on Computational Geometry*. SCG '86. Association for Computing Machinery, 1986, pp. 285–292. DOI: 10.1145/10515.10546.

[2] S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. "Better Guarantees for k-Means and Euclidean k-Median by Primal-Dual Algorithms." In: *SIAM Journal on Computing* 49.4 (2016). DOI: 10.1137/18M1171321.

[3] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. "NP-hardness of Euclidean sum-of-squares clustering." In: *Machine Learning* 75 (2009), pp. 245–248. DOI: 10.1007/s10994-009-5103-0.

[4] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973. ISBN: 9781483191393.

[5] D. Arthur and S. Vassilvitskii. "K-Means++: The Advantages of Careful Seeding." In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. URL: http://dl.acm.org/doi/10.5555/1283383.1283494.

[6] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. Kirby, J. Freymann, K. Farahani, and C. Davatzikos. *Segmentation Labels and Radiomic Features for the Pre-operative Scans of the TCGA-LGG collection*. The Cancer Imaging Archive. 2017. DOI: 10.7937/K9/TCIA.2017.GJQ7R0EF.

[7] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. Kirby, J. Freymann, K. Farahani, and C. Davatzikos. *Segmentation Labels for the Pre-operative Scans of the TCGA-GBM collection*. The Cancer Imaging Archive. 2017. DOI: 10.7937/K9/TCIA.2017.KLXWJJ1Q.

[8] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. S. Kirby, J. B. Freymann, K. Farahani, and C. Davatzikos. "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features." In: *Scientific Data* 4 (2017). DOI: 10.1038/sdata.2017.117.

[9] S. Bakas et al. *Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge*. 2018. URL: https://arxiv.org/abs/1811.02629.

[10] L. Caldeira, P. Almeida, and J. Seabra. "MR Brain Tumor Segmentation using Clustering." In: *European Society for Magnetic Resonance in Medicine and Biology* (2009).

[11] H. Al-Dmour and A. Al-Ani. "MR Brain Image Segmentation Based on Unsupervised and Semi-Supervised Fuzzy Clustering Methods." In: *Proceedings of the Eighteenth International Conference on Digital Image Computing: Techniques and Applications*. DICTA '16. IEEE, 2016, pp. 1–7. DOI: 10.1109/DICTA.2016.7797066.

[12] M.-P. Dubuisson and A. K. Jain. "A modified Hausdorff distance for object matching." In: *Proceedings of 12th International Conference on Pattern Recognition*. Vol. 1. ICPR '94. IEEE, 1994, pp. 566–568. DOI: 10.1109/ICPR.1994.576361.

[13] E. B. Fowlkes and C. L. Mallows. "A Method for Comparing Two Hierarchical Clusterings." In: *Journal of the American Statistical Association* 78.383 (1983), pp. 553–569. DOI: 10.1080/01621459.1983.10478008.

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets." In: *Proceedings of the 27th International Conference on Neural Information Processing Systems*. Vol. 2. NIPS '14. MIT Press, 2014, pp. 2672–2680. URL: http://dl.acm.org/doi/10.5555/2969033.2969125.

[15] A. Grønlund, K. G. Larsen, A. Mathiasen, J. S. Nielsen, S. Schneider, and M. Song. "Fast Exact k-Means, k-Medians and Bregman Divergence Clustering in 1D." In: *CoRR* abs/1701.07204 (2017). URL: http://arxiv.org/abs/1701.07204.

[16] L. Hubert. and P. Arabie. "Comparing partitions." In: *Journal of classification* 2.1 (1985), pp. 193–218. DOI: 10.1007/BF01908075.

[17] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Vol. 37. ICML '15. JMLR.org, 2015, pp. 448–456. URL: https://dl.acm.org/doi/10.5555/3045118.3045167.

[18] F. Isensee et al. "Automated brain extraction of multi-sequence MRI using artificial neural networks." In: *Human Brain Mapping* (2019), pp. 1–13. DOI: 10.1002/hbm.24750.

[19] F. Isensee, J. Petersen, S. A. A. Kohl, P. F. Jäger, and K. H. Maier-Hein. "nnU-Net: Breaking the Spell on Successful Medical Image Segmentation." In: *CoRR* abs/1904.08128 (2019). URL: http://arxiv.org/abs/1904.08128.

[20] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks." In: *IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 5967–5976. DOI: 10.1109/CVPR.2017.632.

[21] M. Jenkinson, M. Pechaud, and S. Smith. "BET2: MR-Based Estimation of Brain, Skull and Scalp Surfaces." In: *Eleventh Annual Meeting of the Organization for Human Brain Mapping* (2005).

[22] R. Jonker and A. Volgenant. "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems." In: *Computing* 38.4 (1987), pp. 325–340. DOI: 10.1007/BF02278710.

[23] K. Kamnitsas, C. Ledig, V. F. J. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker. "Efficient Multi-Scale 3D CNN with Fully Connected CRF for Accurate Brain Lesion Segmentation." In: *Medical Image Analysis* 36 (2016). DOI: 10.1016/j.media.2016.10.004.

[24] P. Kickingereder et al. "Automated quantitative tumour response assessment of MRI in neuro-oncology with artificial neural networks: a multicentre, retrospective study." In: *Lancet Oncology* 20.5 (2019), pp. 728–740. DOI: 10.1016/S1470-2045(19)30098-1.

[25] P.-Y. Laffont, Z. Ren, X. Tao, C. Qian, and J. Hays. "Transient Attributes for High-Level Understanding and Editing of Outdoor Scenes." In: *ACM Transactions on Graphics* 33.4 (2014). URL: http://dl.acm.org/doi/10.1145/2601097.2601101.

[26] G. Lin, C. Nagarajan, R. Rajaraman, and D. P. Williamson. "A General Approach for Incremental Approximation and Hierarchical Clustering." In: *SIAM Journal on Computing* 39.8 (2010), pp. 3633–3669. DOI: 10.1137/070698257.

[27] S. Lloyd. "Least squares quantization in PCM." In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[28] O. Maier et al. "ISLES 2015 - A public evaluation benchmark for ischemic stroke lesion segmentation from multispectral MRI." In: *Medical Image Analysis* 35 (2016). DOI: 10.1016/j.media.2016.07.009.

[29] M. Malathi and P. Sinthia. "MRI Brain Tumour Segmentation Using Hybrid Clustering and Classification by Back Propagation Algorithm." In: *Asian Pacific journal of cancer prevention* 19.11 (2018), pp. 3257–3263. DOI: 10.31557/APJCP.2018.19.11.3257.

[30] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008. ISBN: 9780521865715.

[31] B. H. Menze et al. *The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS).* 2015. DOI: 10.1109/TMI.2014.2377694.

[32] M. Mirza and S. Osindero. "Conditional Generative Adversarial Nets." In: *CoRR* abs/1411.1784 (2014). URL: http://arxiv.org/abs/1411.1784.

[33] G. Mirzaei and H. Adeli. "Segmentation and clustering in brain MRI imaging." In: *Reviews in the Neurosciences* 30.1 (2018), pp. 31–44. DOI: 10.1515/revneuro-2018-0050.

[34] D. Müllner. "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python." In: *Journal of Statistical Software 53* 9 (2013), pp. 1–18. URL: http://www.jstatsoft.org/v53/i09/.

[35] F. Pérez-García, R. Sparks, and S. Ourselin. "TorchIO: a Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning." In: *CoRR* abs/2003.04696 (2020). URL: http://arxiv.org/abs/2003.04696.

[36] M. Perkuhn, P. Stavrinou, F. Thiele, G. Shakirin, M. Mohan, D. Garmpis, C. Kabbasch, and J. Borggrefe. "Clinical Evaluation of a Multiparametric Deep Learning Model for Glioblastoma Segmentation Using Heterogeneous Magnetic Resonance Imaging Data From Clinical Routine." In: *Investigative radiology* 53.11 (2018), pp. 647–654. DOI: 10.1097/RLI.0000000000000484.

[37] R. T. Rockafellar and R. J.-B. Wets. *Variational Analysis.* Springer-Verlag Berlin Heidelberg, 1998. ISBN: 9783642024313.

[38] S. M. Savaresi and D. L. Boley. "On the performance of bisecting K-means and PDDP." In: *Proceedings of the First SIAM International Conference on Data Mining.* SDM '01. Society for Industrial and Applied Mathematics, 2001, pp. 1–14. DOI: 10.1137/1.9781611972719.5.

[39] S. M. Smith. "Fast robust automated brain extraction." In: *Human brain mapping* 17.3 (2002), pp. 143–55. DOI: 10.1002/hbm.10062.

[40] R. Tyleček and R. Šára. "Spatial Pattern Templates for Recognition of Objects with Regular Structure." In: *Proceedings of the German Conference on Pattern Recognition.* GCPR '13'. Springer, Berlin, Heidelberg, 2013, pp. 364–374. DOI: 10.1007/978-3-642-40602-7_39.

[41] N. Vinh, J. Epps, and J. Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance." In: *Journal of Machine Learning Research* 11 (2010), pp. 2837–2854. URL: http://dl.acm.org/doi/10.5555/1756006.1953024.

[42] S. Winzeck et al. "ISLES 2016 and 2017-Benchmarking Ischemic Stroke Lesion Outcome Prediction Based on Multispectral MRI." In: *Frontiers in neurology* 9 (2018). DOI: 10.3389/fneur.2018.00679.

[43] X. Wu. "Optimal Quantization by Matrix Searching." In: *J. Algorithms* 12.4 (1991), pp. 663–673. DOI: 10.1016/0196-6774(91)90039-2.

A Clustering Evaluation

A.1 Shaded Clustering Results

In this section, we show some images created by the hierarchical clustering algorithms. The Leveled k-means (Figure 57a) and the Leveled Nesting (Figure 57b) result were produced with the same number of clusters per level. The cluster list given to Leveled Nesting is $[201, 3]$ while the one given to Leveled k-means is $[3, 67]$. Both produce a three-level dendrogram with three elements in the first level and 201 in the second. The lists differ because of the implementation of the two methods, where Leveled k-means divides each part of the clustering produced by the first number by the second number, while Leveled Nesting creates two independent clusterings and then maps them. The products of these two algorithms are very similar, which is not surprising since they both use k-means1d.

The Nesting procedure creates its own list of cluster amounts: $[77, 54, 38, 27, 19, 13, 9, 6, 5, 3]$. The maximum value ($k_{\text{max}}$) chosen for this instance is 100. Because of the larger number of levels of this cluster, Figure 57c has much more color diversity in each main cluster.

The Agglomerative clustering result in Figure 57d is produced by taking the full dendrogram and cutting it at the interesting points to remove the unnecessary levels. The chosen leveled list is the same as the one for Leveled Nesting ($[201, 3]$). This particular result is different from the previous ones because the sub clusterings are not sorted. The hierarchical clusterings with k-means1d always produce sorted results, which creates a smooth shading transition. This same effect could be realized with the Agglomerative result by sorting each level, but it is not its standard solution.

