Zeszyty Naukowe WWSI, No 26, Vol. 16, 2022, pp. 7-19
DOI: 10.26348/znwwsi.26.7

CUDA accelerated Medical Segmentation metrics with MedEval3D

Jakub Mitura* [1] and Beata E. Chrapko[2]

[1]National Information Processing Institute, Medical University of Lublin
[2]Chair and Department of Nuclear Medicine, Medical University of Lublin

Abstract

Medical segmentation metrics are crucial for development of correct segmentation algorithms in medical imaging domain. In case of three dimensional large arrays representing studies like CT, PET/CT or MRI of critical importance is availability of library implementing high performance metrics. MedEval3D is created in order to fulfill this need thanks to implementation of CUDA acceleration. Most of implemented metrics like Dice coefficient, Jacard coefficient etc. are based on confusion matrix, what enable effective reuse of calculations across multiple metrics improving performance in such use case. Additionally algorithms like interclass correlation and Mahalanobis distance are also introduced. In both cases their implementations are significantly faster then their counterparts from other available libraries. Lastly programming interface to all of the metrics was created in Julia programming language.

Keywords — CUDA, Computer Tomagraphy, PET/CT, medical image segmentation

1 Introduction

In order to evaluate performance of any algorithm one need to define appropriate metric. In the field of medical segmentation arguably most influential works describing segmentation metrics were those published by Renard et al [1] and by Taha and Hanbury [2]. Those works stressed that optimal metric may be different depending on the problem, hence pointing out the need of a tool that would implement whole set of metrics.

The goal of this package is to implement CUDA accelerated framework independent metrics that are suggested in most influential works, and create interface in Julia programming language to give scientific community of researchers working on medical segmentation access

*E-mail: jakub.mitura14@gmail.com

Manuscript received January 10, 2022

on rapidly growing environment of scientific computing in Julia language. Additionally the presented package is part of the bigger medical segmentation framework developed by the author [3].

Currently in the subfield of machine learning concentrated around medical imaging there are two main frameworks Pymia [4] and Monai [5], in both cases user interface is written in Python. Monai [5] implements multiple highly optimized metrics that are particularly prevalent in case of neural networks with available CUDA acceleration, yet lacks most of those described in [2]. It is also based on PyTorch [6]. Pymia [4] implements all of the algorithms described in [2], is framework independent, but lacks GPU acceleration. GPU acceleration can drastically increase performance in case of large data objects − and such case is very common in modern medical imaging.

2 Preliminaries

In the section below there are presented mathematical formulas that were used in implemented algorithms, and are borrowed from the work of Taha and Hanbury [2].

2.1 Notation

$[\ ]$	–	represents empty list, if not stated otherwise lists are 1 indexed (not $0-$ indexed);
$X = x_1 ..., x_n$	–	set of all voxels representing single image (for example CT image), where $\|X\| = w * h * d = n$ and w, h, d represents width, height and depth;
$S_g = S_g^1, S_g^2$	–	ground truth segmentation of X.;
S_t	–	segmentation of X that one is comparing with ground truth segmentation;
$f_g^i(x)$	–	membership function that given a voxel x_i return value marking its membership;
S^1, S^2 if not stated otherwise S^1	–	is class of interest for example representing organ of interest and S^2 is background;
TP	–	true positive;
FP	–	false positive;
FN	–	false negative;
TN	–	true negative;
MI	–	Mutual Information;
P	–	the set of 2 element tuples that represent all possible object pairs in $X \times X$;
ICC	–	Interclass Correlation;
KAP	–	Cohen Kappa Coefficient;
MHD	–	Mahalanobis Distance.

2.2 Binary classification

In case of binary classification metric we calculate the TP, FP, FN and TN as shown in [2]:

$$m_{ij} = \sum_{r=0}^{|X|} f_g^i(x_r) f_t^j(x_r) \tag{1}$$

In the formula above $TP = m_{11}$, $FP = m_{10}$, $FN = m_{01}$ and $TN = m_{00}$. Function $f_g^i(x)$ is a membership function that given a voxel x_i provides a label understood as membership to a set representing a given class (for example background class and object class). In case of binary classification $f_g^I(x) = 1 \; iff \; x \in S_g^i$ and if $x \in S_g^j \; f_g^i(x) = 0$.

