Need for speed

Achieving fast image processing in acute stroke care

Sales Barros, R.

Publication date
2022

Document Version
Final published version

NEED FOR SPEED

Achieving fast image processing in acute stroke care

Renan Sales Barros

Need for speed

Achieving fast image processing in acute stroke care

Renan Sales Barros

Printing: ProefschriftMaken || www.proefschriftmaken.nl

Need for speed

Achieving fast image processing in acute stroke care

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op dinsdag 1 november 2022, te 15.00 uur

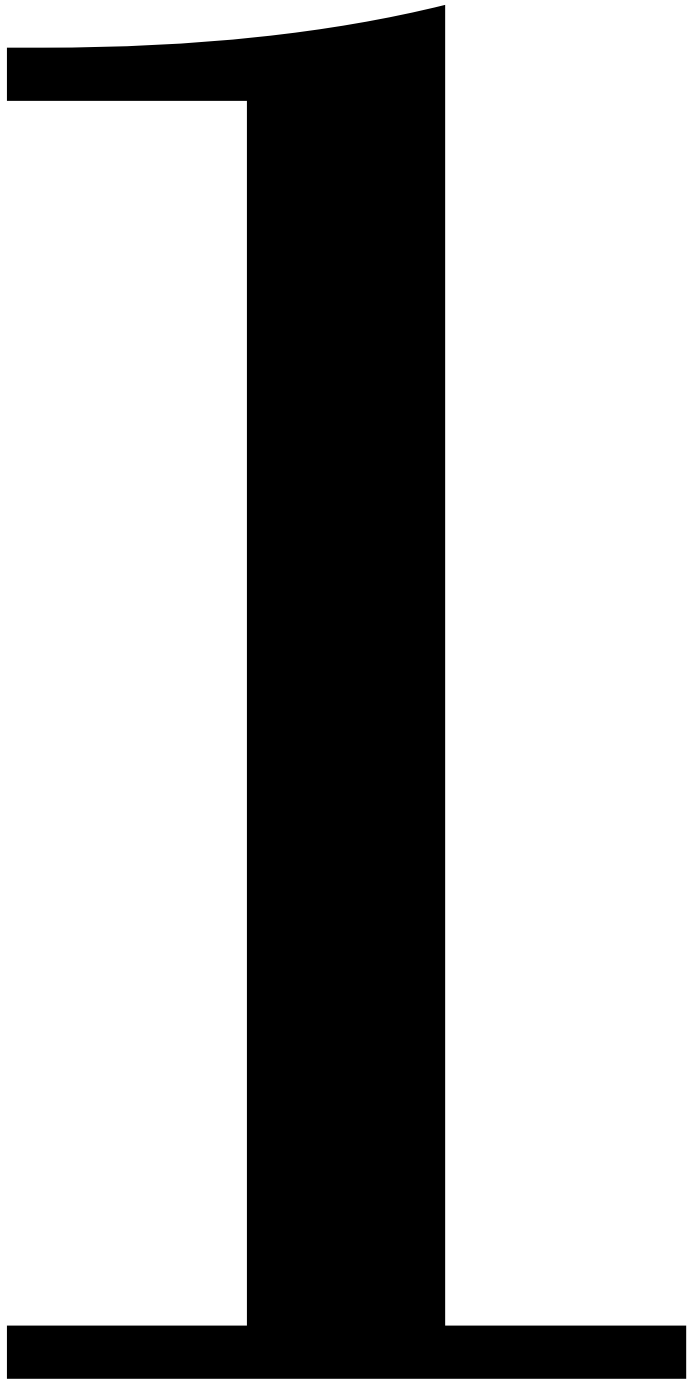
door Renan Sales Barros
geboren te Santana do Ipanema

Promotiecommissie

Promotores:	prof. dr. ing. A.H.C. van Kampen AMC-UvA prof. dr. H.A. Marquering AMC-UvA
Copromotores:	dr. S.D. Olabarriaga AMC-UvA
Overige leden:	prof. dr. I. Išgum AMC-UvA prof. dr. Y.B.W.E.M. Roos AMC-UvA prof. dr. ir. A.G. Hoekstra Universiteit van Amsterdam dr. B.J. Emmer AMC-UvA prof. dr. ir. H.W.A.M. de Jong Universiteit Utrecht prof. dr. R.V. van Nieuwpoort Universiteit van Amsterdam

Faculteit der Geneeskunde

Contents



Chapter 1

Introduction

Introduction

Stroke

A stroke patient loses around 22 days of life for every minute without treatment [1]. This is equivalent to losing more than 4 years of life for every hour of delayed treatment. Therefore, fast treatment in stroke is crucial to increase the chances of good outcome. These numbers are even more impressive when we consider how frequent stroke affects people globally. Stroke is the second leading cause of death and the leading cause of morbidity. One in every 4 people older than 25 will have a stroke during their lifetime [2]. Furthermore, the incidence of stroke is expected to increase in the coming decades because of aging and population growth.

There are two main types of stroke: hemorrhagic and ischemic. Hemorrhagic stroke patients represent 13%-15% of all stroke patients. Hemorrhagic stroke patients have a bleeding in their brain usually due to the rupture of an aneurysm. The remaining 85%-87% of the strokes are ischemic strokes. Ischemic strokes are caused by the obstruction of a cerebral artery by a blood clot.

Stroke workflow

An important moment during the assessment of a suspected stroke patient is determining if the patient suffers a hemorrhagic stroke or ischemic stroke. This is predominantly done via non-contrast computed tomography (NCCT) scans. If hemorrhagic stroke diagnosis is discarded then the patient should be immediately treated with intravenous thrombolysis (IVT). Up until 2015, IVT was the only proven treatment for ischemic stroke patients. Successful IVT treatment dissolves the blot clot and reestablishes the cerebral blood flow. However, IVT is not likely to be effective in patients with large vessel occlusions (LVOs). Fortunately, it has been shown in 2015 that combining IVT with the mechanical removal of the blood clot is an effective treatment option for ischemic stroke patients with LVOs. This mechanical removal of the blood clot, which is also known as endovascular treatment or endovascular therapy (EVT), can only be performed in specialized hospitals. To determine if the patient is eligible for EVT it is necessary to know the location of the vessel occlusion. Determining the location of a blood clot is typically done in computed tomography angiography (CTA) scans.

Originally, the time window available for EVT was 6h from the onset of the symptoms. However, more recent studies have shown that EVT can be effective to some ischemic stroke patients beyond these 6h window. Determining if a patient is eligible to EVT outside this 6h window is primarily done by checking if the infarct core volume in that patient is above a certain threshold. This infarct core volume is generally measured via computed tomography perfusion (CTP) imaging. More recently, the status of

the collateral flow has been shown as an additional clinical decision-making tool for determining the eligibility of ischemic stroke patients for EVT [3].

Quantitative imaging biomarkers

The assessment of stroke patients is heavily depended on computed tomography (CT) imaging. Traditionally, CT scans are inspected by radiologists with little or no help of image processing or computer vision algorithms. The time spent on interpreting CT scans is a considerable part of the EVT workflow [4]. As can be seen in Figure 1, the study by Ng et al. [4] reported that approximately one hour is needed from the moment the CT scan is acquired to the moment a request is made to transfer the stroke patient to a hospital where EVT can be performed. Furthermore, the evaluation of CT scans by human experts suffers from high variability [5] and, in the case of inexpedient radiologists, it is also prone to unsatisfactory accuracy.

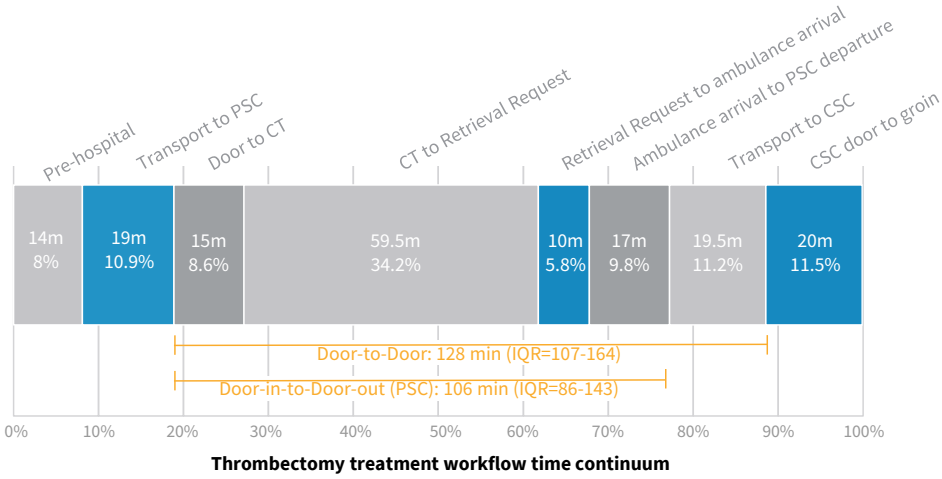


Figure 1: Breakdown of thrombectomy treatment workflow time, adapted from Ng *et al*. [4].

Automated analysis of CT scans has the potential to improve the stroke workflow by reducing variability, reducing the time spent on interpreting these scans, and also by increasing accuracy. These automated methods used to support the clinical decision-making in stroke are also referred to as quantitative imaging (QI) biomarkers [6]. In a general way, a QI biomarker is an objective characteristic extracted from an in vivo image which is used as an indicator of a normal biological process, a pathogenic process, or response to a medical intervention. Designing and implementing QI biomarkers for stroke often requires the use of advanced image processing and computer vision techniques. Furthermore, it is not uncommon that CT scans have around 1500 slices and each of these slices have 512x512 voxels. Image modalities such as multiphase CTA

and CTP are composed of several CT acquisitions at different times which results in a several gigabytes of image data. The combination of advanced processing techniques and large input images often results in a trade-off between accuracy and speed. Unfortunately, accurate QI biomarkers with slow processing time, or fast QI biomarkers with low accuracy have little practical value in the clinical treatment decision of acute stroke patients.

High performance computing

High performance computing (HPC) commonly refers to the set of techniques for combining computing power to achieve much higher performance than what a typical desktop computer or workstation can offer. HPC is the go-to solution for tackling the computational challenges faced by medical research. HPC has been successfully used for supporting biomarker discovery in cancer research [7]. In the context of medical image analysis, HPC has been used in image registration, segmentation, reconstruction, filtering, and classification [8]. HPC makes use of computing platforms such as cloud computing, computer clusters, FPGAs, GPUs, massive parallel processors, multicore CPUs, etc. Often, HPC is achieved via the combination of many of these computing platforms.

Aim and outline

The aim of this thesis is to investigate the use HPC techniques for implementing QI biomarkers which are suitable for clinical decision making in acute stroke care. That is, we want to demonstrate that HPC allows the development of QI biomarkers with the accuracy and speed that are required by the clinical stroke workflow.

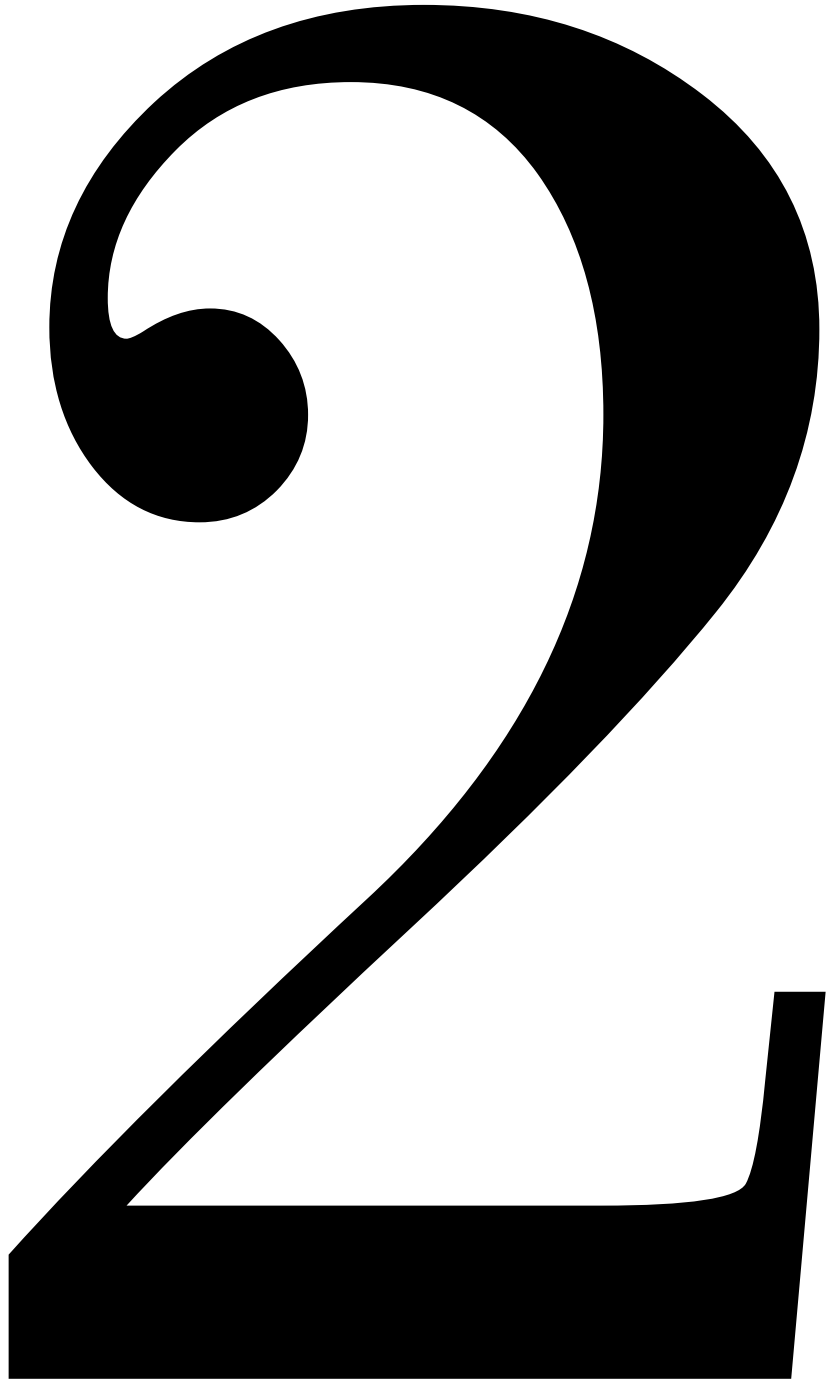
In the first part of this thesis, we investigate different HPC techniques and its applicability to image analysis in stroke. In Chapter 2, we evaluate the use heterogeneous platforms for delivering the high-performance capabilities which are needed by some stroke image processing applications. In Chapter 3, we propose a novel data compression technique that allows the efficient processing of CTP images in GPUs. Subsequently, we compare the effect of such data compression technique in a well stablished image processing CTP algorithm in Chapter 4. In Chapter 5, we present a cloud-based platform that enables fast medical image exchange and HPC image processing in the context of acute stroke care.

The second part of this thesis focuses on developing or improving QI biomarkers for stroke. These QI biomarkers are implemented by using the HPC practices discussed in Part 1. In Chapter 6, we propose a new method for subarachnoid hemorrhage segmentation based on convolutional neural networks. In Chapter 7, a new approach for the segmentation of infarcted brain tissue in follow-up CT scans based on convolutional

neural networks is proposed. In both Chapter 6 and Chapter 7, the techniques used for training and deploying convolutional neural networks are grounded on HPC technologies such as general-purpose computing on GPUs. To conclude, we wrap up all topics considered in this thesis in a general discussion in Chapter 8.

References

1. Saver, Jeffrey L. "Time is brain—quantified." *Stroke* 37.1 (2006): 263-266.
2. GBD 2016 Lifetime Risk of Stroke Collaborators. "Global, regional, and country-specific lifetime risks of stroke, 1990 and 2016." *New England Journal of Medicine* 379.25 (2018): 2429-2437.
3. Powers, William J., *et al.* "Guidelines for the early management of patients with acute ischemic stroke: 2019 update to the 2018 guidelines for the early management of acute ischemic stroke: a guideline for healthcare professionals from the American Heart Association/American Stroke Association." *Stroke* 50.12 (2019): e344-e418.
4. Ng, Felix C., *et al.* "Deconstruction of interhospital transfer workflow in large vessel occlusion: real-world data in the thrombectomy era." *Stroke* 48.7 (2017): 1976-1979.
5. Grotta, James C., *et al.* "Agreement and variability in the interpretation of early CT changes in stroke patients qualifying for intravenous rtPA therapy." *Stroke* 30.8 (1999): 1528-1533.
6. Abramson, Richard G., *et al.* "Methods and challenges in quantitative imaging biomarker development." *Academic radiology* 22.1 (2015): 25-32.
7. Blayney, Jaine, *et al.* "Biomarker discovery, high performance and cloud computing: a comprehensive review." *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2015.
8. Gulo, Carlos ASJ, Antonio C. Sementille, and João Manuel RS Tavares. "Techniques of medical image processing and analysis accelerated by high-performance computing: A systematic literature review." *Journal of Real-Time Image Processing* (2019): 1-18.



Chapter 2

Heterogeneous platform programming for high performance medical imaging processing

Renan Sales Barros[a], Sytse van Geldermalsen[b], Anna M.M. Boers[a,c,d], Adam S. Z. Belloum[b], Henk A. Marquering[a,c], and Silvia D. Olabarriaga[a]

[a] Biomedical Engineering & Physics, Academic Medical Center, University of Amsterdam, Meibergdreef 9, 1105 AZ, Amsterdam, The Netherlands
[b] Department of Computational Science, University of Amsterdam, Science Park 107, 1098 XG, Amsterdam, The Netherlands
[c] Department of Radiology, Academic Medical Center, University of Amsterdam, Meibergdreef 9, 1105 AZ, Amsterdam, The Netherlands
[d] University of Twente, Postbus 217, 7500 AE, Enschede, The Netherlands

- Barros, Renan Sales, et al. "Heterogeneous platform programming for high performance medical imaging processing." *European Conference on Parallel Processing*. Springer, Berlin, Heidelberg, 2014.
- First author, workshop proceedings, https://link.springer.com/chapter/10.1007/978-3-642-54420-0_30

Abstract

Medical imaging processing algorithms can be computationally very demanding. Currently, computers with multiple computing devices, such as multi-core CPUs, GPUs, and FPGAs, have emerged as powerful processing environments. These so called heterogeneous platforms have potential to significantly accelerate medical imaging applications. In this study, we evaluate the potential of heterogeneous platforms to improve the processing speed of medical imaging applications by using a new framework named FlowCL. This framework facilitates the development of parallel applications for heterogeneous platforms. We compared an implementation of region growing based method to automated cerebral infarct volume measurement with a new implementation targeted for heterogeneous platforms. The results of this new implementation agree well with the original implementation and they are obtained with significant speed-up comparing to the sequential implementation. Keywords: dataflow, framework, heterogeneous computing, heterogeneous platforms, medical imaging processing, OpenCL, parallel programming.

1 Introduction

In medical imaging applications large amounts of data must be processed quickly and accurately, which requires the usage of high performance computing systems. Commodity computer architectures are rapidly developing into systems with multi-core CPUs and with additional accelerated hardware devices such as graphics processing units (GPUs) and field programmable gate arrays (FPGAs). These heterogeneous platforms provide a new alternative to design and implement computationally demanding applications. Consequently, the computation power provided by these heterogeneous platforms should be explored for medical image processing.

Expertise of new programming constructs and concepts is however required for application developers to effectively utilize these platforms. The OpenCL [5] technology was developed with the aim of facilitating heterogeneous platforms usage. OpenCL includes a language for writing functions, called kernels, that execute on diverse computing devices. It also includes an application programming interface (API) that is used to control the heterogeneous platforms. A benefit of OpenCL is that the kernels that are coded according to this standard can run on different devices without any modification. This makes it possible to take advantage of computationally powerful devices that are well suited for different tasks. Nevertheless, OpenCL still requires application developers to deal with low level concerns such as the overhead of the code, memory management, and synchronization. In order to evaluate the potential of heterogeneous platforms in medical imaging processing, we needed an easier programming platform. A new framework named FlowCL was developed to provide an intuitive way to create applications utilizing heterogeneous platforms. This framework eliminates the OpenCL API usage, but maintains the OpenCL programming language for writing kernels. A brief description of this framework is presented in Section 2.

We used the FlowCL framework to implement a modified version of a previously developed method of automated measurement of cerebral infarct volume of patients after acute ischemic stroke. This method, which was developed and validated by Boers *et al.* [3], was modified for heterogeneous platforms. In Section 3, we explain the automated cerebral infarct volume measurement method and the modification implemented in this study. We compared the execution times of the original (sequential) implementation with the new parallel implementation for heterogeneous platforms using FlowCL. We also evaluated the differences between the results of both implementations in Section 4. Finally, the conclusions regarding this work and future improvements are presented in Section 5.

2 FlowCL Framework

During the development of OpenCL applications, programmers have to deal with a low level C library, which requires specialized expertise for effective and efficient code development. There are various frameworks to mitigate this problem. For example, the Many GPUs Package (MGP) [2] was built on top of the Virtual OpenCL Layer (VCL). VCL is a transparent layer that accesses and manages OpenCL devices in clusters and presents these devices as a single node. MGP is a layer that facilitates the programing using clusters by hiding low level functions. A library named Maestro [10] also tries to reduce the complexity of OpenCL applications development by providing functions for automatic data transfer and task decomposition across OpenCL devices. However, MGP and Maestro do not use extensive optimizations strategies. A more complete framework is StarPU [1], which is a runtime system capable of scheduling tasks over heterogeneous devices using several optimizations strategies. However, the framework API uses the C programming language and this hampers the usage of high level programming concepts.

FlowCL is a new high level framework that supports rapid prototyping and development with OpenCL, which makes it possible to closely control the execution across all OpenCL devices on one computer system. More details about FlowCL are found in [4]. It hides all low level calls to the OpenCL library API from the application developer. Only the OpenCL kernel code that is designed to run on a selected device must be provided to the framework. Also, FlowCL provides an object-oriented declarative API to easily build applications with the concept of dataflow. The programmer simply declares a set of memory objects and operations. Each operation runs each single kernel function on any available device. This framework automatically applies optimization strategies such as overlapping communication and computation, and asynchronous data transfers and kernel executions.

To use the FlowCL framework, application developers just have to deal with four classes of objects: memory, context, device, and operation. Figure 1 illustrates the relationships between these classes. By having only four classes with limited relationships, FlowCL provides a simpler approach that is easier to understand than OpenCL.

In short, FlowCL framework addresses the following key aspects: it facilitates application development with OpenCL; it provides an object-oriented API to build applications with the dataflow concept; it eliminates the OpenCL API, except for kernel code; it automatically manages all devices on heterogeneous platforms; it supports concurrent kernel execution and asynchronous data transfers; and it supports multiple operating systems.

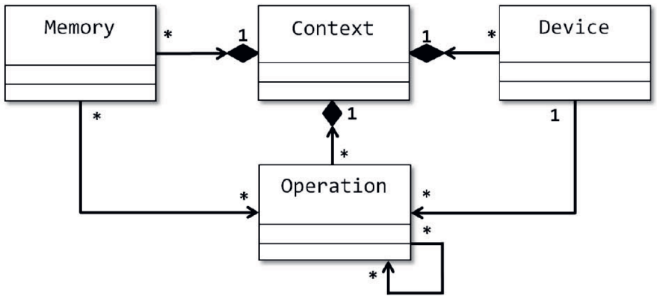


Figure 1: FlowCL cardinality diagram. A Context is instantiated with kernel codes; these codes are usable on all available computing devices. Memory objects act as arguments for operations; they are created by a context with a given size and become available to all devices in this context. Device represents a computing device. Operation runs a specified kernel function on a selected device.

The framework is designed to run in the C++ language and only requires the FlowCL header file inclusion. To illustrate FlowCL usage, the following source code is shown:

```
#include ”FlowCL . hpp”
using namespace FlowCL ;

int main ( )
{
        Context con;
        con. CompileFile (”source. cl” ) ;

        Memory memcpu = con . CreateMemory (1e8* sizeof (int));
        Memory memgpu = con . CreateMemory (1e8* sizeof (int));

        Operation genrand = con. CreateOperation(con. GetCPUDevice ( ), ”GenRand” );
        genrand. SetArg (0, memcpu); // CPU already has access to memory
        genrand. SetArg (1, memgpu);
        genrand. SetWorkSize (1e8); // Set finest granularity

        Operation sortcpu = con. CreateOperation(con. GetCPUDevice ( ), ”SortCPU” );
        sortcpu. SetArgDependency(genrand, 0, memcpu); // Wait for genrand
        sortcpu. SetWorkSize (1e8 );

        Operation sortgpu = con. CreateOperation(con. GetGPUDevice ( ),”SortGPU”);
        sortgpu. SetArgDependency(genrand, 0, memgpu); // Wait for genrand
        sortgpu. SetArgOutput(0, memgpu);
        sortgpu. SetWorkSize (1e8);
        con. Run( ); // Blocking run
}
```

This example is visualized in Figure 2. The first operation executes on the CPU and generates random numbers that are sorted in the next two operations that execute in parallel on the GPU and the CPU. There is no data transferred to the sort operation that runs on the CPU because this data is readily available. Once the GPU operation is completed, the data is transferred back and the execution is finished.

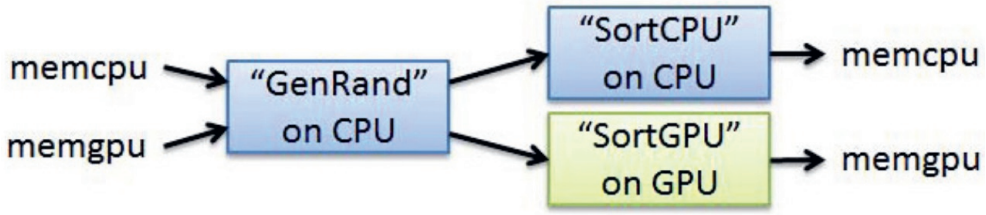


Figure 2: Sample code visual representation.

3 Case Study

In this section we present a case study using heterogeneous computing for measurement of cerebral infarct volume (CIV) of patients with acute ischemic stroke. The CIV has been suggested as an important measure for the effective treatment of these patients [9]. This volume can be manually measured in early follow-up non-contrast CT scans by the delineation of the whole infarct volume. An alternative is to estimate this volume by using the ABC/2 formula, which was originally designed for the estimation of hemorrhage volume [6]. The manual delineation is a tedious and time-intensive task, and the CIV estimation based on the ABC/2 rule only approximates the total CIV. Aiming to address these problems, Boers *et al.* [3] proposed a method to automatically calculate the CIV in follow-up non-contrast CT scans. This method was validated by comparing it with manual delineations performed by experienced radiologists.

The method proposed by Boers *et al.* was implemented using MATLAB [7] and took a long time to run (in the order of 10 min). It uses an intensity-based region growing (IRG) algorithm, which is responsible for more than 95% of the total processing time. To evaluate the potential benefit of heterogeneous computing for this application, we replaced this method with a new version of the IRG algorithm developed with the FlowCL framework. The integration of the new IRG algorithm with the previous MATLAB implementation was straightforward because MATLAB allows external code calls.

In short, the objective of this case study is to understand how heterogeneous platforms can be used and what is their potential value for medical imaging applications. To achieve this objective, we run the method for automated CIV calculation with two different implementations of the IRG algorithm, one based on the original MATLAB code and the other using FlowCL. Below we provide an overview of the complete method for automated CIV measurement used in this case study, and then we describe both the sequential and the parallel IRG implementations.

3.1 Automated Cerebral Infarct Volume Measurement

The automated CIV measurement proposed by Boers *et al.* was designed to process non-contrast CT scans of the whole brain of the patients. The volume measurements are performed for a part of the brain that is segmented from the image using a region growing algorithm (IRG). In this algorithm, a voxel is added to the segmented volume if the difference between its intensity and the average intensity of the segmented volume so far is smaller than a specific threshold. To compute the CIV, this algorithm was repeated for 7 different values of threshold, going from 1.5 until 4.5 with steps of 0.5 Hounsfield units (HU). The algorithm requires a position as starting point (called seed point) in the infarcted lesion. The seed position is set by an experienced radiologist and this assures that the correct infarcted area was selected.

The brain midline is used to prevent the segmented region from leaking to into the contralateral hemisphere, i.e., the region cannot grow into the other side of the brain. This midline is detected based on the geometric center and the most extreme mid-sagittal bone or nasal cartilage structures present on the scan. Also, to avoid leaking into the hypo-attenuated ventricles, the hypo-attenuated region close to the geometric center is segmented and excluded from the segmentation of the infarcted area. All the steps of the segmentation process are illustrated in Figure 3. This process is repeated for 7 thresholds. In the end of this process, the observer must select the best result that most agrees with the scans.

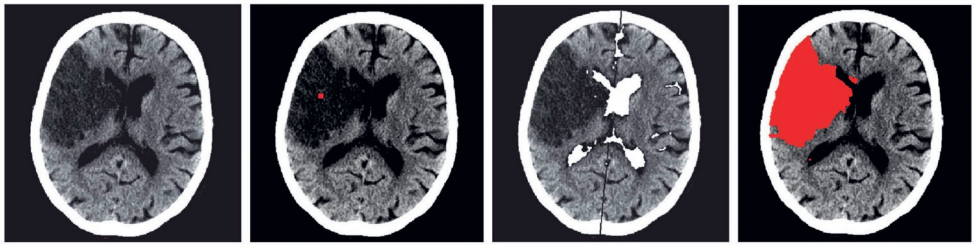


Figure 3: CIV segmentation steps. From the left to right: a CT scan showing an infarct in the right hemisphere (left of the image); the seed position defined by a radiologist; the determined midline and the ventricles segmentation; and the final segmentation representing the CIV.

3.2 Intensity-Based Region Growing Algorithm

Region growing is a segmentation technique to select an image area that is connected according to a specific condition [8]. In intensity-based region growing (IRG), the intensity of the voxel is used as criterion to include or not a voxel to the region. Starting from a given seed point, the IRG algorithm iteratively adds voxels to the region such that the following condition is satisfied: $|I - A| \leq T$, where I is the intensity of the processed voxel and A is the average intensity of the selected image area. The voxel I

must be in the neighborhood of the selected area, and it is included in this area when its intensity is smaller than or equal to the threshold T.

Different neighborhood definitions can be applied (e.g., for 3D images it can take 9 or 26 neighbors into account), and the order in which the voxels are considered for inclusion may influence the final result. The IRG is also sensitive to the chosen threshold T; for this reason, 7 different thresholds are used in the CIV method, and the user can pick the best result.

The sequential implementation of IRG in the original CIV measurement method updates the average of the selected image area immediately after the inclusion of each voxel, and the updated average is used in the test to include the next neighboring voxel. In the parallel implementation of IRG, computing devices in the heterogeneous platform simultaneously process the voxels based on the same value of A. The average intensity A is only updated after all the neighbors of a given voxel are considered for inclusion. Therefore the sequential and the parallel algorithms perform inclusion tests based on potentially different values of A, and can deliver different results.

4 Experimental Results and Comparisons

To evaluate the speed-up obtained with the heterogeneous platform, 53 CT scans were processed in two different hardware configurations with the original and the GPU implementations of the automated CIV measurement method. The complete method was executed in both cases, however for timing purposes only the IRG part was considered. Both hardware configurations have 12 CPU cores and 192 GPU cores, however one is slower than the other – see Table 1.

Table 1: Hardware configuration.

Hardware Detail	Configuration 1	Configuration 2
CPU Name	Intel Core i7-3930K	Intel Xeon E5-2620
CPU Clock	3.20 GHz	2.00 GHz
GPU Name	NVIDIA GeForce GTX 550 Ti	NVIDIA Quadro K600
GPU Clock	900 MHz	876 MHz
GPU Memory Clock	4104 MHz	1782 MHz
GPU Driver Version	9.18.13.1106	9.18.13.2000

The CT scans include the entire brain and were performed with thin-section acquisition by using 8 different multi-section CT scanners with at least 16 sections, but mostly

with 64 or more sections. The 32-bit MATLAB version 8.0.0.783 (R2012b) was used to run the MATLAB code. Microsoft Visual C++ 2010 was used to compile the region growing algorithm for heterogeneous platforms. These software were executed on 64-bit Microsoft Windows 7 Enterprise operating system on both hardware configurations. Execution times were measure before and after calling the IRG function in the MATLAB code. Note that this time includes overhead of internal function call for the sequential implementation, as well as for the external call of the program for the heterogeneous platform implementation.

All the CT scans were analyzed using exactly the same parameters for 7 threshold values (1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5). Parameters that must be manually configured, such as the region growing seed position, were defined only once and used in all runs of the method. Because different threshold values have a great influence in the size of the segmented volume and, consequently, also in the algorithm execution time, we compare the execution time separately for each threshold value.

Figure 4 shows the speed-up factor for the parallel respectively to the original IRG implementation. As we can see, speed-ups of 36 times were obtained. In general, larger gains are obtained for higher threshold values. Higher thresholds produce bigger segmented volumes, which require more computations and, consequently, result in more expressive speed gains. For lower thresholds, the execution time varies among different scans (larger standard deviation). Small thresholds generate smaller segmented volumes, which are more sensitive to the inclusion of neighboring voxels.

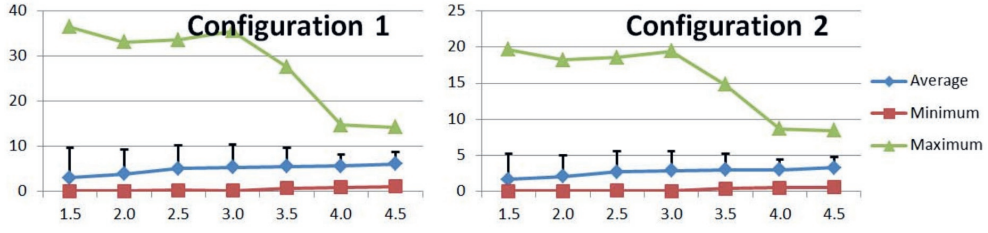


Figure 4: Speed-up on heterogeneous platforms (vertical axis) for each threshold value (horizontal axis). Bars indicate standard deviation from the mean speed-up calculated for 53 scans.