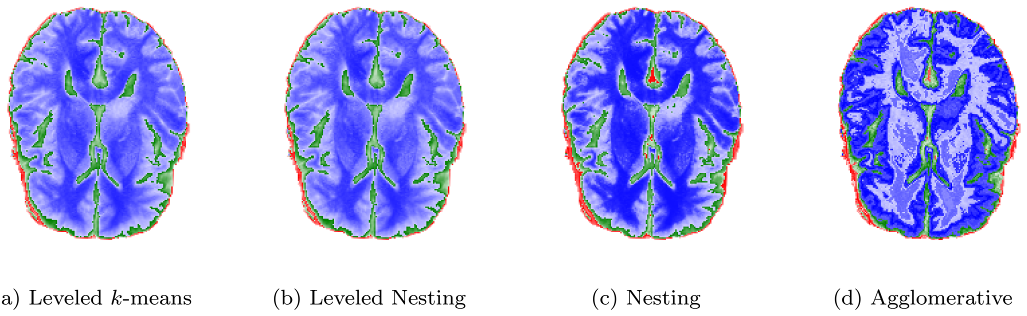
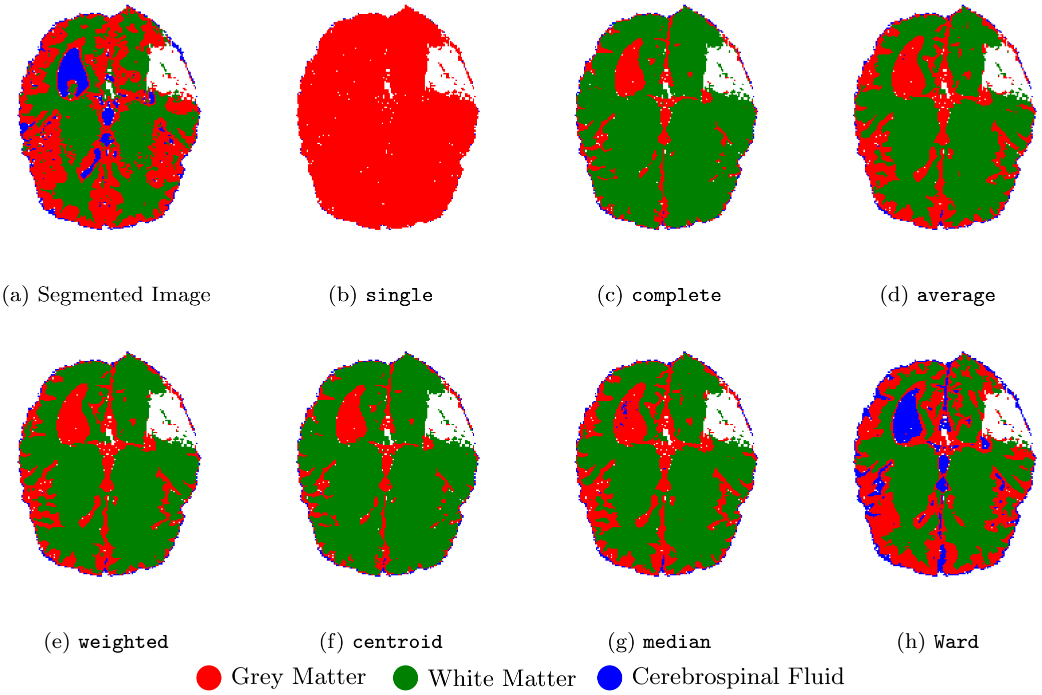

(a) Leveled k-means (b) Leveled Nesting (c) Nesting (d) Agglomerative

FIGURE 57: [Best viewed in color] Comparison of the clusterings obtained with Leveled k-means, Leveled Nesting, Nesting and Agglomerative for patient 2 (BraTS 19).

A.2 Agglomerative Comparison

In Figure 60 and Figure 61 we present all the evaluation metrics for the linkage evaluation. Note that this evaluation has been run on the middle transverse slice of the brain and not on the full volume. The micro precision and recall, the macro precision and recall and the weighted precision and recall have comparable values. However, in the combinatorial recall there are very different values, and `single` is the best method, which is another proof that this method does not fit our use case. In this case most of the points have been assigned to one single cluster and, similarly to the case of `all` for three clusters (Figure 17f), this makes the number of combinatorial false negatives very small. We additionally add some images (Figure 58) to show how the methods



(a) Segmented Image (b) `single` (c) `complete` (d) `average`

(e) `weighted` (f) `centroid` (g) `median` (h) `Ward`

● Grey Matter ● White Matter ● Cerebrospinal Fluid

Figure 58: [Best viewed in color] Segmentation and different Agglomerative clusterings for T1 Weighted of patient 5 (BraTS 13).

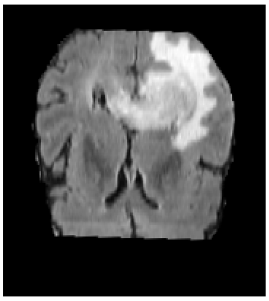


Figure 59: T2 FLAIR of patient 3 (BraTS 13) is not complete.

behave in practice. It is clear that the only image that looks similar to the segmentation is the one produced by `Ward`.

A.3 Comparison of k-means1d, Nesting, Agglomerative

In this section, we present the scores for the evaluation between k-means1d, Nesting and Agglomerative for each single patient. In Figure 62, Figure 63, Figure 64 and Figure 65 we have respectively the scores for patient 1, 2, 3 and 5. These values have been used to compute the averages in Figure 23. In T1 Contrast Enhanced of patient 1 we obtain almost perfect values. The values for this patient are generally better than average, but consistent with the ones shown in Figure 23. Patient 2's T1 Weighted is peculiar, because it is the only T1 Weighted that is best clustered with Agglomerative. This patient, together with patient 3, achieves the best T1 Contrast Enhanced clustering with Nesting. Patient 3 has very high values for Agglomerative in

T2 Weighted, but all methods are similar when clustering T2 FLAIR. This might be due to the fact that the T2 FLAIR scan of this patient was partially missing (Figure 59). This is rather common because doctors only acquire a partial scan for T2 FLAIR when they know the general location of a tumor. For patient 5 the best T1 Contrast Enhanced method is Agglomerative. In T1 Weighted the best method is Nesting and in T2 Weighted the best method is k-means1d.

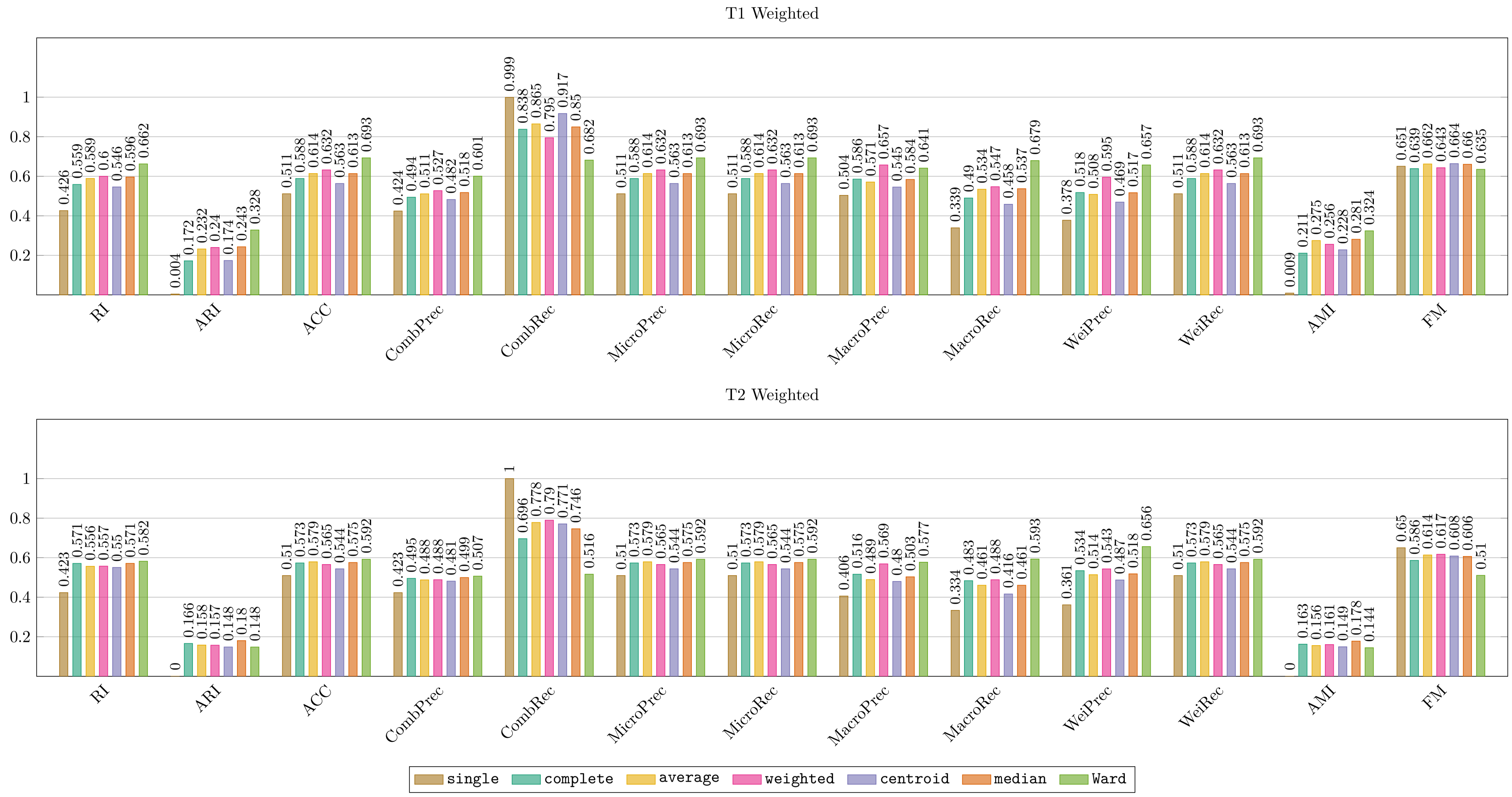


Figure 60: [Best viewed in color] Comparison between different hierarchical clustering metrics using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). The values are grouped by sequence and represent the average of the scores of the four segmented patients. For all measures a higher value indicates a better score.

