MIRF 2.0 — a framework for distributed medical images analysis

Alexandra Shvyrkova
St. Petersburg State University
Saint Petersburg, Russia
shvyrkova.s@gmail.com

Alexey Fefelov
St. Petersburg State University
Saint Petersburg, Russia
fefaleksey@gmail.com

Yurii Litvinov
St. Petersburg State University
Saint Petersburg, Russia
y.litvinov@spbu.ru

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

Egor Ponomarev
St. Petersburg State University
Saint Petersburg, Russia
egorponomarev93@gmail.com

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

Alexander Savelev
St. Petersburg State University
Saint Petersburg, Russia
algsavelev@gmail.com

***Abstract*—MIRF is an open-source library for a convenient creation of applications which process medical data. Architectural style of that library is Pipes and Filters which means that components that process and transform data are connected together in the pipeline. In this paper we describe a new version of the library based on microservice architecture where services are deployed and maintained independently. This could solve a problem with a lack of computational resources needed for image processing. Some new applications of MIRF library are also presented, namely ECG arrhythmias diagnostics and intracranial hemorrhage detection. Implementation of the ECG processing in MIRF was especially interesting due to the fact that ECG is a signal, not an image. Detailed description of ECG processing tool and results of our experiments are also presented.**

***Index Terms*—medical images, microservices, electrocardiogram, convolutional neural network**

Introduction

Medical Images Research Framework (MIRF, [1]) is an open-source platform for rapid development of applications for medical images processing. The project aims to fill the gap between research in automated medical images processing and real-world medical practice, allowing medical specialists with basic programming skills to quickly experiment with computer vision and machine learning algorithms. Research in medical images processing is growing exponentially in recent years, there are many existing libraries, such as MITK[1], international competitions in medical image analysis, such as BraTS challenge[2], conferences (MICCAI[3] and topics in CVPR[4], ICLR[5]). And there is a very pragmatic reason for such an interest in this field: the most interesting medical images are MRI or CT scans, each scan consists of several dozens of slices, and a radiologist should find an anomaly that could be visible only in a very few slices, and should process several slices in a day. It is a great amount of work and the lives of patients depend on it. Thus semi-automatic processing (for example, flagging "interesting" slices for future review) can be crucial. And yet, in a real-world clinics advanced automatic processing is the exception rather than the rule.

MIRF project was started in 2018 as an informal collaboration between students of St. Petersburg State University and radiologists of one of the private clinics in St. Petersburg. It was designed as a framework written in Kotlin for a development of desktop and mobile applications for processing DICOM and NifTI scans. It is integrated with TensorFlow, thus allowing to use neural networks, has its own utilities for DICOM and NifTI processing, can generate PDF reports [1]. Although it quickly became obvious that desktop and mobile applications can not be very effective in a real-world radiology due to the problems with data storage and amount of computational resources needed for 3D images processing. As a second version MIRF was redesigned to use microservices architectural style, to be distributed, scalable and web-oriented. Thanks to this, users do not need to install MIRF on their computer, they can just use the web service, and we can manage the load on the servers.

This article presents our experience in creating a new distributed architecture of Medical Images Research Framework

[1]The Medical Imaging Interaction Toolkit (MITK), URL: http://mitk.org/wiki/MITK (accessed: 11.02.2020)
[2]Multimodal Brain Tumor Segmentation Challenge 2019, URL:http://braintumorsegmentation.org/ (accessed: 11.02.2020)
[3]MICCAI home page, URL: http://www.miccai.org/ (accessed: 16.02.2020)
[4]CVPR home page, URL: http://cvpr2020.thecvf.com/ (accessed: 16.02.2020)
[5]ICLR home page, URL: https://iclr.cc/ (accessed: 16.02.2020)

and a several new applications including automated diagnostic of cardiovascular diseases by electrocardiogram and intracranial hemorrhage detection. Electrocardiogram processing is especially interesting because electrocardiogram is not actually an image, so a successful electrocardiogram diagnostic application shows that MIRF is able to work effectively with more general medical data (despite "Images" in its name). As most of the medical tasks, it is an extremely challenging problem, solving which c help save many lives.