2.3 Dice coefficient

Dice coefficient is one of the most widely used overlap based metrics [2], and is calculated in [2] as seen in equation:

$$DICE = \frac{2TP}{2TP + FP + FN} \tag{2}$$

2.4 Jacard coefficient

Jacquard coefficient is similar to Dice Coefficient yet it is always larger than Dice apart from extremum of 1 and 0 value where they are equal as we can see from the following equation:

$$JAC = \frac{TP}{TP + FP + FN} \tag{3}$$

2.5 Global consistency error

Let $R(S, x)$ is set of all voxels in the same segmentation region S as voxel x. For two segmentations S_t and S_g we can define how similar they are using equations:

$$GCE(S_t, S_g) = \frac{1}{n} min \left(\sum_i^n E(S_t, S_g, x_i), \sum_i^n E(S_g, S_t, x_i) \right), \tag{4}$$

where E is defined in [2] in equation:

$$E(S_t, S_g) = \frac{|R(S_t, x) / R(S_g, x)|}{|R(S_t, x)|} \tag{5}$$

GCE can be also defined as shown in [2] in equation below.

$$GCE = \frac{1}{n} min \left(A, B \right), \tag{6}$$

where A and B are auxiliary variables:

$$A = \frac{FN(FN+2TP)}{TP+FN} + \frac{FP(FP+2TN)}{FP+TN}, \tag{7}$$

$$B = \frac{FP(FP+2TP)}{TP+FP} + \frac{FN(FN+2TN)}{TN+FN}. \tag{8}$$

2.6 Volume based metrics

This type of metric is measuring differences in volumes, which is particularly important in 3d studies such as for example CT scan. Volumetric similarity was defined in [2] as seen in equation:

$$VS = 1 - \frac{\left|\,|S_t^1| - |S_g^1|\,\right|}{|S_t^1| - |S_g^1|} = 1 - \frac{|FN-FP|}{2TP+FP+FN} \tag{9}$$

2.7 Pair counting based metrics

Given all possible voxel pairs in set P $(x_i, x_j) \in P$ we can divide those in 4 groups:

- *Group I* – x_i and x_j are given the same label in both segmentations that we are comparing (usually it would be gold standard and algorithm on which one is working).
- *Group II* – x_i and x_j have the same labels in segmentation S_g but different in S_t.
- *Group III* – x_i and x_j have the same labels in segmentation S_t but different in S_g.
- *Group IV* – x_i and x_j have different labels in both S_t and S_g.

Group I and IV represents agreement between segmentations and Groups II and III disagreement.

As presented in [2] there is an algorithm that prevents the need to analyze all of the point pairs and enable calculation of binary classification metric using TP, TN, FP and FN.

$$a = \frac{1}{2}[TP(TP-1) + FP(FP-1) + TN(TN-1) + FN(FN-1)] \tag{10}$$

$$b = \frac{1}{2}[(TP+FN)^2 + (TN+FP)^2 - (TP^2+TN^2+FP^2+FN^2)] \tag{11}$$

$$c = \frac{1}{2}[(TP+FP)^2 + (TN+FN)^2 - (TP^2+TN^2+FP^2+FN^2)] \tag{12}$$

$$d = \frac{n(n-1)}{2} - (a+b-c) \tag{13}$$

In the expression (13, n is the number of all possible pairs of points, hence $n = |P|$.

2.7.1 Adjusted Rand Index

Adjusted Rand Index [7] is a modification of Rand Index [8], and can be used as a metric for fuzzy segmentation. Equation (14) is presenting calculation of Adjusted Rand Index (ARI):

$$ARI = \frac{\sum_{ij} \binom{m_{ij}}{2} - \frac{\sum_i \binom{m_i}{2} \sum_j \binom{m_j}{2}}{\binom{n}{2}}}{\frac{1}{2}\left[\sum_i \binom{m_i}{2} + \sum_j \binom{m_j}{2}\right] \frac{\sum_i \binom{m_i}{2} \sum_j \binom{m_j}{2}}{\binom{n}{2}}} \tag{14}$$

In the equation (14) n is the object count and m_{ij} is confusion matrix.