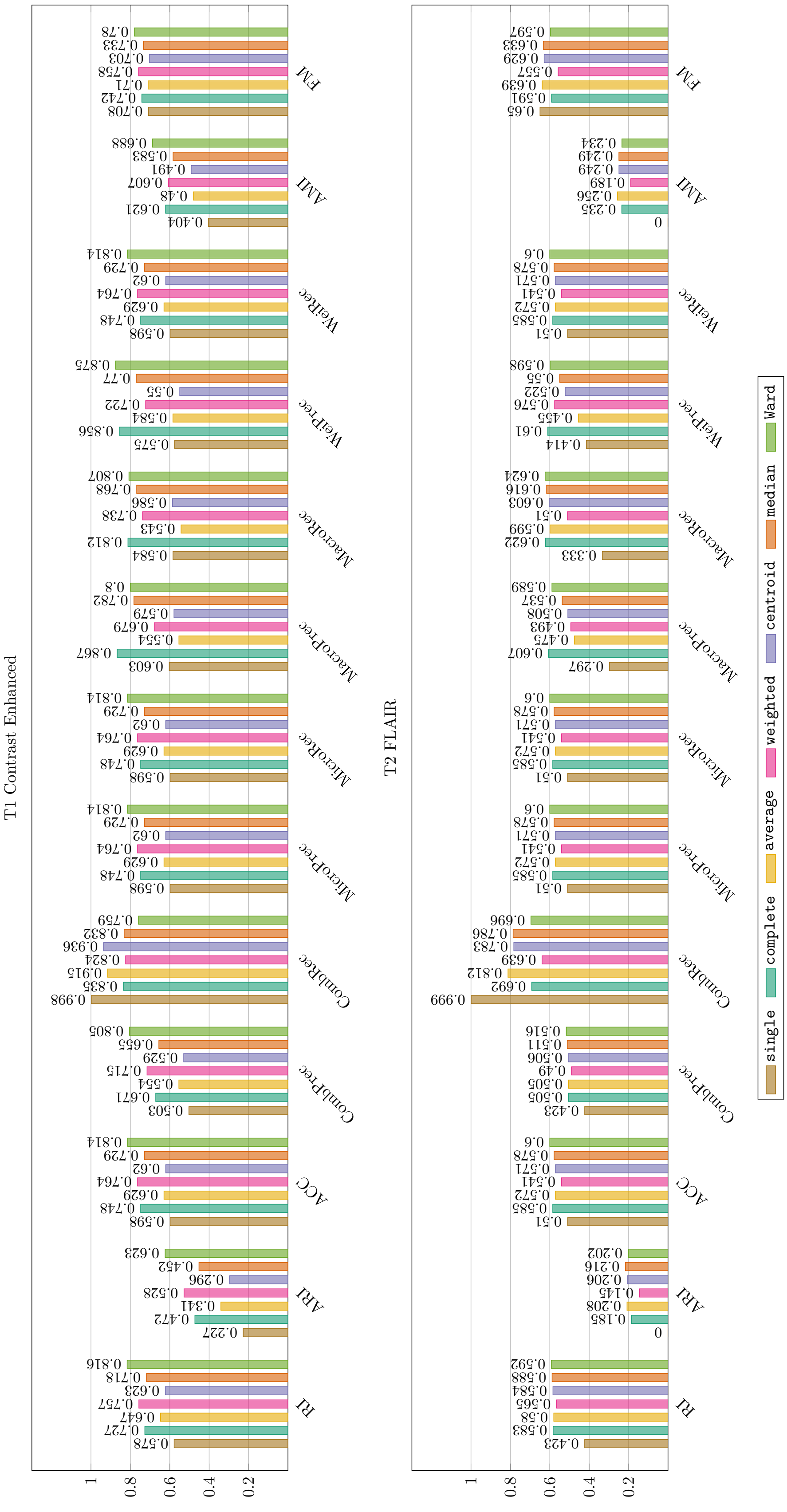


FIGURE 61: [Best viewed in color] Comparison between different hierarchical clustering metrics using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). The values are grouped by sequence and represent the average of the scores of the four segmented patients. For all measures a higher value indicates a better score.

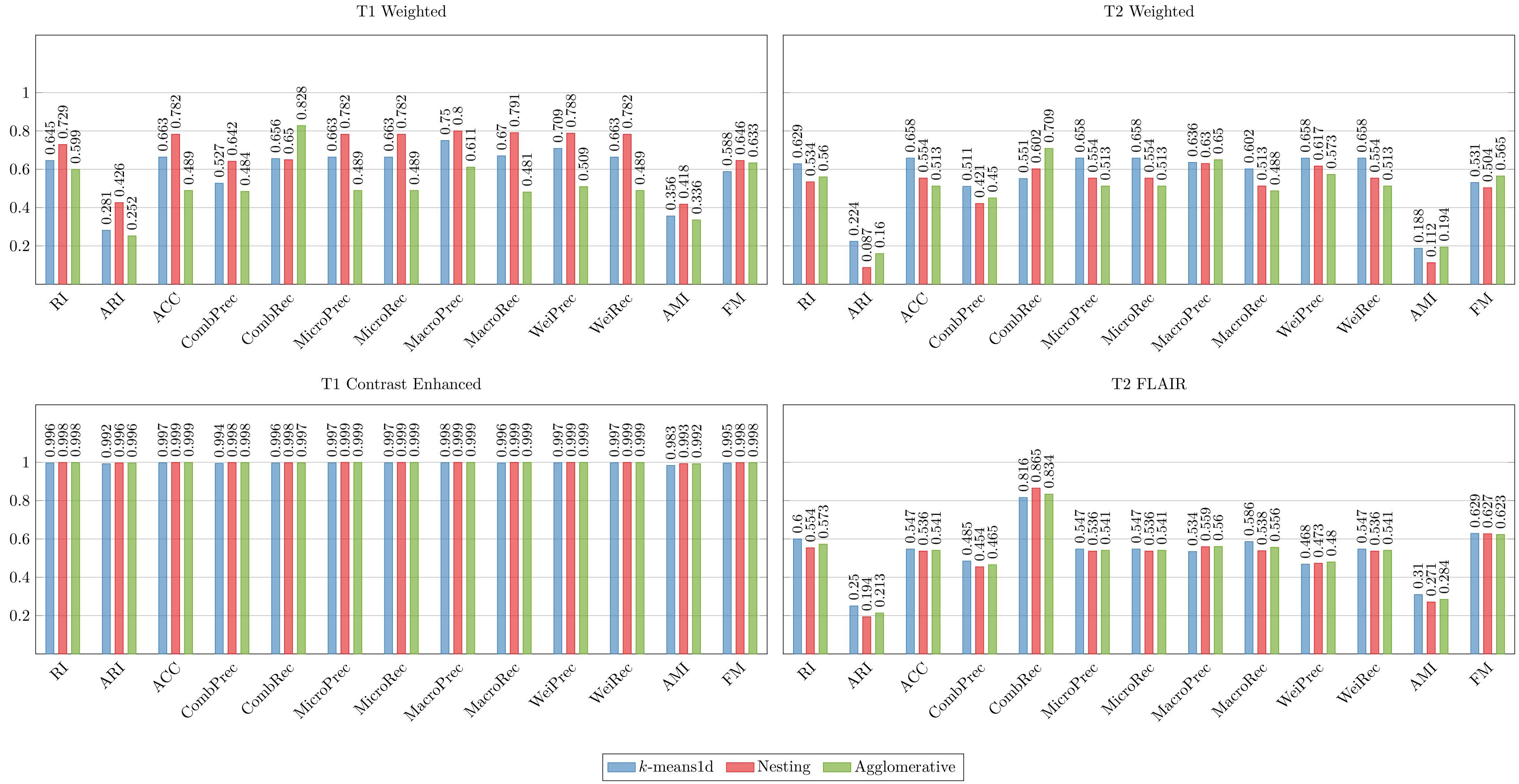


Figure 62: [Best viewed in color] Comparison between k-means1d, Nesting and Agglomerative (`Ward`) using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). These values are for patient 1 (BraTS 13) and are grouped by sequence. For all measures a higher value indicates a better score.

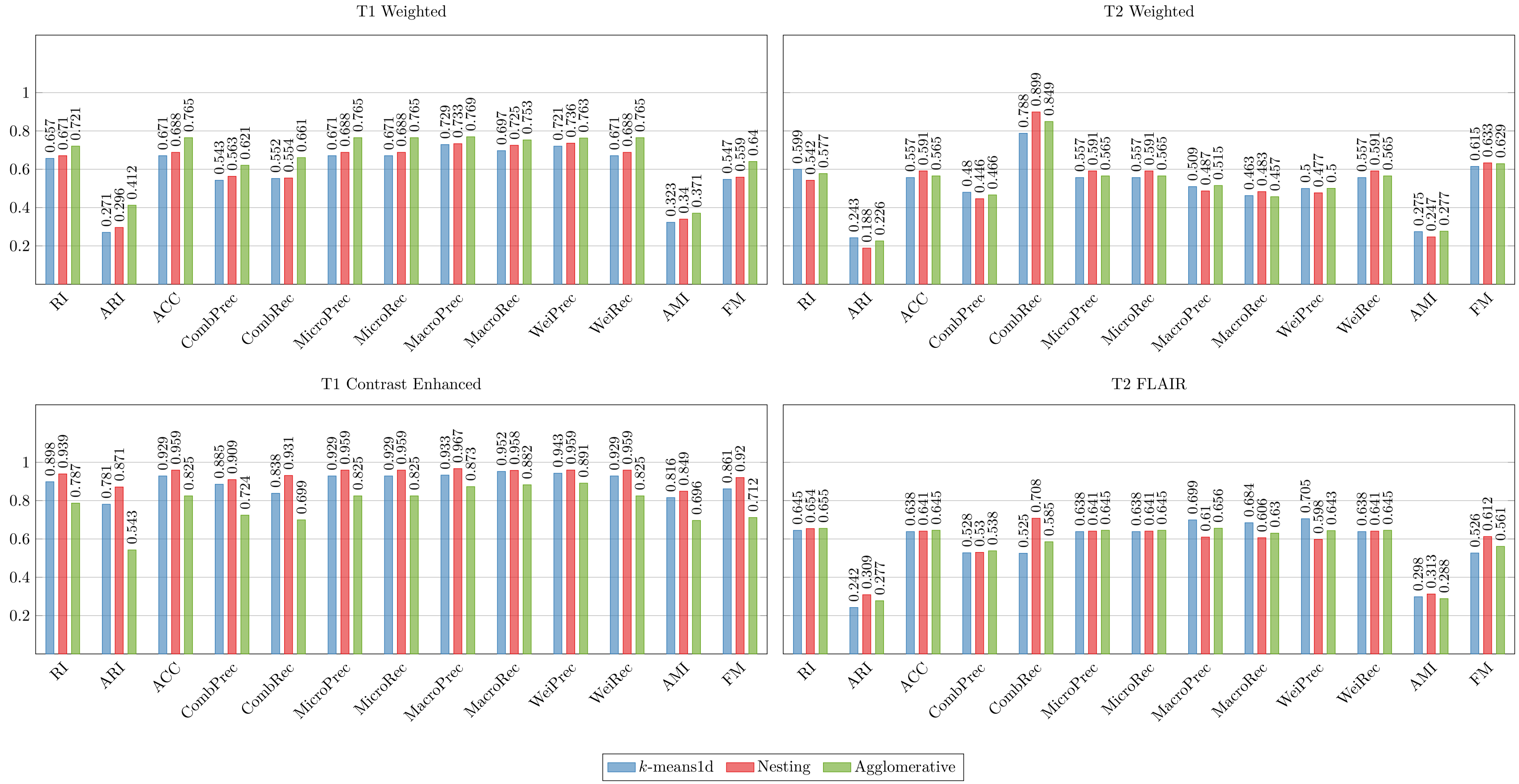


Figure 63: [Best viewed in color] Comparison between k-means1d, Nesting and Agglomerative (`Ward`) using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). These values are for patient 2 (BraTS 13) and are grouped by sequence. For all measures a higher value indicates a better score.