I. MIRF 2.0 ARCHITECTURE

MIRF is divided into 2 global packages — *core* and *features*.

- *core* contains the basic types of libraries: abstract classes and interfaces representing blocks, filters and the base classes for medical data — image series, ECG series.
- *features* contains the different MIRF functionality — API to access the repositories, reporting tools, various data formats parsers, image series segmentation and visualization tools

Data in MIRF is represented as a child classes of the abstract class *Data*. The main objective of the class is to provide convenient access to the metadata stored as list of attributes (the *AttributeCollection* class in MIRF), as well as to ensure pipeline integrity — only *Data*-derived classes are to transfer via pipelines.

Computational elements are presented as the implementations of the *Algorithm* functional interface. It is expected that *Algorithm* would be implemented only with the pure functions. Such a reduction of possible implementations along with the fact that the algorithm can be easily created from one method (using the *SimpleAlg* class) provides opportunities for flexible algorithm and hierarchies creation. A typical example is static classes containing handler methods for data of one types, each of which can be converted into an algorithm instead class institutions for each method.

The main purpose of the MIRF library is to create pipelines — series of data handlers. In the terminology of Pipes & Filters architecture, the filters are instances of *Algorithm* encapsulated in *PipelineBlock* — a class designed to connect *Algorithms* among themselves. Data transfer between the blocks is based on event-oriented model, and *PipelineBlocks* implement Observer pattern. Some blocks are used only for linking algorithms (such as the *AlgorithmHostBlock* class), whereas others may also be used for the data aggregation or have a specific purpose. So, for example, the *AccumulatorBlock* class can subscribe to several blocks and signal the result only if the results of all input blocks are ready, but *ConsumerBlock* does not allow subscribe to itself and serves as a pipeline terminator.

To improve performance and to ensure the possibility of using our system by an unlimited number of people, as well as to save users from the need to install the library on a local computer, we have developed and implemented a microservice architecture. Each Block of MIRF is implemented as a microservice. This allows to flexibly add, remove blocks, change the network configuration, track performance issues. The whole system consists of a set of *Blocks*, *Repository* and *Orchestrator*.

A. Block

Block is a microservice that supports one or more MIRF blocks (a microservice wrapper for *PipelineBlock* class). Blocks do not have an internal state and continuously process incoming data using contained *Algorithm* objects.

B. Repository

Repository is a shared (possibly distributed) database that provides a convenient access to data. This entity is necessary because some blocks require more than one data set (for example, the result of processing images in two projections) and the microservices that support such blocks do not have to store them locally and wait until the data is ready. They just store data in the repository and start processing other data sets if there is at least one such ready-made set, or wait until the complete data set appears in the repository.

C. Orchestrator

Orchestrator is a microservice that distributes the load between other microservices. It receives data and specifications from users and sends them to the repository. Specification is a connected acyclic graph that contains information about data processing. A graph always has one input vertex and one output vertex. Graph nodes are the types of blocks that should process data at the current stage. As soon as the repository receives data, the Orchestrator asks the blocks for status, looks which block is the least loaded and sends it a message that the data is loaded and ready.

D. Online medical files storage

Online storage is used when there is a large amount of data which cannot be stored on the local machine. For such purpose server for storing medical files and supporting Kotlin library was created.

Server: there are several realizations which could be used as a server. The first one is to create your own server which stores files. The reason why it is not suitable is because there are medicine oriented PACS[6] which has its own server realizations. On the other hand, you can use NAS[7]. It has possibilities to use compression while transferring files (for example, JPEG 2000[8]) and more advanced network protocol therefore, data transfer speed may increase significantly [23], but installation complexity and configuration increases, since there are no ready-made NAS solutions. Hence decision to use PACS server was made.

