Medical Images Research Framework

Sabrina Musatian
Saint Petersburg State University
Saint Petersburg, Russia
Email: sabrinamusatian@gmail.com

Alexander Lomakin
Saint Petersburg State University
Saint Petersburg, Russia
Email: alexander.lomakin@protonmail.com

Angelina Chizhova
Saint Petersburg State University
Saint Petersburg, Russia
Email: chilina4@gmail.com

***Abstract*—with a growing interest in medical research problems and the introduction of machine learning methods for solving those, a need in an environment for integrating modern solutions and algorithms into medical applications developed. The main goal of our research is to create medical images research framework (MIRF) as a solution for the above problem. MIRF is a free open–source platform for the development of medical tools with image processing. We created it to fill in the gap between innovative research with medical images and integrating it into real–world patients treatments workflow. Within a short time, a developer can create a rich medical tool, using MIRF's modular architecture and a set of included features. MIRF takes the responsibility of handling common functionality for medical images processing. The only thing required from the developer is integrating his functionality into a module and choosing which of the other MIRF's features are needed in the app. MIRF platform will handle everything else. In this paper, we overview and compare existing applications for handling operations with medical images, as well as describing basic ideas and functionality behind our own MIRF framework.**

1. Introduction

Over the past decade, many kinds of approaches for solving problems in the field of medical images were explored. Because of these researches, the scientific community can now rapidly open new and more challenging tasks. However, these studies should go beyond just algorithmic decisions related to diagnosis and treatment using CT (Computed Tomography) and MRI (Magnetic resonance imaging) images. Doctors require high–performance real–time software systems that can assist in the diagnosis's determination of the patient and solve various related tasks. Hence, it is necessary not only to develop highly efficient algorithms for medical images analysis but also to integrate them into a convenient environment in which many other instruments essential for physicians may be seamlessly used. A set of medical tasks share many of these tools, which means that these tools can be provided within a single platform. In this paper, we investigate existing software systems for medical images and introduce our own framework (MIRF) for medical diagnosis, simplifying the development of medical instruments. The objectives of this work are to create an extensible platform for the development of medical instruments and to show successful applications of this library on some real medical cases.

2. Existing systems for medical image processing

There are many open–source packages and software systems for working with medical images. Some of them are specifically dedicated for these purposes, others are adapted to be used for medical procedures.

Many of them comprise a set of instruments, dedicated to solving typical tasks, such as images pre–processing and analysis of the results – ITK [1], visualization – VTK [2], real–time pre–processing of images and video – OpenCV [3].

Others solve problems related to image analysis of certain organs or diseases. For example, brain images analysis (FreeSurfer [4], SPM [5] and others). The extension of such software systems for solving a wide range of tasks in medicine is quite complicated or even impossible since most often the architecture of such applications was written for solving a specific task and it may be hard to generalize these approaches.

There are also many general–purpose medical imaging applications. Such systems provide basic functionality for working with images. However, they cannot be expanded to address any specific tasks (for example, segmentation or finding features inherent in certain diseases). Such systems are: Ginkgo CAD [6] and ClearCanvas [7].

Another class of medical software form expandable medical applications that focus primarily on the final usage by the doctors. They already provide all the basic methods in an integrated user interface, for example, Slicer [8], Weasis [9] and OsiriX [10]. The last one is an expensive commercial product and is not available to a wide audience. Such applications can be expanded with specifically written plugins for these platforms. However, this approach does not give the developers enough flexibility to create and adjust their own systems and functionality.

The most generalized and flexible product for working with medical images is MITK [11] – an open–source framework for developing interactive medical software systems. MITK combines the algorithms presented in ITK [1]

with the visualization algorithms from the VTK library [2]. MITK also supplements the functionality of these two libraries with some unique features, allowing its users to create a variety of medical programs from a broad range of functions. While MITK is a cross–platform framework, some versions have not been supported for years. Because it is originally written in C++, it requires to be built separately for each platform. Moreover, the developers have to use a custom build procedure provided by MITK to create and add new modules.