Also note that, due to this sensitivity regarding the small volumes, the implementation for heterogeneous platforms can be slower than the original implementation. Note however that only a minor performance reduction is noticed. In most situations the performance of the new implementation is better as presented in figure 5. We must highlight that the automated CIV measurement requires the region growing algorithm to run with 7 different threshold values and, because of this, the performance gains

obtained for higher thresholds values compensate for the loss for smaller thresholds in the total processing time. In no case the new implementation had a total processing time slower than the original implementation when all 7 thresholds are considered. The new implementation was faster in 82% of the scans using the hardware configuration 1 and in 75% of the scans using the hardware configuration 2.

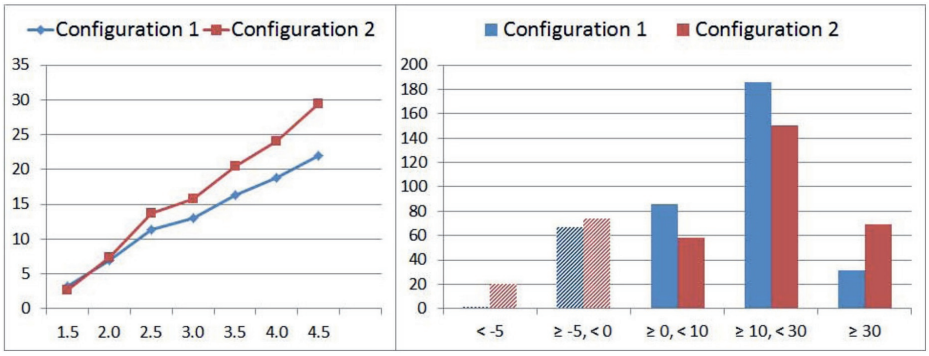


Figure 5: Differences in execution time between both implementations (new - old). *Left:* **Average difference in seconds (vertical axis) for each threshold value (horizontal axis) calculated for 53 scans.** *Right:* **Histogram of differences in execution times in seconds for all thresholds. The negative ranges indicate the runs where the original implementation was faster than the new implementation.**

To evaluate the differences in the quality of results obtained with both implementations we calculate the Dice coefficient for each threshold value individually - see Table 2. Similarly to what we found in the execution time analysis, the greater differences are measured for smaller thresholds. However, this variation in the results does not have a great impact in the method, because the final segmentation must be selected by a human observer which will filter out the segmentations that are not consistent with the images. Moreover, usually the selected segmentation is generated with one of the middle threshold values. The extreme threshold values are used as a safe margin to assure that the most adequate segmentation will be inside this interval.

As shown in Table 2, the Dice coefficients for the threshold between 2.5 and 4 are higher than 0.9. These results indicate good agreement between segmentations when compared to variations found in results obtained with manual segmentation by experts. For example, during the validation of the original method, [3] found that the Dice coefficient for segmentations manually defined by two experienced radiologists were 0.84 ± 0.08 ranged from 0.63 to 0.94 [3].

Table 2: Dice coefficients for each threshold.

Threshold Value:	1.5	2.0	2.5	3.0	3.5	4.0	4.5
Maximum:	0.99994	1.00000	1.00000	1.00000	0.99247	0.99141	0.97592
Average:	0.85936	0.89620	0.94856	0.93499	0.91252	0.91199	0.89875
Minimum:	0.34349	0.25384	0.68682	0.84418	0.82026	0.76991	0.75164
Standard Deviation:	0.18170	0.16985	0.07482	0.05082	0.05052	0.06906	0.07345

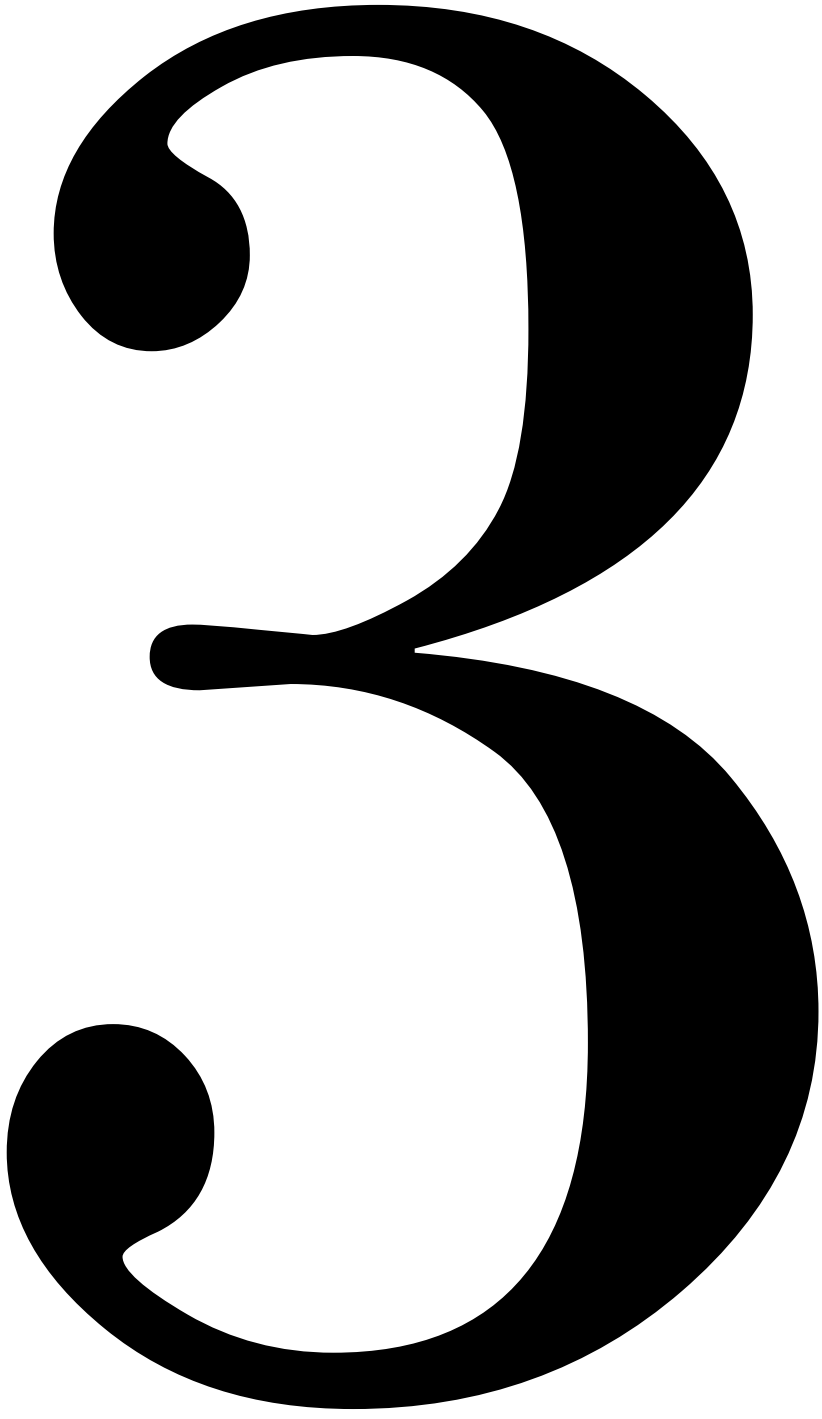
5 Conclusions and Future Work

In this work we presented how the FlowCL framework, which was developed for intuitive heterogeneous platform programing, was used in a medical imaging application. Part of a previously developed and validated method for automated cerebral infarct volume measurement was adapted for heterogeneous platforms using the FlowCL framework. Only the code related with the intensity-based region growing algorithm was modified. All other pieces of code and software used in the method were not modified. We compared the two implementations of the automated CIV measurement method in order to investigate the potential of heterogeneous platform in medical imaging applications. The results of the implementation for heterogeneous platform were obtained faster and were also consistent with the results of the original implementation. This study shows that heterogeneous platforms can increase performance in medical imaging applications. This indicates that other computationally demanding medical imaging algorithms could also be adapted to run on heterogeneous platforms in a straightforward manner with the FlowCL framework.

In the present study, only GPUs and multicore CPUs were used as computing devices. However, there are other different computing devices, such as FPGAs, that were not included in this study and which can be also used in a more comprehensive future study.

References

1. Augonnet, Cédric, *et al.* "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures." *Concurrency and Computation: Practice and Experience* 23.2 (2011): 187-198.
2. Barak, Amnon, *et al.* "A package for OpenCL based heterogeneous computing on clusters with many GPU devices." *2010 IEEE international conference on cluster computing workshops and posters (CLUSTER WORKSHOPS)*. IEEE, 2010.
3. Boers, Anna M., *et al.* "Automated cerebral infarct volume measurement in follow-up noncontrast CT scans of patients with acute ischemic stroke." *American Journal of Neuroradiology* 34.8 (2013): 1522-1527.
4. van Geldermalsen, Sytse. *Flowcl-declarative dataflow api for heterogenous platform computing*. Diss. Master's thesis, University of Amsterdam, 2013.
5. Khronos OpenCL Working Group: *The opencl specification* (2012).
6. Kothari, Rashmi U., *et al.* "The ABCs of measuring intracerebral hemorrhage volumes." *Stroke* 27.8 (1996): 1304-1305.
7. MATLAB: version 8.0.0.783 (R2012b). The MathWorks, Inc., Natick, Massachusetts (2012).
8. Pham, Dzung L., Chenyang Xu, and Jerry L. Prince. "Current methods in medical image segmentation." *Annual review of biomedical engineering* 2.1 (2000): 315-337.
9. Saver, Jeffrey L., *et al.* "Infarct volume as a surrogate or auxiliary outcome measure in ischemic stroke clinical trials." *Stroke* 30.2 (1999): 293-298.
10. Spafford, Kyle, Jeremy Meredith, and Jeffrey Vetter. "Maestro: data orchestration and tuning for opencl devices." *European Conference on Parallel Processing*. Springer, Berlin, Heidelberg, 2010.



Chapter 3

Dynamic CT perfusion image data compression for efficient parallel processing

Renan Sales Barros[a, b], Silvia Delgado Olabarriaga[b], Jordi Borst[c], Marianne A. A. van Walderveen[d], Jorrit S. Posthuma[a], Geert J. Streekstra[a, c], Marcel van Herk[a, e], Charles B. L. M. Majoie[c], and Henk A. Marquering[a, c]

[a] Biomedical Engineering and Physics, Academic Medical Center, University of Amsterdam, Location L0, Meibergdreef 15, 1105 AZ Amsterdam, The Netherlands
[b] Department of Clinical Epidemiology, Biostatistics and Bioinformatics, Academic Medical Center, University of Amsterdam, Location B0, Meibergdreef 9, 1105 AZ Amsterdam, The Netherlands
[c] Department of Radiology, Academic Medical Center, University of Amsterdam, Location B0, Meibergdreef 9, 1105 AZ Amsterdam, The Netherlands
[d] Department of Radiology, Leiden University Medical Center, Albinusdreef 2, 2333 ZA Leiden, The Netherlands
[e] Department of Radiation Oncology, The Netherlands Cancer Institute, Plesmanlaan 121, 1066 CX Amsterdam, The Netherlands

- Barros, Renan Sales, et al. "Dynamic CT perfusion image data compression for efficient parallel processing." *Medical & biological engineering & computing* 54.2 (2016): 463-473.
- First author, journal, https://link.springer.com/article/10.1007/s11517-015-1331-6

Abstract

The increasing size of medical imaging data, in particular time series such as CT perfusion (CTP), requires new and fast approaches to deliver timely results for acute care. Cloud architectures based on graphics processing units (GPUs) can provide the processing capacity required for delivering fast results. However, the size of CTP datasets makes transfers to cloud infrastructures time-consuming and therefore not suitable in acute situations. To reduce this transfer time, this work proposes a fast and lossless compression algorithm for CTP data. The algorithm exploits redundancies in the temporal dimension and keeps random read-only access to the image elements directly from the compressed data on the GPU. To the best of our knowledge, this is the first work to present a GPU-ready method for medical image compression with random access to the image elements from the compressed data.

1 Introduction

CT perfusion (CTP) imaging is used as a diagnostic tool for initial evaluation of patients suffering from acute stroke [1]. CTP images are acquired by dynamically tracking the passage of a contrast agent through the cerebral blood vessels and tissue [2]. Analysis of CTP data enables the assessment of the severity of the damages caused by stroke. This information can be used to choose the most adequate treatment for the patient [3]. Currently, CTP datasets can be as large as 3.76 GB, and when dealing with this amount of data, traditional processing methods are slow and delay the acute care. Also, these traditional methods are expensive because of the costs of purchase and maintenance of dedicated software and hardware for image processing.

Cloud architectures have emerged as a cost-effective alternative for medical image processing. Cloud-based solutions make remote on-demand image processing services available for wide use in medical practice. To provide high-performance processing, cloud architectures can make use of graphics processing units (GPUs), which are designed for very efficient parallel processing of large amounts of data. GPUs were demonstrated being capable of considerably speeding up medical image processing applications [4]. Nowadays CPUs are also capable of parallel processing. However, CPUs are designed for general purpose processing, and because of that, the processing power of a GPU can be superior to the processing power of a CPU in several applications. GPUs are used in several common image processing tasks such as filtering and rendering. Thus, it is feasible to assume that the processing of CTP image data can also take advantage of GPU-based architectures. However, to benefit from the GPU computational power, algorithms need to be adapted or developed from scratch.

The size of CTP data poses challenges for their processing on GPU and on cloud infrastructures. The transfer of CTP data to cloud architectures can be time-consuming, which may limit the suitability of cloud applications for dealing with acute patients. In addition, to perform GPU computation, a host application is required, and the CTP data also need to be transferred from the host memory to the GPU memory. The time spent on the transfers from host to GPU has a considerable impact on the overall processing time. In short, due to the large size of CTP datasets, the time to transfer the image data limits its application for remote processing in acute care scenarios.

Data compression techniques can be used to reduce the CTP dataset size and speed up its transfer to the cloud and to the GPU memory. Since time is critical in acute situations, the time required to compress, decompress, and transfer the compressed data should not be larger than the time required to transfer the uncompressed data. Another important constraint is that, in clinical care applications, the compression technique

must be lossless because no information can be removed or modified due to regulations.

The time required to execute the complete CTP data pipeline depends on scanner acquisition, data reconstruction, preprocessing, etc. Several aspects of this pipeline are strictly determined by scanner manufacturers. Figure 1 illustrates which pipeline stages (dark arrows) of the CTP processing in a GPU-based cloud infrastructure are affected by our compression method. Initially, the CTP data are produced at the scanner (A). After that, the CTP data must be compressed in a terminal (B) before the transfer to the GPU-based cloud infrastructure (C). While the CTP data are processed in the cloud infrastructure, several data transfers between host application memory (D) and GPU memory (E) can be required.

Ideally, the compression must be done in a machine capable of GPU processing. However, the compression can be executed in different computing devices such single-core CPUs and many-core CPUs.

The main goal of our compression technique is to reduce the data size for faster transfer and faster GPU processing on cloud architectures. To achieve this, we introduce a fast and lossless compression technique that not only speeds up the transfer of dynamic CTP data to cloud architectures, but also facilitates their parallel processing on GPUs. This technique presents a compression time suitable for acute care situations and produces compressed data that can be processed on a GPU requiring no decompression of the entire CTP dataset. In our technique, intensities of an arbitrary voxel are retrieved from the compressed data using a fixed amount of instructions independent of the input value or size. This means that, in terms of computational complexity, determining the intensity value of a voxel is a constant-time procedure (i.e., checking if a number is odd or even, checking a constant size lookup table), which is the fastest class of algorithms with computational complexity classified as $O(1)$. To the best of our knowledge, this is the first work to present a lossless method for medical image compression with direct access to the image elements from the compressed data.

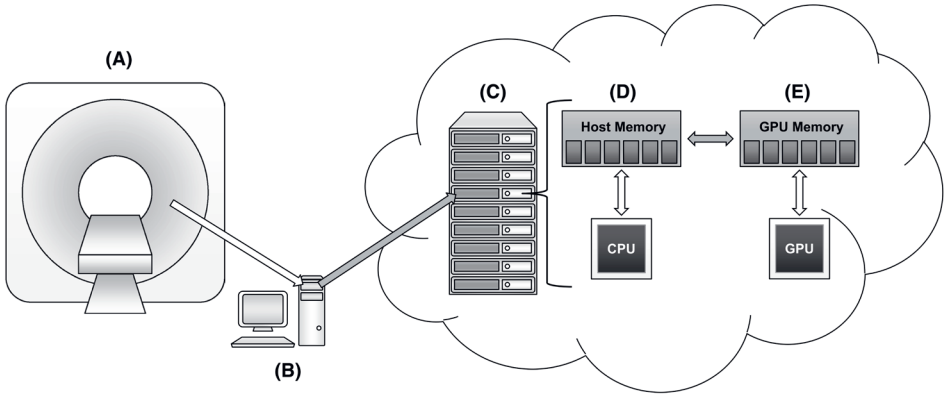


Figure 1: CTP data processing pipeline in a GPU-based cloud infrastructure: the CTP data are produced at the scanner (A), compressed in a terminal (B), sent to the GPU-based cloud infrastructure (C). While being processed, the CTP data can be transferred several times between host application memory (D) and GPU memory (E).

2 Methods

This section describes the characteristics of the CTP data, presents our compression technique, and discusses the relevant aspects that need to be considered during its implementation according to the targeted platform. Subsequently, the configuration of the experiments used to evaluate our compression technique is described.

2.1 Characteristics of CTP data

The datasets used in this study consist of 20 dynamic whole-brain volumes from actual stroke patients. The scans have 320 slices of 512 × 512 voxels with 16 bits/voxel, and each acquisition has 24 time steps. The patients were scanned as part of a Dutch multicenter randomized trial [5]. Approval of the medical ethical committee was obtained. All patients or legal representatives signed informed consent. The volumes are acquired approximately every 2.5 s during the first 35 s, followed by a scan every 5 s until 60 s. Subsequently, five volumes are scanned with a 30 s interval. The size of each volume is 160 MB, and thus the complete dataset has 3840 MB of data that need to be quickly processed for an initial evaluation of the patient condition. Sometimes, an additional CTP dataset is produced to evaluate the treatment progress after around 24 h, resulting in up to 7.5 GB of data per patient. All the image data are saved according to the digital imaging and communications in medicine (DICOM) standard.

Every dataset can be described as $I(\boldsymbol{x}, t)$, which represents the image intensity at position $\boldsymbol{x}$ at time t. The inflow and outflow of contrast agent can be observed in all the brain

tissue. However, the intensity values in the largest part of the brain tissue are expected to vary little over time. To illustrate this characteristic of the data, Figure 2 shows the intensities at $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ along time. The intensities at $\boldsymbol{x}_a$ are not strongly affected by the contrast agent. On the other hand, the intensities at $\boldsymbol{x}_b$ are strongly affected by the inflow and outflow of contrast agent.

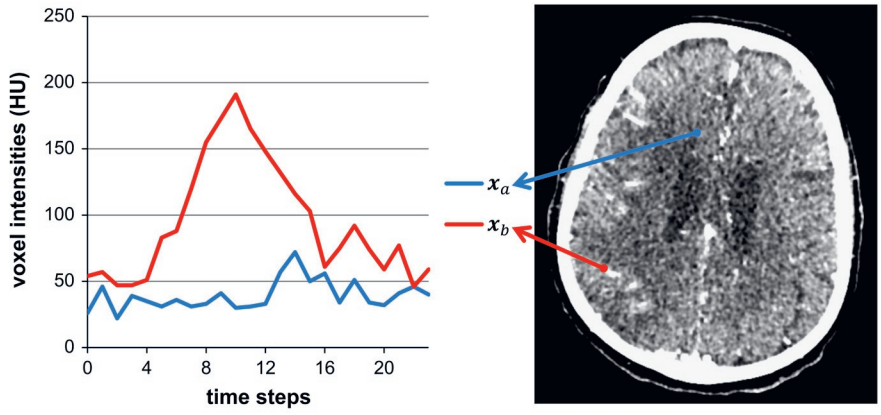


Figure 2: Sample slice of CTP data at the time step 12, and the intensity values of the voxel at x_a and x_b over time. The intensities values at x_a are not strongly affected by the contrast agent, and the intensities valuas at x_b are strongly affected by contrast agent.

Voxel intensities in CT imaging are generally represented using 16 bits. However, the range of voxel values over time is smaller than the range that can be represented by 16 bits. Therefore, fewer bits can be used to represent exactly the same information by storing the variation of these intensities instead of their absolute values. This characteristic is illustrated by using the intensities at $\boldsymbol{x}_b$ as an example. These intensities vary between 46 and 191 HU, so only eight bits are required to represent them ($\lceil \log_2(191 - 46 + 1) \rceil = 8$). For the voxel at $\boldsymbol{x}_a$, a better compression can be obtained because only six bits are required ($\lceil \log_2(72 - 22 + 1) \rceil = 6$), which represents a compression ratio of 2.6 compared with the original representation using 16 bits.

As observed in Fig. 3, only 6 % of the voxels in that slice require more than eight bits to represent their intensities variation over time, and a maximum of 11 bits is required to represent this variation.

The effect of motion artifacts is apparent in Figure 3, and for this reason, a higher amount of bits is required to encode the area around the skull. However, this higher amount of bits (between 9 and 11 bits) is still considerably smaller than 16 bits, which are required for the uncompressed image. Furthermore, the motion affects only a small

portion of the image. In general, when motion is present, there is mainly overlapping of brain tissue with similar intensity values, which does not result in a higher amount of bits for encoding the voxel intensities over time. In short, Figure 3 illustrates that, due to the characteristics of the CTP data, the number of voxels that have a large intensities variation over time is rather small. This indicates that the temporal dimension of the CTP data is a substantial source of redundancies that can be exploited for compression purposes.

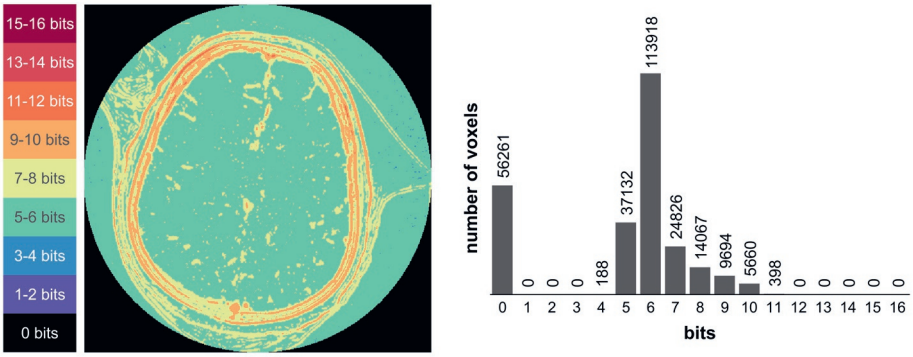


Figure 3: Number of bits required to represent the variation of voxel intensities over time in the selected slice. The effect of motion artifacts is visible, and for this reason, a higher amount of bits is required to represent the area around the skull. Nevertheless, this higher amount of bits (9-11 bits) is considerably smaller than the original 16 bits that are used by the uncompressed data. Furthermore, the motion affects only a small portion of the image. Only 6 % of the voxels require more than eight bits to represent their intensities variation over time.

2.2 Compression algorithm

Our compression technique exploits the time redundancy explained above. Let $I(\boldsymbol{x}, t)$ represent the uncompressed image intensity, where x indicates a 3D coordinate and t indicates a time step between 0 and $n - 1$. In compressed form, the image is represented as $I(\boldsymbol{x}, t) = C(\boldsymbol{x}) + \Delta(\boldsymbol{x}, t)$ with $\min V_x \leq C(\boldsymbol{x}) \leq \max V_x$ and $V_x = \{I(\boldsymbol{x}, t_0), I(\boldsymbol{x}, t_1), \ldots, I(\boldsymbol{x}, t_{n-1})\}$. For simplicity, we use $C(\boldsymbol{x}) = \min V_x$.

The set of values D_x given by $D_x = \{\Delta(x, t_0), \Delta(x, t_1), \ldots, \Delta(x, t_{n-1})\}$ do not present a large variation, so fewer bits can be used to represent them. The exact number of bits required to represent D_x is given by $\lceil \log_2(\max V_x - \min V_x + 1) \rceil$. Thus, D_x is stored by using $\lceil \log_2(\max V_x - \min V_x + 1) \rceil \times n$ bits.

In a sequential processing unit, the voxels are compressed one by one, and the time required to compress a single voxel is proportional to n because n computations are required to determine D_x, $\min V_x$, and $\max V_x$. Thus, when executed sequentially, the

computational complexity of our algorithm is $m \times n$ where m is the number of voxels in the dataset. However, the compression of all the voxels is independent, and consequently, it can be done in parallel. During the compression of a voxel, the computations to calculate D_x are independent, and they can also be parallelized. Moreover, when using parallel processing, $\min V_x$ and $\max V_x$ can be calculated in a time proportional to $\log_2 n$ through parallel reduction [6]. In a parallel implementation, the most expensive computations required by our algorithm correspond to finding $\min V_x$ and $\max V_x$. Consequently, in terms of computational complexity, our algorithm can compress a CTP dataset in a time proportional to $\log_2 n$ when running in parallel.

To retrieve the value of as $I(\boldsymbol{x}, t)$, a sum needs to be performed: $C(\boldsymbol{x}) + \Delta(\boldsymbol{x}, t)$. By using fixed size arrays to store $\Delta(\boldsymbol{x}, t)$ and $C(\boldsymbol{x})$, $I(\boldsymbol{x}, t)$ can be retrieved in constant time. The data stored using less bits, which is $\Delta(\boldsymbol{x}, t)$, do not need to be modified. Thus, in our method, $I(\boldsymbol{x}, t)$ is determined using a single sum of values that can be retrieved in constant time.

2.3 Implementation

The efficiency of our compression method is strongly dependent on the efficiency of the data structures used in its implementation, in particular for $\Delta(\boldsymbol{x}, t)$ and $C(\boldsymbol{x})$. $\Delta(\boldsymbol{x}, t)$ is an element of the set D_x. All the elements in a set D_x are represented using the same number of bits. For instance, by considering the voxels at $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ in Figure 2, six and eight bits are required to represent the elements in D_{xa} and D_{xb} respectively. Thus, because $n = 24$ in our datasets, D_{xa} requires $24 \times 6 = 144$ bits, and D_{xa} requires $24 \times 8 = 192$ bits to be represented. Because the amount of bits required to represent each D_x set varies, it is not possible to use a single fixed size array to store all the different D_x sets in memory.

Current computers are not capable of addressing memory blocks of an arbitrary amount of bits. Thus, all the D_x sets are contiguously stored in a fixed size array of 32 bits elements named D. A maximum of two elements from D need to be accessed to store and retrieve a particular $\Delta(\boldsymbol{x}, t)$ using a fixed amount of bit shift operations. The computational cost of these operations is constant, so they do not increase the computational complexity of reading and storing the values in D.

An offset is provided to determine where a D_x begins in the array D. All the offsets are stored in a fixed size array of 32 bits elements named O. Another fixed size array of eight-bit elements, named B, is used to store how many bits are used to represent the elements in D_x. In this way, different elements in D_x can be distinguished. The offsets can be quickly calculated by traversing B. However, O is provided to keep instant access to any D_x in D. Finally, an array of 16-bit elements, named C, is used to

store all the $C(\boldsymbol{x})$ values. The elements of C have 16 bits because they contain original intensity values from the 16-bit voxels. Figure 4 illustrates the data structures used in our implementation. The size of the resulting compressed data is the sum of the sizes of the arrays C, O, B, and D.

Three different implementations of our dynamic image compression for parallel processing (DICOPP) were developed:

- DICOPP CPU—a parallel implementation compressing the voxels using multiple threads in a many-core CPU and using a sequential method to calculate $\min V_x$ and $\max V_x$;
- DICOPP CPU PR—another parallel implementation targeted for a many-core CPU using multiple threads to calculate $\min V_x$, and $\max V_x$ through the parallel reduction method; and
- DICOPP GPU—a parallel implementation running on the GPU and calculating $\min V_x$ and $\max V_x$ sequentially.

During the implementation, we observed that using parallel reduction to calculate $\min V_x$ and $\max V_x$ on the GPU requires a more complex organization of the data in the GPU memory, which slows down the memory operations and results in an inefficient GPU implementation. For this reason, this alternative was abandoned. Also, only 24 values need to be evaluated to calculate $\min V_x$ and $\max V_x$, and at this scale, the benefits of using parallel reduction are not noticed. Our implementations use the .NET framework version 4.0 [7] and C# [8] as programming language. These technologies were chosen because our implementations need to be integrated in an existing platform for medical image processing based on .NET. Our implementations use Fellow Oak DICOM (FO-DICOM) for .NET version 1.0.36 [9], which is a high-performance API for handling DICOM files. For the GPU computations, OpenCL 1.1 [10] was used. OpenCL is a framework for the development and execution of programs across platforms consisting of different types of processors such as CPUs, GPUs, digital signal processors, field-programmable gate arrays. OpenCL. NET version 2.2.9 [11] was used to integrate OpenCL with .NET. OpenCL.NET is a library that wraps the original OpenCL 1.1 API for .NET.

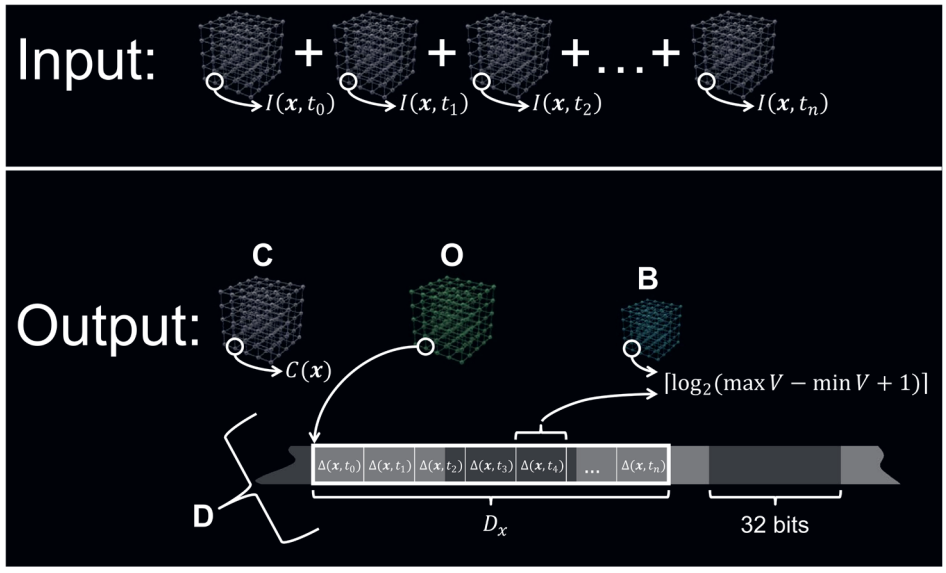


Figure 4: Data structures used in the implementation. B is a constant size array of 8-bit elements that stores the amount of bits used to encode the intensity values of a voxel. C is a constant size array of 16-bit elements used to store all the $C(x)$ values. D is a constant size array of 32-bit elements used to store all the D_x sets. O is an offset to determine where a set D_x begins in the array D

2.4 Evaluation setup

All the compression techniques that are incorporated in the DICOM format were selected for comparison with our method. However, according to the DICOM specification, MPEG2 and MPEG-4 compressions are inherently lossy, and for this reason, they were excluded of our comparison. JPEG 2000 lossless was also excluded from our comparison because it is much slower than the other methods, without a considerable better compression ratio. Consequently, only the following techniques from the DICOM standard were used in our experiments:

- JPEG lossless, more precisely the JPEG process 14 (first-order horizontal prediction [selection value 1], DPCM, non-hierarchical with Huffman coding);
- JPEG LS lossless; and
- Run-length encoding (RLE).

Very efficient low-level implementations of the techniques from the DICOM standard were used in our comparison. For JPEG and JPEG 2000, an open-source C library named FreeImage [12] was used. Regarding JPEG LS, an open-source and optimized C++ library named CharLS [13] was used. Finally, for the RLE compression, the C++ implementation provided with the FO-DICOM library was used. Regarding our

method, the three implementations described in Sect. 2.3 were used in our comparison. All the selected techniques from the DICOM standard were used only to perform 2D compression, and as a result, they were used to independently compress all the slices in a CTP dataset. These techniques are not designed to be executed in massively parallel architectures. Thus, to provide a fair comparison of the compression time with our implementations, which were designed for these architectures, the compression of all slices were divided equally among the CPU threads available by a multithread application. In this manner, the thread overhead was minimized, and the usage of the CPU for the compression task was maximized. Regarding our method, the same approach was used in our CPU implementations, i.e., use all the available CPU threads and distribute load equally. In the GPU implementation, the compression time includes the time required by the transfers between the host application and the GPU device.

Ideally, GPU implementations of the other compression techniques should be used for the comparison. However, to the best of our knowledge, there is no GPU implementation available for these methods. For JPEG, there are many GPU-based codecs, but none of them presents the lossless compression mode.

To compare the time to access the decompressed data, intensities of all time steps of 320 voxels in 320 slices were retrieved sequentially in an application running on the CPU and accessing the compressed data in the host application memory. Our method does not require complete decompression of a CTP dataset, and in this manner, accessing the decompressed value of a single voxel is a straightforward way to compare the decompression performance of the evaluated methods. The compressed data produced by the three different implementations of our method are identical; therefore, reading time was computed only for one of the results.

To evaluate the impact of the number of processing units in the compression time of our method, the DICOPP CPU implementation was executed using from 1 up to 6 threads. The maximum of six threads was defined because this is the number of independent processing units available in the hardware configuration used (see Table 1).