[6]Picture Archiving and Communication System, URL: https://en.wikipedia.org/wiki/Picture_archiving_and_communication_system (accessed: 11.02.2020)

[7]Network Attached Storage, URL: https://en.wikipedia.org/wiki/Network-attached_storage (accessed: 11.02.2020)

[8]JPEG 2000, URL: https://en.wikipedia.org/wiki/JPEG_2000 (accessed: 11.02.2020)

Library: as MIRF is written in the Kotlin programming language[9], the goal was to use capabilities of the language and the libraries written in the language to achieve maximum usability. For network communication, choice of libraries was between those written in Java (Kotlin is free to use with such libraries) or in Kotlin. The second option was picked, since it uses the "coroutine" paradigm[10], which is not supported in Java libraries and significantly simplifies writing asynchronous code. At the moment, there are not a lot of alternatives, as a result, the ktor[11] library was used, due to the fact that it is written and supported by the developers of the Kotlin language, therefore it is stable and well documented. Server responds to requests with JSON objects, so the kotlinx.serialization library was selected for serialization/deserialization, because it is lightweight (485 KB), at the same time provides minimal necessary functionality, and is part of the extended standard library of the language. In the library two abstractions are used: first one represents local file which can be processed or uploaded to the server (so called MedImage) and remote file which can be downloaded or be processed on the server. Now work on remote files processing is underway.

II. Electrocardiogram processing

At the time of creation, MIRF worked only with images. Nevertheless we thought that its functionality could be expanded to solve problems related to electrocardiogram signal processing. The electrocardiogram itself is a record of electrical currents generated by heart. These currents are measured with electrodes placed on the surface of the different parts of body. Potential difference between electrodes is called a Lead. In a heartbeat of a healthy person QRS-complex could be defined as well as many different waves, segments and intervals. The morphology of ECG signal is illustrated in Fig. 1.

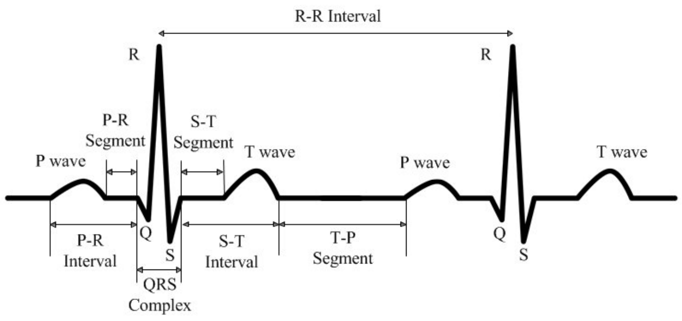

Fig. 1. ECG signal morphology [2]

ECG data preprocessing and classification could be done in several consistent steps. Each of them then would be wrapped in a separate MIRF block. Pipes and Filters architecture gives us an opportunity to combine these blocks, which serve for different purposes, in a one ECG-processing pipeline.

A. Database

Two biggest public datasets with annotated ECG records are PTB Diagnostic ECG Database [3] and MIT-BIH Arrhythmia Database [4], [5]. MIT-BIH Arrhythmia database contains records with different types of arrhythmias whereas PTB database serves as a Myocardial infarction database where most of the records belong to one class. That is why we have chosen MIT-BIH database. It includes 48 ECG recordings with 30-minute duration. Three files are associated with every patient: header file, file with two-lead raw signals and annotation file with heartbeat classifications from doctors.

In order to upload electrocardiogram data in MIRF we first need to decode it. Signals have 11-bit resolution and 360Hz sampling rate. Header file contains all necessary information required for correct data decoding such as checksums and initial values. ECG decoding will be one of the blocks in our pipeline.

B. Denoising

As an electrical signal, ECG is distorted by different kinds of noises such as the Gaussian white noise and baseline wander noise.

White noise is a sequence of independent numbers with constant spectral density over all real values. ECG signals are usually contaminated with Gaussian white noise which has a normal distribution with zero mean. This kind of noise is caused by muscle contractions.

ECG is a non-stationary signal, therefore standard filtering methods could not be used (e.g., Fourier Transform). Commonly used methods for removing white noise from ECG are adaptive filtering, discrete wavelet transform (DWT), Savitzky-Golay filtering [6]. In a comparative study of mentioned methods the best results are obtained while using DWT [6].

Wavelets are functions that are localized in time and frequency domain. Discrete wavelet transform is based on a signal representation as a linear combination of dilated and shifted versions of mother wavelet.