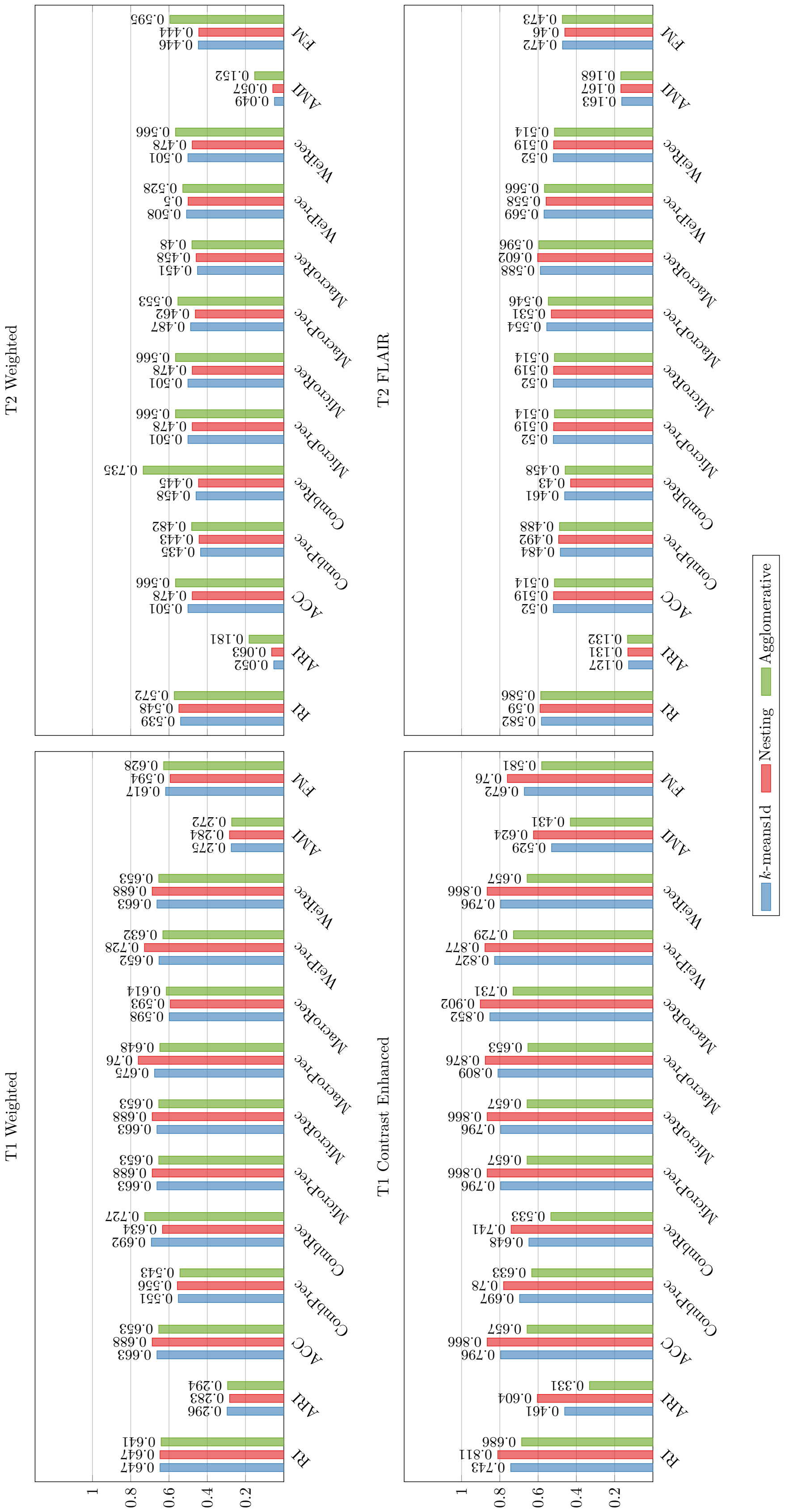


FIGURE 64: [Best viewed in color] Comparison between *k*-means1d, Nesting and Agglomerative (`Ward`) using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). These values are for patient 3 (BraTS 13) and are grouped by sequence. For all measures a higher value indicates a better score.

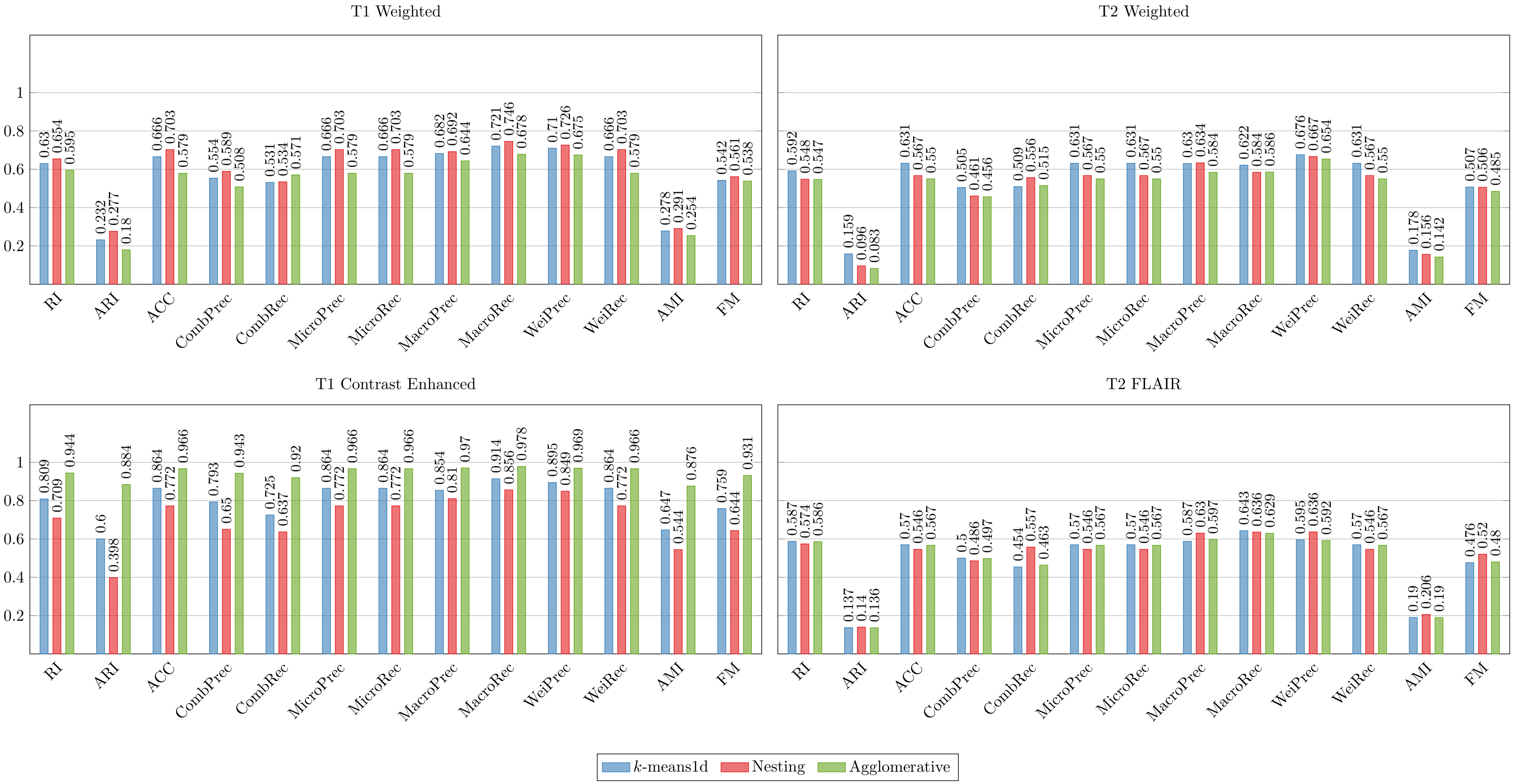


FIGURE 65: [Best viewed in color] Comparison between *k*-means1d, Nesting and Agglomerative (`Ward`) using Rand index (RI), adjusted Rand index (ARI), accuracy (ACC), combinatorial precision and recall (CombPrec, CombRec), micro/macro/weighted precision and recall (Micro/Macro/WeiPrec, Micro/Macro/WeiRec), adjusted mutual information score (AMI) and Fowlkes-Mallows index (FM). These values are for patient 5 (BraTS 13) and are grouped by sequence. For all measures a higher value indicates a better score.

B Mapping Evaluation

B.1 Choice of Main Clusters

In this section, we present the metrics used to evaluate the performance of the segmentation produced by DeepMedic and HD-GLIO with different numbers of main clusters. In Figure 66 and Figure 67 we present the full comparison of the evaluation metrics comparing the tumor ground truth to the segmentations of DeepMedic and HD-GLIO with a transformed image created by BrainClustering k-meanswith different numbers of main clusters. The number of sub clusters is always 100. We also present the values obtained by the segmentation with four original images (in blue).
The segmentation with the original images is indeed better than the substitution, as can be seen by all measures (except in the accuracy of HD-GLIO, but this measure could also be improved by random chance). The results of other measures diverge: For the case of sensitivity, specificity and Dice score three clusters seems to be a wiser choice, but we have better Hausdorff distances using four clusters.