In this paper, we introduce our own open–source medical images research framework (MIRF) as an alternative to existing software systems for medical applications development. MIRF is written in Kotlin programming language with a focus on enabling a smooth integration between modern research in medical imaging. With the Kotlin at its core, MIRF can be smoothly integrated into any projects with Java Environment. We pay close attention to the possibility of integration of artificial intelligence and various machine learning approaches for diagnosis and treatment of various diseases. This is because nowadays the most effective solutions for medical image analysis problems are solved using machine learning or deep learning algorithms [12].

3. MIRF architecture

3.1. Structure

MIRF framework is represented as a collection of generic modules for various tasks. These modules are divided into two global packages:

- Core – the minimum set of necessary modules for the correct operation of the MIRF framework. This package includes modules that are used for transferring data into the internal representation, communication between modules and creating data processing pipelines.
- Features – contains modules with core user functionality that are needed to facilitate development: mechanisms for accessing data storages, adapters for various medical data formats, various pre–processing filters and image analysis tools. Any custom modules should extend the capabilities of this package.

3.2. Pipeline

Execution of any workflows in MIRF is implemented with Pipes & Filters [13] approach. For these purposes, various data handlers should be used to stick individual blocks together.

In the framework, any computational logic must extend the Algorithm interface. The Algorithm is a handler class that, when invoked, changes only the data submitted to it at the input. The Algorithm does not invoke any third–party code associated with data processing. It does not save data and acts solely as a data handler. This approach provides opportunities for flexible creation of algorithms and organization of hierarchies.

Algorithm instances act as filters in our architecture. The Algorithm class is encapsulated by the PipelineBlock class, which is the main entity used to transfer data between algorithms. The communication between the blocks is based on the Observer pattern – after the block executes the algorithm, it informs all its listeners about the completion of the calculations. Some blocks may also be engaged in the aggregation of data for the following blocks or have another specific purpose (for example, they indicate the completion of calculations in the pipeline).

The core architecture of MIRF may be seen at figure 1.

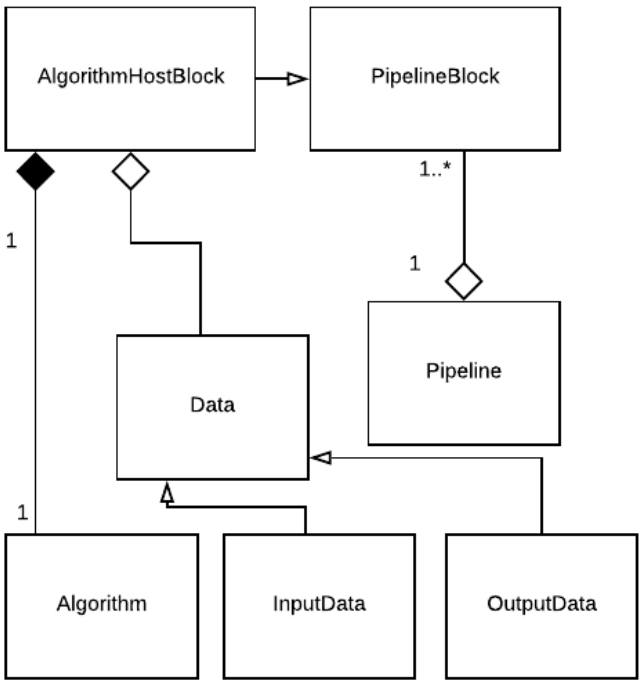


Figure 1. The Core Architecture of MIRF.

3.3. Data representation

Any data in MIRF should be derived from an abstract class Data. The main task of this class is to take over the management of the metadata, namely the list of attributes (AttributeCollection class). Any class inherited from the Data class should be used only as a data storage object. Instances of Data class are passed through the MIRF pipeline and act as Pipes in our Pipes & Filters approach. This ensures the clarity of the entity's purpose within the framework.