Denoising with the help of DWT is performed in several steps. First step is decomposition of a signal which gives us approximate and detail coefficients. Approximation coefficients represent low frequency parts, i.e., main features of a given signal. Some of the coefficients are being removed or scaled down using threshold filters. After that step the filtered signal is reconstructed from new coefficients [7]. Symlet and Daubechies mother wavelets are usually used because their waveforms resemble the form of a QRS-complex which means that ECG-signal could be well represented by them. Symlet was chosen as a mother function considering good results in papers [8], [6]. Soft thresholding function is used for noise reduction. It sets to zero detail coefficients with absolute values less than a threshold, or reduces the coefficient by threshold otherwise. These parameters are used in most cases

[9]Kotlin official documentation, URL: https://kotlinlang.org (accessed: 11.02.2020)

[10]Kotlin Coroutines official documentation, URL: https://kotlinlang.org/docs/reference/coroutines/coroutines-guide.html (accessed: 11.02.2020)

[11]Ktor official documentation, URL: https://ktor.io (accessed: 11.02.2020)

of white noise removal in ECG and they give good results in [6], [8], [9].

Another kind of noise is baseline wander. It is caused by respiration or body movements. The method using wavelets proposed in [10] was chosen since it preserves important clinical information. This algorithm is based on the idea that baseline wander and pure ECG signal are independent parts of the ECG signal. Baseline wander could be extracted using discrete wavelet transform. The level of decomposition is calculated using the preset cut-off frequency. Fig. 2 and Fig. 3 show noise cancellation tested on the record 100 from MIT-BIH database.

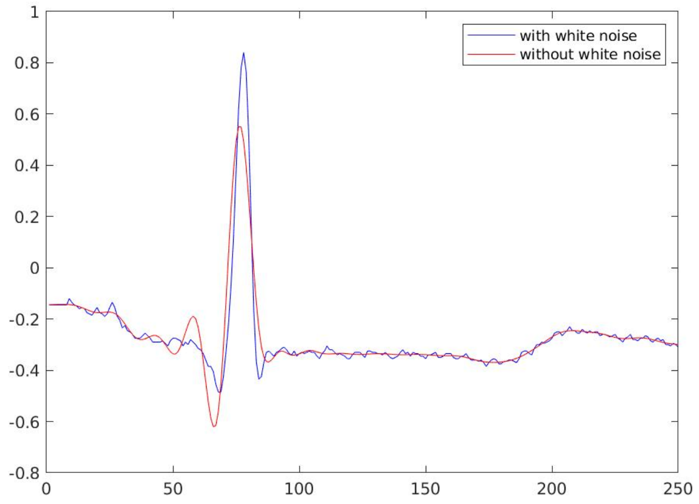

Fig. 2. Result of white noise cancellation.

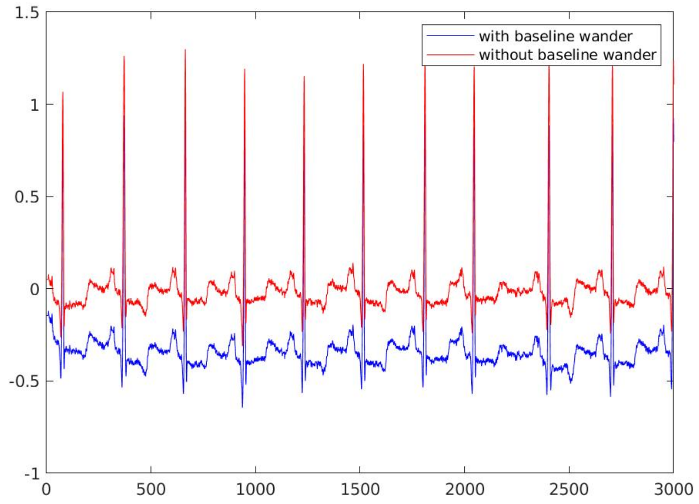

Fig. 3. Result of baseline wander cancellation.

C. *Arrhythmia classification*

Many different algorithms have been proposed to classify abnormalities in ECG records. They could be divided in two classes.