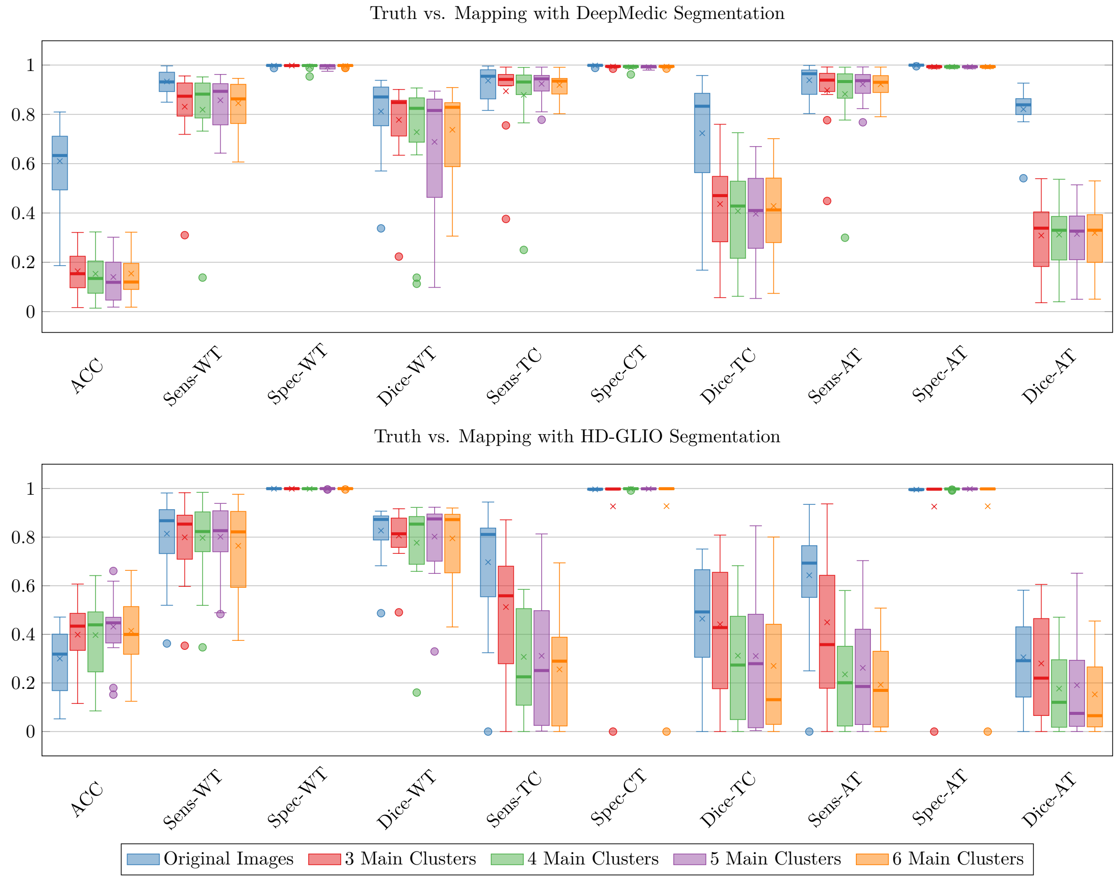


FIGURE 66: [Best viewed in color] Comparison of BrainClustering k-means (search) with different numbers of main clusters with respect to the true tumor segmentation. The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 14 patients from `testUK`. For all measures a higher value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

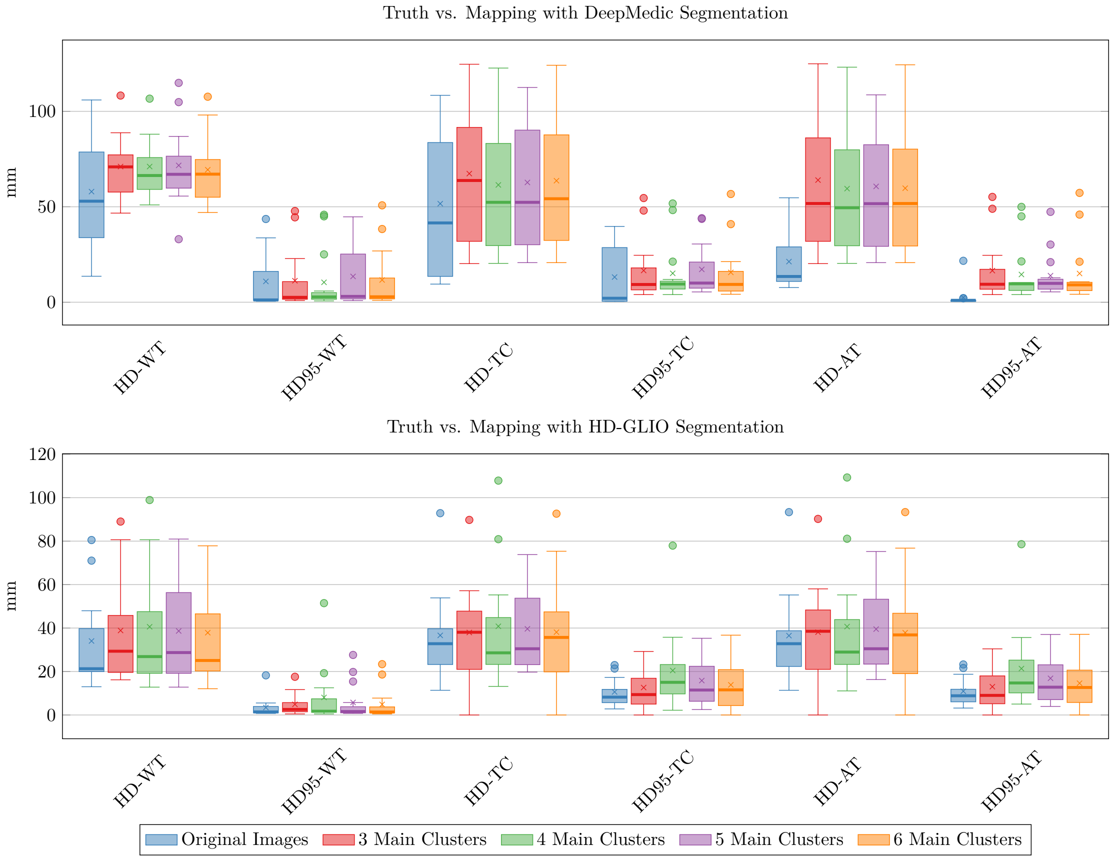


FIGURE 67: [Best viewed in color] Comparison of BrainClustering k-means (search) with different numbers of main clusters with respect to the true tumor segmentation. The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 14 patients from `testUK`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

B.2 Transformed T2 Weighted Evaluation

In this section, we present additional measures for the evaluation of the T2 Weighted sequences generated with our approaches. The evaluation in this appendix refers to the segmentation that can be produced using the transformed images and DeepMedic and HD-GLIO. In fact, we assess the quality of the segmentation that is produced with the real T1 Weighted, T1 Contrast Enhanced and T2 FLAIR and a generated T2 Weighted. For the data sets for which we have the tumor ground truth (`testUK` and `val`) we present a comparison with the real segmentation, while for `testNGT` we compare the results to the original segmentation with four images. In the first case we also present the values obtained with the original segmentation (in blue).
First, we consider the results for the `testUK` data set (Figure 68 and Figure 69). Let us first consider the bounded measures, which are collected in Figure 68. Both for DeepMedic and HD-GLIO all approaches have very high specificity, meaning that the healthy brain voxels are mostly classified correctly. In DeepMedic the segmentation with the original images is always clearly the best, while in HD-GLIO some approaches achieve better results (for example, BrainClustering k-means's Dice score in Whole Tumor). The accuracy measure in HD-GLIO produces contradicting results with respect to the other measures: BrainClustering Agglomerative has the best accuracy, but is the method with the worst scores in general. The search BrainClustering methods produce worse results than their Train&Test counterparts in HD-GLIO, while the results are similar in DeepMedic. Pix2Pix and Complementary perform similarly, but with Complementary being better most times.

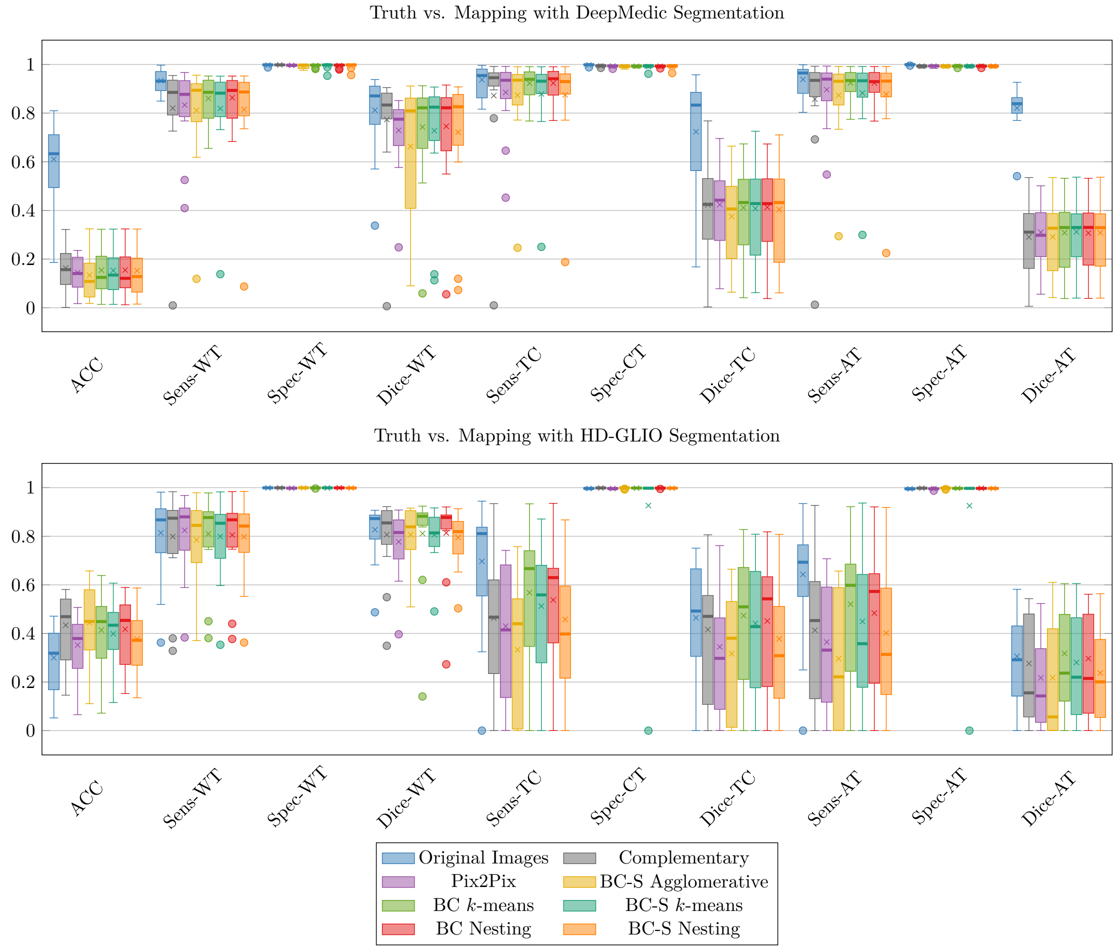


FIGURE 68: [Best viewed in color] Comparison of different approaches for the segmentation obtained with a transformed T2 Weighted with respect to the true tumor segmentation. The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 14 patients from `testUK`. For all measures a higher value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