3.4. Pipeline initialization

MIRF provides common blocks that may assemble custom pipelines and control the functions that the user wants to be executed on the provided data. The pipeline initialization can be done with a few simple steps:

```
val pipe = Pipeline("Pipeline name")
// Creating the blocks
val firstBlock = PipelineBlock(
                    Block parameters
                 )
...Initialization of other blocks...
// Creating connections between blocks
firstBlock.dataReady +=
    secondBlock::inputReady
...Initialization of other connections...
// Setting the first block and input
pipe.rootBlock = firstBlock
pipe.run(Data)
```

4. Medical images representation

4.1. MedImage

There are several common medical images formats, for example, DICOM [14] or NIfTI [15]. For a unified workflow with these formats and applications of common analysis algorithms, we have implemented a general class for representing medical images in MIRF. MedImage is a class, that contains a list of attributes extracted by certain rules, depending on the source format and the pixel representation of the image. Thus, all algorithms for working with medical images work with the MedImage class, which allows the library user to reuse and extend the existing code.

4.2. DICOM

DICOM format is represented as a set of key–value items, and the image itself is also stored by key, as a value. All sets of keys for DICOM images are strictly defined and are used everywhere by the medical community. To read DICOM images, we considered several libraries for working with this format in Kotlin: ImageJ [16], DCM4CHE [17], and PixelMed [18]. While ImageJ supports the DICOM reading, it does not provide the functionality to output images with this format. DCM4CHE is a rich toolkit for working with DICOM images, it provides a lot of functions to work with those images, using medical servers. Because we don't want to overwhelm our library with unnecessary dependencies, we made our final decision towards PixelMed, which supports reading, working with attributes and writing of DICOM images without complicated workflows such as in DCM4CHE. After reading the list of attributes for a DICOM image MIRF converts it to the MedImage class by creating an internal representation of the attributes and extracting an array of images from the original format.

4.3. NIfTI

Another popular type of medical images is NIfTI [15]. There are a few differences between DICOM and NIfTI file formats, such as the data they store and storage representations. For instance, NIfTI metadata does not include patients or hospital related information. It only stores the image and MRI settings metadata. Also, NIfTI stores a set of medical slices within one file (a set of medical images), while DICOM usually stores them as separate files. To enable NIfTI usage in our framework, we used ImageJ [16]. Then, similarly to DICOM images, we convert the information received from the NIfTI to our internal MedImage representation, to make it possible for the same algorithms to work with different file formats.

5. Unique features

5.1. Tensorflow models integration

Because modern researchers are often using deep learning techniques for solving various problems in medicine, we paid special attention to the possibility of integration of those approaches effortlessly within our framework. We started with the most commonly used deep learning frameworks such as Tensorflow [19] and Keras [20]. As a result, integrating Tensorflow models is possible within MIRF Tensorflow block. Since Tensorflow provides a Java API for working with its models, it was possible for us to create a block which may run the provided models. To run inference on the prepared Tensorflow model, the Tensorflow Block with the models parameters should be instantiated. It is sufficient to pass in the path to the saved model and the names of the input and output nodes.

Also, since Tensorflow package provides Keras interfaces, it is possible to integrate not only Tensorflow models but Keras as well.

To the best of our knowledge, no other software for creating medical applications, provide such integration within its core functionality. We believe that this feature is very important in the modern medical applications development because it completely encapsulates the integration of the complex artificial intelligence models in real medical applications and enables developers to focus on creating new algorithms in their preferred languages and environments.

To use Tensorflow API in C++ or Java developers have to specify the Graph of the model and define many fields before they can run it. However, MIRF users can set up the Tensorflow block within just a few lines:

```
val tensorflowModel = TensorflowModel(
    MODEL_NAME, INPUT_NODE_NAME,
    OUTPUT_NODE_NAME, OUTPUT_DIMS
    )
val tensorflowModelRunner =
AlgorithmHostBlock<Data, Data>(
    {
        tensorflowModel.runModel(
```