Algorithms from the first class are based on extraction of morphological features such as amplitudes, segment and interval lengths. These features are then used as an input to the classification algorithm. As a classification method we can use k-nearest members method [11], method based on linear discriminant analysis [12] or support vector machine [13].

Algorithms from the second class work with images obtained from raw digital data. One of the most commonly used technique for ECG image classification are convolutional neural networks (CNN) as they are able to learn spatial hierarchies of patterns. Researches made in [14], [15] show that CNN has an advantage over other classification algorithms. For this reason a model proposed in [14] was implemented.

MIT-BIH database consists of records with 14 different types of abnormalities. Healthy heartbeats and heartbeats with 7 clinically significant types of arrhythmias were extracted from MIT-BIH database. Each chosen heartbeat belongs to one of 8 classes: normal beat, right bundle branch block beat, left bundle branch block beat, premature ventricular contraction beat, paced beat, atrial premature contraction beat, ventricular flutter wave beat, and ventricular escape beat. In database annotations they are shortened to NOR, RBB, LBB, PVC, PAB, APC, VFW and VEB respectively. We obtained 100852 distinct heartbeat images from MIT-BIH database.

Due to the imbalanced distribution across classes when more than 50% of data belongs to NOR class, images that belong to arrhythmias need to be augmented. Nine cropping methods are applied and all images are resized to 128×128 before feeding them to a model.

The model has 11 Conv2D, MaxPooling and Dense layers. The model summary could be seen in Table I. To avoid overfitting batch normalization layers and dropout regularization layers are added. Xavier initializer is used to set initial random weights to layers. According to the original publication, ELU activation function showed better results compared to ReLU and LReLU. Cross-entropy function serves as loss function and optimized with Adam.

To evaluate model after training special cases could be defined: true positive (TP)–model correctly detects arrhythmia, false positive (FP)–model detects arrhythmia in a healthy ECG, true negative(TN)–model does not detect arrhythmia in a healthy ECG, false negative(FN)–model incorrectly classifies ECG with arrhythmia.

Certain characteristics based on these cases exist to check if the diagnosis tool produces good results. Such metrics are accuracy, sensitivity, specificity and positive predictive value. They are defined below:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} * 100\%$$

$$\text{Specificity} = \frac{TN}{FP+TN} * 100\%$$

$$\text{Sensitivity} = \frac{TP}{FN+TP} * 100\%$$

$$\text{Positive Predictive Value} = \frac{TP}{TP+FP} * 100\%$$

Using these metrics the model could be evaluated and compared to the other classification solutions. The comparison

TABLE I
MODEL ARCHITECTURE

	Type of layer	Output shape
0	Input	(128, 128, 1)
1	Conv2D ELU Batch normalization	(128, 128, 64)
2	Conv2D ELU Batch normalization	(128, 128, 64)
3	MaxPooling	(64, 64, 64)
4	Conv2D ELU Batch normalization	(64, 64, 128)
5	Conv2D ELU Batch normalization	(64, 64, 128)
6	MaxPooling	(32, 32, 128)
7	Conv2D ELU Batch normalization	(32, 32, 256)
8	Conv2D ELU Batch normalization	(32, 32, 256)
9	MaxPooling	(16, 16, 256)
10	Dense ELU Batch normalization Dropout	(2048)
11	Dense softmax	(8)

could be seen in Table II.

The model does not reach the classification performance of other solutions. Nevertheless, the results are considered to be satisfying since the aim of this paper is to show that MIRF is a convenient tool for a quick medical application development.

The model was implemented in Python language using open-source library Keras[12]. For training the network NVIDIA Tesla P100 GPU with CUDA 10.1 was used.

TABLE II
SOLUTIONS COMPARISON

Classifier	Accuracy	Specificity	Sensitivity	PPV
Proposed	95.6	94.3	**98.7**	87.7
Acharya et al. [16]	93.4	91.6	96.0	**97.8**
Kiranyaz et al. [17]	**99**	98.9	93.9	90.6
Jiang and Kong [18]	98.8	**99.4**	94.3	95.8

The pipeline that processes ECG in MIRF is illustrated with Fig. 4. Custom functions are wrapped in AlgorithmHostBlocks which are later connected to each other.