The main goal of our compression technique is to enable faster transfer to cloud architectures. To evaluate this, the total transfer time of each compression method used in our comparison was computed. This time is calculated by adding: the compression time, the time to transfer the compressed data, and the time to read the compressed data. The time to transfer the compressed data was calculated by considering the theoretical transfer rate of the following network standards: OC-3/STM-1 [14], OC-12/STM-4 [14], 1000BASE-T [15], and OC-48/STM-16 [14], or 155, 622, 1000, and 2400 Mbps, respectively. 1000BASE-T is a standard for gigabit Ethernet networks. The other

standards specify the transmission bandwidth for digital signals that can be carried on fiber-optic networks.

Our compression technique enables GPU processing directly from the compressed data. By processing the compressed CTP data, less data need to be transferred between host and GPU. This feature can speed up the total GPU processing time considerably because, in some applications, most of the time in a GPU computation is spent on data transfers. In order to evaluate the GPU processing time improvement, a GPU application that creates a mask from the CTP data was developed. The mask, which is defined by the double threshold 0–15 HU, is part of a noise reduction filter for dynamic CTP data described in [16]. In our evaluation, the developed GPU application computes this mask in two different ways: using the uncompressed data and using the compressed data generated by our method. In both ways, the time to compute the mask is measured including the time spent by the transfers between host and GPU.

All the evaluations described in this section were performed in the same hardware configuration (see Table 1) using Windows 7 Enterprise 64 bits as operating system. For all the time measurements, the high-resolution timing counters provided by the Win32 API were used.

Table 1: Hardware configuration used to execute the compression methods evaluated in our experiments.

CPU name	Intel Xeon E5-2620
CPU clock	2.00 GHz
CPU cores	6
CPU threads	12
RAM memory	64 GB
GPU name	GeForce GTX TITAN
GPU driver version	331.65
GPU cores	2688
GPU clock	836 MHz
Dedicated video memory	6 GB GDDR5

3 Results

Table 2 shows the performance results of the evaluated compression techniques applied to 20 CTP datasets described in Sect. 2.1. The DICOPP CPU PR implementation achieved a better compression time than the DICOPP CPU implementation in 85 %

of the executions. As mentioned in Sect. 2.3, the CTP datasets time dimension is too short to substantially benefit from parallel reduction for computing $\min V_x$ and $\max V_x$.

In our evaluation setup, all the data are transferred to CPU memory before being accessed or decompressed. Thus, all the reading and decompression operations are executed only in the host application. The reported time corresponds to the reading time of only 320 × 24 voxels, and not to the entire CTP dataset. Our method does not require full decompression of a dataset, and because of this, it achieved a read time many times lower than the best result from the other methods.

JPEG 2000 lossless took 132 and 470 s to compress and read the compressed data of a single CTP dataset. This is more than six times slower than the results in Table 2.

Table 2: Compression time, reading time, and compression ratio for 20 datasets (mean ± SD [min., max.]) using different compression methods. The best results are underlined.

Compression method	Compression time (ms)	Reading time (ms)	Compression ratio
JPEG LS	09911 ± 0398 [08879, 10806]	58267 ± 2546 [49924, 62052]	4.64 ± 0.29 [4.14, 5.55]
JPEG	14552 ± 0742 [12234, 16095]	43443 ± 1791 [37033, 44997]	2.09 ± 0.16 [2.74, 3.55]
RLE	09679 ± 0947 [08286, 11110]	15554 ± 0634 [13468, 16669]	2.31 ± 0.10 [2.12, 2.66]
DICOPP CPU	20350 ± 2602 [14157, 24239]	0.15 ± 0.36 [0, 1]	2.20 ± 0.17 [1.95, 2.75]
DICOPP CPU PR	17718 ± 1413 [14934, 20712]		
DICOPP GPU	05944 ± 0711 [04826, 07873]		

The number of processing units used to execute our compression method has a major impact in its compression time. To illustrate this, Figure 5 shows the compression time obtained by using different number of threads for compressing 20 CTP datasets using the DICOPP CPU implementation. The standard deviations of the compression time of the executions using from 1 to 6 threads are, respectively, 14.27, 7.30, 5.76, 4.32, 3.83, and 2.86 s.

Table 3 shows the total transfer time (compression time + time to transfer compressed data + decompression time) for the 20 CTP datasets using the maximum transfer rate of four different types of network. As a reference, the first row of Table 3 shows the only the transfer time of an uncompressed dataset. DICOPP GPU achieved the lowest transfer time in all the network types listed in Table 3. However, in networks slower than the ones listed in Table 3, JPEG LS achieves a better transfer time because it has a better compression ratio. In faster networks, it takes longer to compress and transfer the data than to transfer the original data without compression.

Regarding the GPU processing time, the GPU processing of the mask using the original and the compressed data took 2818 ± 382 [2664, 4392] and 1903 ± 186 [1712, 2668] milliseconds, respectively. Accordingly to these results, the GPU processing using the compressed data was, on average, more than 30 % faster than the processing of the original data.

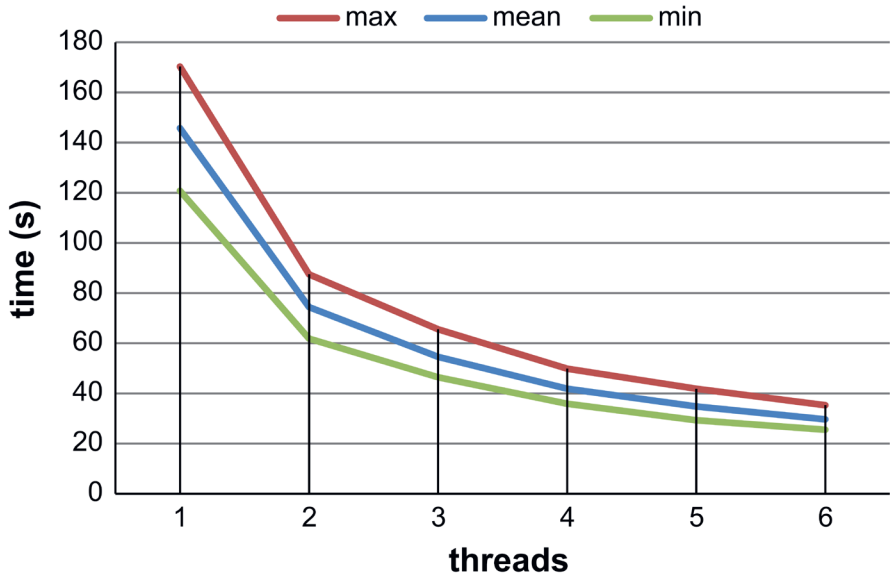


Figure 5: Maximum, mean, and minimum times *(vertical axis)* **spent to compress 20 CTP datasets by using different number of threads** *(horizontal axis)*.

Table 3: Total transfer time (in s) for 20 datasets compressed by different methods and using different network speeds (mean ± SD [min.,max.])

	OC-3/STM-1 (s)	OC-12/STM-4 (s)	1000BASE-T (s)	OC-48/STM-16 (s)
Original Data	207.82	51.79	32.21	13.42
JPEG LS	113 ± 5.3 [096, 122]	79 ± 3.2 [68, 84]	75 ± 3.2 [64, 79]	71 ± 3.0 [61, 75]
JPEG	127 ± 5.4 [107, 134]	75 ± 3.0 [63, 78]	68 ± 2.7 [58, 71]	62 ± 2.5 [53, 64]
RLE	115 ± 4.7 [100, 123]	47 ± 1.9 [41, 50]	39 ± 1.6 [34, 41]	31 ± 1.4 [27, 33]
DICOPP CPU	115 ± 7.9 [089, 127]	44 ± 3.5 [32, 48]	35 ± 3.0 [25, 38]	26 ± 2.7 [19, 30]
DICOPP CPU PR	112 ± 7.4 [090, 123]	41 ± 2.5 [33, 45]	45] 32 ± 2.0 [26, 36]	36] 23 ± 1.6 [19, 27]
DICOPP GPU	100 ± 6.9 [080. 111]	29 ± 1.9 [23, 32]	20 ± 1.3 [16, 22]	12 ± 0.8 [09, 14]

4 Discussion

The compression time of the DICOPP GPU implementation is notably faster than the other methods. Even simple algorithms, such as RLE running in parallel, are around 1.6 times slower than the DICOPP GPU implementation. Note that the implementations of our method used more abstraction layers than the other implementations used in the comparison. For instance, memory management in .NET applications is different from low-level applications, and this can result in a slower execution time when compared with C or C++ applications, which is the case of the other methods. However, despite the higher level of abstraction, the compression time of our CPU implementations is approximately only 5 to 10 s slower than the other methods.

As expected, our method presents negligible times to read the voxel intensities from the compressed data, as illustrated in Table 2. This is possible because our method is the only to provide direct access to the voxel intensities. The random access to voxel values has many advantages, and it enables the application of several imaging operations to the entire image data in the compressed form. Because of this direct access, operations such as local filtering and threshold-based segmentations can be performed without decompression. By doing this, our method saves memory (the compressed data are processed) and processing time (the decompression step is skipped). The direct access to voxel values provided only by our method also speeds up the GPU processing. This is possible because our method reduces the amount of data that need to be transferred between host application and GPU, which is also a common bottleneck in GPU-based computing. As presented in Sect. 3, the GPU computation of a mask from the CTP data was speeded up more than 30 % by using the compressed data produced by our method. We must highlight that exactly the same instructions were executed in the compressed and uncompressed representations of the CTP data. This speedup is only possible because of the direct access to the voxel values from the compressed data on the GPU.

As previously stated, the main goal in acute care is to provide fast results, and as observed in Table 3, the implementations of our method achieve better transfer times than all the others. In networks slower than 100 Mbps, our method was overcome by other compression methods. However, it is reasonable to assume that current cloud infrastructures provide connections with speeds that are higher than 100 Mbps. In fact, most of the current cloud providers offer direct connections up to 10 Gbps. For instance, Microsoft Azure [17] offers connections from 200 Mbps up to 10 Gbps, and Amazon Web Services [18] offers connections from 50 Mbps up to 10 Gbps. In these very fast connections, transferring the uncompressed data is faster than transferring the compressed data. However, these very fast connections are expensive and priced

according to the offered speed. This means that our method enables a cost-effective usage of these connections. Our method can also reduce the GPU processing time of the CTP data. Because of this feature, our method not only contributes for a faster analysis of CTP data, which is crucial in acute stroke cases, but also to cheaper analysis on pay-per-use infrastructures.

Thus, regarding the processing pipeline of CTP data on GPU-based cloud infrastructures, our compression method enables fast transfers and fast GPU processing, which consequently results in reducing costs and providing the faster image processing required when dealing with acute stroke patients.

Our compression technique was developed to be executed in massively parallel architectures. Thus, it is possible to achieve faster results when using more parallel processing units (see Figure 5). Also, as observed in Table 3, our compression technique is the only one that enables reducing transfer times in fast data connections because of its fast compression and because it does not require a decompression step prior to processing. Because of these characteristics, our compression technique is better suited for future computational infrastructures than the other compression techniques evaluated, since it can benefit from massively parallel processing and fast data connections. We must emphasize that, if ignoring the cost aspect, there are connections speeds currently available that are fast enough to be used for transferring uncompressed CTP data. However, with more powerful parallel processing devices, our method can become beneficial even with these connection speeds. Thus, because of these trends, we believe that our method is beneficial not only in current cloud infrastructures but also in the upcoming cloud infrastructures.

The compression ratio of our method is inferior to the compression ratio of the other methods. To improve our compression ratio, different preprocessing operations could be applied. However, this preprocessing can make the execution of our compression technique considerable longer. To avoid that, the CTP processing pipeline has to be carefully analyzed to identify whether the adoption of preprocessing steps will effectively result in a faster data transfer, which is the main goal of our work. For instance, usually the CTP analysis requires the application of a noise reduction filter. In a new pipeline configuration, this noise reduction can be done before the compression in order to achieve a better compression ratio. Noise reduction may also improve the compression ratio of our method because noise strongly influences the variation of the voxel values over time. It is expected that thick slices have less noise, and it may result in better compression ratios. A detailed study to assess the effects of different noise levels in the performance of our compression method can be performed. However, in this paper, we focused on the evaluation of our compression method in the image data that are

generated in clinical practice.

Apart from noise, motion artifacts can also affect the compression ratio of our method. Again, a possible solution is a preprocessing step for motion correction before the compression step [19]. However, this will result in increasing processing time. We evaluated our method in actual patient data, which included motion artifacts, and as shown in Figure 3, the effects of motion do not have a strong impact on our compression ratio. Motion does not affect the compression ratio of our method considerably because, in different time frames, different types of tissue rarely overlap, and thus constant geometrical locations still have similar intensity values. The only exceptions are the areas around the skull, which are a small portion of the image data. However, even in these areas, the amount of bits required to represent the compressed data are still considerably smaller than the original amount of bits used in the uncompressed data.

Perhaps, the most effective preprocessing step that could be applied is a simple threshold segmentation and removal of useless data (i.e., the air around the patient). Nevertheless, we focused in evaluating our method in original patient data. An extensive analysis of the different techniques that can be combined with our compression method was beyond the scope of this study.

Our goal was to provide a compression technique to be used in a specific clinical practice rather than to be used as a general compression technique. In clinical practice, we are dealing with large datasets that are very precisely defined (±24 time steps of approximately 320 slices of 512 × 512 pixels of 16 bits) and that are well accepted worldwide. Since CTP acquisitions are performed tens of thousands times per year, we believe that a specific and applied compression technique is worth studying. Although our technique is applied to and focused on CTP data, we believe that any other medical image time series could be potentially suited for compression by our algorithm. For example, all the medical images used in the experiments described in [2] have the necessary characteristics to be exploited by our compression algorithm, which is a small variation of voxel values over time.

4.1 Related work

Previous works also explored the redundancies in the temporal dimension of medical image data for compression purposes. The work presented in [20] calculates the differences between two contiguous images from a medical image time series and store these differences using eight bits when this is possible. When this difference cannot be expressed using eight bits, the original 16 bits are used. Because of this approach, the theoretical maximum compression ratio achieved by this method is 2. In this method, to retrieve the intensities from a particular time step, it is necessary to decompress

all intensities from the previous time steps. The main differences between this and our method are: our method achieves compression ratios greater than 2, and in our technique, any arbitrary image intensity in the four-dimensional space can be retrieved independently with a constant computational complexity.

Other compression techniques explore the effect of motion in 4D medical images. Motion is a feature especially present in 4D cardiac images. In the context of exploring motion for compression purposes, [21] proposed a technique based on the combination of a predictive image compression and a motion compensation technique. The work presented in [22] evaluates the motion in 4D medical images for compression purposes using motion fields that produce input parameters for a neural network used for motion estimation. [22] combines motion analysis with segmentation, block matching, and expert knowledge, to develop a framework for 4D medical compression. The authors of [23] apply recursively a multiframe motion compensation process that employs 4D search, variable blocksizes, and bidirectional prediction for reducing redundancies in spatial and temporal dimensions. All these three techniques were developed for achieving high compression ratios, and because of their complexity of compression and decompression, they are not well suited for the fast processing as required in acute care situations. Also, differently from our technique, they require a decompression step before processing.

Another common approach is to adapt or use existing sound, image, or video compression techniques for 4D medical image data. However, most of these compression techniques, like MPEG-2 and MPEG-4, are lossy and, for this reason, cannot be used in the same context as the proposed technique. Regarding lossless compression, the authors of [24] proposed a technique for 4D medical images based on the H.264/AVC standard for video compression. Again, this compression technique was designed to achieve high compression ratios, being too complex for producing fast response.

In CTP data, any particular voxel can be considered as an independent time series. Time series compression techniques can be applied independently for each voxel. However, most time series compression techniques are fundamentally lossy [25] and consequently cannot be used for the purposes of this study.

Regarding the lossless compression of time series, current techniques focus on the compression of long time series and are based on very complex models [26–29] that may even require the usage of a database for prediction purposes [28]. Because these techniques are developed for compressing long time series, it is not feasible to use them in CTP datasets, which present only 24 time steps. To illustrate this problem, the smaller model mentioned in [29] requires 192 bits only to store the initial conditions

of themodel equations describing a time series. This represents half of the size of entire time series of a particular voxel in CTP datasets (24 × 16 bits). The lossless time series compression can be also based on features that are not available in CTP datasets, such as multichannel [30] or multispectral information [31]. In short, the usage of state-of-the-art lossless time series compression in the time series from CTP datasets would not be effective because of the short length of these time series.

5 Conclusion

In this paper, we presented a new method to compress CTP data that take advantage of data redundancy in the time dimension. The proposed algorithm reduces the image size by using fewer bits to represent data that do not vary much through time. This method focuses on providing faster transfer of CTP data to GPU-based cloud infrastructures; therefore, a balance between compression ratio and compression time has been pursued, which is different from many compression methods which pursue good compression ratios. Our algorithm was designed for massively parallel architectures, and it is well suited for many-core CPU or GPU execution.

The proposed method was applied to 20 datasets and obtained the faster results compared to the lossless compression techniques adopted in the DICOM standard, despite its inferior compression ratio.

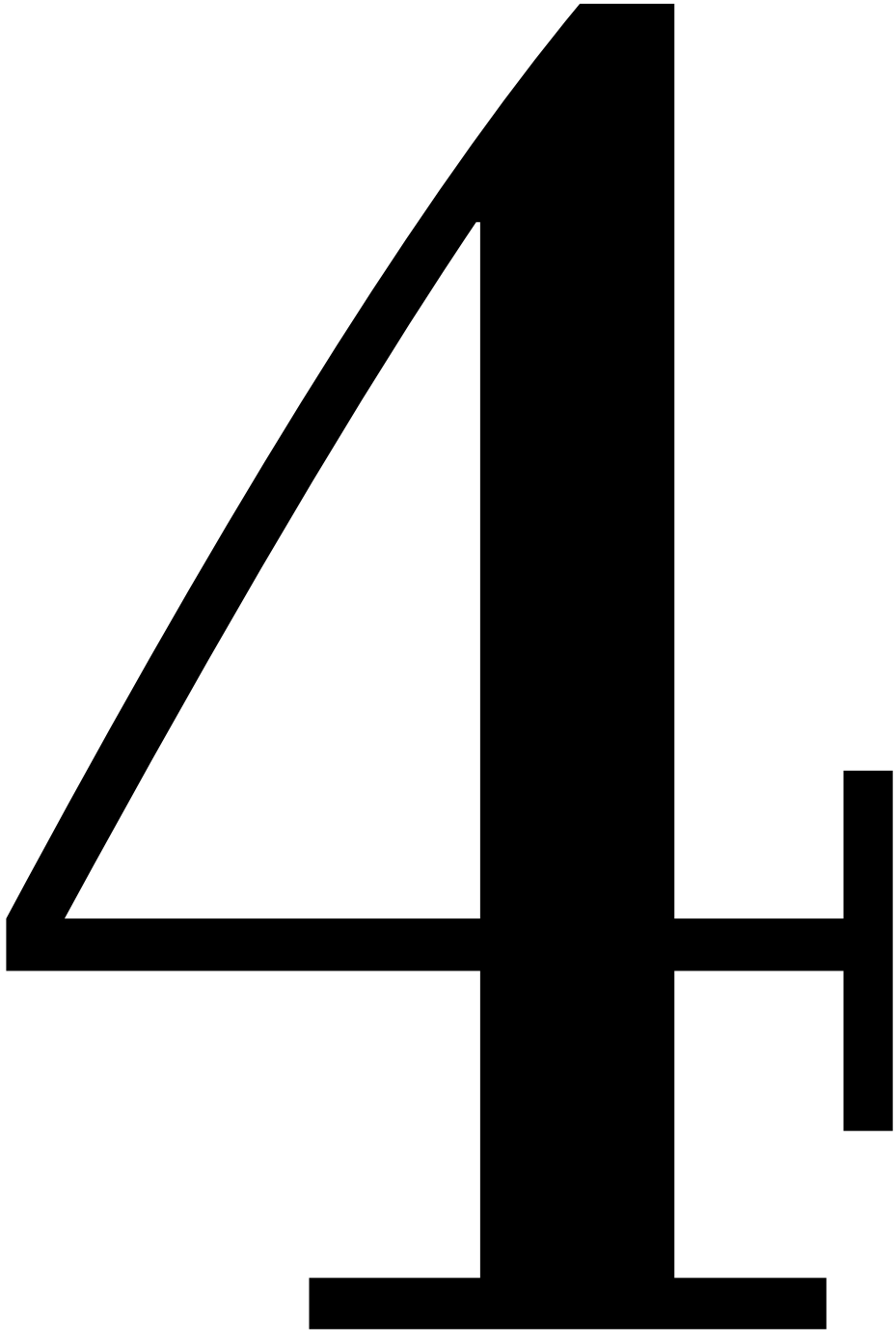
The resulting data representation offers direct random access for subsequent GPU processing, which is a feature not found in the other compression methods. Because of this, our time for retrieving information from the compressed data is negligible. This feature also makes it possible to reduce the time to transfer CTP data between host application and GPU because only the compressed form of the CTP data needs to be used in these transfers. Consequently, the GPU processing of CTP data can be speeded up when using the data in compressed form.

Currently, different ways to improve the compression ratio of our method are being investigated. This investigation focuses on the usage of fast techniques for noise reduction, motion identification, and segmentation of meaningless image elements. All these techniques need to be compatible with current clinical practices adopted when analyzing CTP data.

References

1. Fahmi, Fahmi, *et al.* "The effect of head movement on CT perfusion summary maps: simulations with CT hybrid phantom data." *Medical & biological engineering & computing* 52.2 (2014): 141-147.
2. Wintermark, Max, *et al.* "Perfusion-CT assessment of infarct core and penumbra: receiver operating characteristic curve analysis in 130 patients suspected of acute hemispheric stroke." *Stroke* 37.4 (2006): 979-985.
3. Allmendinger, Andrew Mark, *et al.* "Imaging of stroke: Part 1, perfusion CT — overview of imaging technique, interpretation pearls, and common pitfalls." *American Journal of Roentgenology* 198.1 (2012): 52-62.
4. Eklund, Anders, *et al.* "Medical image processing on the GPU–Past, present and future." *Medical image analysis* 17.8 (2013): 1073-1094.
5. Berkhemer, Olvert A., *et al.* "A randomized trial of intraarterial treatment for acute ischemic stroke." *n Engl J Med* 372 (2015): 11-20.
6. Harris, Mark, Shubhabrata Sengupta, and John D. Owens. "Parallel prefix sum (scan) with CUDA." *GPU gems* 3.39 (2007): 851-876.
7. .NET Framework 4. http://msdn.microsoft.com/en-us/library/ vstudio/w0x726c2(v=vs.100).aspx. Accessed 13 Apr 2015
8. C# Reference. http://msdn.microsoft.com/en-us/library/618ayhy6 (v=vs.100).aspx. Accessed 13 Apr 2015
9. Dillion C (2013) Fellow Oak DICOM for .NET. https://github. com/rcd/fo-dicom. Accessed 13 Apr 2015
10. Munshi A (2011) The OpenCL Specification. http://www.khronos. org/registry/cl/specs/opencl-1.1.pdf. Accessed 13 Apr 2015
11. (2013) OpenCL.NET. http://openclnet.codeplex.com/. Accessed 13 Apr 2015
12. Drolon H (2013) FreeImage. http://freeimage.sourceforge.net/. Accessed 13 Apr 2015
13. Vaan J de (2010) CharLS, a JPEG-LS library. http://charls.codeplex.com/. Accessed 13 Apr 2015
14. Robertazzi, Thomas. "SONET and WDM." *Basics of Computer Networking*. Springer, New York, NY, 2012. 59-64.
15. (2012) IEEE Standard for Ethernet. http://standards.ieee.org/ about/get/802/802.3.html. Accessed 13 Apr 2015
16. Mendrik, Adriënne M., *et al.* "TIPS bilateral noise reduction in 4D CT perfusion scans produces high-quality cerebral blood flow maps." *Physics in Medicine & Biology* 56.13 (2011): 3857.
17. Pricing Details—ExpressRoute | Microsoft Azure. http://azure. microsoft.com/en-us/pricing/details/expressroute/. Accessed 21 Oct 2014
18. AWS Direct Connect | Pricing. http://aws.amazon.com/directconnect/pricing/. Accessed 21 Oct 2014
19. Fahmi, Fahmi, *et al.* "3D movement correction of CT brain perfusion image data of patients with acute ischemic stroke." *Neuroradiology* 56.6 (2014): 445-452.
20. Cohen, Mark S. "A data compression method for image time series." *Human brain mapping* 12.1 (2001): 20-24.

21. Yan, Pingkun, and Ashraf Kassim. "Lossless and near-lossless motion-compensated 4D medical image compression." *IEEE International Workshop on Biomedical Circuits and Systems, 2004.*. ieee, 2004.
22. Žagar, Martin, Mario Kovač, and Daniel Hofman. "Framework for 4D medical data compression." *Teh Vjesn* 19 (2012): 99-105.
23. Sanchez, Victor, Panos Nasiopoulos, and Rafeef Abugharbieh. "Efficient 4D motion compensated lossless compression of dynamic volumetric medical image data." *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2008.
24. Sanchez, Victor, Panos Nasiopoulos, and Rafeef Abugharbieh. "Lossless compression of 4D medical images using H. 264/AVC." *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*. Vol. 2. IEEE, 2006.
25. Oinam, S. B., and Patil SB HK P. "Compression of time series signal using wavelet decomposition, wavelet packet and decimated discrete wavelet compression transforms techniques and their comparison." *Int J Adv Res Comput Commun Eng* 2 (2013): 1540-1544.
26. Takezawa, Tetsuya, Koichi Asakura, and Toyohide Watanabe. "Lossless compression of time-series data based on increasing average of neighboring signals." *Electronics and Communications in Japan* 93.8 (2010): 47-56.
27. Lang, Willis, Michael Morse, and Jignesh M. Patel. "Dictionary-based compression for long time-series similarity." *IEEE transactions on knowledge and data engineering* 22.11 (2009): 1609-1622.
28. Izumi, Tetsuya, and Youji Iiguni. "Data compression of nonlinear time series using a hybrid linear/nonlinear predictor." *Signal processing* 86.9 (2006): 2439-2446.
29. Ogorzalek, Macilej J. "Approximation and compression of arbitrary time-series based on nonlinear dynamics." *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No. 01CH37196)*. Vol. 3. IEEE, 2001.
30. Kamamoto, Yutaka, *et al.* "An efficient lossless compression of multichannel time-series signals by MPEG-4 ALS." *2009 IEEE 13th International Symposium on Consumer Electronics*. IEEE, 2009.
31. Spring, James M., and G. G. Langdon. "Experiments in the lossless compression of time series satellite images using multispectral image compression techniques." *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No. 97CB36136)*. Vol. 2. IEEE, 1997.



Chapter 4

High performance analysis of compressed dynamic CT perfusion image data for acute care of ischemic stroke

Renan Sales Barros[a,b*] , Edwin Bennink[c] , Jorrit Posthuma[a] , Jaap Oosterbroek[c] , Charles Majoie[d] , Hugo de Jong[c] , Silvia Delgado Olabarriaga[b] and Henk Marquering[a,d]

[a] Biomedical Engineering and Physics; Academic Medical Center; University of Amsterdam; Meibergdreef 9, 1105 AZ, Amsterdam, The Netherlands.
[b] Department of Clinical Epidemiology, Biostatistics and Bioinformatics; Academic Medical Center; University of Amsterdam; Meibergdreef 9, 1105 AZ, Amsterdam, The Netherlands.
[c] Image Sciences Institute; University Medical Center Utrecht; Utrecht University; Heidelberglaan 100, 3584 CX, Utrecht, The Netherlands
[d] Department of Radiology; Academic Medical Center; University of Amsterdam; Meibergdreef 9, 1105 AZ, Amsterdam, The Netherlands.

- Barros, Renan Sales, et al. " High performance analysis of compressed dynamic CT perfusion image data for acute care of ischemic stroke." *The 8th International Workshop on High Performance Computing for Biomedical Image Analysis (HPC-MICCAI)*. 2015.
- first author, workshop, https://web.archive.org/web/20151106234137/http://www.bme.ufl.edu/hpcmiccai2015/program.html

Abstract

Background: Patients suspected of acute ischemic stroke benefit from fast diagnosis and treatment decisions. Automated analysis of computed tomography perfusion (CTP) images allows assessing the severity of stroke and supports optimal treatment selection. However, CTP imaging can result in more than 3.5 GB of image data per patient. Automated analysis of this large amount of data can be rather slow, which hinders its application in acute clinical practice.

Methods: To enable fast CTP analysis, we introduce a novel approach for processing compressed CTP image data. This approach does not require the processing algorithms to be altered for handling the compressed data, facilitating its application to existing methods. By processing compressed image data, the analyses are performed faster and memory usage and data transfer time are reduced. This approach also supports massive parallel processing of compressed data on Graphic Processing Units (GPUs). To validate the compressed data analysis, we developed GPU implementations of a double threshold segmentation and a well-established bilateral filter for processing compressed and uncompressed CTP image data.

Results: Our results show that the analysis of compressed data uses between 2 and 2.8 times less memory and improves GPU execution time between 1.2 and 1.7 times with identical analysis results compared to the uncompressed version.

Conclusion: This study shows that GPU processing of compressed CTP data can be used for speeding up diagnosis and treatment selection based on CTP image analysis, potentially improving outcome of patients with acute ischemic stroke.

Keywords: acute care; brain imaging; computed tomography perfusion; data compression; GPU processing; high performance computing; stroke

Background

Patients suspected of acute ischemic stroke benefit from fast diagnosis and fast treatment decisions. CT perfusion (CTP) imaging is a diagnostic tool for initial evaluation of patients suspected of acute ischemic stroke and it can be useful for predicting functional patient outcome [1]. Analysis of CTP data allows assessment of the severity of stroke damage and supports in choosing the most adequate treatment [2]. CTP images are acquired by dynamically tracing the flow of contrast agent through the cerebral blood vessels and cerebral tissues [3]. The combination of accurate postprocessing methods and CTP data analysis may eventually provide a powerful diagnostic tool in acute ischemic stroke management [4]. However, the processing of CTP data can be time consuming because CTP datasets can be larger than 3.5 GB, which causes image transfer and traditional processing methods to become slow and unsuitable for acute care.

Data compression techniques can be used to reduce the size of CTP datasets and speed up transfers. Moreover, the analysis of compressed CTP data has the potential to considerably reduce the amount of data to be processed, which may result in a reduction of execution time and memory usage. Previous non-medical imaging studies [5–7], which are mainly based on lossy compression, have shown the potential of compressed-domain processing. However, in clinical applications, the applied compression techniques are required to be lossless because of legal regulations. Moreover, processing of compressed data using the techniques provided by [5–7], requires large modifications to the original processing algorithms or they only allow the usage of a very limited set of image processing operations.

To provide high-performance data processing, graphic processing units (GPUs) can be used. GPUs are designed for very efficient parallel processing of large amounts of data. GPUs were demonstrated capable of considerably speeding up medical imaging processing in many different applications [8]. Nevertheless, the size of CTP datasets poses challenges for processing on GPUs. To perform a GPU computation, a host application is required to run on the CPU, and the CTP data need to be transferred from the host memory to the GPU memory. The time required in these transfers has a considerable impact on the overall GPU processing time, being a well-known bottleneck on GPU processing applications [9].

In this study, we introduce and validate a novel approach for parallel processing of compressed CTP data that supports GPU computing. The CTP data is compressed by using a fast lossless algorithm [10] that enables direct access to any image element from the compressed data. To evaluate the GPU processing of compressed CTP fata we used a well-established filter for noise reduction in dynamic CTP data [11] and a double threshold segmentation that operates on the 4 dimensions of the dynamic CTP data.

Methods

We evaluated the performance and accuracy of compressed CTP data processing using datasets from the recently completed MR CLEAN multi-center trial [12]. To compress the CTP datasets, we used a GPU-based compression algorithm specifically designed for CTP datasets. To process compressed and uncompressed CTP data on GPUs, we implemented a noise reduction filter and a double threshold segmentation. To validate our method, we compared the differences in execution time, memory usage, and filtering results of the different implementations of the filter and of the segmentation.

CTP image data

We used dynamic whole-brain volume datasets of 29 patients with acute ischemic stroke. Twenty of these scans have 320 slices with pixel spacing of 0.4mm×0.4mm and slice thickness of 0.5mm. The other 9 scans have only 6 slices with pixel spacing of 0.5mm×0.5mm and slice thickness of 4.8mm. Each slice has 512×512 voxels with 16 bits/voxel and 24 time frames. The patients were scanned as part of a multi-center randomized trial [12]. Approval of the medical ethical committee was obtained. All patients or legal representatives signed informed consent. The volumes were acquired approximately every 2.5 seconds during the first 35 seconds followed by a scan every 5 seconds until 60 seconds. Subsequently, a few additional volumes were scanned with a 30 seconds interval. The size of each volume with 320 slices is 160 MB. Thus, a single CTP series for one patient can have up to 3840 MB of data.

Compression algorithm

The compression algorithm used in this study is described in detail in [10]. Nevertheless, the main characteristics of this compression method are presented here for providing a better understanding of how the CTP data is stored in the compressed form.

Each CTP dataset can be represented as $I(x, t)$, which denotes the image intensity at position x at time t. Intensities in a CTP dataset are encoded by using 16 bits. However, at a given position, the range of intensities of the voxels over time is considerably smaller than the range of values that can be encoded by 16 bits. Consequently, fewer bits can be used to encode the same information when considering the range of these intensities, instead of their absolute values. This concept is illustrated in Figure 1, which shows that the intensities over time at positions x_a and x_b can be represented by using only 6 and 8 bits respectively ($\lceil \log_2(72 - 22 + 1) \rceil = 6$ and $\lceil \log_2(191 - 46 + 1) \rceil = 8$). This allows a compression ratio respectively of 2.6 and 2 for x_a and x_b compared to the original pixel representation in 16 bits. In the sample slice shown Figure 2, a compression ratio of at least 2.6 is possible in 79% of the voxels.