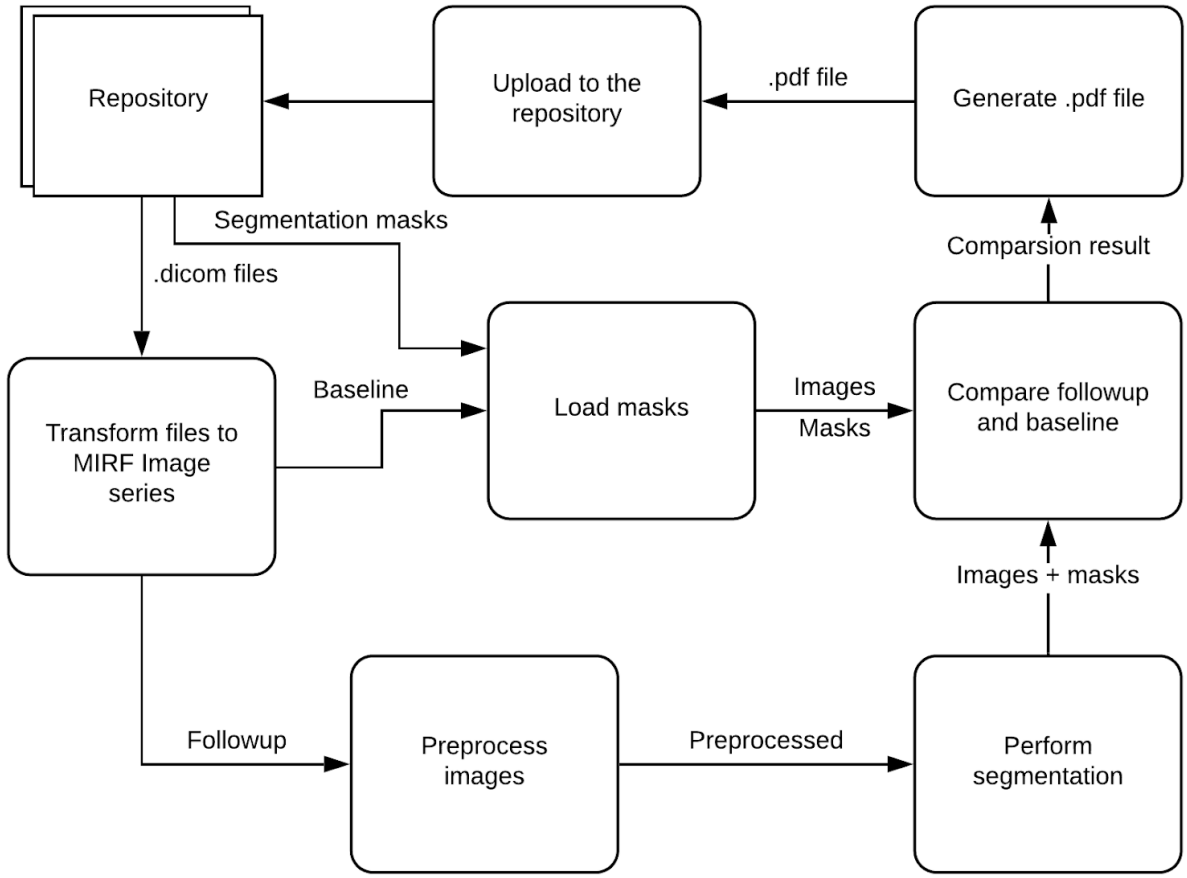


Figure 2. The data flow diagram for the multiple sclerosis analysis pipeline. MIRF reads a set of DICOM images and loads lesion masks from a baseline set. The follow–up set of images is pre-processed and the segmentation masks are calculated. Then, MIRF compares the baseline and follow–up images and generates a report based on this data.

```
            it, INPUT_DIMS)
    },
    pipelineKeeper = pipe
)
```

5.2. PDF reports generation

There are various types of medical documentation that doctors generate after the patients appointment. Those documents usually include CT and MRI images as additions to the final diagnosis paper and recommendation. The outline and contents of medical reports are strictly regulated by the government standards and they vary by different criteria, such as organs, diseases or medical procedures performed. Doctors have to fill in those reports manually or semi-automatically and include images into them. MIRF provides tools for creating these reports automatically, based on the results of the specified pipelines. MIRF generates the report in PDF format and has all the necessary images already included.