III. INTRACRANIAL HEMORRHAGE DETECTION

Medical image analysis competitions have gained great popularity among researchers in the field of machine learning. A large number of people take part in competitions and discuss their results. Our team have decided to integrate the best solutions from the competition into MIRF. With this approach, the library would be expanded with new blocks. What is more,

[12]Keras, URL: https://github.com/fchollet/keras (accessed: 10.11.2019)

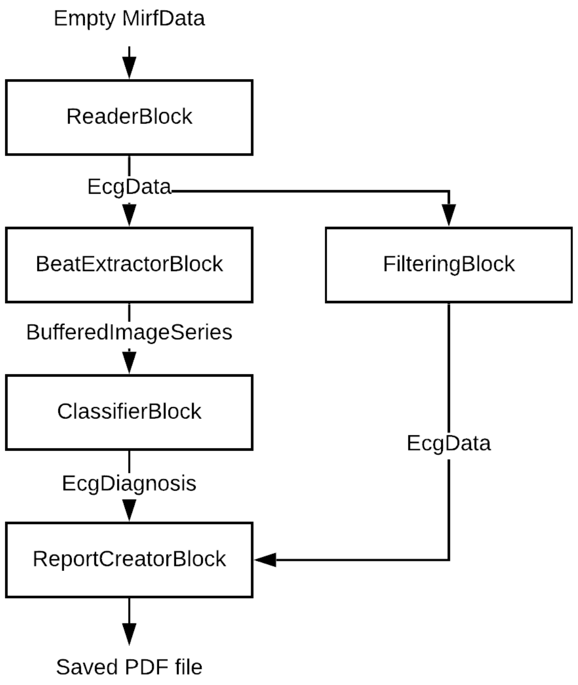

Fig. 4. ECG processing pipeline in MIRF

it could draw attention of other researchers and users interested in this topic.

There is a large number of venues where data analysis contests are held. The most popular of them are: Kaggle, Codalab, CrowdAI. Different venues differ from each other in the way they present data and the prize pool which greatly affects the number of participants and the level of competition.

It was decided to add CNN based solution from RSNA Intracranial Hemorrhage Detection because this competition became one of the most popular competitions on Kaggle in 2019. The purpose of this competition is to classify intracranial hemorrhage into 6 categories: epidural, intraparenchymal, intraventricular, subarachnoid, subdural and any other hemorrhage, including its absence. Labeled data for training and testing was provided by the organization comitee. The success of the solution was evaluated by the LogLoss metric [19].

At the moment, our team is developing a block at MIRF which implements the solution from the competition.

IV. RELATED WORK

There are some other systems that allow to create medical imaging processing applications and build research and clinical software prototypes. The main criteria for the comparisons of these systems is a support for different medical imaging types, tool functionality, languages and type of platform to run in. Also some of the medical imaging processing tools are open source or have free version to be used non-commercially, while others have only commercial versions. For instance, the Medical Imaging Interaction Toolkit (MITK) is a software system for development of an interactive medical image processing software. MITK framework is a software tool that combines the Insight Toolkit (ITK) and the Visualization Toolkit (VTK) [13]. One of the main disadvantages of MITK is that tools are built separately for each platform. NiftyRec could also be mentioned. It is one of the projects developed at University College London. NiftyRec is a software for tomographic reconstruction, providing the fastest GPU-accelerated reconstruction tools for emission and transmission computed tomography[14]. Besides that NiftyRec works with only brain tomography, it has only Matlab and Python interfaces. Another existing tool is Slicer [15] which is a software modular platform for medical image processing, and three-dimensional visualization. Slicer can be expanded for necessary task with specifically written plugins for these platforms, but this approach does not give the developers enough flexibility to create and adjust their own systems and functionality.