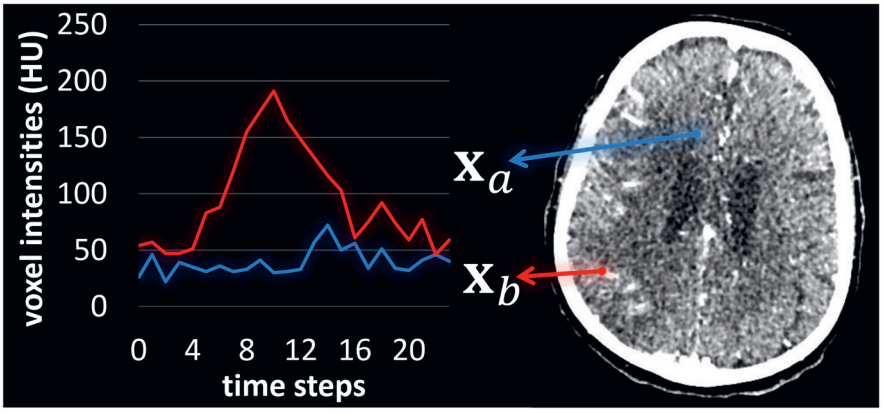


Figure 1: Intensities over time at positions x_a and x_b in Hounsfield units (HU).

By taking advantage of these smaller ranges of CTP data, we can represent the compressed image as $I(x, t) = C(x) + \Delta(x, t)$ with $\min V_x \leq C(x) \leq \max V_x$ and $V_x = \{I(x, t_0), I(x, t_1), \ldots, I(x, t_{n-1})\}$. For simplicity, we use $C(x) = \min V_x$. The set of values D_x given by $D_x = \{\Delta(x, t_0), \Delta(x, t_1), \ldots, \Delta(x, t_{n-1})\}$ may present a small variation. In this case, fewer bits can be used to encode their elements. The exact number of bits required to represent a single element in D_x is given by $\lceil \log_2 (\max V_x - \min V_x + 1) \rceil$. Thus, D_x is stored by using $\lceil \log_2 (\max V_x - \min V_x + 1) \rceil \times n$ bits.

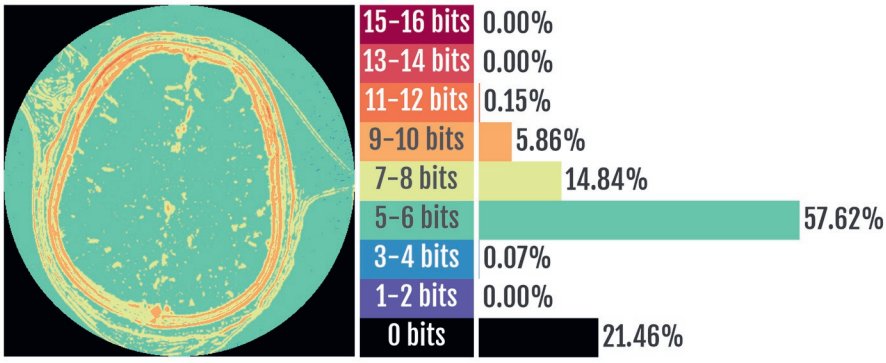


Figure 2: Number of bits requires to encode the variation of the intensities of a voxel over time.

Because the number of bits required to represent each D_x set varies for each x, all the D_x values are contiguously stored in an array **D**. A maximum of two elements from D need to be accessed to store and retrieve a particular $\Delta(x, t)$ using a fixed amount of bit shift operations. As we can see in Figure 5, when two elements from D need to be accessed, two bit shift operations are needed (lines 16-17 in Figure 5). Otherwise, a single bit shift

operation is required (line 14 in Figure 5)

Figure 3 illustrates the data structures used in the implementation of this compression technique. An offset is provided to determine where a particular D_x begins in the array D. All the offsets are stored in an array **O**. Another array, named **B**, is used to store how many bits are required to represent the elements in D_x. In this way, different elements in D_x can be distinguished. The offsets can be quickly calculated by traversing B. However, O is provided to keep instant access to any D_x in D. Finally, an array **C** is used to store all the *C*(x) values.

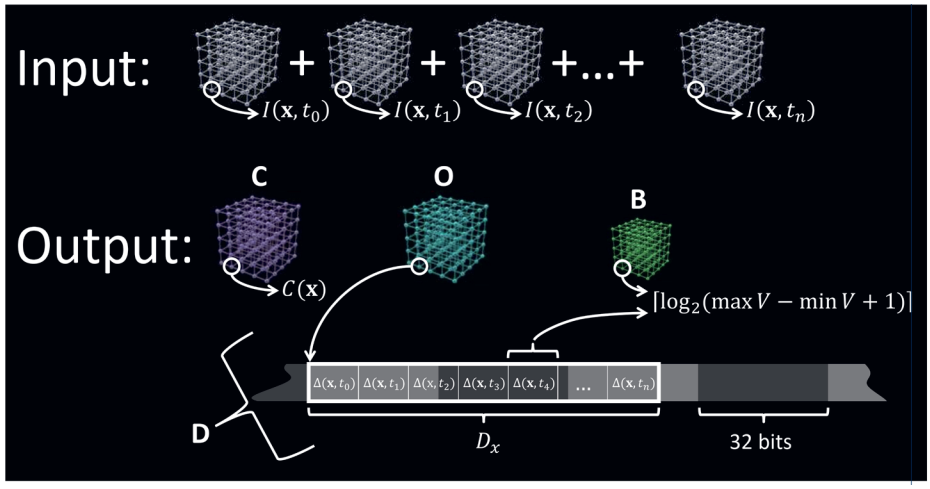


Figure 3: Illustration of the data structures to store the compressed CTP data. The input is a series of volume images in different time steps. B is an array of 8 bits elements. D and O are arrays of 32 bits elements. C is an array of 16 bits elements because its elements contain original intensity values from the 16-bit voxels. All these arrays are fixed-size.

The time required to compress a CTP dataset must be considered in our proposed strategy for processing CTP data. Since the compression method is processed on the GPU, a very low execution time is achieved. Figure 4 shows the compression time for 20 datasets of 3.75GB, which are the biggest datasets and, therefore, the slowest to compress. The size of the compressed files produced by this compression algorithm is larger than other lossless compression methods (Figure 4). However, as detailed in [10], other compression algorithms have a very long compression or decompression time and, for this reason, they are not suitable for speeding up the analysis of dynamic CTP data.

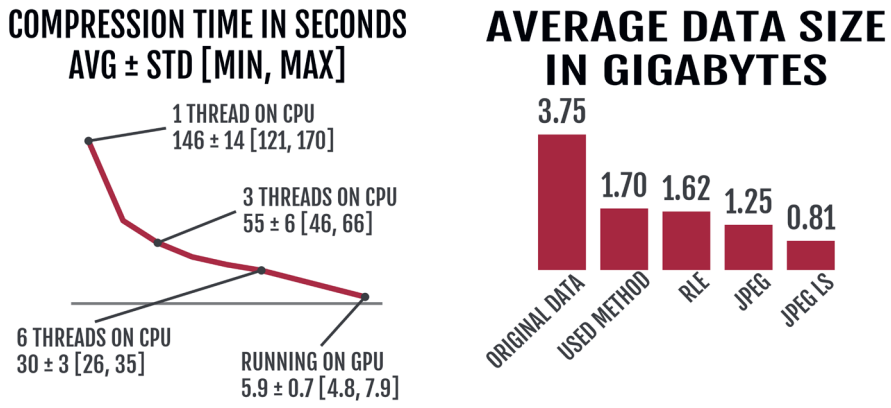


Figure 4: Compression time and compressed file size of the adopted compression algorithm. Data retrieved from the compression of the 20 CTP datasets of 3840MB. Left: compression time when using different number of CPU threads and when using GPU (for hardware configuration details see Table 1). Right: uncompressed dataset size and average compressed dataset size produced, respectively, by the method proposed by [10], run-length encoding (RLE), the lossless mode of JPEG, and JPEG LS.

Thresholding and filtering

In CTP images, the level of noise is commonly high due to the limited amount of radiation used during acquisition. Algorithms to determine perfusion parameters based on dynamic CTP data are sensitive to noise. Thus, noise reduction is an important goal in the processing of CTP image data. A time-intensity profile similarity (TIPS) bilateral filter [11] was demonstrated to produce high quality images by reducing noise in CTP datasets. Because of the importance of this method for CTP analysis, this filter was used here to demonstrate the potential of the analysis of compressed CTP image data. The TIPS filter is defined as a regular 3D bilateral filter in which the pixel similarity function was replaced by a TIPS function. The TIPS function considers the temporal similarity to determine the agreement of 2 voxels and is defined as

$$\rho(\xi, x) = \exp\left(-\frac{1}{2}\left(\frac{\varsigma(\xi, x)}{\sigma\varsigma}\right)^2\right),$$

where $\rho(\xi, x)$ is the sum of the squared differences (SSD) between the intensities of a voxel at x and a voxel at ξ in time. Thus, the SSD is defined as:

$$\varsigma(\xi, \mathrm{x}) = \sum_{t=0}^{T-1} (I(\xi, t) - I(\mathrm{x}, t))^2,$$

where T is the number of time steps. In this way, we calculate a new (filtered) intensity value $h(\mathrm{x}, t)$ for each intensity value $I(\mathrm{x}, t)$ in the CTP dataset. $\mathrm{h}(\mathrm{x}, t)$ is defined as:

$$\frac{1}{r(x)} \sum_{i=-m}^{m} \sum_{i=-n}^{n} \sum_{i=-o}^{o} I\,(\xi(x+i, y+j, z+k), t) c(\xi, \mathrm{x}) \rho(\xi, \mathrm{x}),$$

where m, n, and o are, respectively, the half kernel sizes in the x, y, and z directions

$$\mathrm{c}(\xi, \mathrm{x}) = \exp\left(-\frac{1}{2}\left(\frac{d(\xi, \mathrm{x})}{\sigma d}\right)^2\right),$$

$$r(x) = \sum_{i=-m}^{m} \sum_{i=-n}^{n} \sum_{i=-o}^{o} c\,(\xi, \mathrm{x}) \rho(\xi, \mathrm{x}),$$

and $d\,(\xi, \mathrm{x})$ is the Euclidean distance between the voxel at x and the voxel at ξ.

Besides the TIPS filtering, we also performed a double threshold segmentation on the CTP data. This segmentation can be used, for instance, to create a binary mask of the cerebral spinal fluid (0-15 HU). This mask is useful for the automatic calculation of parameters such as the noise level in a CTP image [11]. The double threshold segmentation of dynamic CTP datasets evaluates all the voxels in the 4 dimensions and produces a mask with the same dimension as the input image data.

Implementation details

Regarding the TIPS filter, we used the original CPU-based implementation (CPUFIL) from [11] and two additional GPU implementations: one processing uncompressed data (FIL) and another processing compressed data (FIL-COM). Regarding the thresholding segmentation, we only used two GPU implementations: one of them handles compressed CTP data (SEG-COM) and the other does not (SEG).

The host applications of all GPU implementations were developed in the .NET Framework version 4.0 [13]. The library Fellow Oak DICOM version 1.0.36 [14] was used for handling DICOM files. For the GPU computations, OpenCL 1.1 [15] was used and OpenCL.NET version 2.2.9 [16] was used to integrate OpenCL with .NET.

CPU-FIL was implemented in C++, being accessed and executed through MATLAB.

This implementation only uses CPU computations and it was optimized for minimizing cache misses. In FIL and FIL-COM, the filtered intensities over time of a voxel are stored in a single OpenCL data structure (see lines 1-4 in Figure 6). In these implementations, the function $p(\xi, x)$ was implemented as an OpenCL function (see lines 44-45 in Figure 6); $h(x, t)$ was implemented as an OpenCL kernel; $c(\xi, x)$ was implemented as a lookup table with the same size of the kernel ($2_m \times 2_n \times 2_o$) (see lines 27-28 in Figure 6); and $r(x)$ is computed together with the calculations of h(x, t) in an OpenCL kernel (see lines 59-64 in Figure 6). Similarly to FIL and FIL-COM, the segmentation results of SEG and SEG-COM of a single voxel over time are grouped in the same OpenCL data structure. Each binary segmentation result is represented in a byte (0 = false, other values = true).

Since the compression method allows direct access to the individual voxels, the main difference between FIL and FIL-COM, and also between SEG and SEG-COM, is the way in which the data is transferred to the GPU and how the input data is accessed. These differences are highlighted in Figure 6 and in Figure 7. In FIL and SEG, the data is sent to the GPU in an array of OpenCL data structures (see lines 6-7 in Figure 6). Each data structure contains all the intensity values of one voxel over time in its original 16 bits integer representation. In FIL-COM and SEG-COM, the input data is sent in the arrays C, O, B, and D using the compressed representation (see lines 1-5 in Figure 7). A few extra operations (bit shift, add) need to be performed to reconstruct the original intensity value of a voxel. The OpenCL code used for these recostructions is shown in Figure 5. In both GPU implementations, all the data stored in global memory used in the kernel computations is copied to local memory prior to processing.

Code for retrieving a specific $\Delta(\mathbf{x}, t)$ from the array **D**

```
1. constant uint Masks[17] =  {0, 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023,
2.                             2047, 4095, 8191, 16385, 32767, 65537};
3.
4. int GetBitVectorValueUsingOffset(int offset, int index, int bits, global uint *vector)//vector = array D
5. {
6.     int currentValueBitIndex = index * bits;
7.     int nextValueBitIndex = currentValueBitIndex + bits;
8.     int firstWordIndex = offset + currentValueBitIndex / sizeof(uint);
9.     int secondWordIndex = offset + (nextValueBitIndex - 1) / sizeof(uint);
10.    int nextValueBitIndexMod32 = nextValueBitIndex % sizeof(uint);
11.    int rightShift = sizeof(uint) - nextValueBitIndexMod32;
12.    if (firstWordIndex == secondWordIndex)
13.    {
14.        return (vector[firstWordIndex] >> rightShift) & Masks[bits];
15.    }
16.    uint firstWordValue = (vector[firstWordIndex] & Masks[bits - nextValueBitIndexMod32]) << nextValueBitIndexMod32;
17.    uint secondWordValue = (vector[secondWordIndex] >> rightShift) & Masks[nextValueBitIndexMod32];
18.    return firstWordValue | secondWordValue;
19.}
```

Figure 5: Code for accessing the a specific $\Delta(x; t)$ from a set D_x stored in the array D, which is referred in the code as vector. Note that only operations with constant computational complexity are used. Therefore, the computational complexity for accessing the intensity value of a single voxel is also constant. One bit shift operation is required when accessing a single element of D (line 14). When two elements are accessed, two bit shifts are needed (lines 16-17).

Validation

We performed four experiments to validate the analysis of compressed CTP image data. Two of these experiments verify the differences between the results produced by the processing of compressed and uncompressed data. The other two experiments quantify the differences in the usage of computational resources while processing compressed and uncompressed CTP data on GPUs. We performed a voxel-wise comparison of the results of CPU-FIL, FIL, and FIL-COM to establish whether the outputs of all implementations are equivalent. In this experiment, we used 20 datasets with 320 slices because they have more noise than the datasets with 6 slices, which could intensify the differences in the results from the different implementations. These 20 datasets were corrected for motion using rigid 3D registration [17] prior to filtering and compression. We used the following values for the parameters of the filter: kernel size = 41 × 41 × 35 voxels, σ_ζ = 150, σ_d = 2.5, and clamping the voxels at the border of the image.

To evaluate the usage of computational resources in FIL and FIL-COM we measured the memory usage (size of the GPU memory buffers with the input data), the GPU processing time (excluding the time for data transfers), the time to transfer data to and from the GPU, and the total GPU time (processing + transfers). In this experiment we used all the 29 datasets without motion correction. Also, we used different filter parameters, since some datasets only have 6 slices: kernel size = 5 × 5 × 5 voxels, σ_ζ = 150, σ_d = 0.5, and clamping the voxels at the border of the image.

The total GPU processing time spent by SEG-COM and SEG was measured to evaluate their differences in the usage of computational resources. The same parameters used for the voxel-wise comparison were used in this experiment. All the GPU computations were performed in the same hardware configuration (see Table 1) using Windows 7 Professional 64 bits as operating system. For time measurements, high-resolution timing counters provided by the Win32 API were used.

Table 1: Hardware configuration on which the validation experiments were performed.

CPU Name:	Intel Xeon E5-2620
CPU Clock:	2.00 GHz
CPU Cores:	6
CPU Threads:	12
RAM Memory:	64 GB
GPU Name:	GeForce GTX TITAN
GPU Driver Version:	331.65
GPU Cores:	2688
GPU Clock:	836 MHz
Dedicated Video Memory:	6 GB GDDR5

Results and discussion

The filtering results obtained from FIL and FIL-COM are exactly the same. However, a few differences were observed when comparing the results from the GPU implementations with the results from the CPU implementation. In Figure 8 we can observe the differences between these results for one dataset. In Figure 9 we show a histogram of the differences in Hounsfield units between the filter results from the original and the GPU implementations for all datasets.

The original TIPS filter implementation is targeted to CPU computing and it takes around 30 minutes to process a complete CTP dataset. To prevent the original implementation to take even longer, some kernel computations are truncated on purpose. This is the main reason for the differences between the results from CPUFIL and the results from FIL and FIL-COM. Since the GPU implementations are much faster, we do not need to truncate the kernel computations. Nevertheless, even if the original implementation would perform all the computations, some differences are still expected due to rounding-off operations and different ways that different APIs implement basic operations like square root, power, etc.

Table 2 summarizes the gain in the usage of computational resources by FILCOM and FIL. The values in Table 2 are calculated by dividing a measurement of computational resource usage (time, memory) collected from FIL by that collected from FIL-COM (uncompressed/compressed).

When comparing the computational resources used by FIL and FIL-COM we opted for not using motion correction because it interferes with the final compression ratio, which consequently interferes with the memory usage. The motion correction may reduce the range of values of a voxel over time. Since the compression algorithm explores this characteristic, a higher compression ration can be achieved.

Table 2: Gains from running the TIPS filter on GPU with compressed CTP data relative to uncompressed data.

	minimum	average	maximum
speed up in time to send input data	1.9	2.2	2.6
speed up in processing time	1.2	1.3	1.4
speed up in time to get output data	0.8	1.0	1.5
compression ratio of input data	2.0	2.2	2.8
speed up in total time (transfers + processing)	1.2	1.3	1.6

As can be observed in Table 2, the processing of compressed CTP data has several advantages over processing of uncompressed CTP data. By performing direct processing of compressed CTP data, it is possible to reduce not only data transfer time and memory usage, but also GPU processing time. One of the main advantages of our approach is that it enables fitting more data into the GPU memory, and consequently also doing more computations before transferring new input data from the CPU to the GPU. This can reduce the overhead caused by the communication between GPU and host application.

The only aspect that was not improved when comparing FIL and FIL-COM data was the transfer of the results from the GPU memory. This happened because the filter results were not compressed after processing, even though this is possible. By compressing the filter results, the time to retrieve the output data from the GPU memory should be reduced in a similar way as the time to send the input data. However, the time required to perform the compression will be added to the total GPU processing time. Moreover, no delay or overhead was introduced in the time to get the output data of FIL-COM compared to FIL. Figure 10 shows in detail the time spent in each step of the execution of FIL and FIL-COM when processing the CTP datasets of 320 slices, which are bigger and consequently have slower execution time compared to the datasets of 6 slices.

Regarding SEG-COM to SEG, the segmentation results of these implementations were identical. When considering all 29 CTP datasets, the total GPU usage of SEGCOM was between 1.2 and 1.7 times faster than SEG, with an average speedup of 1.5 times. Also, no differences were observed in the time to transfer the results from the GPU memory to host application memory. However, in this case the segmentation results cannot be compressed because they are binary masks and the compression algorithm used can only compress grey scale time series images. Figure 11 shows the total GPU processing time of SEG and SEG-COM implementations. As we can observe in Figure 11, the total GPU processing time of the segmentation was reduced in the same proportion as the total processing time of the filter.

As observed in Figure 10 and Figure 11, a few seconds were reduced in the total execution time when processing compressed CTP data. However, by using this approach in all steps of an entire CTP analysis pipeline, more considerable gains are expected. Furthermore, segmentation of infarct core, generation of arterial input function, and calculation of many other parameters from CTP data can benefit from postprocessing techniques as the TIPS filter [4, 11]. Thus, a processing time 1.3 times faster and an input data 2 times smaller would be very beneficial in a CTP analysis pipeline with extensive use of postprocessing techniques, thin slices CTP scans, and multimodal data.

Conclusions

We have presented a novel approach for processing compressed dynamic CTP data. This approach takes advantage of the data structures used to store the compressed data to enable massively parallel data processing on GPUs, without requiring major modifications to the CTP processing algorithms. We have shown that this approach was able execute the GPU implementations of a well-established TIPS filter and a generic bi-threshold segmentation 1.3 times faster. This speedup was achieved with no modification in the source code implementing the filter or the segmentation calculations. The only modification required is related to the access to a voxel intensity, which is represented as $C(\mathrm{x})+\Delta(\mathrm{x}, t)$ instead of the original value $I(\mathrm{x}, t)$.

Our strategy does not add costly computational operations for data decompression, and it reduces memory usage and processing time considerably. Because of that, it is expected that any other GPU-based processing of CTP data could also benefit from our approach. Consequently, our method may contribute to achieving faster treatment and diagnosis based on GPU processing of CTP image analysis, which is relevant for optimal treatment selection of patients suspected of acute ischemic stroke.

References

1. Borst, Jordi, et al. "Value of computed tomographic perfusion–based patient selection for intra-arterial acute ischemic stroke treatment." *Stroke* 46.12 (2015): 3375-3382.
2. Allmendinger, Andrew Mark, et al. "Imaging of stroke: Part 1, perfusion ct??? Overview of imaging technique, interpretation pearls, and common pitfalls." *American Journal of Roentgenology* 198.1 (2012): 52-62.
3. Wintermark, Max, et al. "Perfusion-CT assessment of infarct core and penumbra: receiver operating characteristic curve analysis in 130 patients suspected of acute hemispheric stroke." *Stroke* 37.4 (2006): 979-985.
4. Geuskens, Ralph REG, et al. "Characteristics of misclassified CT perfusion ischemic core in patients with acute ischemic stroke." *PLoS One* 10.11 (2015): e0141571.
5. Mukhopadhyay, J.: Image and Video Processing in the Compressed Domain. CRC Press, ??? (2011)
6. Hu, Qinrui, and Guoqiang Xiao. "Image Segmentation Based on Kernel Clustering in Compressed Domain." *2014 International Conference on Information Science & Applications (ICISA)*. IEEE, 2014.
7. Thies, William, Steven Hall, and Saman Amarasinghe. "Manipulating lossless video in the compressed domain." *Proceedings of the 17th ACM international conference on Multimedia*. 2009.
8. Eklund, Anders, et al. "Medical image processing on the GPU–Past, present and future." *Medical image analysis* 17.8 (2013): 1073-1094.
9. Brodtkorb, André R., Trond R. Hagen, and Martin L. Sætra. "Graphics processing unit (GPU) programming strategies and trends in GPU computing." *Journal of Parallel and Distributed Computing* 73.1 (2013): 4-13.
10. Barros, Renan Sales, et al. "Dynamic CT perfusion image data compression for efficient parallel processing." *Medical & biological engineering & computing* 54.2-3 (2015): 463-473.
11. Mendrik, Adriënne M., et al. "TIPS bilateral noise reduction in 4D CT perfusion scans produces high-quality cerebral blood flow maps." *Physics in Medicine & Biology* 56.13 (2011): 3857.
12. Berkhemer, Olvert A., et al. "A randomized trial of intraarterial treatment for acute ischemic stroke." *n Engl J Med* 372 (2015): 11-20.
13. NET Framework (2015). http://www.microsoft.com/net
14. Fellow Oak Dicom for .NET (2015). https://github.com/rcd/fo-dicom
15. The OpenCL Specification (2011). https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf
16. OpenCL.NET (2013). https://openclnet.codeplex.com/
17. Fahmi, Fahmi, et al. "3D movement correction of CT brain perfusion image data of patients with acute ischemic stroke." *Neuroradiology* 56.6 (2014): 445-452.

Noise reduction on uncompressed data

```
1. typedef struct Voxel
2. {
3.     short TimeSteps[T];
4. } Voxel;
5.
6. kernel void NoiseReduction(global Voxel *inputVoxels,
7.                            global Voxel *outputVoxels)
8. {
9.     size_t i = get_global_id(0);
10.
11.    size_t kernelIndices[rowsTimesColumnsTimesSlices];
12.    short3 pos = From1DTo3D(i);
13.    float gaussianClosenessArray[rowsTimesColumnsTimesSlices];
14.    int index = 0;
15.    for (int x = -halfKernelX; x <= halfKernelX; ++x)
16.    {
17.        for (int y = -halfKernelY; y <= halfKernelY; ++y)
18.        {
19.            for (int z = -halfKernelZ; z <= halfKernelZ; ++z)
20.            {
21.                short3 newPos = pos;
22.                newPos.x = clamp(pos.x + x, minX, maxX);
23.                newPos.y = clamp(pos.y + y, minY, maxY);
24.                newPos.z = clamp(pos.z + z, minZ, maxZ);
25.                kernelIndices[index] = From3DTo1D(pos);
26.
27.                //c(ξ, x)
28.                gaussianClosenessArray[index] = GaussianClosenessLookUpTable[abs(x) +
29.                                                                             abs(y) * (halfKernelX + 1) +
30.                                                                             abs(z) * (halfKernelX + 1) * (halfKernelY + 1)];
31.                ++index;
32.            }
33.        }
34.    }
35.
36.    Voxel currentVoxel = inputVoxels[i];
37.
38.    Voxel kernelVoxels[rowsTimesColumnsTimesSlices];
39.    float tipsArray[rowsTimesColumnsTimesSlices];
40.    for(index = 0; index < rowsTimesColumnsTimesSlices; ++index)
41.    {
42.        kernelVoxels[index] = inputVoxels[kernelIndices[index]];
43.
44.        //p(ξ, x)
45.        tipsArray[index] = TimeIntensityProfileSimilarity(currentVoxel, kernelVoxels[index], sigmaZeta);
46.    }
47.
48.    Voxel outputVoxel;
49.    for(int t = 0; t < T; ++t)
50.    {
51.        float normalizationFactor = 0.0f;
52.        float intensityAccumulator = 0.0f;
53.
54.         for(index = 0; index < rowsTimesColumnsTimesSlices; ++index)
55.        {
56.
57.            intensityAccumulator += currentVoxel.TimeSteps[t] * gaussianClosenessArray[index] * tipsArray[index];
58.
59.            //r(x)
60.            normalizationFactor += gaussianClosenessArray[index] * tipsArray[index];
61.        }
62.
63.        //h(x, t)
64.        outputVoxel.TimeSteps[t] = (short)(intensityAccumulator / normalizationFactor);
65.    }
66.
67.    outputVoxels[i] = outputVoxel;
68. }
```

Figure 6: Code snippet of the OpenCL kernel that implements the noise reduction filter on uncompressed CTP data. The lines of code highlighted in light gray are the only lines that need to be changed to adapt this code for processing compressed CTP data. These lines need to be changed because they are related to the way the input data is stored in the GPU memory.

Noise reduction on compressed data

```
1.  kernel void NoiseReductionWithCompression(global uint* transformedData, //array D, all the sets Dx are stored in this array
2.                                            global int* offset,           //array O
3.                                            global short* minimum,        //array C, C(x) = minVx
4.                                            global uchar* bits,           //array B
5.                                            global Voxel *outputVoxels)
6.
7.      Voxel currentVoxel;
8.      int minValue = minimum[i];
9.      int bitValue = bits[i];
10.     int offsetValue = offset[i];
11.     for(int t = 0; t < T; ++t)
12.     {
13.         //I(x, t) = C(x) + Δ(x, t)
14.         currentVoxel.TimeSteps[t] = minValue + GetBitVectorValueUsingOffset(offsetValue, t, bitValue, transformedData);
15.     }
16.
17.         int newMinValue = minimum[kernelIndices[index]];
18.         int newBitValue = bits[kernelIndices[index]];
19.         int newOffsetValue = offset[kernelIndices[index]];
20.         for(int t = 0; t < T; ++t)
21.         {
22.             //I(x, t) = C(x) + Δ(x, t)
23.             kernelVoxels[index].TimeSteps[t] = newMinValue +
24.                                                GetBitVectorValueUsingOffset(newOffsetValue, t, newBitValue, transformedData);
25.         }
```

Figure 7: Code snippet of the OpenCL kernel that implements the noise reduction filter on compressed CTP data. Only the lines of code that need to be changed from the source code presented in Figure 6 are shown. These lines are: the kernel signature (1-5), which needs to be changed to receive the compressed data as input; the code to access all the intensity values of the current voxel over time (7-15), in this case each of these values need to be retrieved from the array D; and the code to access the intensity values over time of the voxels in the kernel (17-25).

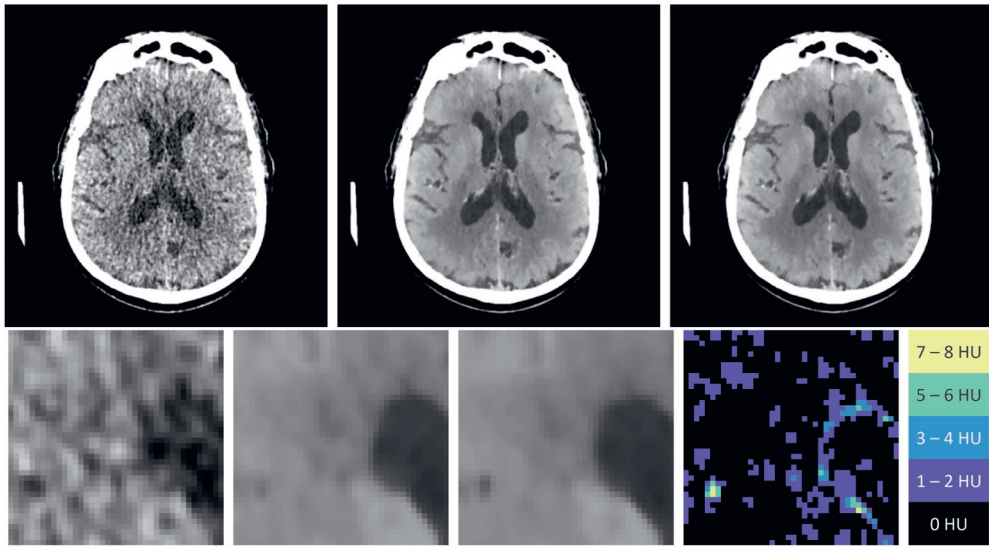


Figure 8: CTP filtering results. From left to right, the top row shows: a sample slice from the input data, the result produced by the original CPU implementation, and the result from the GPU implementations. The bottom row shows in detail: input data, CPU result, GPU result, and absolute differences in Hounsfield units (HU).

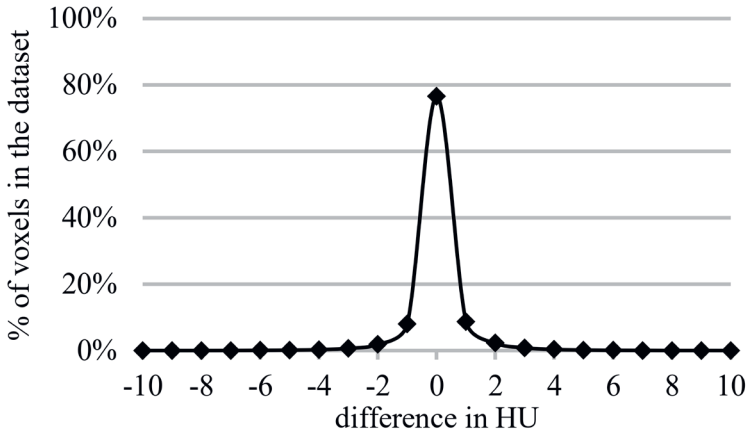


Figure 9: Histogram of differences in voxel intensities between the results of the original CPU implementation and the GPU implementations of the TIPS filter. All voxels from all datasets were evaluated and in more than 99% of them the difference is ±5 Hounsfield units (HU) or less.

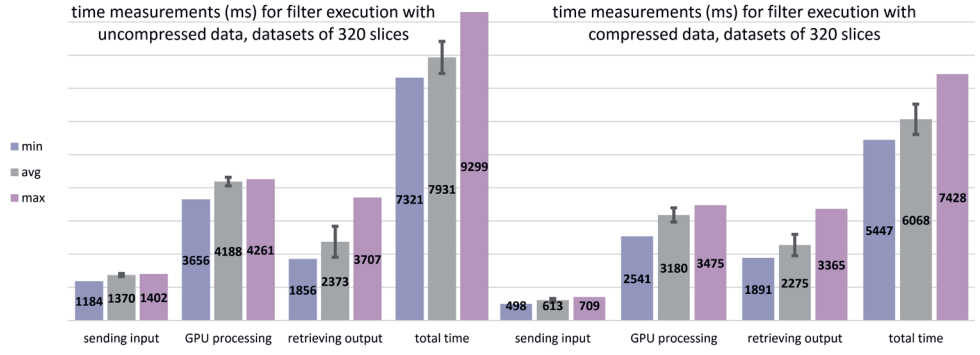


Figure 10: Time measurements of the TIPS filter (in milliseconds) when processing compressed and uncompressed CTP data.