Another way to calculate the ARI is presented in [2] in equation which is based on equations (10)−(13):

$$ARI = \frac{2(ad - bc)}{c^2 + b^2 + 2ad + (a + d)(c + b)}. \tag{15}$$

2.8 Information theoretic based metrics

Information theoretic based metrics are based on a notion of Entropy and mutual information from information theory. Mutual information (MI) in case of the image segmentations is defined in [2] as follows in equation:

$$H(S_1, S_2) = -\sum_{ij} p(S_1^i, S_2^j) \log p(S_1^i, S_2^j) \tag{16}$$

In the equation above $p(x, y)$ is joint probability. The equation (16) can be also represented in terms of TP, FP, TN and FN as shown in [2] in the following equations:

$$p(S_g^1) = (TP + FN)/n \tag{17}$$

$$p(S_g^2) = (TN + FN)/n \tag{18}$$

$$p(S_t^1) = (TP + FP)/n \tag{19}$$

$$p(S_t^2) = (TN + FP)/n \tag{20}$$

$$N = TP + FP + TN + FN \tag{21}$$

$$p(S_1^i, S_2^j) = \frac{|S_1^i \cap S_2^i|}{n} \tag{22}$$

$$p(S_1^1, S_2^1) = \frac{TP}{n} \tag{23}$$

$$p(S_1^1, S_2^2) = \frac{FN}{n} \tag{24}$$

$$p(S_1^2, S_2^1) = \frac{FP}{n} \tag{25}$$

$$p(S_1^2, S_2^2) = \frac{TN}{n} \tag{26}$$

2.9 Probabilistic metrics

2.9.1 The Interclass Correlation

The Interclass Correlation (ICC) in the context of segmentation is defined in [2] as in equation below:

$$ICC = \frac{\sigma_S^2}{\sigma_S^2 + \sigma_\epsilon^2} \tag{27}$$

In the equation above σ_S is variance caused by differences between segmentations and σ_ϵ denotes variance caused by differences between the points in the segmentations. In practice as shown in [2] the ICC can be calculated using equations:

$$ICC = \frac{MS_b - MS_w}{MS_b + (k-1)MS_w}, \tag{28}$$

where MS_b is representing mean squares between segmentations and MS_w denotes mean square within the segmentations:

$$MS_b = \frac{2}{n-1}\sum_x (m(x) - \mu)^2, \tag{29}$$

$$MS_w = \frac{1}{n}\sum_x (f_g(x) - m(x))^2 + (f_t(x) - m(x))^2. \tag{30}$$

2.9.2 Cohen Cappa

Cohen Kappa Coefficient (KAP) is a measure o agreement between the samples that takes chance into account. It can be calculated as shown in [2] as in equations:

$$KAP = \frac{F_a - F_c}{N - F_c}, \tag{31}$$

where:

N is the number of voxels,
$F_a = TP + TN$,
$F_c = \frac{1}{N} \cdot ((TN + FN)(TN + FP) + (FP + TP)(FN + TP))$.

2.10 Spatial distance based metrics

Mahalanobis Distance takes into account correlation of all points in the point cloud, and is calculated as shown in [2] in equations:

$$MHD(X, Y) = \sqrt{(\mu_x - \mu_y)^T S^{-1} (\mu_x - \mu_y)} \tag{32}$$

Algorithm 1: Calculating TP, FP, FN, TN

```
boolGold = goldGPU[i]==numberToLookFor
boolSegm = segmGPU[i]==numberToLookFor
x= threadIdxX()
y = threadIdxY()
z = (boolGold & boolSegm + boolSegm +1)
shmemblockData[x,y,z] +=(boolGold — boolSegm)
```

In this equation μ_x, μ_y are means of point clouds X, Y that represents segmentations that we compare. Covariance matrix S is calculated as weighted matrix:

$$S = \frac{n_1 S_1 + n_2 S_2}{n_1 + n_2}, \tag{33}$$

where S_1 and S_2 constitutes covariance matrices of voxel sets and n_1 and n_2 represents number of voxels in each set.

3 Data preparation

Programming model is based on the two phase metric evaluation:

- First phase is invoked as a preparation step in order to calculate variables that are constant across kernel given image array dimensions. Those constants include thread block dimensions and number of required thread blocks to optimize occupancy using Occupancy API. Other constants are mainly related to precalculation of loop sizes and appropriate GPU memory allocations. Preparation step is designed to be invoked once for each dataset and can be potentially cached and reused given image array type and GPU hardware will not change.
- Second phase is invoked with an image array and gold standard segmentation together with variables calculated in the preparation step. Additionally to enable reliable calculation in multiple function invocations all data structures are set to initial values (ussually 0).