V. Future work

Our future research directions will be focused on improvement of existing functionality and adding new features for a quick and efficient development. By means of new blocks creation, we will provide an ability to work with text data and histology images. These additions are related to the clinical practice guidelines that consist of recommendations for optimizing patient care, and according to that doctor should take into account all patient data at diagnosis, including laboratory tests and histological techniques. Likewise, one of the most important goals of our future research is to make machine learning algorithms accessible to the doctors and medical researchers. It would save their time in routine operations such as compiling a report and would give an opportunity to compare doctor's diagnosis with a machine learning tool prediction. At the beginning of our work with MIRF framework, our main advisors among the medical community were radiologists. A radiologist is a physician who obtains and interprets medical images directly. Thereby, radiology was the first medical specialization to computerize and, historically, radiologists use wide range of software tools. For these reasons, most radiologists differ from the other medical specialists, so far as having technical skills such as programming. Working on the tasks of the ECG analysis, we collaborated with cardiologists and had completely realized that most doctors do not have any previous programming experience and do not have time to get it. The actual challenge for us is to make MIRF framework available for an average doctor who does not have any coding experience. With that in mind, we directed our attention to visual programming technology, that allows non-developers to build applications. One of advantages of visual languages is that skills in writing code are not needed. Syntax errors are most common types of errors made by programmers. These types of errors are avoided with block-based approach. Doctors will be able to create tools based on machine learning algorithms by dragging icons and fitting them together.

The block-based approach of visual programming gained popularity while introducing programming to kids. But recently, the largest technology company and research groups have been releasing graphical and drag and drop development tools for different tasks, including machine learning (e.g., Watson Studio by IBM, Microsoft drag and drop machine learning tool, MIT project Northstar [20], University of Copenhagen [21]).

MIRF framework is evolving in the ecosystem of Software Engineering Department, we are well acquainted with the REAL.NET research project. REAL.NET framework uses multilevel metamodeling approach for fast creation of graphical programming tool which includes new language, editor and code generator for wide range of tasks and areas [22]. Through cooperation with REAL.NET team we plan to make convenient and highly specialized visual platform for creating medical applications based on machine learning algorithms. Hence, this will allow more doctors and medical researchers to improve daily clinical practice and to exploit new ideas by a way of prototyping new tool within a few hours and ensure smooth integration of it into the doctor's work environment.

Conclusion

In this paper we described additions and improvements made to the MIRF library.

Firstly, we moved MIRF to microservices. A system of Blocks, Repository and Orchestrator, where each Block is implemented as a microservice, was developed. Thus, we aimed to improve performance and made it easier to add, remove blocks, change the network configuration and track performance issues. However, performance experiments have not yet been conducted. This is an important area of further work.

Secondly, we examined storage implementation variants and selected the best one — PACS. Basic repository access library functionality was implemented. PACS server gave us opportunity to conveniently process, upload and download medical data to our server. It would also allow us to process files remotely in the future.

We also added new medical functionality to MIRF and showed that this library is capable of processing medical data apart from images. We investigated various techniques which are used for ECG processing and chose the ones that showed the best results. Algorithms for denoising and decoding were implemented and a convolutional neural network for arrhythmia classification was trained.

Finally, our plans for future MIRF development were described such as adding blocks for processing histology images. We also want our tools to be used by doctors in real clinical practice. For that purpose we are planning to develop visual language which will allow to create tools for medical data processing by dragging icons and fitting them together.

[13]The Medical Imaging Interaction Toolkit (MITK), URL: http://mitk.org/wiki/The_Medical_Imaging_Interaction_Toolkit_(MITK) (accessed: 09.02.2020)

[14]NiftyRec 2.0, URL: http://niftyrec.scienceontheweb.net/wordpress/ (accessed: 09.02.2020)

[15]3D Slicer, URL: https://www.slicer.org/ (accessed: 09.02.2020)

References

[1] S. Musatian, A. Lomakin, A. Chizhova, "Medical images research framework" in CEUR Workshop Proceedings, Volume 2372 (4th Conference on Software Engineering and Information Management, SEIM 2019), Apr. 2019, pp. 60–66.

[2] T. N. Gia et al. "Fog computing in body sensor networks: An energy efficient approach" in IEEE International Body Sensor Networks Conference, pp. 1–7, January 2015.

[3] R. Bousseljot, D. Kreiseler, A. Schnabel, (1995). "Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet" in Biomedical Engineering, Vol. 40(s1), Jan 1, 1995, pp. 317–318.