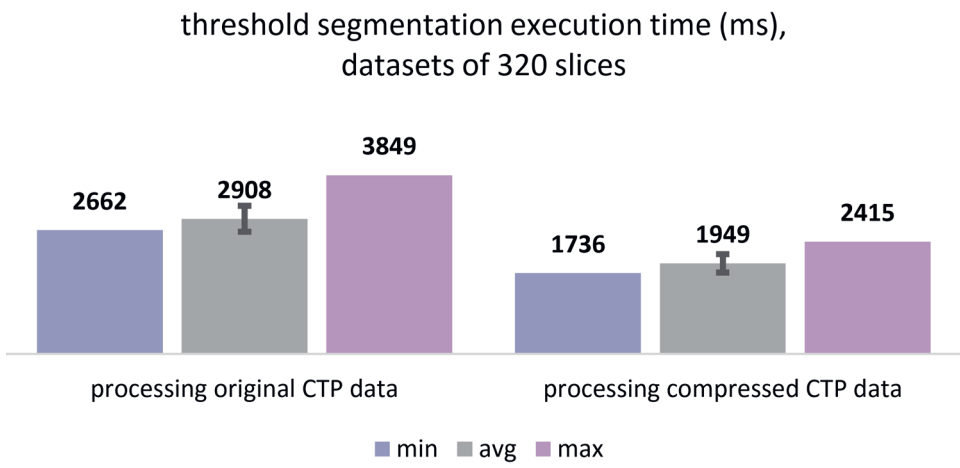
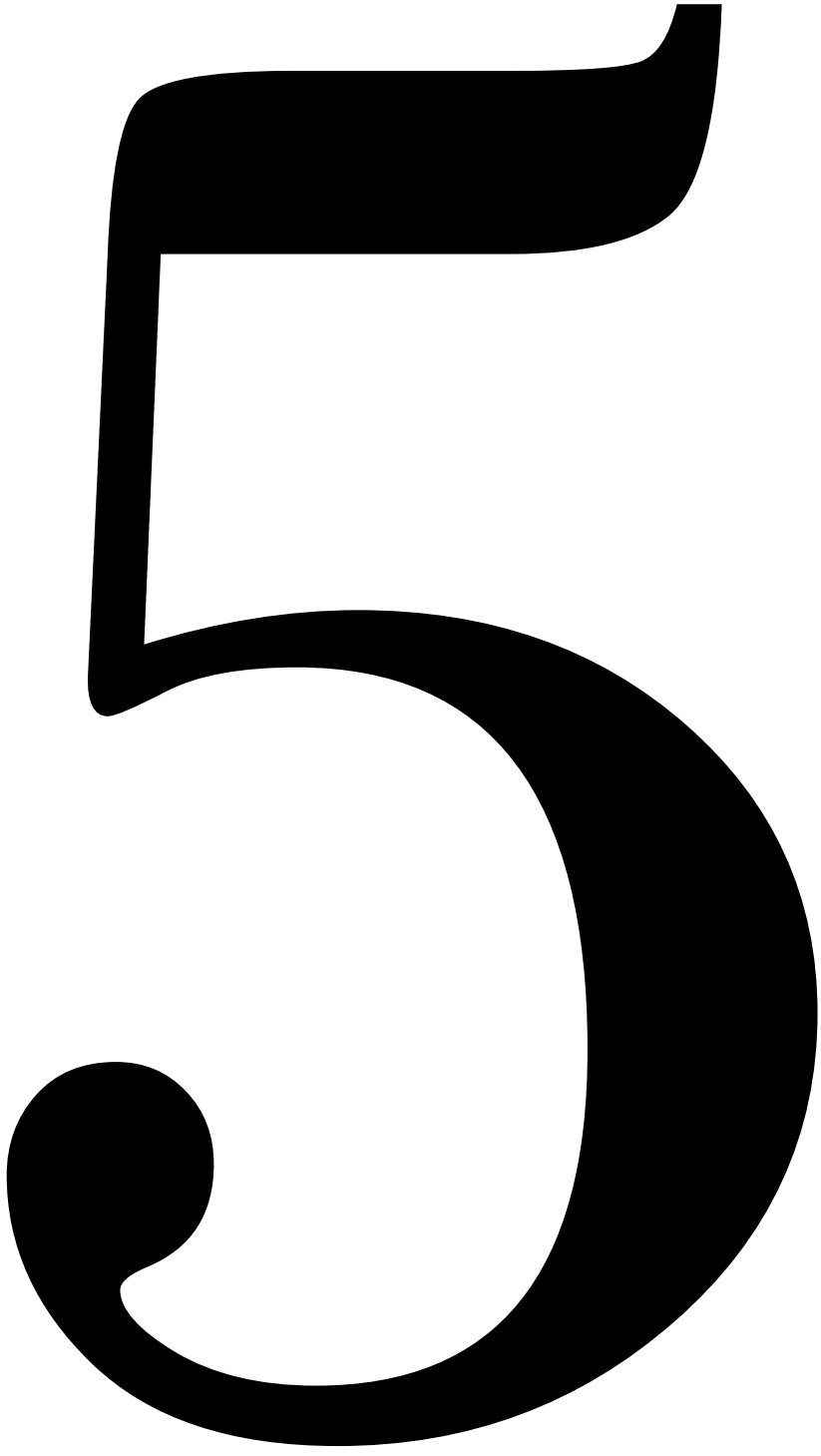


Figure 11: Total GPU usage time (data transfers + processing) in milliseconds for the double threshold segmentation when processing uncompressed (left) and compressed (right) CTP data for 20 datasets with 320 slices.



Chapter 5

Remote collaboration, decision support, and on-demand medical image analysis for acute stroke care

Renan Sales Barros[a](*) , Jordi Borst[a] , Steven Kleynenberg[b] , Céline Badr[c] , Rama-Rao Ganji[d] , Hubrecht de Bliek[e] , Landry-Stéphane Zeng-Eyindanga[f] , Henk van den Brink[g] , Charles Majoie[a] , Henk Marquering[a] , and Sílvia Delgado Olabarriaga[a]

[a] Academic Medical Center, University of Amsterdam, Amsterdam, The Netherlands
[b] Sopheon, Maastricht, The Netherlands
[c] Prologue, Les Ulis, France
[d] ARTEMIS Department, Telecom SudParis, Evry, France
[e] Philips Healthcare, Eindhoven, The Netherlands
[f] Bull, Grenoble, France
[g] Technolution, Gouda, The Netherlands

- Barros, Renan Sales, et al. "Remote collaboration, decision support, and on-demand medical image analysis for acute stroke care." *European Conference on Service-Oriented and Cloud Computing*. Springer, Cham, 2015.
- First author, conference proceedings, https://link.springer.com/chapter/10.1007/978-3-319-24072-5_15

Abstract

Acute stroke is the leading cause of disabilities and the fourth cause of death worldwide. The treatment of stroke patients often requires fast collaboration between medical experts and fast analysis and sharing of large amounts of medical data, especially image data. In this situation, cloud technologies provide a potentially cost-effective way to optimize management of stroke patients and, consequently, improve patient outcome. This paper presents a cloud-based platform for Medical Distributed Utilization of Services & Applications (MEDUSA). This platform aims at improving current acute care settings by allowing fast medical data exchange, advanced processing of medical image data, automated decision support, and remote collaboration between physicians in a secure and responsive virtual space. We describe a prototype implemented in the MEDUSA platform for supporting the treatment of acute stroke patients. As the initial evaluation illustrates, this prototype improves several aspects of current stroke care and has the potential to play an important role in the care management of acute stroke patients.

Keywords: Acute care · Cloud computing · Decision support · High performance computing · Medical image analysis · Remote collaboration · Stroke · Telemedicine

1 Introduction

Acute ischemic stroke is the leading cause of disability and fourth cause of death [1]. In acute ischemic stroke, a blood clot obstructs blood flow in the brain causing part of the brain to die due to the lack of blood supply. The amount of brain damage and the patient outcome is highly related to the duration of the lack of blood flow ("time is brain"). Therefore, fast diagnosis, decision making, and treatment are crucial in acute stroke management.

Medical data of a stroke patient is collected during the transport by ambulance to the hospital (e.g. vital signs, patient history, and medication). At arrival, various types of image data are acquired following protocols that involve opinions and decisions from various medical experts. Sometimes, a patient needs to be transferred to a specialized hospital and, in this case, it is important that all the data collected in the ambulance and at the referring hospital is available to the caregivers that will continue the treatment. Often, various medical specialists need to collaborate based on available information for determining the correct diagnosis and choosing the best treatment. Usually, this collaboration is based on tools that are not connected to each other and, because of that, they may not deliver the necessary information rapidly enough.

In addition to these challenges, the amount of patient medical data is growing fast [2]. This fast increase is especially observed in radiological image data, which is also a consequence of new medical imaging technologies [3, 4]. The management, sharing, and processing of medical image data is a great challenge for healthcare providers [3, 4] and they can be greatly improved by the usage of cloud technologies [5]. Cloud technologies also enable collaboration and data exchange between medical experts in a scalable, fast, and cost-effective way [5]. Mobile devices, remote collaboration tools, and on-demand computing models and data analysis tools supported by cloud technologies may play an important role to help in optimizing stroke treatment and, consequently, improve outcome of patients suffering from stroke.

In this paper, we present a cloud-based platform for Medical Distributed Utilization of Services & Applications (MEDUSA). This platform aims at improving current acute care settings by allowing fast medical data exchange, advanced processing of medical image data, automated decision support, and remote collaboration between physicians through a secure responsive virtual space. We discuss a case study implemented using the MEDUSA platform for supporting the treatment of acute stroke patients, presenting the technical details of the prototype implementation and commenting on its initial evaluation.

2 Related Work

The development of cloud-based platforms for collaboration and processing of medical data is a challenging task. Many authors [4, 5, 6, 7] put forward that these platforms hold the potential to define the future of healthcare services. Also, the analysis of medical data can be an important way to improve quality and efficiency in healthcare [8, 9].

The work presented in [10, 11] focuses on the development of a cloud-based solution aimed at only the storage and sharing of medical data. In other words, they propose solutions based on cloud infrastructures to facilitate medical image data exchange between hospitals, imaging centers, and physicians. A similar solution is presented in [12], however focusing on medical data sharing during emergency situations. A cloud-based system is presented in [13] for storage of medical data with an additional functionality that enables content-based retrieval of medical images. Still focusing on cloud-based data storage and sharing, [14] presents a solution to help managing medical resources for the prevention and treatment of chronic stroke patients.

In addition to storage and sharing, some studies also include the possibility of using the cloud infrastructure for processing of medical data. A simple cloud-based application is presented in [15] to monitor oxygenated hemoglobin and deoxygenated hemoglobin concentration changes in different tissues. Cloud computing is also used in [16] not only to support data storage and sharing, but also to visualize and render medical image data. In [17] the authors also propose a cloud application for rendering of 3D medical imaging data. This application additionally manages the cloud deployment by considering scalability, operational cost, and network quality.

Complete cloud-based systems for medical image analysis are presented in [18, 19, 20]. However, in these systems, image upload and download is manually performed by the user, while the system focuses on the remote processing, storage, and sharing of medical image data. The MEDUSA platform not only provides cloud-based storage, sharing, and processing of medical image data, but also real-time communication between medical experts, real-time collaborative interaction of the medical experts with the medical data, and a real-time decision support system that continuously processes patient data and displays relevant notifications about the patient condition.

The MEDUSA platform also includes a cloud management layer that coordinates the use of resources in the cloud infrastructure. Other studies also present some cloud management features. In [21] the authors propose a cloud architecture that reserves network and computing resources to avoid problems regarding load-balancing mechanisms of cloud infrastructures and to reduce the processing delays for the medical

applications. Also, [2] proposes an algorithm to optimize the organization of medical image data and associated processing algorithms in cloud computing nodes to increase the computing performance. Finally, [3] presents a cloud-based multi-agent system for scalable management of large collections of medical image data.

The project presented in [22] tries to speed up current stroke care by integrating and sharing data from stroke patients using mobile networks. In this scenario, a hospital can, for instance, be prepared with the right resources before the arrival of the patient. This project also includes decision support, which suggests a predefined path through the emergency procedures according to the structure of mandatory and other supplementary healthcare protocols. However, differently from MEDUSA, this project does not include any image processing based feature.

3 Acute Stroke Care

Currently, treatment decision of stroke patients is increasingly driven by advanced imaging techniques. These imaging techniques consist of non-contrast computed tomography (ncCT), computed tomography angiography (CTA), and computed tomography perfusion (CTP). Because of the extensive usage of imaging techniques, it is common to produce gigabytes of image data per patient.

The primary treatment for patients with acute ischemic stroke is intravenous administration of alteplase (thrombolysis). Patients who are not eligible for treatment with alteplase or do not respond to the treatment can be treated by mechanical removal of the blood clot via the artery (thrombectomy). Thrombectomy is only available in specialized hospitals and often a patient must be transferred for treatment.

This transfer is arranged via telephone and imaging data created in the initial hospital is not available for the caregivers in the specialized hospital until the patient and imaging data arrive via the ambulance. On a regular basis it happens that the imaging data was wrongly interpreted in the initial hospital and that the patient is not eligible for thrombectomy. Also, often new imaging acquisitions have to be redone due to broken DVDs, wrong data, or insufficient quality. These problems result in futile transfers and loss of valuable time.

4 MEDUSA Platform

The MEDUSA platform was designed to support remote collaboration and high performance processing of medical data for multiple healthcare scenarios. The platform

is accessible to final users through the MEDUSA Collaboration Framework (MCF), which is a web application that is compatible with any web browser that supports HTML5. The MCF is a special type of MEDUSA application that provides to the users an entry point to access other MEDUSA applications. A cloud management layer controls the deployment and execution of all MEDUSA applications in one or more cloud providers. Figure 1 illustrates the architectural design of the MEDUSA platform.

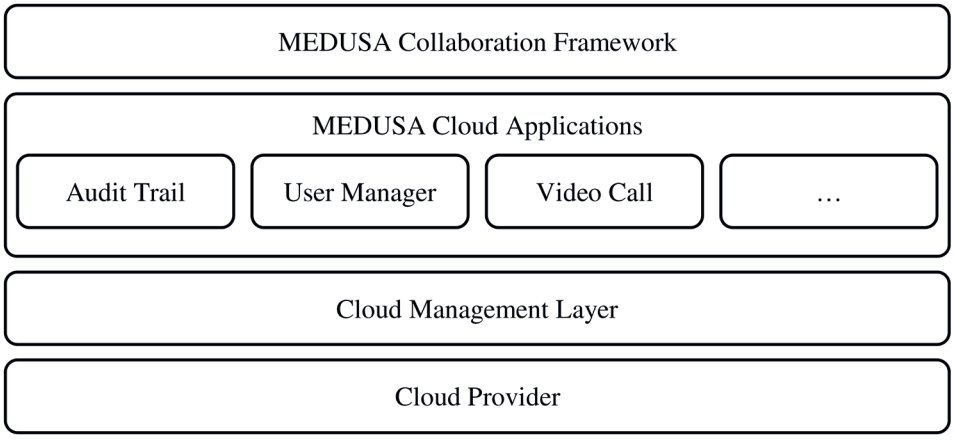


Figure 1: The MEDUSA platform architecture.

4.1 MEDUSA Cloud Applications

The MEDUSA platform has a number of cloud applications that are available in all healthcare scenarios: Audit Trail, which reports the events generated by the other MEDUSA applications; User Manager, which allows assigning roles to users and defining which MEDUSA applications they can use; and Video Call, which allows communication between users of the MEDUSA platform.

The MEDUSA applications are started as part of a MEDUSA session. Multiple users in a session can interact with these applications, and these interactions are visible to all the users in the session. The handling of multiple user interactions is done by each MEDUSA application. The applications in the MEDUSA platform can be web applications or regular desktop applications. The desktop applications are integrated in the MEDUSA platform through a virtualization server that uses the technologies described in [23] and [24]. The multi-user interaction of the desktop applications is handled by the virtualization server.

4.2 Cloud Provider

The MEDUSA applications can be deployed in different cloud providers. Currently, these applications are being deployed in the High Performance Real-time Cloud for

Computing (HiPeRT-Cloud) of Bull. The HiPeRT-Cloud is mainly designed for realtime computationally-intensive workloads. This solution is fully compatible with the Cloud Computing Reference Architecture of the National Institute of Standards and Technology (NIST) and provides infrastructure services under any cloud broker solution. The HiPeRT-Cloud is used in the MEDUSA platform because it provides solutions for handling complex applications in the field of real-time computational and data-intensive tasks in the cloud.

4.3 Cloud Management Layer

In order to take advantage of the on-demand, flexible, high-performance, and costeffective options that cloud providers can offer, the cloud management layer, implemented by Prologue, manages the cloud deployment in the MEDUSA platform. This layer orchestrates the allocation and release of resources on the cloud provider's infrastructure. It also oversees the lifecycle of the deployed resources, ensures their availability and scalability, and links the desktop applications from the virtualization server back to the MCF. The cloud management layer is designed according to the Service-Oriented Architecture model and its functionalities are accessible through a Representational State Transfer Application Programming Interface (REST API). The cloud management layer also incorporates a monitoring service that operates by accessing directly the deployed virtual machines (VMs). The technology behind the cloud management layer is aligned with the NIST architecture and based on the Open Cloud Computing Interface specifications.

In the MEDUSA context, technical requirements for computing, storage, network, and security resources have been identified for each MEDUSA application to be deployed. All requirements are then translated into machine-readable code that is used to provision the cloud resources.

The components of the MEDUSA platform are hosted on the cloud through a security-aware, need-based provisioning process. By supporting on-demand hybrid and multi-cloud deployments, as well as monitoring, load balancing, and auto-scaling services through an agent embedded in each VM, the cloud management layer thus ensures a high resilience of the MEDUSA platform.

4.4 Security

The security of the MEDUSA platform is currently mainly based in the use of digital certificates, which are used to authenticate MEDUSA applications (VMs), to secure the data exchanges through the network, and to provide strong authentication of MEDUSA users.

The VMs containing the applications are deployed dynamically, and thus server certificates need to be created dynamically, during the deployment. A web service was developed to provide dynamic generation of server certificates for the different VMs in the MEDUSA platform. These server certificates must be created during the deployment of the VMs and there must be one certificate per application and VM (identified by the IP address).

Regarding the user authentication, an authentication module is called when a user opens a MEDUSA session. This module authenticates a user by checking the provided credentials against the user management component, which has access to a special internal directory containing the certificates used for strong authentication of MEDUSA users.

The MEDUSA platform also uses robust image watermarking and fingerprinting methods to prevent and detect unauthorized modification and leaking of medical images by authorized users by. However, due to legal regulations, an important requirement when dealing with medical images is the capability reconstructing the original image data. Because of this, reversible or semantic-sensitive techniques for watermarking and fingerprinting can be used in the MEDUSA platform. These techniques enable to completely recover the original image data or at least the recovery of the regions of these images that are relevant for the user or application.

5 MEDUSA Stroke Prototype

The MEDUSA platform was designed to support various medical scenarios. Here, we focus on a prototype for supporting acute stroke care. The MEDUSA Stroke Prototype (MSP) is built by combining the default MEDUSA applications with three applications specifically configured to support the treatment of stroke patients: Advanced Medical Image Processing, Decision Support System, and 3D Segmentation Renderer. All the applications of the MSP are executed in VMs running on the HiPeRT-Cloud. The cloud management layer is in charge of the deployment of these VMs.

5.1 Advanced Medical Image Processing

For supporting the assessment of the severity of a stroke, several medical image processing algorithms (MIPAs) have been developed. These algorithms perform quantitative analysis of the medical image data and the result of these analyses can be used to support the treatment decisions. The output of these algorithms are, for example, the segmentation of a hemorrhage in the brain [25], the segmentation of a blood clot [26], and the segmentation of the infarcted brain tissue [27]. The MIPAs are linked together

into processing pipelines with well-defined input, output, and policies that control their execution. The execution of these pipelines is automatically orchestrated to deliver the lowest execution time based on a set of optimization strategies (e.g. task parallelism, data parallelism, and GPU computing).

The MIPAs are implemented as plugins for the IntelliSpace Discovery (ISD) platform, an enterprise solution for research, developed by Philips Healthcare. Figure 2 shows the output of the plugin for infarct volume calculation in the ISD. The collection of MIPAs specially developed to support acute stroke care that are included in the ISD constitutes the Advanced Medical Image Processing application of the MSP.

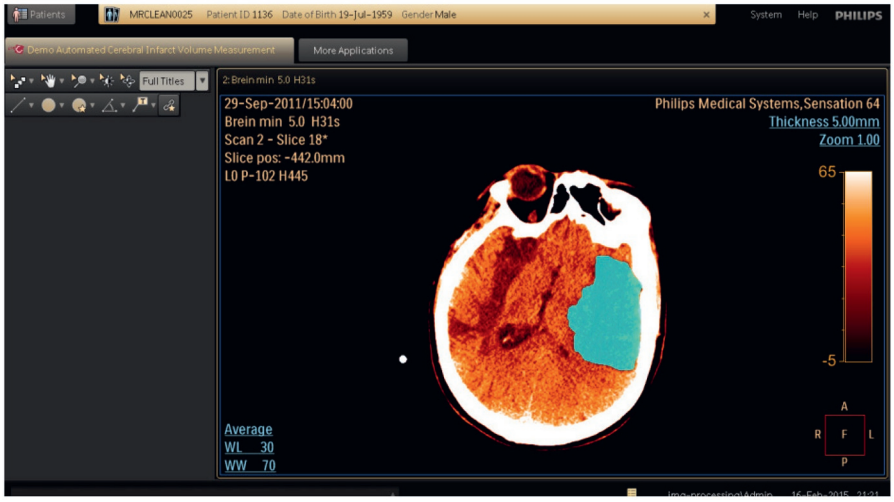


Figure 2: Plugin for automated measurement of the cerebral infarct volume in the ISD.

The ISD is a Windows desktop application developed by using the .NET Framework. The development of the MIPAs is also based in the .NET Framework. For GPU-based computations, OpenCL 1.1 was used. OpenCL is a framework for the development and execution of programs across platforms consisting of different types of processors such as CPUs, GPUs, etc. OpenCL.NET was used to integrate OpenCL with the .NET. Framework.

The data generated by the MIPAs are exported to the DSS by using JavaScript Object Notation (JSON) files through WebSockets. (Anonymized) Patient information is sent to the MIPAs by using the tags of the medical image data used as input. The information about the current session is directly sent to the ISD and forwarded to the MIPAs.

5.2 Decision Support System

The Decision Support System (DSS) by Sopheon provides real-time process support to medical professionals collaborating on the stroke case. The DSS is rule-based: the rules specify the conditions under which actions are to be advised (delivered as notifications). The Decision Support rules are part of a medical protocol and thus defined and approved by medical professionals.

In the MSP, the DSS runs a set of rules specifically designed for dealing with stroke patients. It gathers real-time input from vital sign sensors and MIPAs. For instance, a rule could state that an infarct volume larger than 70 milliliters is associated with a poor outcome for the patient. When the DSS detects an infarct volume value of e.g. 80 milliliters, it will display the notification associated with this condition. The DSS also selects relevant information from the data generated by the MIPAs and forwards it to the audit trail and to the 3D Segmentation Renderer.

The DSS runs on Node.js, which is a platform built on Google Chrome's JavaScript runtime. The DSS is deployed on Fedora, which is an operating system based on the Linux kernel.

5.3 3D Segmentation Renderer

The 3D Segmentation Renderer by Sopheon is responsible for displaying 3D segmentations generated by the MIPAs. This application was developed by using the WebGL library, which enables to render 3D graphics in the browser without installing additional software. Figure 3 shows the GUI of this application rendering the segmentation of brain tissue (in green and blue) and the segmentation of the infarcted region (in red).

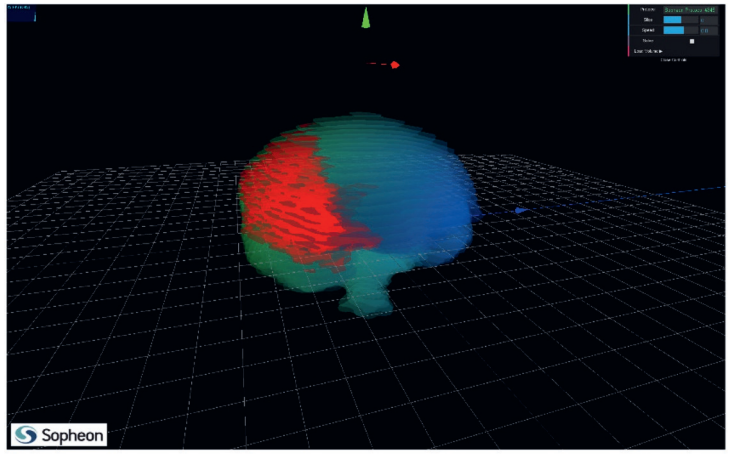


Figure 3: 3D segmentation renderer showing the segmentation of brain tissue (green and blue) and the infarction in the brain (red).

6 Initial Evaluation

As this is an on-going project, the discussion presented below is based upon an evaluation of the first fully-integrated prototype.

The MSP integrates very heterogeneous applications, which run on different operational systems (Windows, Linux) and use different development technologies (Java, OpenCL, C#, C++). These applications are seamlessly available for the user from a single interface. Also, the deployment of the applications is transparently handled by the platform. This solution is provided in a smooth and transparent manner, hiding the complex details from the user.

In the MEDUSA platform, the data and user input need to cross several software layers, which might introduce overheads and decrease performance. However, such poor performance was not noticed in the initial MSP prototype. For instance, the Advanced Medical Image Processing application, which requires data exchange between different architectural components, was almost instantaneously ready for use without noticeable interaction delays.

The MSP implements a complete acute stroke use case, which has been demonstrated live in various occasions. Impressions have been collected informally to assess the potential value of this prototype system. Table 1 compares the current stroke care situation in the Netherlands versus the stroke care that could be supported by the MEDUSA platform based on the functionalities currently present in the MSP.

Because of its complexity, a detailed and quantitative evaluation of the MEDUSA platform involves several software components and requires a careful planning. The design of this evaluation was already defined in the first year of the project. It is scheduled to take place during the last 6 months of the MEDUSA project (end of 2015).

Table 1: Current stroke care vs. stroke care with MEDUSA.

	current	**with MEDUSA**
Data availability	images are not available	images are available online
Time to access data	transport by car of physical media (minutes to hours)	online data transfer (few seconds)
Potential value for decision	automated quantitative analysis not used yet for clinical decision	results of MIPAs readily available as decision parameters
Infrastructure	static, proprietary, fixed scale	pay-per-use, scalable, and portable to different cloud providers
Remote collaboration	by phone	by video-conference with access to the patient data

Concerning the image processing functionality, most of the MIPAs included in the MSP are too computationally expensive to be executed on a local machine according to the time constraints of an acute stroke patient. HPC capabilities delivered by cloud computing were crucial to improve the processing of these algorithms from hours to minutes, making them suitable for acute stroke care. For instance, the time to run the method used to reduce noise in CTP data was reduced from more than half an hour to less than 2 minutes [28].

7 Discussion and Conclusion

The development of the MEDUSA platform started in 2013. Back then, this kind of cloud-based solutions was not common. Today, however, there is a clear trend in the healthcare industry towards the usage of cloud computing, collaboration, and automated analyses of medical data. In addition, when dealing with processing of medical data constrained by the requirements of acute care situations, a lot of benefits can be derived from the use of cloud computing: scalability, pay-per-use model, high performance computing capabilities, remote access, etc.

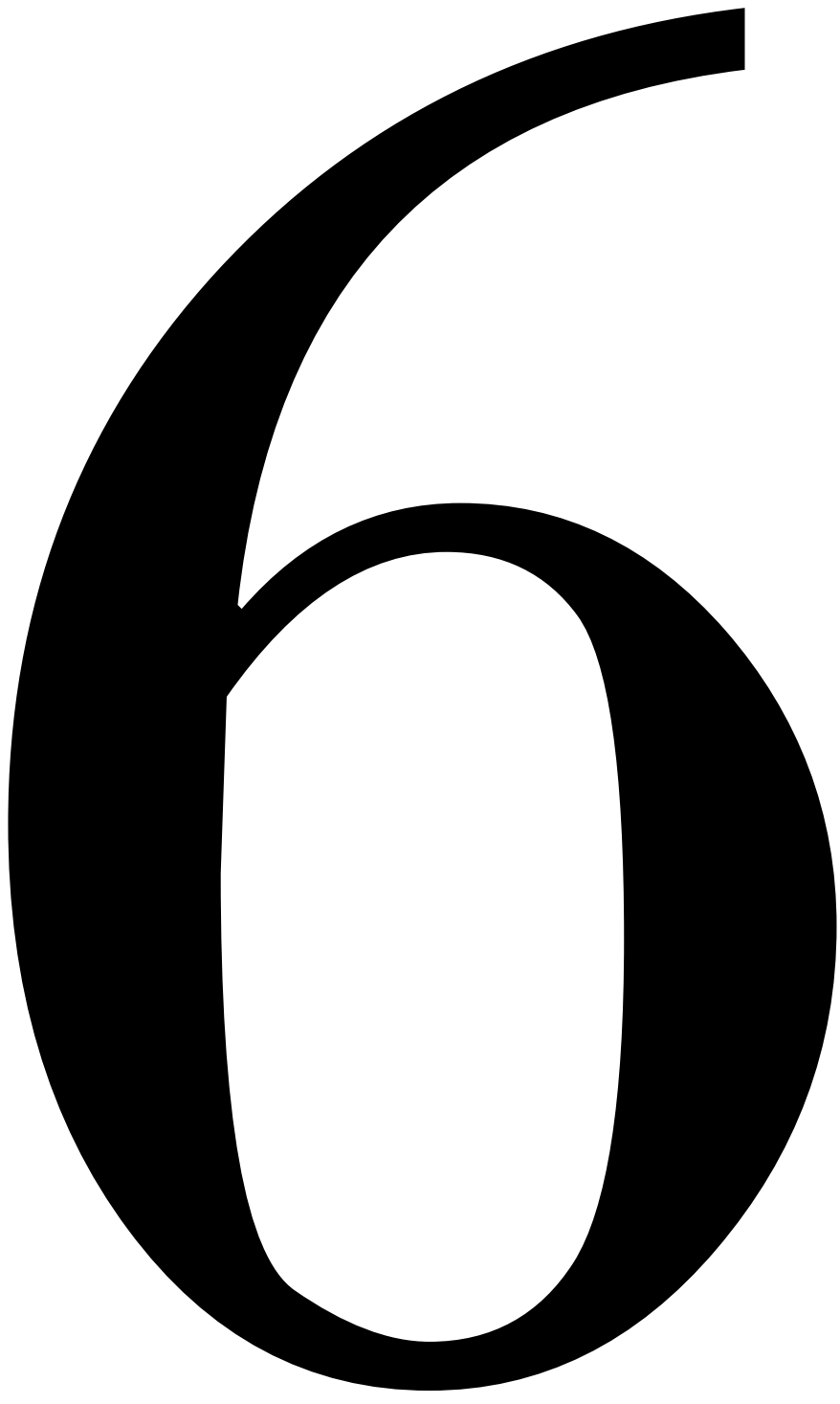
There are innumerous technical challenges for enabling the execution and communication of software components in a platform like MEDUSA. Regarding stroke care, the software components execute in different computing devices (CPUs, GPUs, etc.) and based on different software platforms (web, Linux, Windows, etc.). In the MEDUSA platform these challenges are tackled using SOA approach and a virtualized infrastructure. Because of the variety of application types, a uniform way of establishing communication between the MEDUSA applications has not been developed yet. Nevertheless, the direct communication between applications based on the exchange of well-defined file formats through WebSockets was demonstrated to be effective, without a negative impact in the development and integration of these applications. The current functionalities present in the MSP have the potential to improve several aspects of current stroke care.

The MEDUSA platform is still under development. Thus, most of the components to implement security are still not completely integrated in the platform yet. Defining and developing the security aspects of a platform like MEDUSA is also a very challenging task, since it is necessary to cope with different legal constraints, in particular across countries. The development process of the MEDUSA platform includes the implementation and validation of the platform in three different hospitals. This validation is currently being carried out in one hospital. Preliminary evaluation of the platform indicates that the solution is promising and has potential large value for improving treatment of these patients.

References

1. Go, Alan S., et al. "Heart disease and stroke statistics—2013 update: a report from the American Heart Association." *circulation* 127.1 (2013): e6-e245.
2. Hallett, Shane, et al. "Cloud-based healthcare: towards a SLA compliant network aware solution for medical image processing." (2012).
3. Alonso-Calvo, Raúl, et al. "Cloud computing service for managing large medical image data-sets using balanced collaborative agents." *Advances on Practical Applications of Agents and Multiagent Systems*. Springer, Berlin, Heidelberg, 2011. 265-270.
4. Shini, S. G., Tony Thomas, and K. Chithraranjan. "Cloud based medical image exchange-security challenges." *Procedia Engineering* 38 (2012): 3454-3461.
5. Kagadis, George C., et al. "Cloud computing in medical imaging." *Medical physics* 40.7 (2013): 070901.
6. Jeyabalaraja, V., and M. S. Josephine. "Cloud computing in medical diagnosis for improving health care environment." *International Journal of Computing Algorithm* 2 (2013): 458-462.
7. Pino, Carmelo, and Roberto Di Salvo. "A survey of cloud computing architecture and applications in health." *International conference on computer science and electronics engineering*. 2013.
8. Jee, Kyoungyoung, and Gang-Hoon Kim. "Potentiality of big data in the medical sector: focus on how to reshape the healthcare system." *Healthcare informatics research* 19.2 (2013): 79-85.
9. Murdoch, Travis B., and Allan S. Detsky. "The inevitable application of big data to health care." *Jama* 309.13 (2013): 1351-1352.
10. Kanagaraj, G., and A. C. Sumathi. "Proposal of an open-source cloud computing system for exchanging medical images of a hospital information system." *3rd International Conference on Trendz in Information Sciences & Computing (TISC2011)*. IEEE, 2011.
11. Yang, Chao-Tung, et al. "Implementation of a medical image file accessing system on cloud computing." *2010 13th IEEE International Conference on Computational Science and Engineering*. IEEE, 2010.
12. Koufi, Vassiliki, Flora Malamateniou, and George Vassilacopoulos. "Ubiquitous access to cloud emergency medical services." *Proceedings of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine*. IEEE, 2010.
13. Zhuang, Yi, et al. "Efficient and robust large medical image retrieval in mobile cloud computing environment." *Information Sciences* 263 (2014): 60-86.
14. Gu, Hua, Lei Huang, and Bei Xi. "A cloud computing based collaborative service pattern of medical association for stroke prevention and treatment." *Management & Engineering* 21 (2015): 3.
15. Sharieh, Salah, Franya Franek, and Alexander Ferworn. "Using cloud computing for medical applications." *Proceedings of the 15th Communications and Networking Simulation Symposium*. 2012.
16. Parsonson, Louis, et al. "A cloud computing medical image analysis and collaboration platform." *International Conference on Cloud Computing and Services Science*. Springer, New York, NY, 2011.