3.0.1 Algorithm to calculate TP, FP, FN and TN

In order to calculate most metrics that are presented below one needs to calculate TP, FP, FN and TN first. In order to be able to use the GPU acceleration the ***Algorithm 1*** would be used as a CUDA kernel applied to the ground truth and the algorithm output arrays. Each block will iterate over number of elements that will be calculated as ceiled division of total number of voxels with number of available thread blocks.

Crucial part of the impementation is presented in ***Algorithm 1***. To improve readability some implementation details are omitted in case a reader would be intrested in them one is invited to inspect the source code. Algorithm is executed inside the parallelized loop where i is linear index and is calculated on the basis of the id of given thread and thread block.

Variable $goldGPU$ is the GPU array holding gold standard segmentation, while $segmGPU$ is the array of the same dimensions but holding the output of a segmentation algorithm we want to compare with gold standard. $shmemblockData$ holds in shared memory sums of FN, FP and TP, each thread has distinct 3 entries for each sum. Line 1 $boolGold$ is true if in a given location of $goldGPU$ $numberToLookFor$ can be found, analogically the same is stored for $segmGPU$ in $boolSegm$. $goldGPU$ and $segmGPU$ are accessed using linear index to simplify contiguous memory access. Next the indexes for accesing shared memory are calculated in line 3 and 4 indexes in x and y dimension are set to be identical as thread indicies in 2 dimensional thread block.

The equation $boolGold\&boolSegm+boolSegm+1$ (line 5) will evaluate to 3 for TP, 2 for FP and 1 for either FN or TN. However in case of the TN the equation $(boolGold|boolSegm)$ (line 6) will evaluate to 0 hence will not cause any change in FN count. Such way of accessing the array will avoid branching statements which is important to increase performance in GPU architectures. The sum of TN will be obtained in the end by subtracting from total number of voxels sum of TP, FP and FN.

After completing above algorithm next step is reduction of accumulated values. Reduction is achieved in 3 stages first warp level reduction, than shared memory reduction across thread block and atomic reduction across all GPU. Such reduction algorithm is proven to be usually most effective in reducing overall time of communications between threads.

On the basis of calculated above TP, FP, FN, TN and algorithms presented by Taha et all [2] which are summarized in the appendix one can easily calculate multple metrics like Dice Coefficient, Jaccard Coefficient, Global Consistency Error, Volume Metric, Rand Index,Kohen Cappa, Mutual Information and Variation Of Information. Details of how to use exactly those functions is described in the GitHub readMe file.

Last two described metrics Interclass Correlation and Mahalanobis distance needs separate kernels in order to be calculated. However Interclass Correlation algorithm is similar to the kernel described above, with the exception of using cooperative kernel methods (grid synchronization) that will be described also below. Hence detailed explanation will not be presented.

3.0.2 Mahalanobis distance

All of the metrics described above are invariant to the location of non overlapping voxels − i.e. it matters only how many voxels have the same or different labels in both of the arrays, but not how they are distributed. Depending on the algorithm it may lead to severe misrepresentation of actual algorithm performance. In order to remedy this one should take into account relative positions of the voxels − and metrics that do it are called distance metrics. Most commonly used distance metrics in case of medical image segmentations are Mahalanobis distance and Hausdorff distance, the latter will be described in separate work.

Algorithm 2: Calculating Mahalanobis distance

Data: goldGPU
Data: segmGPU
Data: numberToLookFor
1 boolGold = goldGPU[x,y,z]==numberToLookFor
2 boolSegm = segmGPU[x,y,z]==numberToLookFor
3 **if** *boolGold* **then**
4 | addSumXYZGold()
5 **end**
6 **if** *boolSegm* **then**
7 | addSumXYZSegm()
8 **end**
9 syncGrid()
10 **if** *boolGold* **then**
11 | getVariancesAndCovGold()
12 **end**
13 **if** *boolSegm* **then**
14 | getVariancesAndCovSegm()
15 **end**
16 syncGrid()
17 $a = sqrt(varianceX)$
18 $b = (covarianceXY)/a$
19 $c = (covarianceXZ)/a$
20 $e = sqrt(varianceY - b^2)$
21 $d = (covarianceYZ - (c * b))/e$
22 $f = (meanXgold - meanXsegm)/a$
23 $g = (meanYgold - meanYsegm) - b * f)/e$
24 $h = (meanZgold - meanZsegm) - g * d - f * c)/sqrt(varianceZ - c^2 - d^2)$
25 $result = sqrt(f^2 + g^2 + h^2)$
26 **return** *result*

Mahalanobis distance is calculated based on covariance matrix of x, y, z coordinates of points in both of the arrays as described in ***Algorithm 2***:

- Line 1 and subsequent shows similarly like in previous algorithm defining Boolean values on the basis whether given entries in tested arrays are meeting our equality predicate.