[4] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," IEEE Engineering in Medicine and Biology Magazine, Volume 20, no. 3, pp. 45–50, May-June 2001.

[5] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. Circulation", Volume 101, no. 23, pp. e215–e220, 2000.

[6] A. Mohammed, R. H. Bryan, "Performance Study of Different Denoising Methods for ECG Signals" in Procedia Computer Science, 2014, Volume 37, pp. 325–332.

[7] M. Stéphane, "A Wavelet Tour of Signal Processing (Third Edition)" in Academic Press, 2009, pp. 89–153.

[8] M. Aqil, A. Jbari, A. Bourouhou, "ECG Signal Denoising by Discrete Wavelet Transform" in International Journal of Online Engineering, 2017, p. 51.

[9] A. Fedotov, "Myographic Interference Filtering from ECG Signals Using Multiresolution Wavelet Transform" in Biomedical Engineering, January 2019.

[10] B. Mozaffary, M. A. Tinati, "ECG Baseline Wander Elimination using Wavelet Packets" in World Academy of Science, Engineering and Technology, 2005, pp. 14-16.

[11] J. Park, K. Lee, K. Kang, "Arrhythmia detection from heartbeat using k-nearest neighbor classifier" in IEEE International Conference on Bioinformatics and Biomedicine, 2013, pp. 15–22.

[12] P. DeChazal, M. O'Dwyer, R. B. Reilly, "Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features" in IEEE Transactions on Biomedical Engineering, Vol. 51, Issue: 7, July 2004, pp. 1196–1206

[13] J.A. Nasiri, M. Naghibzadeh, H.S. Yazdi, "ECG arrhythmia classification with support vector machines and genetic algorithm", UKSim European Symposium on Computer Modeling and Simulation, 2009, pp.187–192

[14] T. J. Jun, H. M. Nguyen, D. Kang, D. Kim, D. Kim, Y.-H. Kim, "ECG arrhythmia classification using a 2-D convolutional neural network", 2018. [Online]. Available: https://arxiv.org/pdf/1804.06812.pdf [Accessed: 09.02.2020]

[15] J. Liu, S. Song, G. Sun, Y. Fu, "Classification of ECG Arrhythmia Using CNN, SVM and LDA" in Artificial Intelligence and Security, 2019, pp. 191–201.

[16] U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, M. Adam, A. Gertych, R. S. Tan, "A deep convolutional neural network model to classify heartbeats" in Computers in Biology and Medicine, 2017, pp. 389–396.

[17] S. Kiranyaz, T. Ince, M. Gabbouj, "Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks" in IEEE Transactions on Biomedical Engineering, 2016, Vol. 63(3), pp. 664–675.

[18] Wei Jiang, Seong Kong, "Block-Based Neural Networks for Personalized ECG Signal Classification" in IEEE Transactions on Neural Networks, 2007, Vol. 18(6), pp. 1750–1761.

[19] V. Vovk, "The fundamental nature of the log loss function", 2015

[20] T. Kraska, "Northstar: an interactive data science system" in Proceedings of the VLDB Endowment, Vol. 11, Issue 12, Aug. 2018

[21] S. Tamilselvam, N. Panwar, S. Khare, R. Aralikatte, A. Sankaran, M. Kumarasamy, K. Senthil, "A Visual Programming Paradigm for Abstract Deep Learning Model Development". [Online]. Available: https://arxiv.org/pdf/1905.02486.pdf [Accessed: 09.02.2020].

[22] Kuzmina, E., Litvinov, Y. "Implementation of "smart greenhouse"" in CEUR Workshop Proceedings, Volume 2372 (4th Conference on Software Engineering and Information Management, SEIM 2019), Apr. 2019, pp. 67–73.

[23] Veeramani, S. and Masood, M. and Sidhu, A. 2014. "A PACS alternative for transmitting DICOM images in a high latency environment", IEEE Conference on Biomedical Engineering and Sciences, pp. 975-978. Kuala Lumpur: IEEE. [Online]. Available: https://espace.curtin.edu.au/handle/20.500.11937/45899 [Accessed: 09.02.2020].