17. Dorn, Karlheinz, Vladyslav Ukis, and Thomas Friese. "A cloud-deployed 3D medical imaging system with dynamically optimized scalability and cloud costs." *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2011.
18. Chiang, Wen-Chung, et al. "Bulding a cloud service for medical image processing based on service-orient archtecture." *2011 4th International Conference on Biomedical Engineering and Informatics (BMEI)*. Vol. 3. IEEE, 2011.
19. Huang, QingZang, et al. "Medical information integration based cloud computing." *2011 International Conference on Network Computing and Information Security*. Vol. 1. IEEE, 2011.
20. Ojog, Iuliana, et al. "A cloud scalable platform for DICOM image analysis as a tool for remote medical support." *The Fifth International Conference on eHealth, Telemedicine, and Social Medicine. France*. 2013.
21. Ahn, Yong Woon, and Albert Mo Kim Cheng. "Autonomic computing architecture for real-time medical application running on virtual private cloud infrastructures." *ACM SIGBED Review* 10.2 (2013): 15-15.
22. Holtmann, Carsten, et al. "Medical opportunities by mobile IT usage–a case study in the stroke chain of survival." *European Conference on eHealth 2007*. Gesellschaft für Informatik e. V., 2007.
23. Joveski, Bojan, et al. "Semantic multimedia remote display for mobile thin clients." *Multimedia systems* 19.5 (2013): 455-474.
24. Joveski, Bojan, Mihai Mitrea, and Rama-Rao Ganji. "MPEG-4 solutions for virtualizing RDP-based applications." *Mobile Devices and Multimedia: Enabling Technologies, Algorithms, and Applications 2014*. Vol. 9030. International Society for Optics and Photonics, 2014.
25. Boers, A. M., et al. "Automatic quantification of subarachnoid hemorrhage on noncontrast CT." *American journal of neuroradiology* 35.12 (2014): 2279-2286.
26. Santos, Emilie MM, et al. "Development and validation of intracranial thrombus segmentation on CT angiography in patients with acute ischemic stroke." *PLoS One* 9.7 (2014): e101985.
27. Boers, Anna M., et al. "Automated cerebral infarct volume measurement in follow-up noncontrast CT scans of patients with acute ischemic stroke." *American Journal of Neuroradiology* 34.8 (2013): 1522-1527.
28. Barros, R.S., et al.: High Performance Image Analysis of Compressed Dynamic CT Perfusion Data of Patients with Acute Ischemic Stroke. Submitted to MICCAI HPC Workshop (2015)



Chapter 6

Automated segmentation of subarachnoid hemorrhages with convolutional neural networks

Renan Sales Barros [a] , Wessel E. van der Steen [a] , Anna M.M. Boers [a] , IJsbrand Zijlstra [a] , Rene van den Berg [a] , Wassim El Youssoufi [b] , Alexandre Urwald [c] , Dagmar Verbaan [a] , Peter Vandertop [a] , Charles Majoie [a] , Silvia Delgado Olabarriaga [a] , Henk A. Marquering [a],*

[a] Amsterdam UMC, Location AMC, University of Amsterdam, Amsterdam, the Netherlands
[b] Artemis Department of the Telecom SudParis, Evry, France
[c] Telecom Physique Strasbourg, Illkirch-Graffenstaden, France

- Barros, Renan Sales, et al. "Automated segmentation of subarachnoid hemorrhages with convolutional neural networks." *Informatics in Medicine Unlocked* 19 (2020): 100321.
- First author, journal, https://www.sciencedirect.com/science/article/pii/S2352914820300769

Abstract

Purpose: To investigate the viability of convolutional neural networks (CNNs) for the detection and volumetric segmentation of subarachnoid hemorrhage (SAH) in non-contrast computed tomography (NCCT).

Materials and methods: We developed and trained a CNN for the SAH segmentation by splitting a set of 302 baseline NCCTs into a training (268) and a validation set (34). Segmentation accuracy was assessed on an additional 473 baseline NCCTs of SAH patients by calculating the intraclass correlation coefficient of the SAH volume and the Dice coefficient of the segmentations. We subsequently evaluated whether the developed SAH segmentation network can be used to discriminate SAH from acute ischemic stroke using 280 scans to optimize the discrimination and 70 scans for testing. Additionally, we tested whether the CNN-based volumetric SAH segmentation can also be used for hemorrhage segmentation in 396 NCCTs of rebleed patients.

Results: The SAH volume agreement was high with an intraclass correlation coefficient of 0.966. The average Dice coefficient of the volumetric SAH segmentation was 0.63 ± 0.16, which is similar to expert interobserver agreement. The differentiation of SAH from ischemic stroke patients achieved an accuracy of 0.96. Despite the common presence of severe metal artifacts in scans of rebleed patients due to coiling, the CNN-based segmentation appears to be suitable for segmentation of rebleeds as well with comparable accuracy. The average CNN detection and segmentation processing time was 30 s.

Conclusion: The proposed CNN is fast and accurate in detecting and segmenting SAH in NCCT scans

1 Introduction

Subarachnoid hemorrhage (SAH) accounts for approximately 5% of all strokes. With a fatality rate of 30% [1] and half of the patients being younger than 55 years, the number of productive life years lost due to SAH is equivalent to ischemic stroke [2,3]. Several scales for assessing the severity of SAH on computed tomography (CT) images have been developed [4]. Currently, the most commonly used scales are the Fisher scale [5], modified Fisher scale [6], and Hijdra sum score [7]. It has been shown that these scores are associated with complications such as vasospasm, delayed cerebral ischemia (DCI), and poor outcome. Nevertheless, these scores have a considerable interobserver variability, which may limit their predictive value [8,9].

Accurate volumetric segmentation and quantitative assessment of SAH in CT scans are a valuable alternative to these radiological scales and provides valuable information for monitoring and predicting the outcome of SAH patients. Previous studies have shown that quantitative segmentation of SAH is possible using image processing techniques [10]. The segmentation of SAH is considered complex compared to other causes of intracranial hemorrhages since the hyperdensity due to the presence of blood is in the hypodensity parts of the brain (e.g. the fissures). Zijlstra et al. [11] showed that indeed the quantified blood as resulted from the method by Boers et al. [10] is significantly associated with DCI. Since the method by Boers et al. is based on the relative increase of Hounsfield units (HU) due to the presence of blood, other image features that may be relevant for accurate SAH segmentation, such as spatial bleeding patterns, may have been ignored. In another approach in which the CT images were analyzed using an autoencoder, it was shown that DCI prediction can strongly be improved using additional image features [12]. However, these image features cannot be interpreted by clinicians, and this approach is therefore not suitable for clinical practice. To overcome these limitations, we propose a novel approach for SAH detection and segmentation in CT scans based on convolutional neural networks (CNNs).

CNNs require high-quality ground truth samples for training. The generation of this ground truth is a demanding task to because the effects of SAH in CT images can be very subtle. The SAH may spread around the subarachnoid space [13] and gets diluted in the cerebral spinal fluid (CSF). When the basal cisterns are filled with blood, the distinction between intravascular and extravascular blood is challenging. Moreover, even when the effects of SAH are visible, only a tiny portion of voxels in the CT scans are affected (see Figure. 1). Next to this, the agreement of manual segmentations by experienced radiologists is moderate with an average Dice coefficient of 0.64. Considering the relatively imprecise training data, the primary goal of this study is to investigate the feasibility and performance of CNNs for fully automated SAH detection

and its volumetric segmentation.

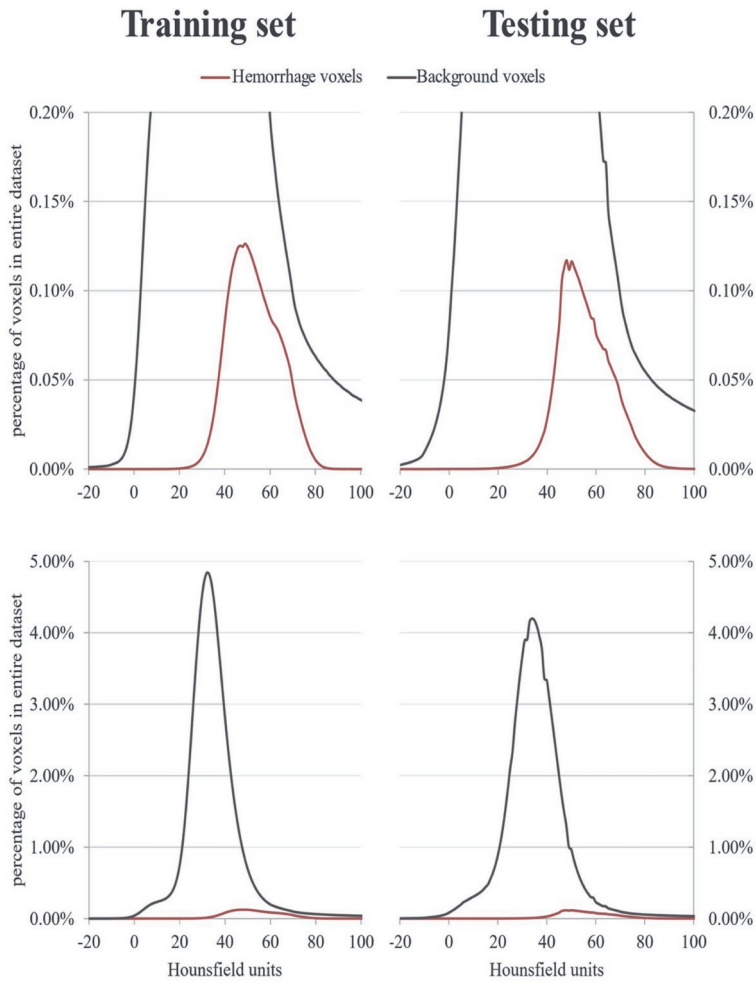


Figure 1: Histogram of Hounsfield unit values in the baseline non-contrast CT scans in the training set (268) and testing set (473). Only voxels inside the intracranial region were considered (that is, voxels representing air, skull, etc., were excluded in these histograms). The first row shows a zoomed version of the histograms in the second row.

2. Materials and Methods

In this section, we describe the used image data, the development of the CNN-based volumetric SAH segmentation, two alternative SAH segmentation approaches which are used for comparison with the CNNbased results, the approach to differentiate

hemorrhagic stroke from ischemic stroke patients, and an additional experiment to evaluate whether the proposed CNN-based approach is also suitable to segment hemorrhages in patients with a rebleed. All CNN-based methods consist of two steps: [1]: the generation of hemorrhage probability maps and [2] the determination of thresholds to transform the probability maps into binary segmentations. The first step is generic, and the latter is application specific. Moreover, the generation of hemorrhage probability maps makes use of "training sets", whereas the latter utilizes "validation sets".

2.1. Image data

In this retrospective study, we used image data from a SAH registry and from an ischemic stroke clinical trial. The SAH registry is composed of prospective consecutive patients with aneurysmal SAH admitted in the Amsterdam UMC between December 2011 and December 2016. The inclusion criteria were (1) SAH visible on baseline non-contrast CT (NCCT) or confirmed after lumbar puncture and (2) confirmation of ruptured aneurysm by angiography imaging. We excluded patients with significant movement or metal artifacts on baseline NCCT or who were included in ongoing trials. We also used baseline NCCTs from ischemic stroke patients included in the in the MR CLEAN trial [14]. A subset of 317 patients from this SAH registry was used in a prior study [12]. The acute ischemic trial patients have been included in many published studies.

A total of 775 baseline NCCT scans from the SAH registry were used for developing the CNN-based SAH segmentation. The training set comprised 268 scans, the validation set consisted of 34 scans, and the test set included 473 scans. The ground truth segmentation of these 775 images was produced as follows: (1) The method by Boers et al. was used to produce an initial segmentation; (2) this segmentation was corrected by a trained observer; and (3) subsequently, this segmentation was validated by an expert radiologist and, when applicable, additional corrections were made. Table 1 shows the properties of CT scans used in this study.

The proposed differentiation of SAH from ischemic stroke patients make use of the hemorrhage probability maps generated by the CNN that was trained for SAH segmentation. Therefore, no training step is required in the SAH ischemic stroke differentiation. For developing the automated SAH differentiation, we used 350 baseline NCCT images of evenly distributed acute ischemic stroke (175) and SAH (175) patients. These 350 scans were used as follows: the validation set consisted of 280 scans with 140 ischemic stroke and 140 SAH patients. The test set included 70 scans from both ischemic stroke (35) and SAH (35) patients.

The medical ethics committee of the Amsterdam UMC exempted this study of an official approval for the usage of the anonymized NCCT scans included in this SAH

registry and informed consent was waived.

Table 1: Properties of the images used in this study. When applicable, the data is shown is average ± standard deviation.

	Number of scans	Slices per scan	Voxel spacing (mm)	Spacing between slices (mm)
Baseline NCCTs included in the training and validation sets of the CNN for SAH volumetric segmentation.	302	32 ± 6.6	0.45 ± 0.05	4.86 ± 0.57
Baseline NCCTs used to test the CNN-based SAH volumetric segmentation.	473	39 ± 9.9	0.45 ± 0.06	4.23 ± 1.01
Follow-up NCCTs used to test the CNN for segmentation of hemorrhages in rebleed patients.	396	32 ± 3.2	0.47 ± 0.05	4.95 ± 0.57
Baseline NCCTs from ischemic stroke patients used to optimize and evaluate the automated SAH detection.	175	30 ± 9.7	0.45 ± 0.07	4.49 ± 1.11

2.2. CNN for volumetric SAH segmentation

The proposed CNN outputs a probability map, which indicates the likelihood of a voxel belonging to the class hemorrhage. Since probability values range from 0 to 1, after training we optimized a threshold value for the binary voxel-wise classification for SAH segmentation and another cut-off value for the SAH – ischemic stroke differentiation using validation data.

The proposed CNN for SAH segmentation was designed to classify a single voxel based on an image patch around that voxel. We performed a grid search to select the optimal CNN architecture for this classification task. We used the average Dice coefficient to select the best CNN architecture. Table 2 shows the hyperparameter values evaluated in this grid search.

To tackle the problem of the unbalanced representation of hemorrhage versus background voxel classes, we used the same number of hemorrhage and background patches during training. We automatically determined the most relevant background patches while training. Figure. 2 lists the steps used during training. Figure. 3 illustrates the result of this automatic selection of training samples of the background class.

Table 2: Evaluated values of hyperparameters during optimization of the proposed convolutional neural network architecture via a grid search.

Hyperparameter	Evaluated values
Number of convolutional layers	2, 3, or 4
Number of fully connected layers	2 or 3 positioned at the end of the network
Nodes of a fully connected layer	64, 128, 256
Max-pooling layers	All permutations of max-pooling layers after convolutional layers
Feature maps of a convolutional layer	64 or 128
Sizes of the convolution kernel of a convolutional layer	3 x 3 x 1, 5 x 5 x 1, 7 x 7 x 1, 9 x 9 x 1, 3 x 3 x 3, 5 x 5 x 3, 7 x 7 x 3, or 9 x 9 x 3
Size of input patch	7 x 7, 15 x 15, 19 x 19, 23 x 23, or 31 x 31
Number of slices in a patch	1, 3, or 5
Processing patches with multiple slices	As different channels with 2D convolutions or as 3D images with 3D convolutions
Dropout	Used or not used
Postprocessing final probability map	Dense conditional random field, 3D Gaussian smoothing, or no postprocessing

2.3. Image pre-processing

We segmentated the intracranial region and used a threshold-based segmentation to exclude voxels that can be trivially classified as background, such as air. Subsequently, random background patches were selected for the first training step. In all subsequent training steps, the background patches with lowest accuracy were used for training. This step was repeated until no improvement was observed in the average Dice coefficient of the validation set.

The segmentation

of the intracranial region is based on the size range of the foramina of the skull, as reported by Ref. [15] and on typical intensity values of the voxels. This segmentation is performed according to the following steps: (1) Thresholding is used to segment bone. We considered all voxels with an intensity above 160 HU as bone. (2) A morphological dilation with 7 mm radius is used to close all foramina of the skull, except the foramen magnum. (3) The centroid of the segmented bone is used as a seed for region growing inside the skull. (4) A morphological dilation with 7 mm radius was applied to the region growing result to bring the segmented intracranial region closer to the skull border. (5) The foramen magnum is detected by evaluating the segmented area in each individual slice from top to bottom. The foramen magnum slice was determined as the highest slice with a segmented area below 900 mm^2 below the slice with the maximum segmented area. All segmented voxels below the foramen magnum slice were excluded from the segmentation.

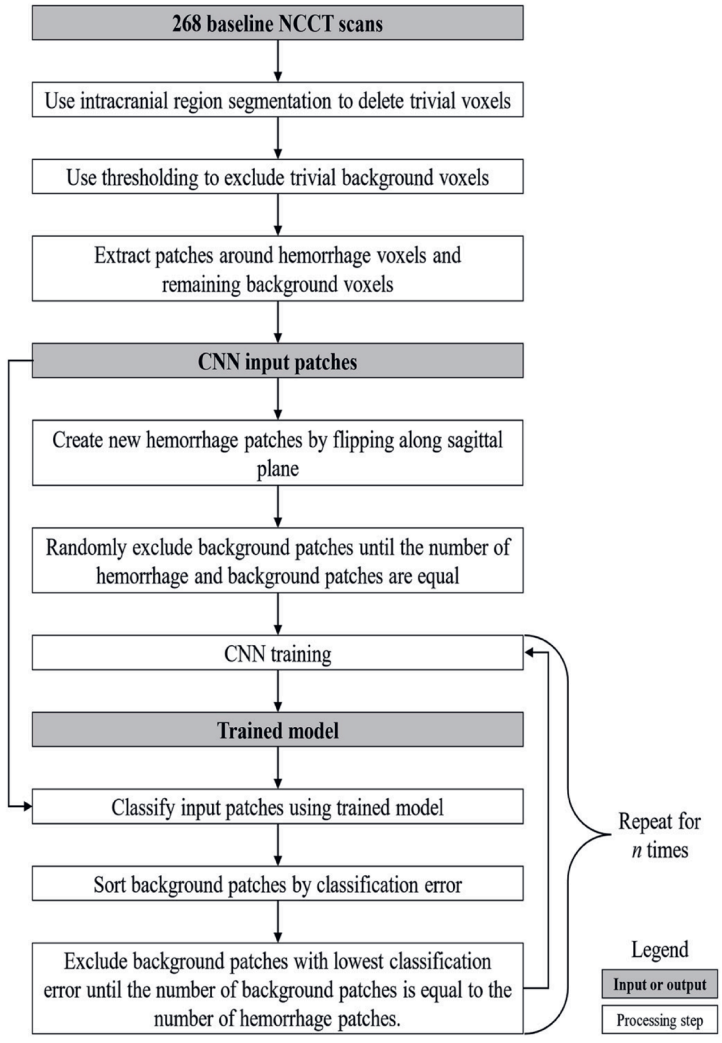


Figure 2: Steps of the training process of the convolutional neural network (CNN) for voxelwise classification of based on image patches.

2.4. SAH segmentations

We used two alternative methods for SAH segmentation to compare with our CNN-based volumetric SAH segmentation: a refined threshold segmentation and the U-Net [16].

We developed the refined threshold segmentation based on three key observations: (1) hemorrhage voxels usually have similar intensity values (see Figure. 1). (2) Most background voxels with intensities similar to hemorrhage voxels are located close to

the skull (see the area highlighted in blue in Figure. 3). (3) A region with SAH voxels is usually larger than just a few voxels.

The refined threshold segmentation was optimized with a grid search that determined the upper and lower limits for a threshold-based hemorrhage segmentation, the parameters for a morphological erosion of the brain mask to exclude the voxels close to skull, and the parameters of a morphological opening to exclude small groups of voxels selected in the threshold-based hemorrhage segmentation. This grid search was performed by evaluating all images in the training and validation sets of CNN-based segmentation. Subsequently, the refined threshold segmentation was evaluated on the test set of the CNN-based segmentation.

The evaluated U-Net architecture had three layers with 32, 64, and 128 feature maps in the first, second, and third layers respectively. The size of the convolution kernels of the U-Net was 3 x 3. The U-Net was trained and evaluated with the same training, validation, and test sets of the proposed CNN.

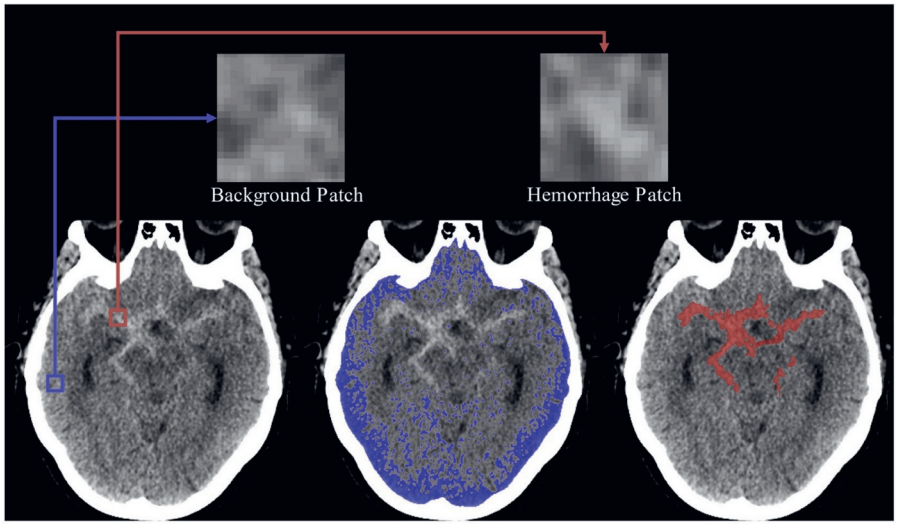


Figure 3: Example of training patches extracted from a CT scan. The background patches are highlighted in blue. The hemorrhage patches that are generated for all voxels in the area highlighted in red. Note that the background patches selected for training are very similar to the hemorrhage patches. The background pixels were selected as the voxels with highest classification error during the training process. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

2.5. SAH detection

We assumed that a scan belongs to a patient with a SAH instead of ischemic stroke when the volume of the SAH segmentation in a NCCT image is above a certain threshold. Since the CNN outputs a probability map with values ranging from 0 to 1, we optimized a specific cut-off value for the binary voxel-wise classification used for SAH detection. Subsequently, we optimized the minimum SAH volume for the differentiation.

2.6. Statistics

The agreement of the automated generation of SAH volumes with the ground truth was assessed by the calculation of the intraclass correlation coefficient and Bland-Altman analysis in the validation set. The accuracy of the SAH segmentations was determined by the calculation of the Dice coefficient in the validation set. The accuracy of the SAH – ischemic stroke differentiation was determined as the percentage of accurate classified patients.

2.7. Volumetric rebleed SAH segmentation

As a proof of concept and to determine whether this approach deems robust for image artifacts, we evaluated the applicability of the SAH segmentation also in patients with a rebleed. The segmentation of rebleeds are expected to be more difficult because commonly these patients have been coiled to treat the initial SAH. These coils commonly result in severe artifacts in NCCT images.

From the SAH registry, 396 follow-up NCCTs of rebleed patients were used. Ground truth SAH segmentations of these rebleed scans were created as follows: (1) These scans were processed by the proposed CNN to generate SAH probability maps. (2) These probability maps were checked by two experts, and an optimal threshold was selected to generate binary maps representing the rebleed region. (3) Subsequently, these two experts corrected the segmentations if needed. It should be noted that no baseline NCCTs from these rebleed patients were included in the training and validation set. Of these 396 rebleed patients, 127 scans had significant metal artifacts.

3. Results

The best CNN architecture is composed of two convolutional layers followed by two fully connected dense layers with 256 nodes each. The size of the input patch of this CNN is 19 x 19 x 3 and each slice is considered as a different image channel. Max-polling layers with kernel 2 x 2 are present after each convolutional layer. The first convolutional layer has 64 features maps and the second has 128 features maps. Both convolutional layers have 5 x 5 kernels. No dropout nor postprocessing of the final

probability map was used.

The SAH volume agreement was excellent with an intraclass correlation coefficient of 0.966 in the validation set. The Bland-Altman analysis showed a bias of 0.0 ml and a relative large spread with the 95% limits of agreement between 20 and 20 ml (see Figure. 4.)

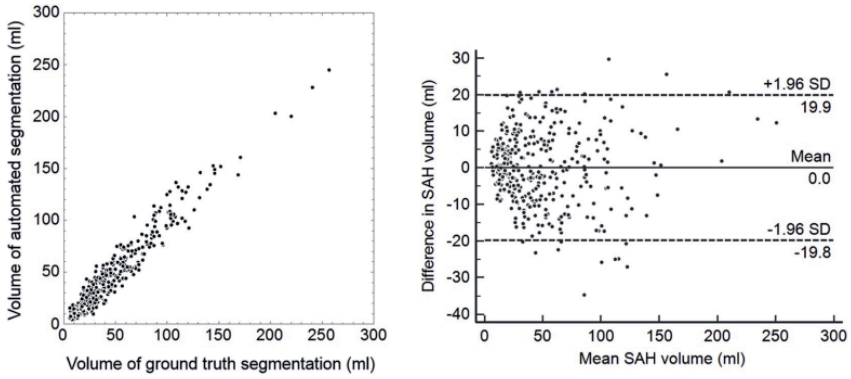


Figure 4: Scatterplot of the subarachnoid hemorrhage volumes as determined by the CNN and by the manual delineation (left) and Bland Altman plot of the differences and averages of these volumes. The scatter plot shows a high agreement (intraclass correlation coefficient of 0.966), the Bland Altman plot show an ignorable bias (0.0 ml) and large spread (95% limits of agreement between 20 and 20 ml).

An average Dice coefficient of 0.63 ± 0.16 ranging between 0.19 and 0.92, was obtained for the test set. Figure. 5 shows images from the test together with the ground truth segmentations, probability maps generated by the CNN, and final binary segmentation. After intracranial region segmentation and threshold segmentation to exclude trivial background voxels, the method achieved a voxelwise classification with accuracy = 0.93, sensitivity = 0.94, specificity = 0.95, and area under the receiver operating characteristic curve = 0.99. The proposed SAH segmentation outperformed the baseline segmentation methods, which achieved an average Dice of 0.42 and 0.40 for the refined threshold segmentation and U-Net segmentation, respectively.

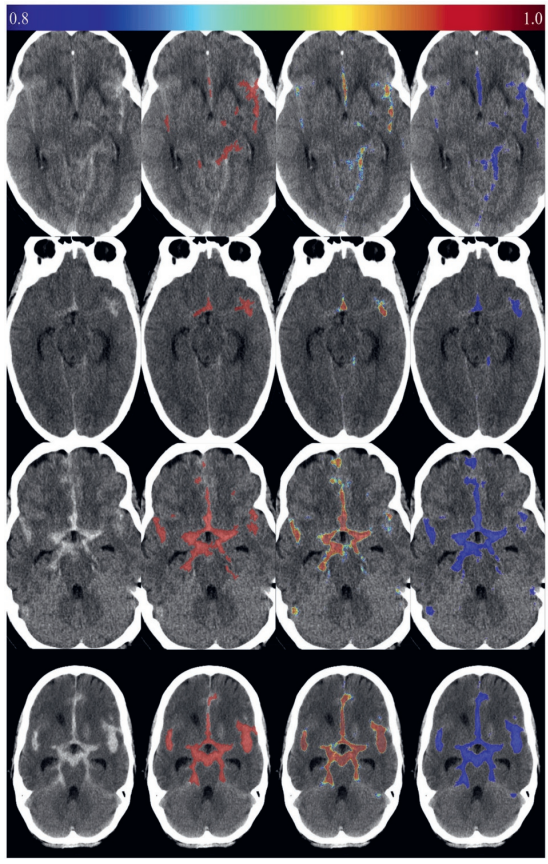


Figure 5: Sample segmentation from the proposed convolutional neural network. From top to bottom, the Dice coefficient is 0.30, 0.45, 0.60, and 0.75. From left to right, the columns show a slice of the input CT scan, the reference segmentation, the output probability map, and the final binary segmentation. The probability value threshold of 0.84 was optimal for the dichotomization into hemorrhage and background. The color scale indicates the different probability values ranging from 0.8 (blue) to 1.0 (red). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

The average Dice coefficient achieved by the CNNs with different hyperparameters ranged from 0.40 to 0.63. The hyperparameter with most impact in the accuracy was the patch size. The only difference in hyperparameter values between the CNNs with highest and lowest average Dice coefficient was the size of the input patch. The input patch size of the CNN with lowest average Dice coefficient was 7 x 7 x 1. Using dropout or any of the evaluated post processing techniques resulted in differences in average Dice coefficients 3% or lower. The remaining hyper parameters had very small impact in the average Dice coefficient. CNNs with 3 and 4 layers had an average Dice

coefficient 1% and 2% inferior respectively. The different values for number of nodes in the fully connected layers resulted in a variation of 1% in the average Dice coefficient. The different values for all remaining hyper parameters resulted in a maximum variation of 2% in the average Dice coefficient.

Regarding the SAH - ischemic stroke differentiation, the optimized cut-off probability value for generating the binary segmentations is 0.9, and the minimum hemorrhage volume for having a positive SAH prediction is 6.6 ml. The SAH detection achieved an accuracy of 0.96, with a sensitivity of 1.00 and a specificity of 0.91.

The average Dice coefficient was 0.66 ± 0.19 for SAH segmentation in the rebleed patients. Figure. 6 shows a sample segmentation from a rebleed patient and illustrates that good segmentation results can be achieved despite the metal artifacts.

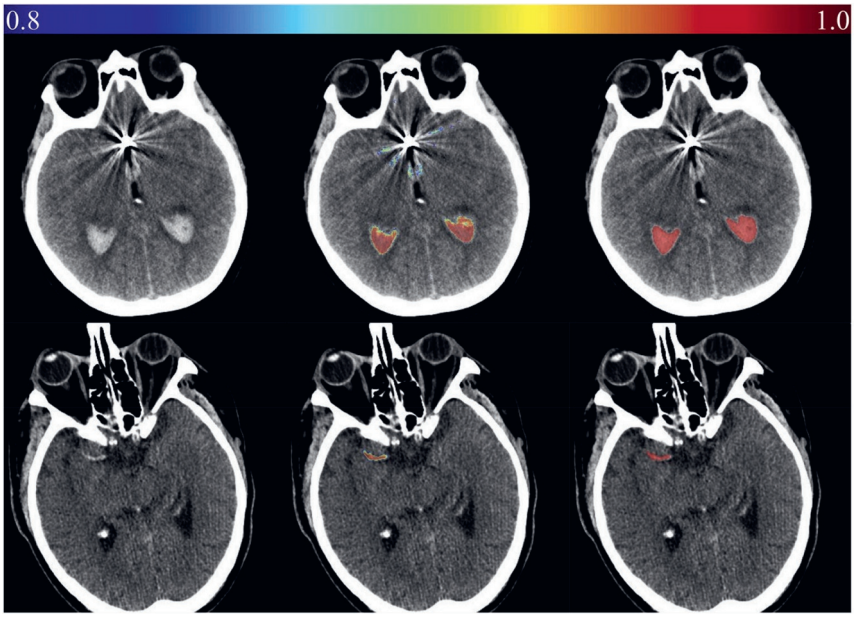


Figure 6: Sample results from a rebleed patient (first row) and an ischemic stroke patient (second row). From left to right we have the original CT scan, the probability map generated by the convolutional neural network, and the automatically generated binary segmentation. In the first row, the final binary segmentation is not affected by the metal artifacts. In the second row, a blood clot is misclassified as hemorrhage. The proposed method for subarachnoid hemorrhage (SAH) detection correctly detected this ischemic stroke patient as not a SAH patient. This happened because the volume of the segmented blood clot does not surpass the minimum required volume for a positive SAH prediction.

4. Discussion

The proposed CNN architecture achieved a high accuracy in SAH volume, segmentation, and differentiation from acute ischemic stroke patients. The accuracy of volumetric SAH segmentation was similar to expert agreement. Moreover, we have shown that this approach is also suitable for SAH detection in patients with rebleeds, for whom the CT images may have severe artifacts.

Supervised automated segmentation methods require accurate reference standards during learning and accuracy assessments. Limited interobserver agreement hampers the generation of high-quality training data and makes it difficult to use CNNs or other supervised learning methods. The study by Boers et al. compared the manual SAH segmentations produced by two experienced radiologists and reported an average Dice coefficient of 0.64 ranging from 0.00 to 0.86. Thus, the average Dice coefficient achieved by the proposed CNN in both the SAH and rebleed patients is comparable with the agreement of expert radiologists.

SAH is a type of intracranial hemorrhage. Various approaches for automated segmentation of intracranial hemorrhages in CT scans have been proposed such as thresholding [17–19], region growing [17], clustering [17,18,20–22], active contour [17], graph cut [23], random-forest [24], level-set [21,22,25,26], and others [27–30]. None of these approaches were thoroughly tested with images from SAH patients. The only available method for automatic SAH segmentation was proposed by Boers et al. and reported an average Dice coefficient of 0.55. Moreover, our proposed method also outperforms refined threshold segmentations and the U-Net based approach.

Another advantage of the proposed CNN over the method by Boers et al. is the processing time. While the proposed CNN only needs around 30 s to segment or detect SAH in a CT scan, the method by Boers et al. required several minutes.