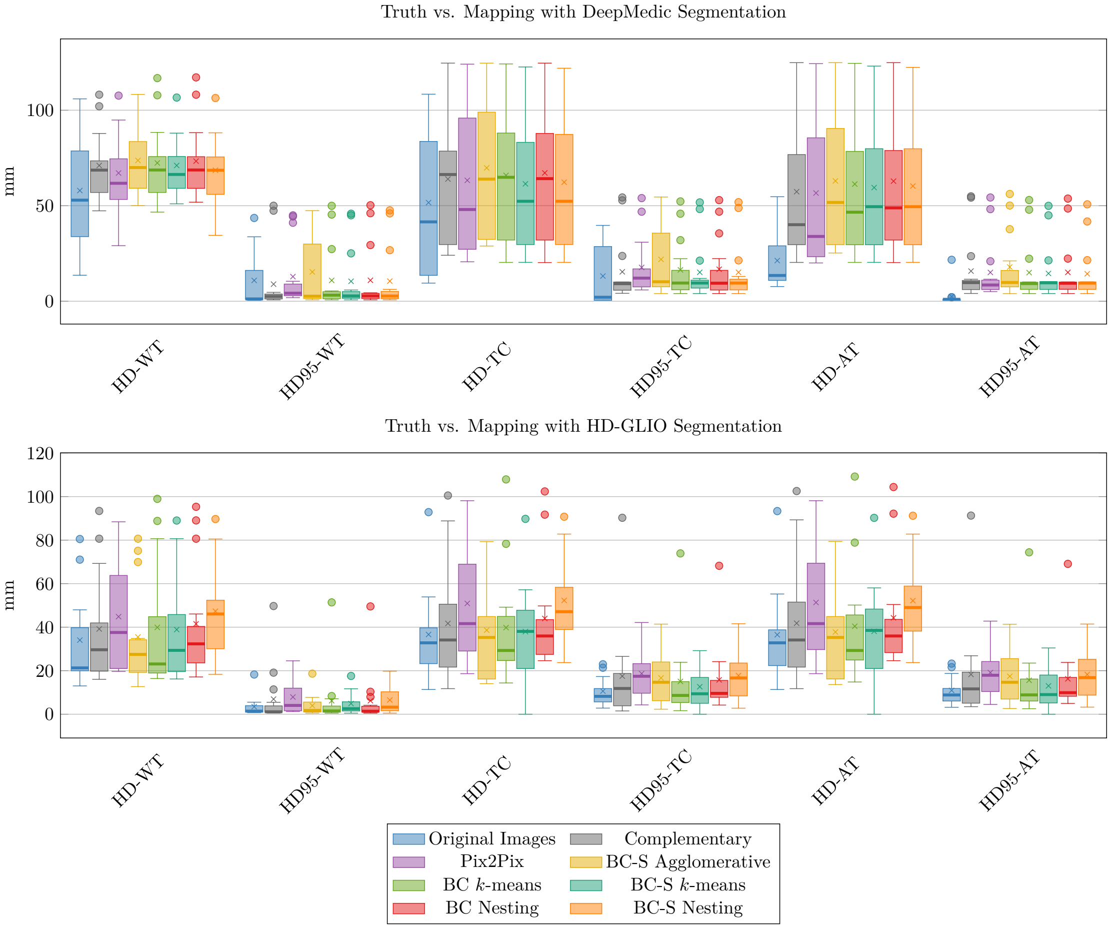


FIGURE 69: [Best viewed in color] Comparison of different approaches for the segmentation obtained with a transformed T2 Weighted with respect to the true tumor segmentation. The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 14 patients from `testUK`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

We now move to the unbounded measures, which are presented in Figure 69. We learn that in DeepMedic the reproduced images yield much worse distances in the Active Tumor area, while in the other regions the distances are worse, but still rather similar. This is reasonable considering that the resulting images proposed in Section 8 have large areas of Active Tumor. However, this is not the case for HD-GLIO, where the distances are worse for the transformed images, but in all regions equally.

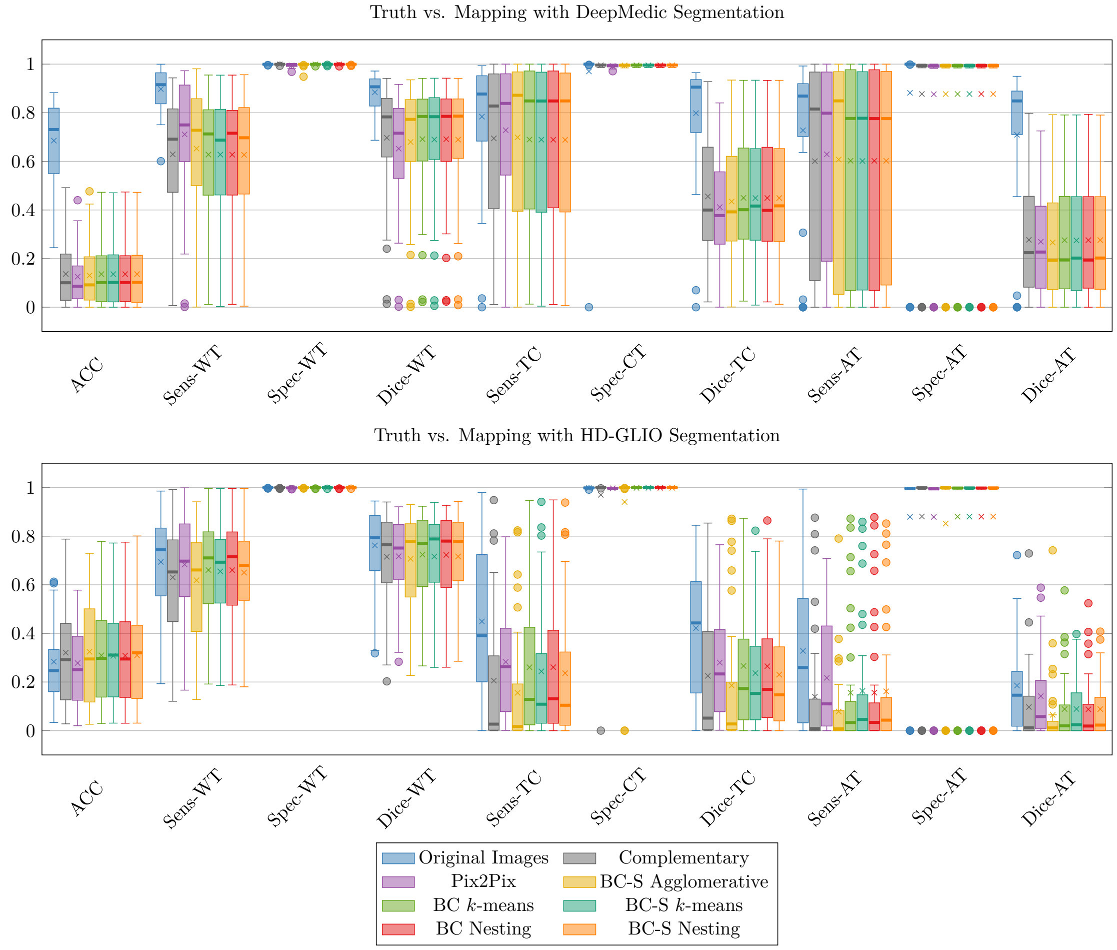


Figure 70: [Best viewed in color] Comparison of different approaches for the segmentation obtained with a transformed T2 Weighted with respect to the true tumor segmentation. The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 34 patients from `val`. For all measures a higher value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

The results for the `val` data set are shown in Figure 70 and Figure 71. Once more, we first discuss the bounded measures (Figure 70). The results for this data set present more outliers in the Active Tumor area. All approaches are equally distributed for DeepMedic, and no approach is clearly outperforming the others. By comparison, for HD-GLIO Pix2Pix is the best approach. In the distance measures (Figure 71) most approaches are still equally distributed for DeepMedic, but now Complementary has the best distances. Additionally, it has also the shortest distances for HD-GLIO.

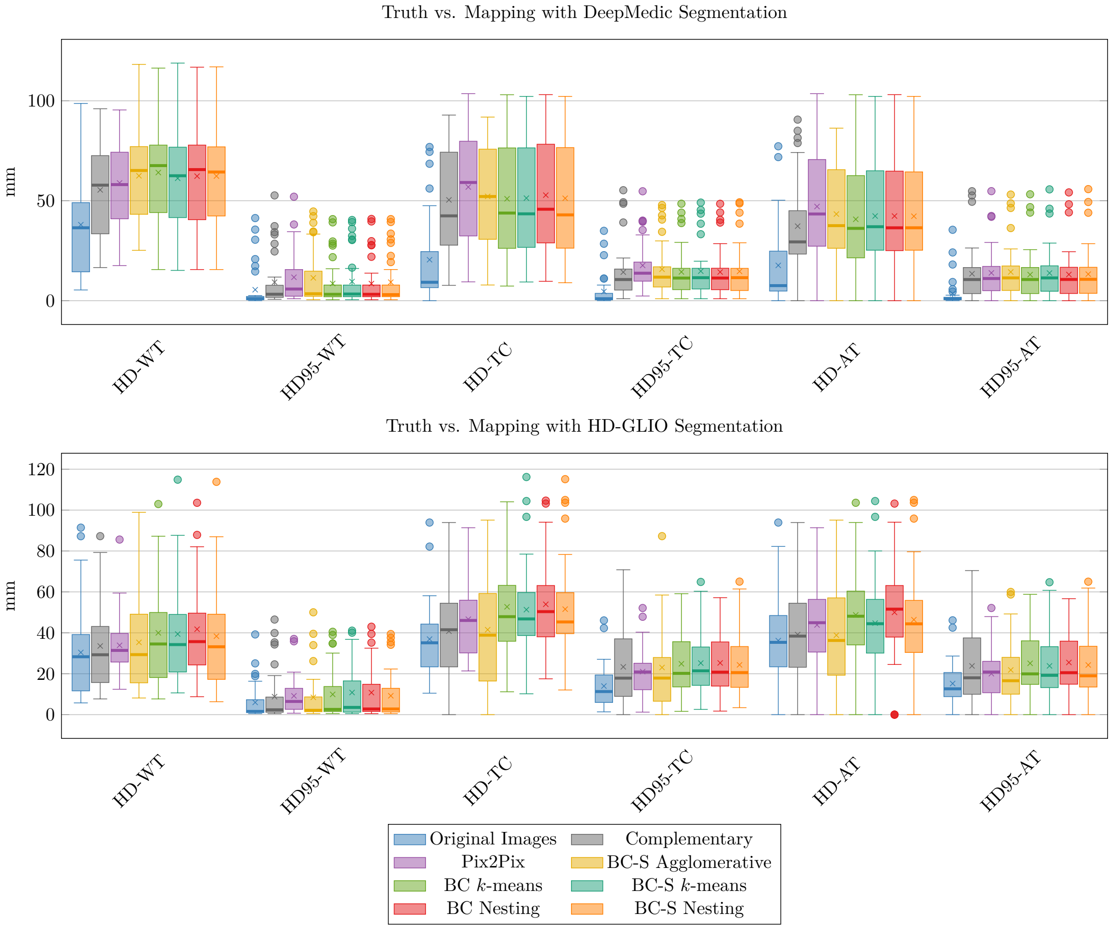


Figure 71: [Best viewed in color] Comparison of different approaches for the segmentation obtained with a transformed T2 Weighted with respect to the true tumor segmentation. The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 34 patients from `val`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