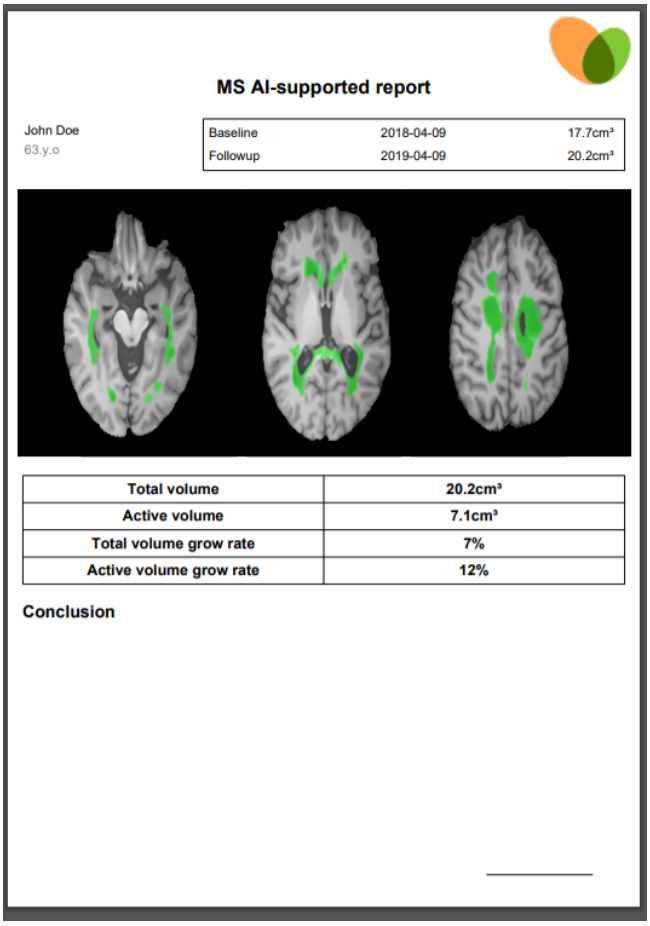
We use algorithm class implementation for this purpose. It generates a report in the form of PdfElementData from input data. The final report is then created by PdfElementsAccumulator class, which takes the sequence of PdfElementData as input and draws them on the document. We use IText 7.1.2 [21] as the main library for working with PDF format.

MIRF provides a set of primitive modules that may be included in the final PDF report. We currently support tables, images, and raw text. If the user needs other instances in his report, he may create his own implementation of PdfElementData and include it in the final report.

6. MIRF applications

6.1. Multiple sclerosis analysis

With our framework, developers may easily create custom pipelines for specific tasks. This automates many manual scenarios and it can bring new features, unused before,



MS AI-supported report

John Doe
63.y.o

Baseline	2018-04-09	17.7cm³
Followup	2019-04-09	20.2cm³

Total volume	20.2cm³
Active volume	7.1cm³
Total volume grow rate	7%
Active volume grow rate	12%

Conclusion

Figure 3. Example of PDF file, that was created with our framework from a multiple sclerosis analysis pipeline.

to doctors workflows. We take Multiple sclerosis analysis as an example of such a workflow.

Multiple sclerosis is an immune–mediated disorder, affecting the central nervous system. Patients with this disease have multiple lesions in the brain. Such patients have to take MRI scans twice a year. Doctors are comparing scans over time and check the growing process of lesions in the brain. They generate the report about this.

We implemented an application, that generates MS reports based on the baseline and follow–up sets of scans. This procedure saves a lot of time for doctors and optimizes their work at several steps.

The data flow diagram for this pipeline may be seen at figure 2. First, it reads a set of DICOM images and loads lesion masks from a baseline set. The follow–up set of images is pre–processed and the segmentation masks are calculated. We use Tensorflow block to perform segmentation on the images. Then, MIRF compares the baseline and follow–up images and generates a report based on this data. An example report for the MS pipeline may be seen at figure 3.

With this application, the segmentation, comparison and report generation is performed automatically for the doctor.