Differently from other CNN-based classifications, the proposed method always generates a SAH probability map that can be visually inspected by human experts or automatically postprocessed for SAH detection. When this probability map was postprocessed for SAH detection instead of segmentation, a high detection accuracy was achieved. However, since the SAH segmentation was trained with only SAH images, we hypothesize that this leads to an overestimation of SAH volume due to false positive predictions. This effect can be visualized in Figure. 6, where a thrombus was misclassified as a hemorrhage. To compensate for these false positive predictions, we opted for defining a minimum segmented SAH volume for a positive SAH detection. We validated and tested the differentiation of SAH and ischemic stroke patients only.

Thus, this detection is not yet suited for clinical use. For example, venous thrombosis could be misclassified as SAH since both conditions are presented with similar image features. On the other hand, these probability maps can help human experts to perform faster NCCT assessments because they highlight the regions of the NCCT scan that are more likely to be SAH. This was the approach used for the generation of the hemorrhage segmentations of the rebleed patients.

Fast and accurate SAH segmentation can lead to more precise prediction of patient outcome. The work by Zijlstra et al. [31] already demonstrated that the SAH volume is associated with DCI. In addition, accurately and quickly eliminating SAH as the type of stroke may have a major impact in patient outcome, since this can lead to a fast start of the ischemic stroke treatment and a safer administration of clot dissolving agent. However, additional research is still needed to demonstrate the value of the proposed CNN in clinical practice.

To conclude, we demonstrated that the proposed CNN can be used for fast detection and volumetric SAH segmentation with similar accuracy as expert radiologists. This was achieved despite the difficulties in producing accurate ground truths and despite the large difference between the number of hemorrhage and background voxels.

References

1. Vergouwen, Mervyn DI, et al. "Time trends in causes of death after aneurysmal subarachnoid hemorrhage: a hospital-based study." *Neurology* 86.1 (2016): 59-63.
2. Van Gijn, Jan, Richard S. Kerr, and Gabriel JE Rinkel. "Subarachnoid haemorrhage." *The Lancet* 369.9558 (2007): 306-318.
3. van Donkelaar, Carlina E., et al. "Predictive factors for rebleeding after aneurysmal subarachnoid hemorrhage: rebleeding aneurysmal subarachnoid hemorrhage study." *Stroke* 46.8 (2015): 2100-2106.
4. van der Steen, Wessel E., et al. "Radiological scales predicting delayed cerebral ischemia in subarachnoid hemorrhage: systematic review and meta-analysis." *Neuroradiology* 61.3 (2019): 247-256.
5. Fisher, C. M., J. P. Kistler, and J. M. Davis. "Relation of cerebral vasospasm to subarachnoid hemorrhage visualized by computerized tomographic scanning." *Neurosurgery* 6.1 (1980): 1-9.
6. Frontera, Jennifer A., et al. "Prediction of symptomatic vasospasm after subarachnoid hemorrhage: the modified fisher scale." *Neurosurgery* 59.1 (2006): 21-27.
7. Hijdra, A., et al. "Grading the amount of blood on computed tomograms after subarachnoid hemorrhage." *Stroke* 21.8 (1990): 1156-1161.
8. Kramer, Andreas H., et al. "A comparison of 3 radiographic scales for the prediction of delayed ischemia and prognosis following subarachnoid hemorrhage." *Journal of neurosurgery*109.2 (2008): 199-207.
9. Van der Jagt, M., et al. "Interobserver variability of cisternal blood on CT after aneurysmal subarachnoid hemorrhage." *Neurology* 54.11 (2000): 2156-2158.
10. Boers, A. M., et al. "Automatic quantification of subarachnoid hemorrhage on noncontrast CT." *American journal of neuroradiology* 35.12 (2014): 2279-2286.
11. Zijlstra, I. A., et al. "Association of automatically quantified total blood volume after aneurysmal subarachnoid hemorrhage with delayed cerebral ischemia." *American Journal of Neuroradiology* 37.9 (2016): 1588-1593.
12. Ramos, Lucas Alexandre, et al. "Machine learning improves prediction of delayed cerebral ischemia in patients with subarachnoid hemorrhage." *Journal of neurointerventional surgery* 11.5 (2019): 497-502.
13. Li, Yonghong, et al. "Automatic detection of the existence of subarachnoid hemorrhage from clinical CT images." *Journal of medical systems* 36.3 (2012): 1259-1270.
14. Berkhemer, Olvert A., et al. "A randomized trial of intraarterial treatment for acute ischemic stroke." *n Engl J Med* 372 (2015): 11-20.
15. Berge, Jennifer K., and Ronald A. Bergman. "Variations in size and in symmetry of foramina of the human skull." *Clinical Anatomy: The Official Journal of the American Association of Clinical Anatomists and the British Association of Clinical Anatomists* 14.6 (2001): 406-413.
16. Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *International Conference on Medical image computing and computer-assisted intervention*. Springer, Cham, 2015.
17. Bhadauria, N. S., et al. "Performance evaluation of segmentation methods for brain CT images based hemorrhage detection." *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2015.

18. Gautam, Anjali, and Balasubramanian Raman. "Automatic segmentation of intracerebral hemorrhage from brain CT images." *Machine intelligence and signal analysis*. Springer, Singapore, 2019. 753-764.
19. Zhang, Yuanxiu, et al. "Detection and quantification of intracerebral and intraventricular hemorrhage from computed tomography images with adaptive thresholding and case-based reasoning." *International journal of computer assisted radiology and surgery* 8.6 (2013): 917-927.
20. Sharma, Bhavna, and K. Venugopalan. "Automatic segmentation of brain ct scan image to identify hemorrhages." *International Journal of Computer Applications* 40.10 (2012): 1-5.
21. Prakash, KN Bhanu, et al. "Segmentation and quantification of intra-ventricular/cerebral hemorrhage in CT scans by modified distance regularized level set evolution technique." *International journal of computer assisted radiology and surgery* 7.5 (2012): 785-798.
22. Singh, Pankaj, Vandana Khanna, and Meenu Kamal. "Hemorrhage segmentation by fuzzy c-mean with Modified Level Set on CT imaging." *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE, 2018.
23. Sun, Mingjie, et al. "Intracranial hemorrhage detection by 3D voxel segmentation on brain CT images." *2015 International Conference on Wireless Communications & Signal Processing (WCSP)*. IEEE, 2015.
24. Scherer, Moritz, et al. "Development and validation of an automatic segmentation algorithm for quantification of intracerebral hemorrhage." *Stroke* 47.11 (2016): 2776-2782.
25. Liao, Chun-Chih, et al. "A multiresolution binary level set method and its application to intracranial hematoma segmentation." *Computerized Medical Imaging and Graphics*33.6 (2009): 423-430.
26. Shahangian, Bahareh, and Hossein Pourghassem. "Automatic brain hemorrhage segmentation and classification algorithm based on weighted grayscale histogram feature in a hierarchical classification structure." *Biocybernetics and Biomedical Engineering* 36.1 (2016): 217-232.
27. Roy, Snehashis, et al. "Intraparenchymal hemorrhage segmentation from clinical head CT of patients with traumatic brain injury." *Medical Imaging 2015: Image Processing*. Vol. 9413. International Society for Optics and Photonics, 2015.
28. Shahangian, Bahare, and Hossein Pourghassem. "Automatic brain hemorrhage segmentation and classification in CT scan images." *2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP)*. IEEE, 2013.
29. Ray, Soumi, et al. "Intensity population based unsupervised hemorrhage segmentation from brain CT images." *Expert Systems with Applications* 97 (2018): 325-335.
30. Soltaninejad, Mohammadreza, et al. "A Hybrid Method for Haemorrhage Segmentation in Trauma Brain CT." *MIUA*. 2014.
31. Zijlstra, I. A., et al. "Ruptured middle cerebral artery aneurysms with a concomitant intraparenchymal hematoma: the role of hematoma volume." *Neuroradiology* 60.3 (2018): 335-342.

Chapter 7

Automatic segmentation of cerebral infarcts in follow-up computed tomography images with convolutional neural networks

Renan Sales Barros,[a] Manon L Tolhuisen,[a,b] Anna MM Boers,[a,c] Ivo Jansen,[b] Elena Ponomareva,[c] Diederik W J Dippel,[d] Aad van der Lugt,[e] Robert J van Oostenbrugge,[f] Wim H van Zwam ,[g,h] Olvert A Berkhemer,[b,e] Mayank Goyal ,[i] Andrew M Demchuk,[j] Bijoy K Menon,[k] Peter Mitchell,[l] Michael D Hill ,[j] Tudor G Jovin,[m] Antoni Davalos,[n] Bruce C V Campbell,[o,p] Jeffrey L Saver,[q] Yvo B W E M Roos,[r] Keith W. Muir,[s] Phil White,[t,u] Serge Bracard,[v] Francis Guillemin,[v] Silvia Delgado Olabarriaga,[b] Charles B L M Majoie,[w] Henk A Marquering [a,b]

[a] Department of Biomedical Engineering and Physics, Amsterdam UMC. location AMC, Amsterdam, the Netherlands
[b] Department of Radiology and Nuclear Medicine, Amsterdam UMC, location AMC, Amsterdam, the Netherlands
[c] Nico-lab, Amsterdam, Netherlands
[d] Department of Neurology, Erasmus MC - University Medical Center, Rotterdam, Netherlands
[e] Department of Radiology and Nuclear Medicine, Erasmus MC - University Medical Center Rotterdam, Rotterdam, Netherlands
[f] Department of Neurology, School for Cardiovascular Diseases (CARIM), Maastricht University Medical Center, Maastricht, the Netherlands
[g] Department of Radiology and Nuclear Medicine, Maastricht University Medical Center, Maastricht, Netherlands
[h] CArduivascular Research Institute Maastricht (CARIM), Maastricht, the Netherlands
[i] Department of Diagnostic Imaging, University of Calgary, Calgary, Alberta, Canada
[j] Department of Clinical Neurosciences, University of Calgary, Calgary, Alberta, Canada
[k] Calgary Stroke Program, University of Calgary, Calgary, Alberta, Canada
[l] Department of Radiology, Royal Melbourne Hospital, Melbourne, Victoria, Australia
[m] Department of Neurology, University of Pittsburgh, Pittsburgh, Pennsylvania, US
[n] Department of Neurology, Hospital Universitari Germans Trias i Pujol, Barcelona, Spain, Badalona, Spain
[o] Department of Medicine, University of Melbourne, Parkville, Victoria, Australia
[p] Department of Neurology, Royal Melbourne Hospital, Melbourne, Victoria, Australia
[q] Department of Neurology, UCLA, Los Angeles, California, US
[r] Department of Neurology, Amsterdam UMC, location AMC, Amsterdam, the Netherlands
[s] 9Institute of Neuroscience & Psychology, University of Glasgow, Queen Elizabeth University Hospital, Glasgow, Scotland, UK
[t] 20Translational and Clinical Research Institute, Faculty of Medical Sciences, Newcastle University, Newcastle upon Tyne, UK
[u] 1Department of Neuroradiology, Newcastle upon Tyne Hospitals, Newcastle upon Tyne, UK
[v] CIC1433-¬Epidémiologie Clinique, Inserm, Centre Hospitalier Régional et Universitaire de Nancy, Université de Lorraine, Nancy, France
[w] Department of Radiology and Nuclear Medicine, Amsterdam UMC, location AMC, Amsterdam, the Netherlands

- Barros, Renan Sales, et al. "Automatic segmentation of cerebral infarcts in follow-up computed tomography images with convolutional neural networks." *Journal of NeuroInterventional Surgery* 12.9 (2020): 848-852.
- First author, journal, https://jnis.bmj.com/content/12/9/848

Abstract

Background and purpose: Infarct volume is a valuable outcome measure in treatment trials of acute ischemic stroke and is strongly associated with functional outcome. Its manual volumetric assessment is, however, too demanding to be implemented in clinical practice.

Objective: To assess the value of convolutional neural networks (CNNs) in the automatic segmentation of infarct volume in follow-up CT images in a large population of patients with acute ischemic stroke.

Materials and methods: We included CT images of 1026 patients from a large pooling of patients with acute ischemic stroke. A reference standard for the infarct segmentation was generated by manual delineation. We introduce three CNN models for the segmentation of subtle, intermediate, and severe hypodense lesions. The fully automated infarct segmentation was defined as the combination of the results of these three CNNs. The results of the three-CNNs approach were compared with the results from a single CNN approach and with the reference standard segmentations.

Results: The median infarct volume was 48mL (IQR 15–125mL). Comparison between the volumes of the three-CNNs approach and manually delineated infarct volumes showed excellent agreement, with an intraclass correlation coefficient (ICC) of 0.88. Even better agreement was found for severe and intermediate hypodense infarcts, with ICCs of 0.98 and 0.93, respectively. Although the number of patients used for training in the single CNN approach was much larger, the accuracy of the three-CNNs approach strongly outperformed the single CNN approach, which had an ICC of 0.34.

Conclusion: Convolutional neural networks are valuable and accurate in the quantitative assessment of infarct volumes, for both subtle and severe hypodense infarcts in follow-up CT images. Our proposed three-CNNs approach strongly outperforms a more straightforward single CNN approach.

Introduction

Measuring the volume of infarcts on non-contrast computed tomography (NCCT) scans provides a quantitative assessment of infarcted brain tissue resulting from ischemic stroke. Follow-up infarct volume measured after 24hours from onset [1] is a valuable predictor of functional outcome. Infarct volume has been suggested as a surrogate endpoint for classic patient outcome scales in multiple randomized controlled trials [2]. By combining infarct volume with infarct location, a more precise prediction of patient outcome can be achieved [3].

The reference standard for infarct segmentation is manual delineation by medical experts. However, manual delineation has several disadvantages as it is time-demanding, subjective, prone to errors, and costly [4]. Accordingly, manual delineation does not work well in large cohort studies.

Convolutional neural networks (CNNs) have outperformed many existing image analysis methods for image classification and image segmentation. CNNs have produced good segmentation results in multiple medical imaging domains, including segmentation of ischemic stroke lesions in magnetic resonance images of the brain [5-7]. In this study, we evaluated the usefulness of CNNs for automatic segmentation of infarcted brain tissue in follow-up NCCT scans from patients with an acute ischemic stroke.

Materials and methods

Image data

We used anonymized image data from the HERMES collaboration [8]. This collaboration combined clinical and image data from seven clinical trials that investigated the efficacy of endovascular therapy in patients with acute ischemic stroke. Central medical ethics committees and research boards of each participating hospital approved each trial and the use of anonymized image data in this retrospective study. All patients, or their legal representatives, provided written informed consent.

We used image data only from patients with follow-up NCCT acquired between 12hours and 2 weeks after stroke onset and for whom a reference infarct segmentation was available. A total of 1026 patients had follow-up NCCT imaging acquired within the selected time window and with an available reference segmentation. Thin-slice image data were reconstructed into scans with 5mm slice thickness.

Reference segmentations

The reference infarct segmentation on the follow-up NCCT scans was manually delineated by one of two experienced observers, as described by Boers *et al* [9]. In short, infarcts were identified as hypodense areas. Infarcted tissue in the ipsilateral hemisphere with characteristics of an old infarct were excluded from the reference segmentation. NCCT scans of patients who underwent decompressive hemicraniectomy were excluded. Parenchymal hemorrhages within or adjacent to the infarcted area were included in the reference segmentation. A standard window width of 30 Hounsfield units (HU) and center level of 35 HU were used to limit variation between observers. If multiple follow-up images were available, reference segmentation was performed in the latest acquired scan. The manual segmentations were checked by one of three expert radiologists and, when necessary, corrections were made.

Preprocessing

To exclude trivial voxels that were of no interest, such as air or skull, we used automatic methods for intracranial region and cerebrospinal fluid (CSF) segmentation. First, we excluded all voxels outside the brain using an intracranial region segmentation. Subsequently, we also discarded all voxels selected by the CSF segmentation. All discarded voxels were neither used to train the CNN nor used for accuracy testing of the CNN.

The intracranial region segmentation uses the size range of the foramina of the skull, as reported by Berge *et al*, [10] and typical HU values of the skull. This segmentation was performed according to the following steps:

- A threshold-based segmentation was performed to segment bones. We considered everything with intensity >160 HU as bone.
- A morphological dilatation with a 7mm radius was used to close all foramina of the skull except the foramen magnum.
- The center of gravity of the segmented bone was used as a seed for a region growing inside the skull.
- A morphological dilatation with a 7mm radius was applied to the region growing result to bring the segmented intracranial region close to the skull border.
- The foramen magnum was detected by evaluation of the segmented area in each individual slice from top to bottom. The foramen magnum slice was determined as the first slice with a segmented area < 900 mm^2 after the slice with the maximum segmented area. All voxels below the foramen magnum slice were excluded from the segmentation.

The CSF segmentation was performed by selecting the voxels around the centroid of the segmented intracranial region as seeds for region growing. All voxels within a maximum distance of 15mm from this centroid and with density values between –5 and 13 HU were used as seeds. The lower and upper thresholds of this region growing were also –5 and 13 HU.

We used a previously presented method for automated intracranial hemorrhage segmentation11 to exclude the parenchymal hemorrhages of the CNN-based infarct segmentation. These hemorrhage voxels were not used to train the CNN. However, for infarct volume accuracy testing, any area that was classified as hemorrhage was added to the infarct segmentation.

CNN-based infarct segmentation

The CNN architecture used in this study was developed in-house. Its hyperparameters were optimized for segmentation of a single foreground structure in head NCCT scans, which in this case was the infarcted brain tissue. Previously, the same CNN architecture was successfully used for intracranial hemorrhage segmentation [11]. This CNN architecture determines the probability of the voxel at the center of an image patch being foreground (infarcted tissue) or background (any other tissue). This probability was subsequently dichotomized using a cut-off value, which was optimized with the data in the validation set.

The CNN architecture has two convolutional layers followed by two fully connected dense layers. Each dense layer has 256 nodes. The size of the input patch was 19×19×3 voxels; 19×19 voxels in the axial plane and three slices high. Each slice of the input patch was processed as a different image channel. After each convolutional layer, there is a max-polling layer with a 2×2 kernel and a 2×2 stride. The first convolutional layer has 64 feature maps and the second has 128 feature maps. Both convolutional layers have kernels with size 5×5.

The hypodensity of the infarcted tissue in NCCT scans is related to breakdown of cells and its fluid content. As shown in figure 1, the infarcted areas in the three NCCT scans have different HU values. In figure 1, we also show the distribution of the average HU values of the infarct reference segmentations. In our population, the HU value distribution depicted three peaks, which we named subtle, intermediate, and severe hypodense infarcts. Because of this observation, we trained three CNNs. Each of these CNNs was trained to classify a different hypodensity distribution of infarcted brain tissue. We grouped all patients according to the hypodensity of the delineated infarct. We used the average HU value of the infarction for this grouping. The average infarct intensity was computed after excluding the hemorrhage voxels of the reference

segmentation. The thresholds that define each infarction class were (14, 22) HU for severe, (22, 32) HU for intermediate, and (32, 44) HU for subtle.

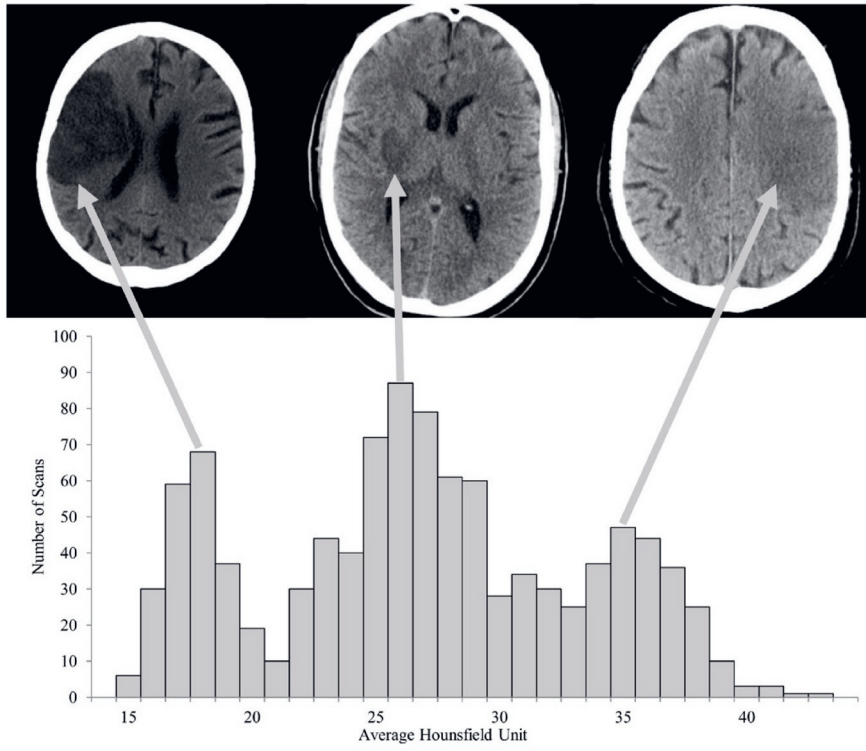


Figure 1: Histogram of average infarct intensities of the manually delineated infarcts. The left CT image at the top displays a relatively old infarct with a severe hypodensity; in the middle, an intermediate old infarct is shown; and the image on the right shows a relatively young infarct with a subtle hypodensity.

We used 570 randomly selected scans to train the three CNNs. We augmented the number of training infarct patches by flipping along the sagittal plane and by rotation. No data augmentation was applied to the non-infarct patches. We used an additional 60 scans to optimize the cut-off value for generating binary segmentations, 20 scans for each CNN. The union of the results of these three CNNs and the result of the intracranial hemorrhage segmentation was considered to be the automated generated infarct segmentation. The remaining 396 scans were used to test segmentation performance.

For comparison, we also trained a single CNN architecture for the segmentation of all types of infarction. The same methodology and data were used for this single CNN approach and the three-CNNs approach.

We used the Dice coefficient as an accuracy measure of the infarct segmentation performance in the test set. We calculated the intraclass correlation coefficients (ICCs) to compare the reference and the automatically generated infarct volumes. ICCs were interpreted according to the American Psychological Association *et al* [12]: < 0.4 is poor; ≥0.4 to < 0.6 is fair, ≥0.6 to <0.75 is good, and ≥0.75 is excellent. We opted not to compare our approach with U-Net or Mask R-CNN architectures. Both these architectures are more extensive than the proposed architecture and, in a straightforward approach, their input would be an entire NCCT slice. Since we used 5mm reconstructions, and not all slices from a NCCT scan have infarction, we did not expect a satisfactory segmentation given the limited number of NCCT slices with infarcted brain tissue that would be used as training samples.

Result

The median infarct volume was 48 (IQR 15–125)mL overall, with 29 (IQR 11–86), 46 (IQR 18–101), and 89 (IQR 35–210)mL for patients with a subtle, intermediate, and severe hypodense infarct, respectively.

The comparison between manually delineated infarct volumes and the volumes from the three-CNNs approach showed an excellent agreement with an ICC of 0.88. Even better agreement was observed for severe and intermediate hypodense infarcts with ICCs of 0.98 and 0.93, respectively. Agreement was good for subtle hypodense infarcts, with an ICC of 0.66. In figure 2, the agreement between the infarct volumes is shown. Agreement of the single CNN approach was poor, with an ICC of 0.34.

The average Dice coefficient achieved by the three-CNNs approach was 0.57±0.26. The average Dice coefficients for each category were 0.78±0.09, 0.61±0.21, and 0.37±0.26, for the severe, intermediate, and subtle hypodense infarcts, respectively. The method based on a single CNN achieved an average Dice coefficient of 0.18±0.23. Table 1 shows a summary of the segmentation performance measures. In figure 3, we show some sample results from the three-CNNs approach.

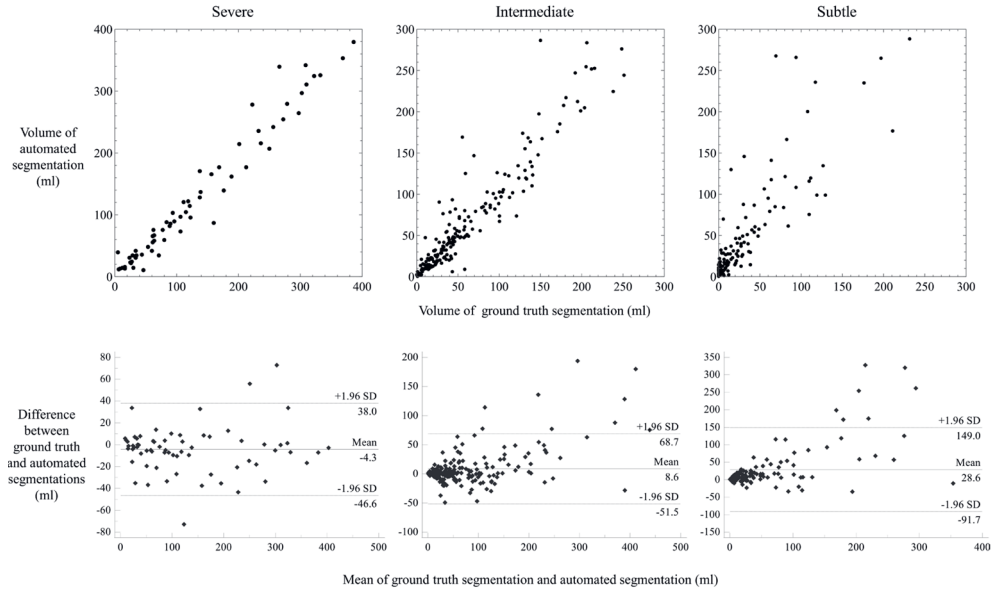


Figure 2: Top: Comparison of the infarct volume of the results from the three-CNNs approach (y axis) with the reference to infarct volume (X axis). Bottom: Bland-Altman plots of the infarct volumes. The difference in the volume determination is given along the Y axis, and the average of the automated and reference infarct volume is depicted along the x axis. The different columns show separate severe, intermediate, and subtle hypodensity infarcts.

Table 1: Results of automated infarct segmentation for severe, intermediate, and subtle hypodense infarcts and the average over the whole test dataset for the three-CNNs approach. for comparison with the accuracy of the single CNN approach.

		ICC	Dice	Test set size
Three-CNNs approach	Severe	0.98	0.78±0.09	67
	Intermediate	0.93	0.61±0.21	204
	Subtle	0.66	0.37±0.26	125
	All infarctions	0.88	0.57±0.26	396
Single CNN approach	All infarctions	0.34	0.18±0.23	396

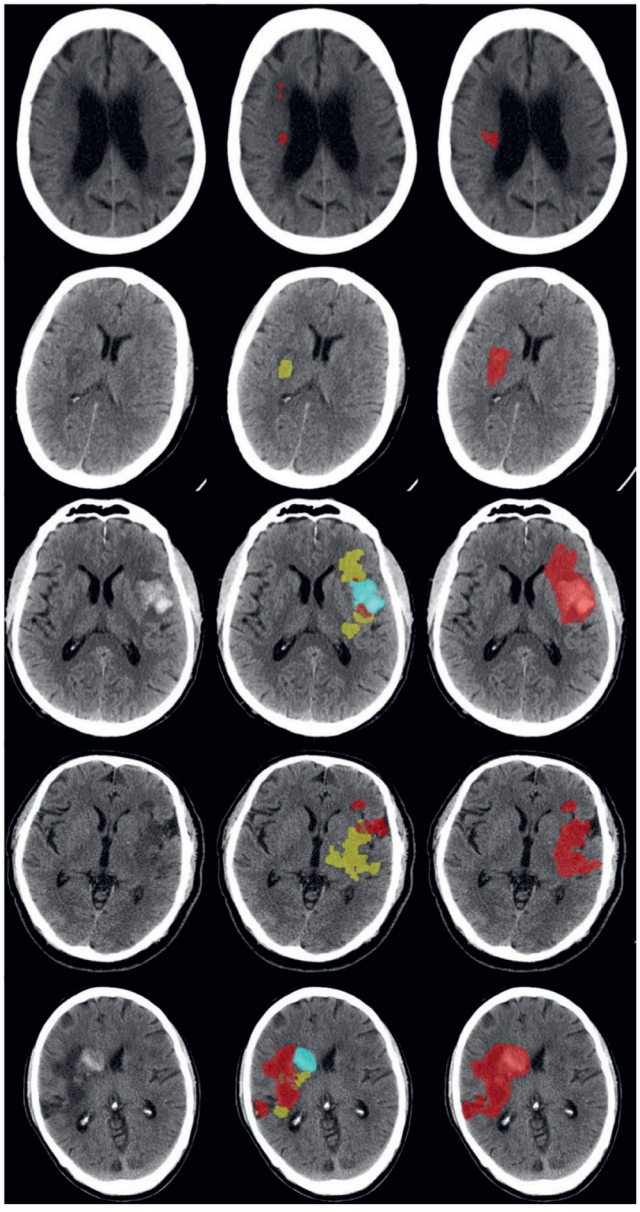


Figure 3: Sample results. from left to right we have input image, union of the segmentation results, and reference segmentation. For simplicity, in the center column we rendered the hemorrhages (blue) over the subtle infarcts (yellow), subtle infarcts over standard infarcts (orange), and standard infarcts over severe infarcts (red). The Dice coefficients from top to bottom were 0.10, 0.26, 0.40, 0.55, and 0.70. In the left colum the original images are shown. The right shows the merged segmentations.

Discussion

We have shown that CNNs are valuable in the automated cerebral infarct segmentation in follow-up CT images of patients with acute ischemic stroke, with excellent agreement with volumetric assessments of expert observers. Owing to the wide variety of the severity of hypodensities, we proposed using the combination of three CNNs, which strongly outperformed a single CNN approach.

Infarct location and infarct volume have been strongly associated with outcome of patients with ischemic stroke in several studies [3, 13]. Reliably segmenting cerebral infarcts is challenging because of pathophysiological heterogeneity, presence of preexisting pathologies such as old infarcts, leukoaraiosis, atrophy, intrinsic differences in attenuation of grey and white matter, and hemorrhagic transformation. Thus, to be able to develop robust automated methods for cerebral infarct segmentation, heterogeneous image data are required. The proposed method was evaluated in a large cohort of patients from seven multicenter randomized trials enrolling in multiple countries. The follow-up NCCT scans used in our study also had a (pragmatically) wide range of follow-up time after stroke onset, ranging from 12hours to 2 weeks. Despite these variations, the proposed approach based on three different CNNs produced accurate cerebral infarct segmentations. The volume of these segmentations had good or excellent correlation with the reference infarct volume. We have shown that accuracy for old, severe hypodense infarcts was higher than for subtle hypodense infarcts. Note that, although we presented the results in a selective manner, exactly the same procedure was applied for the infarct segmentations in all the three different infarct categories

A number of previous studies on automatic infarct core segmentation in various image modalities have been presented. Multiple CNN-based techniques have been introduced recently. On baseline CT perfusion, state-of-the-art infarct segmentation was obtained by a CNN architecture proposed by Liu *et al*, [14] achieving an average Dice coefficient of 0.51±0.31. On MRI the CNN architecture proposed by Kamnitsas *et al* [6] reported an average Dice coefficient of 0.66±0.24. Maier *et al* [7] tested several methods with different types of MR images. Their best reported result was achieved by a CNN with an average Dice coefficient of 0.73±0.18. The current state-of-the-art method for infarct segmentation on MR images is the CNN proposed by Zhang *et al*, [5] which achieved an average Dice coefficient of 0.79 in a test set with 90 images. Although good segmentation results were achieved in CT perfusion and MR images, NCCT scans are still the predominant method for assessment of follow-up infarct in patients with ischemic stroke. Therefore, we focused on using NCCT as input for the proposed cerebral infarct segmentation method.

On NCCT scans, two semiautomated methods are available for infarct segmentation. The semiautomated method by Bardera *et al* [15] was evaluated with 18 patients and reported a Pearson's correlation coefficient of 0.98 and 0.97 compared with the manual segmentations from two different observers. The semiautomated method by Kuang *et al* [16] was evaluated with 16 patients and reported an average Dice coefficient of 0.76±0.10. By contrast, our method is both fully automated, which avoids the variability introduced by the user inputs, and has been tested on a far larger number of patients.

Other fully automated methods for infarct segmentation on NCCT are available. The method by Boers *et al* [17] reported an average Dice coefficient of 0.74±0.13 in a test set with 34 images. The average onset to follow-up scanning time in the study by Boers *et al* was 4.1±2.3 days. The average Dice coefficient between human observers in the study by Boers et al was 0.84 ranging from 0.63 to 0.94, which was somewhat higher than the agreement we achieved. However, it should be noted that the manual delineation was performed for old, hypodense infarcts only. The method by Vos *et al* [18] reported an average Dice coefficient of 0.74±0.09 in a test set with 30 images. In the study by Vos *et al*, the average time between onset and scan acquisition was 3 days ranging between 2 and 5 days. More recently, the method by Gillebert *et al* [19] was evaluated with 12 patients with ischemic stroke and reported Dice coefficients ranging from 0.27 to 0.71. The scans used to evaluate the method by Gillebert *et al* had an average acquisition time after onset of 40hours. Their method was evaluated in a limited set of selected images to illustrate different types of ischemic stroke lesions. In contrast with the methods of Boers *et al*, Vos *et al*, and Gillebert *et al*, our method has been thoroughly evaluated with a large and diverse test set.

The data used in our study included follow-up scans as early as 12hours after stroke onset. Infarcts in these early follow-up scans might be subtle and harder to segment. Thus, it was expected that our method would achieve a lower accuracy in such scans. Moreover, the manual delineation in these scans is more difficult, resulting in more variation among experts. This may also strongly contribute to the lower agreement of the automated method with the reference standard. It some cases (also in figure 3), the network in charge of segmenting subtle infarcts overestimates the infarct region by including subtle hypodense areas which are not part of the infarction. Another common source of misclassifications by our proposed method is the inclusion of cerebral sulci in the results of the network trained to segment severe infarctions (figure 3).