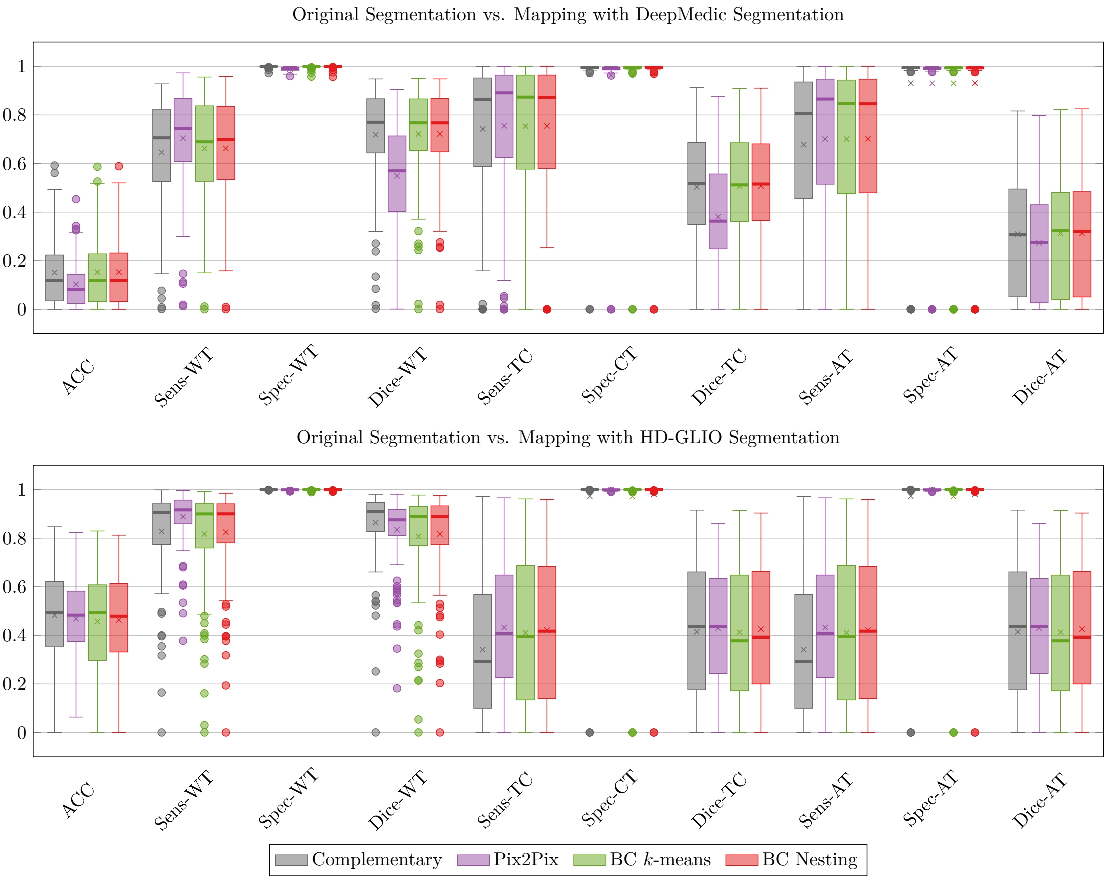


FIGURE 72: [Best viewed in color] Comparison of different approaches for the segmentation obtained with a transformed T2 Weighted with respect to the true tumor segmentation. The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 111 patients from `testNGT`. For all measures a higher value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

For the `testNGT` data set we perform a different evaluation, since we do not have the ground. In this case the measures are computed with respect to the segmentation that can be achieved with four original images. We first look at the bounded measures (Figure 72). The Dice scores for BrainClustering and Complementary are similar, while Pix2Pix performs worst. By contrast, it obtains the best sensitivity scores. In HD-GLIO all approaches perform similarly, but Pix2Pix is the best for most measures. According to the distance measures (Figure 73) in DeepMedic Pix2Pix obtains the worst values. Similarly, Pix2Pix and BrainClustering k-means have the worst distances in HD-GLIO. In both segmentations the results of complementary have the best distances, but in DeepMedic the values of Complementary, BrainClustering k-means and BrainClustering Nesting are similar.

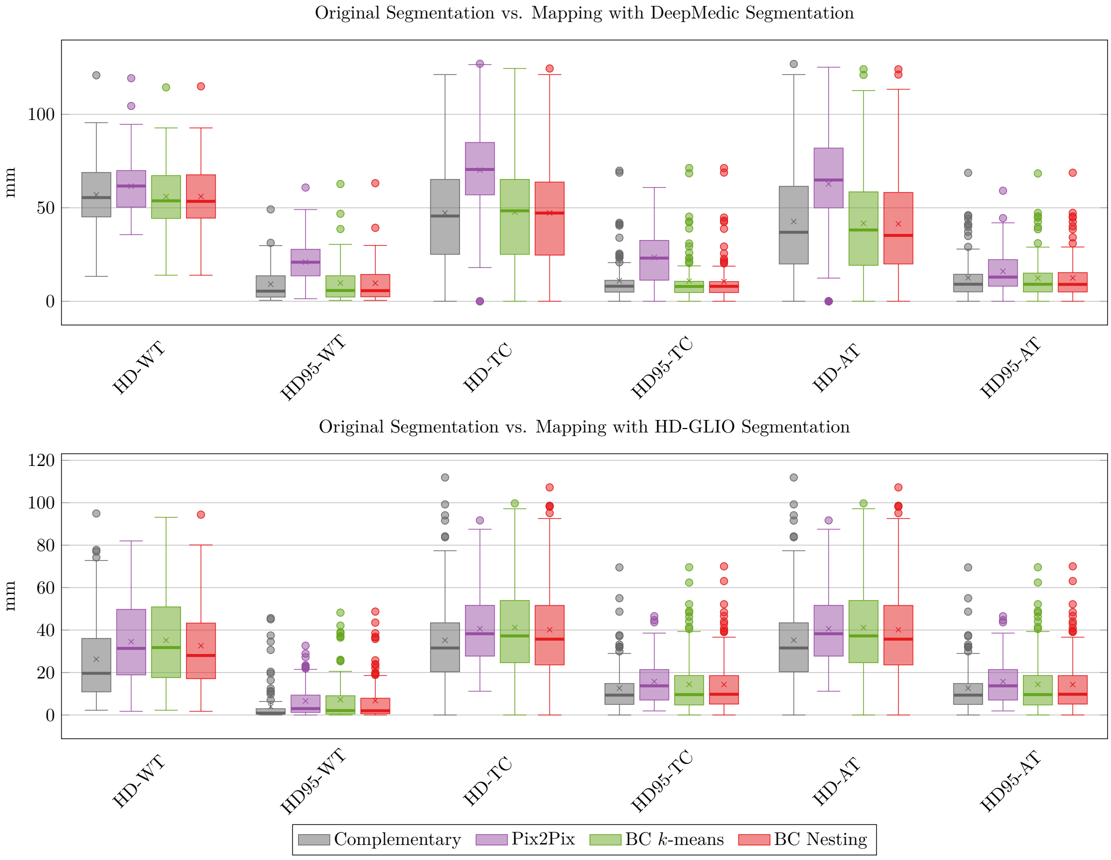


FIGURE 73: Comparison of different approaches for the segmentation obtained with a transformed T2 Weighted with respect to the true tumor segmentation. The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor). The scores represent the distribution of 111 patients from `testNGT`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.

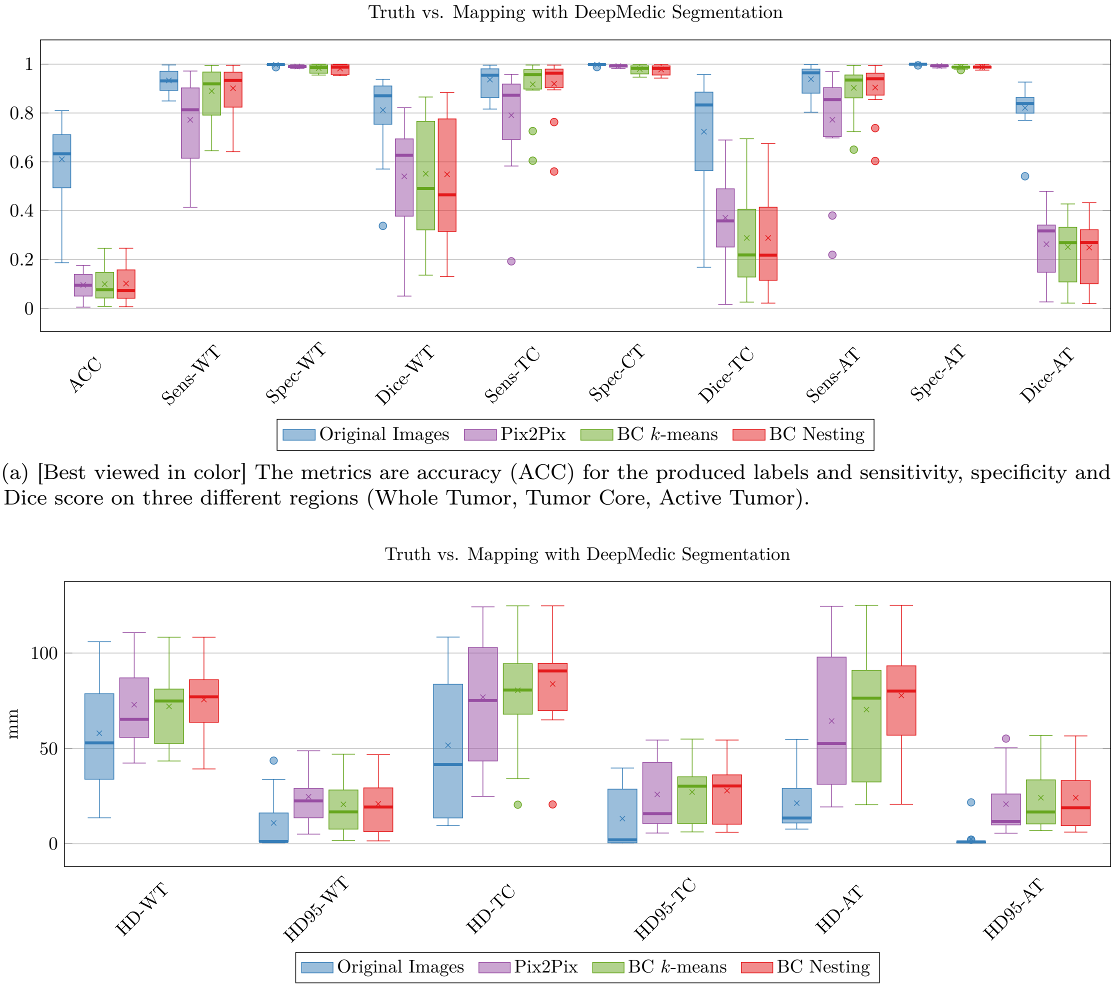
B.3 Transformed T1 Contrast Enhanced Evaluation

In this section, we discuss the results of the Pix2Pix, BrainClustering k-means and BrainClustering Nesting (both only in Train&Test mode) approaches with respect to the recreation of the T1 Contrast Enhanced sequence. The evaluation metrics contained in this appendix refer to the quality of the segmentation that can be achieved using the real T1 Weighted, T2 Weighted and T2 FLAIR and a reproduced T1 Contrast Enhanced. The evaluation of the data sets with the tumor ground truth (`testUK` and `val`) is based on the comparison with the real segmentation, while the evaluation for `testNGT` is based on the comparison with the segmentation with four original images. For `testUK` and `val` we also present the values for the segmentation produced with four original images (in blue). Once again, the evaluation is divided according to the data sets.