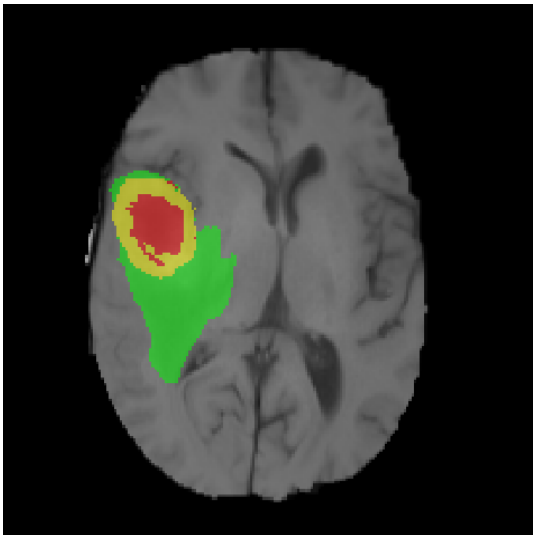


Figure 4. Example of segmented brain image. Different tumor structures are specified as folllowing: edema (green), enhancing core (yellow) and the necrotic core (red).

These steps are usually done manually and require a lot of time.

6.2. Brain tumor analysis

Another example, that shares common functionality with MS analysis is the brain tumor segmentation and a report generation from the obtained information. With this case, we show how MIRF core functions may be used in various scenarios and optimize doctors workflows.

According to [12], brain tumor segmentations are performed either manually or semi–automatically, as well as there is no registered case of bringing the modern research for this problem into real clinical trials. The main information that can be inferred from such segmentation on the early stages of treatment is the tumor volume and its relative volume to the whole patients brain. Hence, these discoveries should be added to the final disease statement. These actions (analyzing MRI scans, calculating the volume and including this information in a report) are performed manually by the specialists. As part of the final tool for working with various medical images, this pipeline may be easily included in our framework. For the brain tumor segmentation, we take an implementation of the state of the art solution of this problem [22]. The algorithm for segmentation is implemented using Tensorflow framework and may be integrated as a model file with our general purpose Tensorflow block, described above. It takes MRI brain images in NIfTI format [15] and creates a mask, indicating different types of tumor tissues, where they are present (figure 4). Since MRI images are represented as a set of slices, where voxels in the slice correspond to some particular volume, it is possible to calculate volume, based on the number of voxels. The information about this encoding is stored in a medical image metadata and depends