A major limitation is the highly selective nature of the HERMES population. All patients had anterior circulation stroke confirmed by CT angiography, mostly within 6 hours of onset. Patients were excluded from most studies if they had prior disability or low Alberta Stroke Program Early CT scores. As a result, many of the background abnormalities

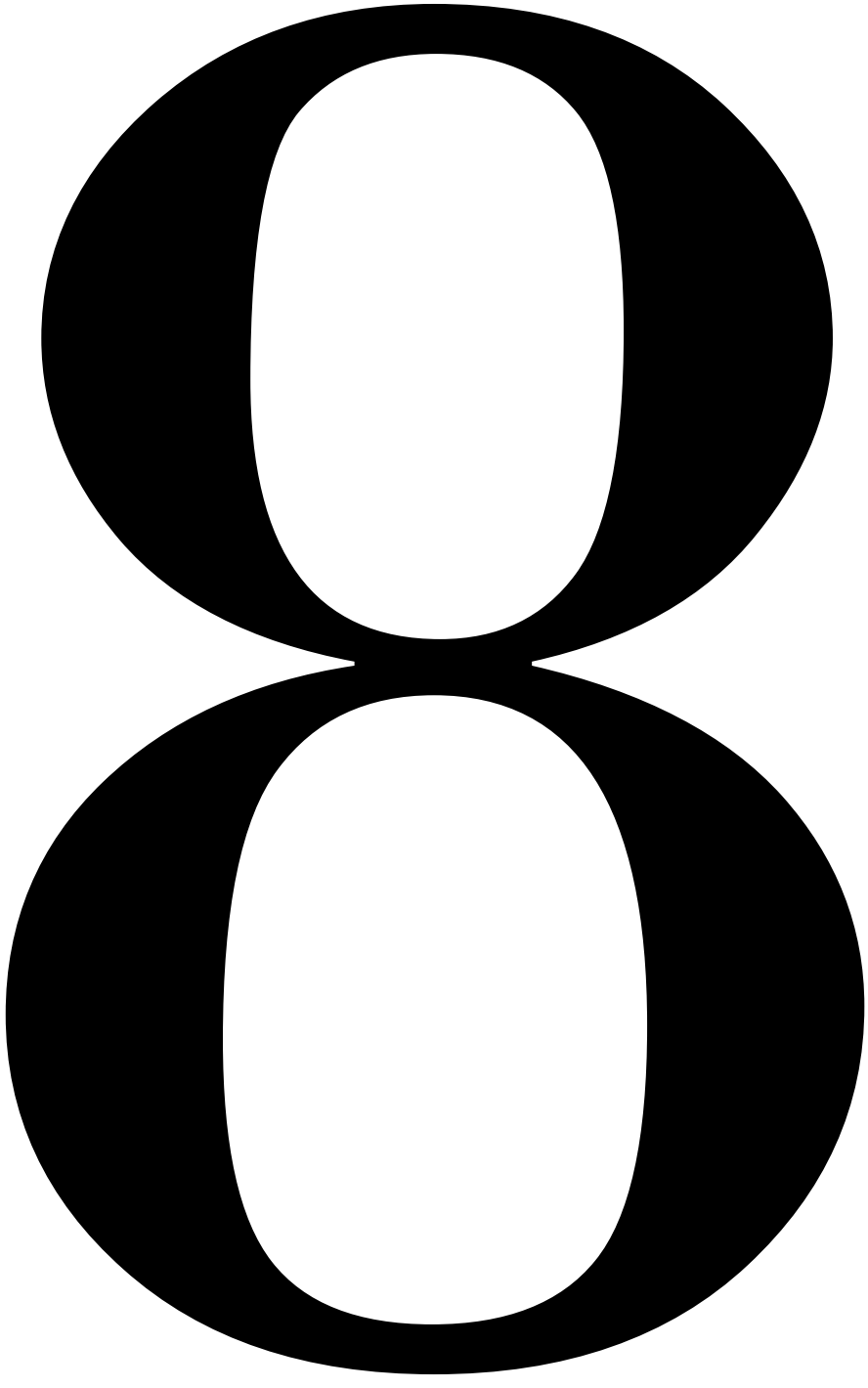
typical in populations with acute stroke were less prevalent in our population. Moreover, average age was around 69, and very elderly patients were under-represented. Despite variation among study populations, these still represent a much more homogeneous group than patients with stroke as a whole.

Overall, the proposed method achieved an excellent correlation with the reference infarct volume. This suggests that our method can be used in clinical trials, replacing tedious manual delineations. Its value in functional outcome prediction for patients with ischemic stroke and its value as a secondary outcome measure in treatment trials still has to be established.

References

1. Berkhemer, Olvert A., et al. "Imaging biomarkers for intra-arterial stroke therapy." *Cardiovascular engineering and technology* 4.4 (2013): 339-351.
2. Warach, Steven J., et al. "Acute stroke imaging research roadmap III imaging selection and outcomes in acute stroke reperfusion clinical trials: consensus recommendations and further research priorities." *Stroke* 47.5 (2016): 1389-1398.
3. Ernst, Marielle, et al. "Association of computed tomography ischemic lesion location with functional outcome in acute large vessel occlusion ischemic stroke." *Stroke* 48.9 (2017): 2426-2433.
4. Doyle S, Forbes F, Jaillard A. Sub-acute and Chronic Ischemic Stroke Lesion MRI Segmentation. In: Brainlesion: glioma, multiple sclerosis, stroke and traumatic brain injuries. *Springer International Publishing*, (2018): 111–22.
5. Zhang, Rongzhao, et al. "Automatic segmentation of acute ischemic stroke from DWI using 3-D fully convolutional DenseNets." *IEEE transactions on medical imaging* 37.9 (2018): 2149-2160.
6. Kamnitsas, Konstantinos, et al. "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation." *Medical image analysis* 36 (2017): 61-78.
7. Maier, Oskar, et al. "Classifiers for ischemic stroke lesion segmentation: a comparison study." *PloS one* 10.12 (2015): e0145118.
8. Goyal, Mayank, et al. "Endovascular thrombectomy after large-vessel ischaemic stroke: a meta-analysis of individual patient data from five randomised trials." *The Lancet*387.10029 (2016): 1723-1731.
9. Boers, Anna MM, et al. "Mediation of the relationship between endovascular therapy and functional outcome by follow-up infarct volume in patients with acute ischemic stroke." *JAMA neurology* 76.2 (2019): 194-202.
10. Berge, Jennifer K., and Ronald A. Bergman. "Variations in size and in symmetry of foramina of the human skull." *Clinical Anatomy: The Official Journal of the American Association of Clinical Anatomists and the British Association of Clinical Anatomists* 14.6 (2001): 406-413.
11. Barros RS, der SWEvan, Boers AMM, et al. (Submmited) automated detection and segmentation of subarachnoid hemorrhages with Convolutional neural networks. *Radiol Artif Intell* (2019);In press.
12. Cicchetti, Domenic V. "Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology." *Psychological assessment* 6.4 (1994): 284.
13. Bivard, Andrew, et al. "Validating a predictive model of acute advanced imaging biomarkers in ischemic stroke." *Stroke* 48.3 (2017): 645-650.
14. Liu, Liangliang, et al. "Multi-scale deep convolutional neural network for stroke lesions segmentation on CT images." *International MICCAI Brainlesion Workshop*. Springer, Cham, 2018.
15. Bardera, Anton, et al. "Semi-automated method for brain hematoma and edema quantification using computed tomography." *Computerized medical imaging and graphics*33.4 (2009): 304-311.
16. Kuang, Hulin, et al. "Joint segmentation of intracerebral hemorrhage and infarct from non-contrast CT images of post-treatment acute ischemic stroke patients." *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, Cham, 2018.

17. Boers, Anna M., et al. "Automated cerebral infarct volume measurement in follow-up noncontrast CT scans of patients with acute ischemic stroke." *American Journal of Neuroradiology* 34.8 (2013): 1522-1527.
18. Vos, Pieter C., et al. "Automatic detection and segmentation of ischemic lesions in computed tomography images of stroke patients." *Medical imaging 2013: computer-aided diagnosis*. Vol. 8670. International Society for Optics and Photonics, 2013.
19. Gillebert, Celine R., Glyn W. Humphreys, and Dante Mantini. "Automated delineation of stroke lesions using brain CT images." *NeuroImage: Clinical* 4 (2014): 540-548.



Chapter 8

Discussion

Discussion

Processing time is brain

Treatment decisions in stroke are heavily dependent on imaging. The same way that newer generations of digital cameras have higher image resolutions and bigger image files, a comparable trend is observed with CT imaging for stroke patients. Early scans had poor resolution, thick slices, and covered only a small portion of the patient head. Nowadays, is not uncommon to find CT perfusion acquisitions with 30 volumes covering the entire patient head with tens of slices containing 512x512 voxels. It is unrealistic to expect that all these gigabytes of information are visually inspected by a radiologist without any help of a computer. It is also not reasonable to expect that all stroke centers around the world have dedicated supercomputers to analyze all these image data as fast as required for the work up of stroke patients.

In other words, there is an increasing demand for processing power to analyze increasingly larger CT scans. The infrastructure to deliver this processing power needs to be scalable, cost effective, and easily accessible. In this context, the goal of this thesis was to investigate if high performance computing (HPC) can contribute for fast and accurate analysis of medical images of acute stroke patients. On top of that, all HPC technologies used throughout this thesis are easily available to any regular consumer. All experiments in this thesis were performed using consumer hardware or public cloud infrastructures. The ambition was also to demonstrate that such HPC infrastructures can enable advanced image analysis in a fast enough manner to be used in the real-world workflow of stroke patients. Fast treatment decision in stroke increases the changes of good patient outcome. Therefore, also processing time is brain.

Amdahl's law

In the first part of this thesis, we investigated how HPC techniques can be used to speed up image processing algorithms that analyze CT scans from stroke patients. Initially, HPC techniques were directly applied to an existing algorithm for cerebral infarct segmentation in follow-up non-contrast CT (NCCT) scans. This process is described in Chapter 2. Since the original cerebral infarct segmentation algorithm was fundamentally sequential, only a limited gain in performance gains was achieved, and the average speed up was only 30 seconds in the best configuration evaluated. The results presented in Chapter 2 suggested that to fully benefit from HPC existing image processing algorithms needed to be modified or entirely redesigned to take advantage of what HPC can offer. In short, one cannot simply throw 10 times more computing power to an existing image processing algorithm and expect it to run 10 times faster. If the algorithm is not suited for taking advantage of the architecture where it is being executed, the obtained performance gains is limited. Amdahl's law already stated that there is a limit to speed up

of an algorithm regardless of the amount of parallel processing that is available.

Stroke is not a game

The most common architecture used to deliver HPC is based on graphical processing units (GPUs). GPUs were originally designed for performing the computations needed to create 30-60 images per second in a video game. In this context, data needed to be loaded one time into GPU memory and after that, the GPU needed to generate these 30-60 images per second through the game session without minimal additional loading of data into the GPU memory. This is what GPUs were made for: load data once, do a lot of computations after using that same data over and over again. The situation is quite different when processing a CT scan. The data that needs to be loaded often does not fit into GPU memory, meaning that the data needs to be split into multiple chunks that are loaded multiple times to be processed only once.

To reduce the impact of this bottleneck, we proposed a compression technique to address the limitations in memory access when processing a CT scan (Chapter 3). To have a better measure of the impact of HPC techniques, we adapted an existing image processing algorithm for running in massive parallel architectures (Chapter 4).

The results presented in Chapter 4 show that the processing of CT perfusion (CTP) scans can be reduced from around 30 minutes to 8 seconds when using HPC techniques. When combining HPC and the compression method proposed in Chapter 3, that number was reduced even further to only 6 seconds. The proposed compression technique achieves a compression ratio of 2 and, when comparing it with commonly available compression algorithms, it was able to compress and decompress data 4 times faster than the fastest available method, which is Run-length encoding (RLE). Although JPEG achieves better compression ratios (around 4.5), the time needed to compress and decompress required by JPEG is 11 times longer than the proposed method.

CTP scans are the largest scans acquired during the stroke workflow, with size ranging between 2.5 GB and 5 GB. Following the results of multiple studies [1 - 3], the processing of CTP scans became part of the clinical guidelines for stroke management. Thus, we can conclude that the methods presented in chapters 3 and 4 are relevant for speeding up the assessment of ischemic stroke patients.

High performing clouds

The analyses presented in chapters 3 and 4 were performed using local workstations. Given the incidence of stroke worldwide, it is important that a more scalable approach for delivering HPC techniques is available. In chapter 5, we investigated the use high performance computing based on cloud infrastructures for improving the stroke

workflow. This investigation concluded that, compared with the traditional workflow, the usage of cloud-based infrastructures presents several advantages:

- It enables sharing of data between hospitals in only a few minutes. Traditionally, several minutes are required to burn a disk with the patient data and much time was required to send such a disk with a cab or ambulance to another hospital.
- It enables the use of large-scale cloud computing resources and advanced algorithms for processing the image data. In the traditional setting, the computing power is limited by the infrastructure that is available in the hospital premises.
- It allows a Pay-per-use model instead of the large upfront costs for acquiring the necessary infrastructure.
- It supports remote collaboration sessions with patient data directly available to all participants.

Jump into the deep

Deep learning emerged as a technique that leverages HPC and large datasets. As the next step on applying HPC to stroke management, part 2 of this thesis focused on the development of new deep learning algorithms for the segmentation of subarachnoid hemorrhages (SAH) and final cerebral infarct volume (FIV). By designing algorithms from the ground up to be implemented in HPC architectures, the expectation was to achieve substantially better performance in terms of execution time as well as in terms of accuracy. In the case of SAH segmentation, the processing time is around 30 seconds with an average Dice coefficient of 0.63. For comparison, the heuristic-based method proposed by Boers et al. [4] takes around 5 minutes with an average Dice coefficient of 0.55. The same approach used was adapted for FIV segmentations. In this case, the same processing time of around 30 seconds was achieved. For comparison, the FIV segmentation method described in chapter 2 took around 10 minutes to compute.

The original implementations of the methods in chapters 2, 4, 6, and 7 use some sort of transformation to reduce the computation load (see summary in Table 1). Those transformations are required because these methods were implemented to run sequentially. The counterpart methods proposed in this thesis do not suffer from such a limitation. Furthermore, the proposed methods have the potential to achieve even faster processing times if more parallel processing is used.

Table 1. Impact image processing algorithms running on HPC platforms.

	Task	Time of proposed method	Time of reference method	Transformation used by reference method	Reference method
Chapter 2	FIV segmentation	10 minutes	2 hours	Thick slice images	Boers et al. [5]
Chapter 4	CTP noise reduction	6 seconds	30 minutes	Truncation of kernel operations	Mendrik et al. [6]
Chapter 6	SAH segmentation	30 seconds	5 minutes	Thick slice images	Boers et al. [4]
Chapter 7	FIV segmentation	30 seconds	10 minutes	Thick slice images	Chapter 2

You only have to look at the Medusa straight on to see her…

It should be noted that when the results of the chapters in Part 1 were made publicly available, there was a strong resistance in the medical field against using cloud computing for processing patient data. This resistance came primarily from concerns regarding security and patient privacy. The work done in this thesis was part of a larger research project called Medical Distributed Utilization of Services & Applications (MEDUSA). This research project pioneered the development of the required technologies for processing medical data in cloud platforms. Nowadays, most of the concepts that were initially evaluated during the MEDUSA project are currently implemented in commercial solutions of companies such as NICO.LAB, Viz, Brainomix, and RapidAI. All these companies have cloud-based solutions that aim at improving the stroke care workflow by using advanced image analysis and cloud-based data sharing.

There were also doubts towards the practical aspects of the deep learning techniques in real clinical workflows. The adoption of deep learning models for radiology-related tasks were seen with skepticism by most medical professionals because of the lack of interpretability and explainability offered by these so called "black box" models. However, as shown in both chapters of Part 2 of this thesis, deep learning models can go beyond producing simple classification based on an input image. The method used in chapter 6, for instance, can highlight the visual features that lead to a positive detection of a SAH. This way, the medical professional can more easily interpret and evaluate the output of the deep learning model. In addition to the technical developments to address these interpretability and explainability concerns, there were also advances in the medical regulations. For instance, the FDA has proposed a regulatory framework for artificial intelligence and machine learning-based medical software (https://www.fda.gov/media/122535/download).

The introduction of more inclusion and exclusion criteria for different stroke treatment options has been a main factor forcing the adoption of deep learning-based methods for supporting stroke treatment decisions. For example, deep learning models can be used for measuring differences in clot characteristics to support treatment selection for mechanical removal of that clot. Alternatively, deep learning models can be used to determine the eligibility for thrombectomy outside the 6h standard treatment window in locations where CTP imaging is not available.

Despite the advancements in the field of deep learning in general, when it comes to application in the medical domain, there is still a need for improvements in terms of generalization. Unfortunately, the largest datasets available for training deep learning models for stroke are still around a few thousand images. This way, rare conditions that happen in, for example, only 5% of cases are poorly represented in such datasets, which limits the applicability of the trained models in these rare cases. Unfortunately, having a deep learning model that only works in the easy cases is less useful for the medical experts. Another common limitation of these models is the need for high quality ground truth labels. Such labels are difficult, expensive, and time consuming to acquire. Fortunately, there is an extensive body of research done about unsupervised learning techniques. Unfortunately, the use of unsupervised learning techniques to support the stroke workflow is still extremely limited.

Conclusion

This thesis addressed various aspects of the combination of HPC-enabled machine learning in combination with cloud infrastructures. It has been shown that this combination has an enormous potential to achieve fast and accurate analysis of large amounts of radiological images of patients suspected of stroke. These addressed technologies can democratize the usage of high-quality diagnostic tools which, in turn, can lead to a significant impact in the stroke care as this thesis shows demonstrations of benefits of HPC techniques in diagnosis of both ischemic and hemorrhagic stroke. We have also demonstrated that the clinical stroke workflow can potentially benefit from cloud-based solutions, and that we can use image analysis algorithms for supporting the assessment of stroke patients.

References

1. Albers, Gregory W., et al. "Thrombectomy for stroke at 6 to 16 hours with selection by perfusion imaging." *New England Journal of Medicine* 378.8 (2018): 708-718.
2. Nogueira, Raul G., et al. "Thrombectomy 6 to 24 hours after stroke with a mismatch between deficit and infarct." *New England Journal of Medicine* 378.1 (2018): 11-21.
3. Brott, Thomas, et al. "Measurements of acute cerebral infarction: a clinical examination scale." *Stroke* 20.7 (1989): 864-870.
4. Boers, A. M., et al. "Automatic quantification of subarachnoid hemorrhage on noncontrast CT." *American journal of neuroradiology* 35.12 (2014): 2279-2286.
5. Boers, Anna M., et al. "Automated cerebral infarct volume measurement in follow-up noncontrast CT scans of patients with acute ischemic stroke." *American Journal of Neuroradiology* 34.8 (2013): 1522-1527.
6. Mendrik, Adriënne M., et al. "TIPS bilateral noise reduction in 4D CT perfusion scans produces high-quality cerebral blood flow maps." *Physics in Medicine & Biology* 56.13 (2011): 3857.

Summary

Summary

The aim of this thesis is to investigate the use of high-performance computing (HPC) techniques in the development of methods for the detection and quantification imaging biomarkers for supporting the clinical workflow of acute stroke patients. In the first part of this thesis, we evaluate various HPC technologies and how these technologies can be leveraged by different image analysis applications used in the context of acute stroke care. The second part of this thesis focuses on developing and improving methods to quantitatively assess imaging biomarkers in computer tomography (CT) scans from stroke patients. In **Chapter 2,** we evaluated how computers with multiple computing devices can be used to accelerate medical imaging applications. These computers with different computing devices, such as multi-core CPUs (central processing units), GPUs (graphical processing units), and FPGAs (field-programmable gate arrays), are referred to as heterogeneous platforms. For the evaluation presented in **Chapter 2**, we used a new framework named FlowCL. We designed FlowCL for the development of parallel medical imaging applications running heterogeneous platforms. We compared an implementation of a region-growing method for cerebral infarct volume measurement with a new implementation targeted for heterogeneous platforms. The results of this new implementation are generated with significant speed-up compared to the original implementation.

Chapter 3 proposes a novel data compression technique that allows the efficient processing of CT perfusion (CTP) images in GPUs. The size of CTP datasets makes data transfers to computing devices time-consuming and therefore not suitable in acute situations. **Chapter 3** introduces a fast and lossless compression algorithm for CTP data to reduce the time spent in such data transfers. The algorithm exploits redundancies in the temporal dimension of the CTP data and keeps random read-only access to the image elements directly from the compressed data on the GPU.

Chapter 4 goes a step further and evaluates the algorithm's usefulness proposed in **Chapter 3** with two different applications: a double threshold segmentation and a time-intensity profile similarity (TIPS) bilateral filter to reduce noise in CTP scans. The results show that the processing of compressed data uses between 2 and 2.8 times less memory and the execution times are between 1.2 and 1.7 times faster than when processing the original data. The outputs when processing compressed data are identical to the outputs when processing the original uncompressed data.

Chapter 5 presents a cloud platform for deploying medical applications. The goal of this platform is to improve acute care workflows by enabling fast medical data exchange, advanced processing of medical image data, automated decision support, and remote

collaboration between physicians in a secure and responsive virtual space. **Chapter 5** describes a prototype implemented in this cloud platform that supports the treatment of acute stroke patients. As a result, this prototype improves several aspects of the acute stroke clinical workflow and has the potential to play an essential role in the management of acute stroke patients.

In **Part 2** of this thesis, **Chapter 6** presents a convolutional neural network (CNN) for the detection and volumetric segmentation of subarachnoid hemorrhages (SAH) in non-contrast CT scans of patients with subarachnoid hemorrhagic stroke. The CNN was trained with 302 baseline non-contrast CT scans. The final segmentation performance was evaluated on an additional dataset with 473 baseline scans, and the SAH volume agreement was high with an intraclass correlation coefficient (ICC) of 0.966. The average Dice coefficient of the volumetric SAH segmentation was 0.63 ± 0.16, which is similar to the interobserver agreement between experts. The CNN was also evaluated for differentiating between ischemic and hemorrhagic stroke patients and achieved an accuracy of 0.96. The average CNN detection and segmentation time was around 30 seconds. In **Chapter 7**, another method based on CNNs was proposed for the quantification of the final infarct volumes in follow-up non-contrast CT scans from ischemic stroke patients. We developed three CNNs for the segmentation of subtle, intermediate, and severe hypodense lesions. The fully automated infarct segmentation was generated by merging the results of these three CNNs. The comparison between the results of the proposed method and the reference segmentations showed an excellent agreement with an ICC of 0.88.

Samenvatting

Samenvatting

Het doel van dit proefschrift is om het gebruik van high-performance computing (HPC) technieken te onderzoeken bij de detectie en segmentatie ontwikkeling van beeldvormende biomarkers ter ondersteuning van de klinische workflow van patiënten met een acute beroerte. In **Deel 1** van dit proefschrift evalueren we verschillende HPC-technieken en hoe dergelijke technieken kunnen worden ingezet bij verschillende beeldanalysetoepassingen in de context van zorg voor een acute beroerte. **Deel 2** van dit proefschrift richt zich op het ontwikkelen en verbeteren van de kwantitatieve bepaling van beeldvormende biomarkers in computertomografie (CT) scans van patiënten met een beroerte.

In **Hoofdstuk 2** hebben we geëvalueerd hoe computers met meerdere computerplatformen kunnen worden gebruikt om medische beeldvormingstoepassingen te versnellen. Deze computers met meerdere computerplatformen, zoals multi-core CPU's (centrale verwerkingseenheden), GPU's (grafische verwerkingseenheden) en FPGA's (field-programmable gate arrays) worden heterogene platformen genoemd. Voor de evaluatie die in **Hoofdstuk 2** wordt gepresenteerd, hebben we een nieuw raamwerk gebruikt met de naam FlowCL. We hebben FlowCL ontworpen voor de ontwikkeling van parallelle medische beeldvormingstoepassingen die op heterogene platformen draaien. We vergeleken een implementatie van een regio groeiende methode voor het meten van herseninfarctvolume met een nieuwe implementatie gericht op heterogene platformen. De resultaten van deze nieuwe implementatie worden aanzienlijk sneller gegenereerd dan bij de oorspronkelijke implementatie.

Hoofdstuk 3 stelt een nieuwe datacompressie-techniek voor die de efficiënte verwerking van CT-perfusie (CTP)-beelden in GPU's mogelijk maakt. De omvang van CTP-datasets maakt de gegevensoverdracht naar computerapparatuur tijdrovend en daarom niet geschikt in acute situaties. **Hoofdstuk 3** introduceert een snel en lossless compressie-algoritme voor CTP-gegevens om de tijd die aan dergelijke gegevensoverdrachten wordt besteed te verminderen. Het algoritme maakt gebruik van redundanties in de temporele dimensie van de CTP-gegevens en houdt willekeurige alleen-lezen toegang tot de afbeeldingselementen rechtstreeks vanuit de gecomprimeerde gegevens op de GPU.

Hoofdstuk 4 gaat een stap verder en evalueert het nut van het algoritme dat in **Hoofdstuk 3** is voorgesteld met twee verschillende toepassingen: een segmentatie met dubbele drempelwaarden en een bilateraal filter voor time-intensity profile similarity (TIPS) om ruis in CTP-scans te verminderen. De resultaten laten zien dat de verwerking van gecomprimeerde gegevens tussen de 2 en 2,8 keer minder geheugen gebruikt en dat de uitvoeringstijden tussen 1,2 en 1,7 keer sneller zijn dan het verwerken van de

originele data. De resultaten bij het verwerken van gecomprimeerde data zijn identiek aan de resultaten bij het verwerken van de originele ongecomprimeerde data.

Hoofdstuk 5 presenteert een Cloud platform voor het inzetten van medische applicaties. Het doel van dit platform is het verbeteren van workflows voor acute zorg door snelle uitwisseling van medische gegevens, geavanceerde verwerking van medische beeldgegevens, geautomatiseerde beslissingsondersteuning en samenwerking op afstand tussen artsen in een veilige en responsieve virtuele ruimte mogelijk te maken. **Hoofdstuk 5** beschrijft een prototype geïmplementeerd in dit Cloud platform dat de behandeling van patiënten met een acute beroerte ondersteunt. Als gevolg hiervan verbetert dit prototype verschillende aspecten van de klinische workflow voor acute beroerte en heeft het de potentie om een essentiële rol te spelen bij de behandeling van patiënten met een acute beroerte.

In **Deel 2** van dit proefschrift presenteert **Hoofdstuk 6** een convolutioneel neuraal netwerk (CNN) voor de detectie en volumetrische segmentatie van subarachnoïdale bloedingen (SAH) in niet-contrast CT-scans. Het CNN werd getraind met 302 baseline non-contrast CT-scans. De uiteindelijke segmentatieprestaties werden geëvalueerd op een aanvullende dataset met 473 baseline-scans en de SAH-volumeovereenkomst was hoog met een intraclass correlatiecoëfficiënt (ICC) van 0,966. De gemiddelde Dice-coëfficiënt van de volumetrische SAH-segmentatie was 0,63 ± 0,16, wat vergelijkbaar is met de overeenkomst tussen experts. De CNN werd ook geëvalueerd om onderscheid te maken tussen ischemische en hemorragische beroertepatiënten en bereikte een nauwkeurigheid van 0,96. De gemiddelde CNN-detectie- en segmentatietijd was ongeveer 30 seconden. In **Hoofdstuk 7** werd een andere methode, gebaseerd op CNNs, voorgesteld voor de kwantificering van de uiteindelijke infarctvolumes in non-contrast CT-scans van patiënten met een ischemische beroerte. We hebben drie CNN's ontwikkeld voor de segmentatie van subtiele, intermediaire en ernstige hypodense laesies. De volledig geautomatiseerde infarctsegmentatie werd gegenereerd door de resultaten van deze drie CNN's samen te voegen. De vergelijking tussen de resultaten van de voorgestelde methode en de referentiesegmentaties toonde een excellente overeenkomst met een ICC van 0,88.

Acknowledgements

Acknowledgements

What a life-changing journey a PhD can be! I am glad I could count on the support of my promoters and co-promoters throughout this journey. Antoine, thanks for always making sure that we had no obstacles with all our Ph.D. plans. Henk, thank you for all the patience, help, and fantastic fun moments during our MEDUSA-related trips. It almost feels likes this Ph.D. is more your accomplishment than mine. I honestly think that another person would not be so patient with this long, long, and let me add an extra long, nine years waiting time. Silvia, I still remember the first time we met and that you mentioned that there was a Ph.D. position in Amsterdam that could be suitable for me. You were always so helpful from the beginning. You even helped me find a flatmate so I could afford the expensive rent here in Amsterdam. Seriously, this is more than anyone can ask from a supervisor. I will always be grateful to you both for being part of this unforgettable journey. I arrived here in the Netherlands with just a suitcase. Now I have a wife, a daughter, a house, a company, and lots of new friends. It was the opportunity that you both gave me that led to so many amazing things. And on top of that, a Ph.D. title.

The good thing about a Ph.D. is that it brings many people with similar interests together. Thus, most people with whom I shared this nine years journey are also close friends. And as good friends, they will forgive me for making a short acknowledgment to them because they know more than anyone how much I need to finish this book. So here we go!

I want to thank my friends from the department of Biomedical Engineering and Physics: Mustafa, Emilie, Merel, Bart, Wessel, Haryadi, Lucas, Jorrit, and Marit. Thanks also to my Brazilian friends that moved to Europe: Rosalia, Vinicius, Carol, Roger, Rafa, PL, Fred, Dizzy, and João. I also want to thank my Amsterdam friends Juan, Maria, Heather, and Nicky. And my Nicolab friends: Elena, Kate, Aashish, Marco, Mart, Henry, Ivo, and Razmara. Thank you all for our trips together, the deep conversations, the silly jokes, etc. I learned a lot from each one of you. Thank you all for the multiple cherishable memories that make me smile whenever I think of each one of you. I love you all.

A special thank you to my paranymphs Roeland and Raquel. You two always remind me that I can find amazing friends no matter where I am in the world. Thanks for being there for me every time I needed it. But more importantly, thank you for being there when I did not need it because I love being around you two at any time. And, obviously, I also love you both.

I also want to thank my parents Elza and Renato; my siblings Nayara, Ellton, and

Clarice; and my wife Elsemiek. Without your support, I surely would not be able to be here. Ellton, you even gave me my first euros to pay my first rent in Amsterdam. Clarice, you helped me organize my agenda so I could free up some time to finish my thesis. Nayara, you are always doing so many things for everyone in the family. I cannot even comprehend how you manage that. Mieka, thanks for helping me organize all figures in the chapters of this thesis. Thank you all for all the love. I love each one of you a lot more than you think. Finally, I also want to say thank you to my baby girl Vesper. Just today, as I came home tired and without the energy to write this, you welcomed me with a loud laugh that gave me all the energy I needed.

P.S. I am a bit sleep-deprived and in a hurry. Thus, if I forgot you, you are entitled to an acknowledgment written by hand in your copy of my thesis as a consolation prize.

About the author

About the author

Renan Sales Barros was born on 29 February 1988 in Santana do Ipanema, State of Alagoas, Brazil. He started coding at age 14 and has been doing formal software engineering education since 2003. He got his bachelor's degree in computer science in 2011 from the Federal University of the State of Pará. In that year, he was awarded the best computer science student graduating. While pursuing his bachelor's degree, Renan was a research assistant in developing automated tools for learning process evaluation and software process modeling, evaluation, and improvement. He also researched software process improvement and quality assurance for over two years. In 2013, Renan got his master's degree in computer science with a thesis about mathematical models for the simulation of human pigmentation disorders and some forms of skin cancer. Renan obtained his master's degree at the Federal University of the State of Rio Grande do Sul, one of Brazil's top 5 computer science master's programs. Since 2013, Renan has been developing high-performance image processing algorithms to enhance the quality of diagnosis and decision-making when dealing with acute stroke patients. This high performance is achieved using parallel computing, GPU processing, super-computers, and cloud-based architectures. Renan has a solid background in software engineering and algorithm design. With a strong experience in software testing and quality assurance, Renan can design and implement robust and fast software solutions for medical image processing. In 2015, Renan co-founded Nicolab. Nicolab is a health-tech company that develops solutions for streamlining emergency workflows by providing physicians with all the necessary information to diagnose patients more accurately and to make faster treatment decisions. Renan is currently the CTO of Nicolab.

Portfolio

Portfolio

Name of PhD student Renan Sales Barros
PhD Period April 2013 – November 2022
Name PhD supervisors: Prof. Dr. H.A. Marquering and Dr. S.D. Olabarriaga

General courses	Year	ECTS
The AMC World of Science	2013	0.7
Systematic Reviews	2013	0.7
Practical Biostatistics	2013	1.4
E-Science	2014	0.7
Entrepreneurship in Health and Life Sciences	2014	1.5
Specific courses	**Year**	**ECTS**
Biomedical Image Analysis Summer School (Institut Henri Poincaré, Paris)	2013	1.5
Medical Imaging Summer School (Favignana, Sicily, Italy)	2014	1.5
Supervising		
Master's Internship – Malo Louvigne	2015	1.5
Master's Internship – Alexandre Urwald	2016	1.5
Master's Internship – Wassim El Youssoufi	2016	1.5
Master's Thesis – Jorrit Posthuma	2017	1.5
Master's Internship – Nil Stolt	2019	1.5
Master's Internship – Maximilian Schlögel	2020	1.5
Master's Internship – Mahsa Mojtahed	2020	1.5
Master's Internship – Marek Oerlemans	2021	1.5
Conferences, workshop, and symposiums		
European Conference on Parallel Processing	2013	1.0
Medical Imaging Symposium for PhD Students	2013	0.2
Medical Imaging Symposium for PhD Students	2014	0.2
Medical Imaging Symposium for PhD Students	2015	0.2
European Conference on Service-Oriented and Cloud Computing	2015	0.6
International Conference on Medical Image Computing and Computer Assisted Intervention	2015	1.0
European Stroke Conference	2018	0.9
International Stroke Conference	2019	0.9

Other		
Medical Distributed Utilization of Services & Applications (MEDUSA) Progress Workshops	2013-2015	4.5
Weekly Cardiovascular Engineering Meeting	2013-2017	7.4
Monthly Ischemic Stroke Meeting	2013-2017	1.7