First, we examine the `testUK` data set (Figure 74a and Figure 74b). Pix2Pix has the best distances and the best scores. However, the BrainClustering methods have higher sensitivity in all three regions, meaning that the BrainClustering methods identify a larger portion of the tumorous voxels.

Next, we observe the results for the `val` data set (Figure 75a and Figure 75b). The BrainClustering methods still have a better sensitivity, but also have a better accuracy and Dice score, which means that in this data set BrainClustering has a higher precision. BrainClustering also achieves better distances for this data set. Note that the difference between this data set and the previous one is that the ground truth comes from different sources, and thus it is completely reasonable that some methods are better with a certain ground truth.

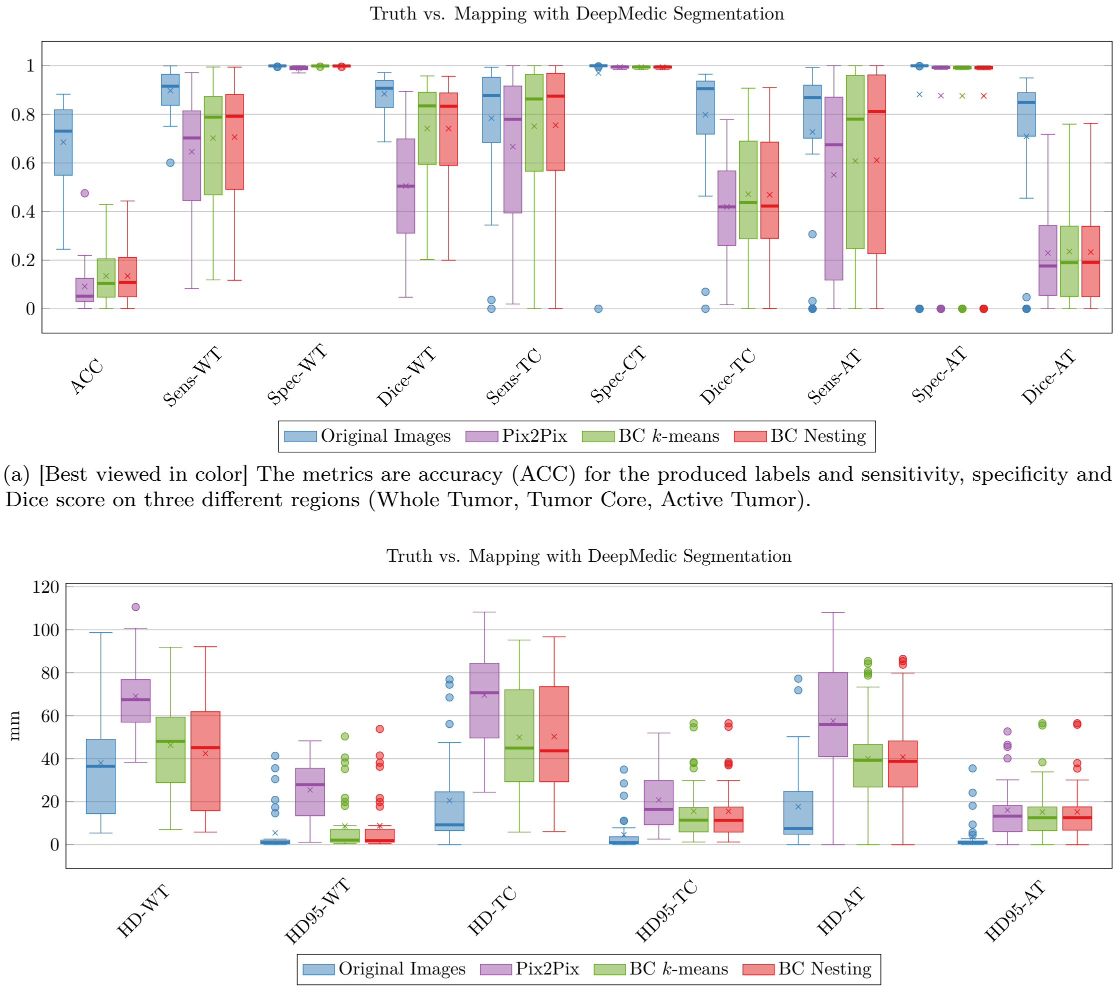
Lastly, we compare the segmentations produced from the transformed T1 Contrast Enhanced with the segmentation produced with four original images for the `testNGT` data in Figure 76a and Figure 76b. The results are similar to the ones of the `val` data set, and the BrainClustering methods obtain better (or equal) scores in all metrics.



(a) [Best viewed in color] The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor).

(b) [Best viewed in color] The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor).

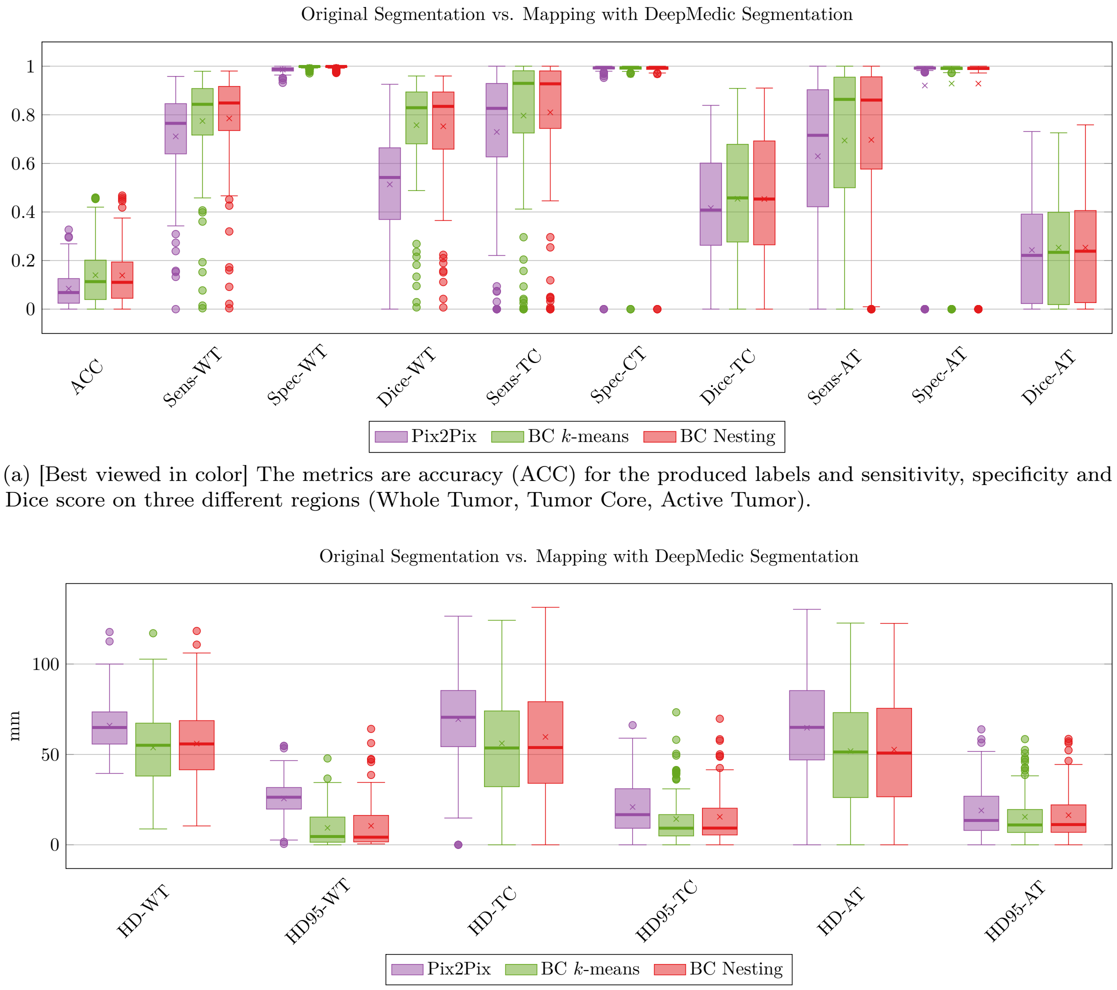
FIGURE 74: Comparison of different approaches for the segmentation obtained with a transformed T1 Contrast Enhanced with respect to the true tumor segmentation. The scores represent the distribution of 14 patients from `testUK`. For all measures a higher value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.



(a) [Best viewed in color] The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor).

(b) [Best viewed in color] The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor).

Figure 75: Comparison of different approaches for the segmentation obtained with a transformed T1 Contrast Enhanced with respect to the true tumor segmentation. The scores represent the distribution of 34 patients from `val`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.



Original Segmentation vs. Mapping with DeepMedic Segmentation

(a) [Best viewed in color] The metrics are accuracy (ACC) for the produced labels and sensitivity, specificity and Dice score on three different regions (Whole Tumor, Tumor Core, Active Tumor).

Original Segmentation vs. Mapping with DeepMedic Segmentation

(b) [Best viewed in color] The metrics are undirected Hausdorff distance and undirected 95% percent Hausdorff distance on three different regions (Whole Tumor, Tumor Core, Active Tumor).

FIGURE 76: Comparison of different approaches for the segmentation obtained with a transformed T1 Contrast Enhanced with respect to the true tumor segmentation. The scores represent the distribution of 111 patients from `testNGT`. For all measures a lower value indicates a better score. The crosses represent the averages, while the thick bars are the medians and the dots are the outliers.