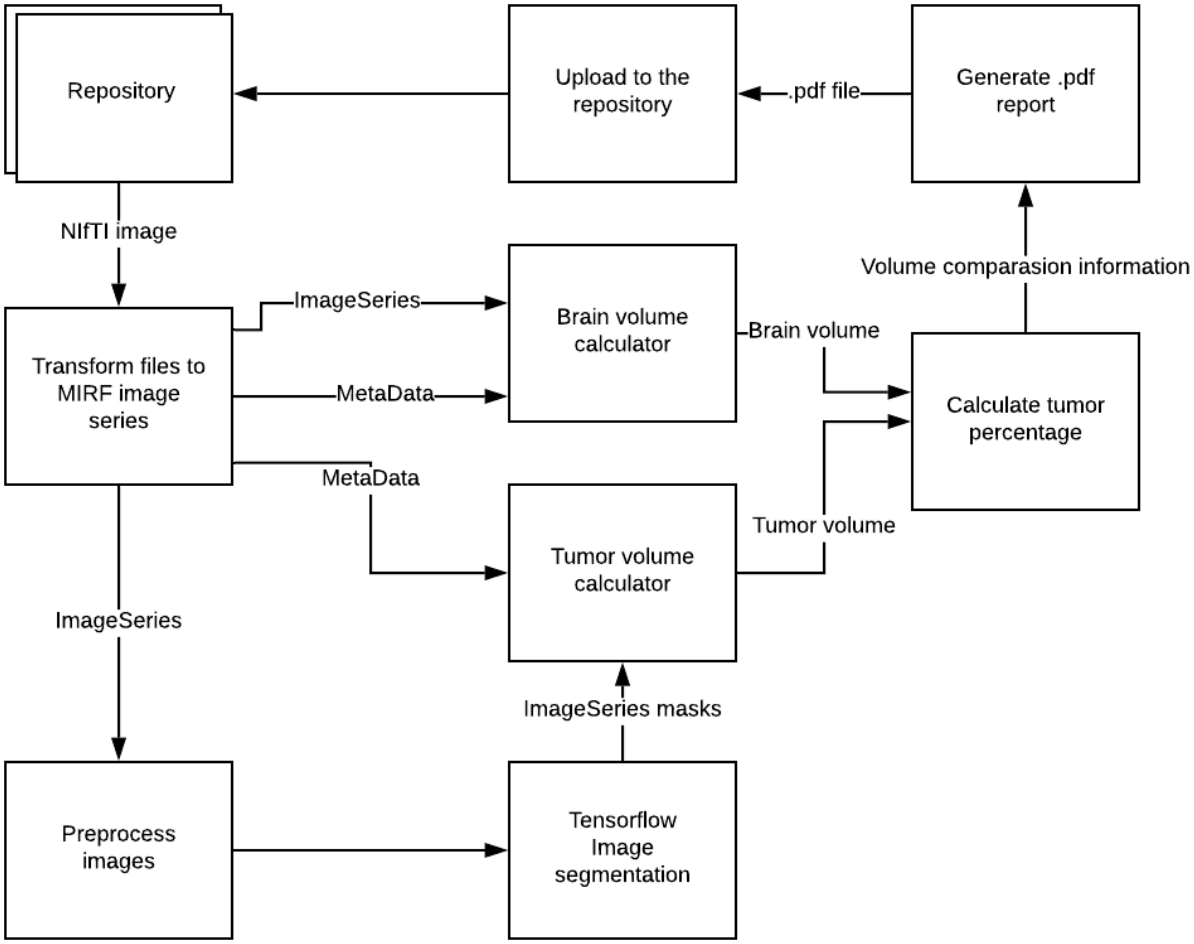


Figure 5. The data workflow and the blocks scheme for the brain tumor segmentation example. MIRF loads a NIfTI image from a local repository and extracts the metadata from raw image data. It pre–processes the image data and sent it to the segmentation block. This block produces a mask with information about detected tumor cells. Then the volume for both tumor and the whole brain is calculated in the corresponding volume blocks. It is essential to pass in the metadata from the NIfTI image to the volume blocks, so the scaling can be inferred. After that, the pdf report from the collected data is generated.

on the MRI machine settings. MIRF calculates the tumor volume based on the segmented mask. Using the initial brain images the whole brain volume may be determined and relativities between those volumes are deduced. Then, MIRF creates a complete report with this information, using PDF generating tools. The data workflow and the blocks used in this pipeline may be seen on figure 5.

This example shares such blocks as image reading, segmentation and report generation with MS analysis workflow. It demonstrates how the core MIRF blocks may facilitate very different workflows from the medical point of view.

6.3. Skin cancer detection on Android

Due to the fact that we develop our framework on Kotlin programming language, it is possible to create not only cross–platform desktop applications but also mobile. This enables developers to deploy the same pipelines and scenarios on a wide range of devices without rewriting any code. To show the benefits of this approach, we created an Android application for skin cancer detection. For image classification, we use an open–source implementation of one of the deep learning algorithms for this problem [23]. It implements the deep learning model with the Keras framework usage. Since it is possible to convert Keras models into pure Tensorflow models, we may use the Tensorflow integration block from MIRF to deploy this model. As a result, we may use the same pipeline as for the desktop app on the phone to detect skin cancer from the phones images. It is required from the developer to write custom layouts on Android for the GUI. We are planning to resolve this issue in the future, by creating a pre–defined library of such

graphical interfaces, so the development of these apps may be performed with more ease and automaticity.

To show the simplicity of our approach, we provide the pipeline code that is used on Android to run this example. It takes an image path as an input and generates a label showing whether the mole is benign or malignant.

```
val pipe = Pipeline("Detect moles")
val assetsBlock =
    AlgorithmHostBlock<Data, AssetsData>
    (...algorithm parameters...)
val imageReader =
    AlgorithmHostBlock<AssetsData,
                       BitmapRawImage>
    (...algorithm parameters...)
val tensorflowModelRunner =
AlgorithmHostBlock<BitmapRawImage,
                   ParametrizedData<Int>>
    (...algorithm parameters...)
val root = PipeStarter()
// Make connections
root.dataReady +=
            assetsBlock::inputReady
assetsBlock.dataReady +=
            imageReader::inputReady
imageReader.dataReady +=
    tensorflowModelRunner::inputReady
// Run
pipe.rootBlock = root
pipe.run(MirfData.empty)
```

7. Conclusion

In this paper, we introduced the Medical Images Research Framework for the development of complex medical applications with various types of medical images. We investigated the existent solutions in this area and argued why we created such a tool.