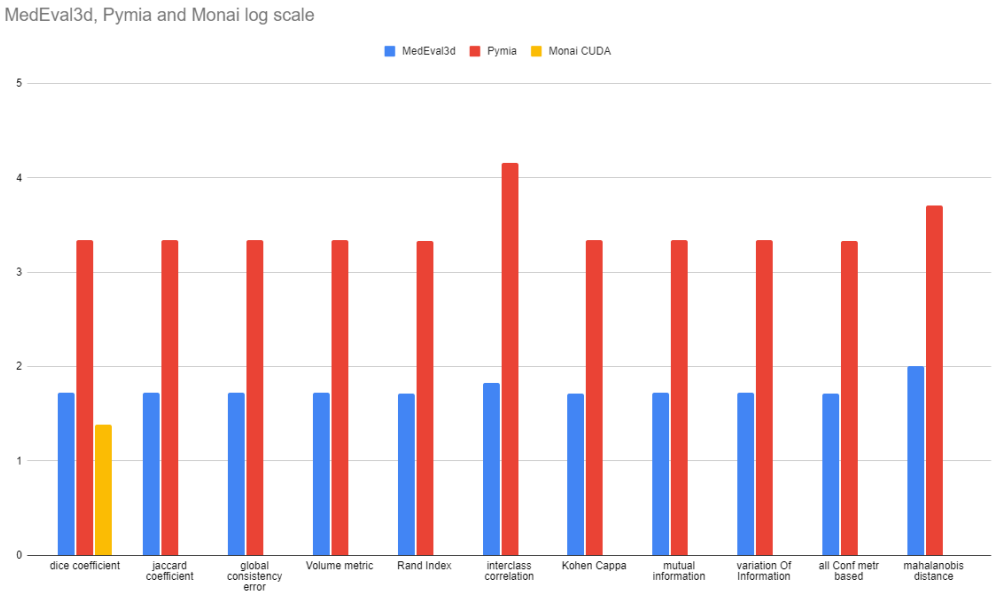

Figure 1: Comparison of median times needed to calculate given metrics in log scale, for Monai only CUDA accelerated algorithm was taken into account. Dimensionality of data was $(512 \times 512 \times 826)$

However in this case we are accessing arrays using x, y, z coordinates instead of linear index.

- Line 3 $-$ separately we need to increment number of voxels that meet set equality predicate for both input arrays, and for all x, y and z coordinates separately. Those sums will be later used to acquire means of x, y, z coordinates of voxels that meet predicate.
- Line 9 and 16 $-$ $syncGrid()$ is a function that synchronizes all of the threads on the device.
- Line 10 $-$ using means calculated above we calculate all of the entries needed to obtain 3×3 covariance matrix of x, y, z coordinates of voxels that meet predicate. Hovewer becouse of the fact that covariance matrix is symmetric number of calculated variables is smaller than number of entries in the covariance matrix of interest

Original calculation as shown in [2] was shown as series of linear algebra operations that includes matrix inversion. In order to optimize the calculation authors had obtained the same result by first doing Cholesky decomposition and then forward substitution. Both operations were unrolled and specialized to the 3×3 covariance matrix. Specialized Cholesky decomposition is seen from line 17 to 21 and forward substitution from line 22 to line 24.

Table 1: Presentation of median times needed to calculate given metrics. Dimensionality of data was ($512 \times 512 \times 826$)

	MedEval3d $[ms]$	Pymia $[ms]$
Dice Coefficient	52.8	2181.9
Jaccard Coefficient	52.9	2174.4
Global Consistency Error	52.7	2181.4
Volume metric	52.8	2172.4
Rand Index	52.0	2153.2
Interclass Correlation	67.7	14513.1
Kohen Cappa	52.0	2172.8
Mutual Information	52.8	2173.4
Variation Of Information	52.8	2188.0
all Conf metr based	51.8	2169.1
Mahalanobis Distance	101.8	5137.9

4 Experiments

All experiments were conducted on Windows PC (Intel Core I9 10th gen., GeforceRTX 3080), and Data from CT-ORG [9] dataset, on image of size ($512 \times 512 \times 826$). Time needed to calculate metrics was estimated in case of Julia code using BenchamrkTools.jl [10]. For testing python libraries internal python module timeit was utilized. Results of experiments mentioned above are summarized in Figure 1, and in Table 1. In all cases data was already in RAM memory for CPU computation or in GPU memory for CUDA computations − hence memory transfer times were not included.

As visible in the implementation of CUDA acceleration of described package in most cases led to from 40 up to 214 times shorter execution times. The only exception is in case of CUDA accelerated Monai Dice metric algorithm MedEval3D is slower (24.1 ms vs 52.8 ms). However from all of the metrics described Monai implements only Dice metric, although admittedly most of others could be potentially calculated from Monai confusion matrix. What is also worth pointing out is a field in a table named "all Conf metr based" this tested calculating all of the metrics jointly apart from interclass correlation and mahalanobis distance − time of execution in such case both in case of MedEval3d and Pymia algorithms was similar to calculation of just one of those metrics. This can be explained by the fact that in all of those cases the most computation intensive work is related to calculation of confusion matrix, and this calculation is reused in both packages between diffrent metrics.

5 Conclusions

As clearly visible in performed experiments CUDA accelerated algorithm gives vast performance improvement, comparing to the state of the art CUDA algorithm provided by Monai [5], MedEval3D did not lead to any improvements. However in case of most metrics described in [2],to the best knowledge of the author there are no easy to use CUDA accelerated metrics implementations, and in this cases current work brings vast increases in performance, with additional benefit of providing interface in rapidly growing in popularity Julia [11] programming language.

Acknowledgments

Development of this package would not be possible without help from Julia language community particularly Tim Besard.

References

[1] F. Renard, S. Guedria, N. De Palma, and N. Vuillerme, "Variability and reproducibility in deep learning for medical image segmentation," *Scientific Reports*, vol. 10, no. 1, p. 13724, Aug. 2020. [Online]. Available: https://doi.org/10.1038/s41598-020-69920-0

[2] A. A. Taha and A. Hanbury, "Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool," *BMC Medical Imaging*, vol. 15, no. 1, p. 29, Aug. 2015. [Online]. Available: https://doi.org/10.1186/s12880-015-0068-x

[3] J. Mitura and E. B. Chrapko, "3D Medical Segmentation Visualization in Julia with MedEye3d," *Zeszyty Naukowe WWSI*, vol. 15, no. 25, pp. 57–67, 2021. [Online]. Available: https://doi.org/10.26348/znwwsi.25.57

[4] A. Jungo, O. Scheidegger, M. Reyes, and F. Balsiger, "pymia: A Python package for data handling and evaluation in deep learning-based medical image analysis," *Computer Methods and Programs in Biomedicine*, vol. 198, p. 105796, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169260720316291

[5] The MONAI Consortium, "Project MONAI," 2020. [Online]. Available: https://monai.io/

[6] A. Paszke, S. Gross, F. Massa *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[7] K. Y. Yeung and W. L. Ruzzo, "Details of the Adjusted Rand index and Clustering algorithms Supplement to the paper "An empirical study on Principal Component Analysis for clustering gene expression data" (to appear in Bioinformatics)," 2001. [Online]. Available: https://faculty.washington.edu/kayee/pca/supp.pdf

[8] F. Zaidi, D. Archambault, and G. Melançon, "Evaluating the Quality of Clustering Algorithms Using Cluster Path Lengths," in *Advances in Data Mining. Applications and Theoretical Aspects*, P. Perner, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 42–56. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-14400-4_4

[9] B. Rister, D. Yi, K. Shivakumar, T. Nobashi, and D. L. Rubin, "CT-ORG, a new dataset for multiple organ segmentation in computed tomography," *Scientific Data*, vol. 7, no. 1, p. 381, Nov. 2020. [Online]. Available: https://doi.org/10.1038/s41597-020-00715-8

[10] J. Revels, "BenchmarkTools.jl," 2021. [Online]. Available: https://github.com/JuliaCI/BenchmarkTools.jl

[11] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: https://doi.org/10.1137/141000671