We introduced the basic overview of the proposed architecture and its benefits in this paper, as well as the unique features of our platform. We believe that our framework will help in mending the gap between innovative research made in medical images analysis and delivering it to the final users. As our research is still early in development, we have many plans for further integration, such as adding most commonly used features (scales, segmentation masks, zooming, working with patients data) and GUI for them. We also plan on creating a visual programming environment based on our framework, so creating medical apps would be possible for people with little programming experience.

Out project is publically available and may be found at https://github.com/MathAndMedLab/Medical-images-research-framework

References

[1] Itk. [Online]. Available: http://www.itk.org [Accessed: 17.12.2018].

[2] Vtk. [Online]. Available: http://www.vtk.org [Accessed: 17.12.2018].

[3] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*, 2nd ed. O'Reilly Media, Inc., 2013.

[4] Freesurfer. [Online]. Available: http://surfer.nmr.mgh.harvard.edu [Accessed: 17.12.2018].

[5] Spm. [Online]. Available: http://www.fil.ion.ucl.ac.uk/spm/ [Accessed: 17.12.2018].

[6] Ginkgo cad. [Online]. Available: https://github.com/gerddie/ginkgocadx [Accessed: 17.12.2018].

[7] Clearcanvas. [Online]. Available: https://www.clearcanvas.ca/ [Accessed: 17.12.2018].

[8] S. Pieper, M. Halle, and R. Kikinis, "3d slicer," in *2004 2nd IEEE International Symposium on Biomedical Imaging: Nano to Macro (IEEE Cat No. 04EX821)*, April 2004, pp. 632–635 Vol. 1.

[9] Weasis. [Online]. Available: http://nroduit.github.io/en/ [Accessed: 17.12.2018].

[10] A. Rosset, L. Spadola, and O. Ratib, "Osirix: An open-source software for navigating in multidimensional dicom images," *Journal of digital imaging : the official journal of the Society for Computer Applications in Radiology*, vol. 17, pp. 205–16, 10 2004.

[11] M. Nolden, S. Zelzer *et al.*, "The medical imaging interaction toolkit: challenges and advances," *International journal of computer assisted radiology and surgery*, vol. 8, 04 2013.

[12] M. S.A., L. A.V. *et al.*, "Medical images segmentation operations," *Trudy ISP RAN/Proc. ISP RAS*, vol. 30, pp. 183–194, 2018.

[13] A. H. Vermeulen, G. Beged-Dov, and P. W. Thompson, "The pipeline design pattern," in *OOPSLA 1995*, 1995.

[14] W. D. Bidgood, S. C. Horii *et al.*, *Understanding and Using DICOM, The Data Interchange Standard for Biomedical Imaging*. Boston, MA: Springer US, 1998, pp. 25–52.

[15] V. Patel, I. D Dinov *et al.*, "Loni mind: metadata in nifti for dwi," *NeuroImage*, vol. 51, pp. 665–76, 03 2010.

[16] Imagej. [Online]. Available: https://imagej.nih.gov/ij/index.html [Accessed: 17.12.2018].

[17] Dcm4che. [Online]. Available: https://www.dcm4che.org [Accessed: 17.12.2018].

[18] Pixelmed. [Online]. Available: http://www.pixelmed.com/dicomtoolkit.html [Accessed: 17.12.2018].

[19] M. Abadi, A. Agarwal *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[20] Keras. [Online]. Available: https://github.com/fchollet/keras [Accessed: 17.12.2018].

[21] Itext. [Online]. Available: https://itextpdf.com/en [Accessed: 07.02.2019].

[22] G. Wang, W. Li *et al.*, "Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks," *CoRR*, vol. abs/1709.00382, 2017. [Online]. Available: http://arxiv.org/abs/1709.00382

[23] Skin cancer detection project. [Online]. Available: https://github.com/dasoto/skincancer [Accessed: 10.02.2019].