Нижегородский государственный университет им. Н.И. Лобачевского  
Институт информационных технологий, математики и механики  
Кафедра Математического обеспечения и суперкомпьютерных технологий

**Архипелаг 20.35**

**Трек «ИИ в здравоохранении»**

**Лабораторная работа №2  
Сегментация ЭКГ с помощью OpenVINO**

*При поддержке компании Intel*

*Васильев Е.П.*

Нижний Новгород  
2020

**Содержание**

[1 Введение 3](#_Toc55808898)

[2 Методические указания 3](#_Toc55808899)

[2.1 Цели и задачи работы 3](#_Toc55808900)

[2.2 Структура работы 3](#_Toc55808901)

[2.3 Рекомендации по проведению занятий 3](#_Toc55808902)

[3 Запуск примеров и демо-приложений OpenVINO на языке Python 3](#_Toc55808903)

[3.1 Настройка окружения OpenVINO 3](#_Toc55808904)

[3.2 Скачивание моделей 4](#_Toc55808905)

[3.3 Конвертация моделей 4](#_Toc55808906)

[3.4 Запуск примеров для сегментации изображений 4](#_Toc55808907)

[4 Разработка приложения для классификации ЭКГ с использованием OpenVINO 5](#_Toc55808908)

[4.1 Конвертация модели 5](#_Toc55808909)

[4.2 Рабочие скрипты 6](#_Toc55808910)

[4.3 Загрузка модели 6](#_Toc55808911)

[4.4 Загрузка и предобработка данных 7](#_Toc55808912)

[4.5 Вывод модели 7](#_Toc55808913)

[4.6 Постпроцессинг 7](#_Toc55808914)

[4.7 Создание тестирующего примера 9](#_Toc55808915)

[4.7.1 Разбор параметров командной строки 9](#_Toc55808916)

[4.7.2 Создание основной функции 9](#_Toc55808917)

[4.8 Запуск приложения 10](#_Toc55808918)

[5 Литература 10](#_Toc55808919)

[5.1 Основная литература 10](#_Toc55808920)

[5.2 Дополнительная литература 10](#_Toc55808921)

[5.3 Ресурсы сети Интернет 10](#_Toc55808922)

# Введение

Настоящая практическая работа является вводной и предусматривает создание инфраструктуры для выполнения последующих работ. Здесь дается описание процедуры установки Intel Distribution of OpenVINO Toolkit [3] и настройки окружения для работы с данным инструментом.

Решение задач, поставленных в ходе выполнения практических работ, осуществляется на языке Python 3. В качестве натренированных моделей глубокого обучения используются модели из множества обученных моделей Open Model Zoo [3]. В данной практической работе рассматривается задача классификации изображений и предлагается общая схема ее решения средствами Intel Distribution of OpenVINO Toolkit.

# Методические указания

## Цели и задачи работы

Цель работы состоит в создании глубокой модели сегментации ЭКГ с применением инструмента Intel Distribution of OpenVINO Toolkit.

Для достижения поставленной цели необходимо решить следующие задачи:

* Установить Intel Distribution of OpenVINO Toolkit.
* Настроить рабочее окружение.
* Изучить структуру и состав Intel Distribution of OpenVINO Toolkit.
* Сконвертировать глубокую модель сегментации ЭКГ.
* Выполнить классификацию изображения.

## Структура работы

В работе приводится руководство по установке Intel Distribution of OpenVINO Toolkit и необходимых зависимостей. Руководство включает описание последовательности действий, которую следует выполнить из командной строки для настройки рабочего окружения и проведения экспериментов по классификации при помощи глубоких моделей. Далее поэтапно разрабатывается программный код, решающий задачу классификации изображений.

## Рекомендации по проведению занятий

При выполнении данной практической работы рекомендуется следующая последовательность действий:

* Выполнить установку Intel Distribution of OpenVINO Toolkit и зависимостей.
* Настроить рабочее окружение.
* Изучить структуру и состав Intel Distribution of OpenVINO Toolkit, опираясь на лекционный материал курса и дополнительные источники.
* Разработать программный код для решения задачи классификации с применением компонента Inference Engine, входящего в состав инструмента Intel Distribution of OpenVINO Toolkit, проверить его работоспособность.

# Запуск примеров и демо-приложений OpenVINO на языке Python

## Настройка окружения OpenVINO

После того, как установлен OpenVINO, создана и активирована виртуальная среда, необходимо добавить Python-библиотеки OpenVINO в переменную окружения PATH с помощью одной из нижеперечисленных команд.

В Windows:

"C:\Program Files (x86)\Intel\openvino\_2021\bin\setupvars.bat"

В Linux:

source ~/intel/openvino\_2021/bin/setupvars.sh

## Скачивание моделей

Open Model Zoo [3] – репозиторий глубоких нейросетевых моделей, содержащий большое количество обученных моделей, которые могут исполняться при помощи OpenVINO. Данный репозиторий хранит не только модели, но и параметры для конвертации моделей из разных фреймворков в промежуточный формат OpenVINO.

Для скачивания моделей из репозитория Open Model Zoo нужно воспользоваться инструментом Model Downloader, точка входа – скрипт download.py.

python "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\tools\model\_downloader\downloader.py" --name <model\_name> --output\_dir <destination\_folder>

где <model\_name> – название скачиваемой модели, а <destination\_folder> – директория, в которую необходимо скачать модель.

Список доступных для скачивания моделей можно получить посредством указания ключа   
--print\_all:

python "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\tools\model\_downloader\downloader.py" --print\_all

Для решения задачи сегментации можно использовать модель DeepLabv3.

## Конвертация моделей

Для конвертации загруженных моделей нужно воспользоваться инструментом Model Optimizer и входящим в него модулем converter.py. Данный модуль имеет доступ к параметрам конвертации моделей из зоопарка моделей.

python "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\tools\model\_downloader\converter.py" --name <model\_name> --download\_dir <destination\_folder>

Для конвертации собственных моделей необходимо узнать дополнительные параметры и использовать модуль mo.py, это будет рассмотрено позже.

## Запуск примеров для сегментации изображений

В пакете OpenVINO содержится файл segmentation\_demo.py, который позволяет классифицировать любое изображение при помощи глубокой нейронной сети. На официальном сайте присутствует полное описание данного примера и инструкций по его запуску [7].

Для запуска примера скачайте, сконвертируйте и запустите модель Squeezenet, последовательность команд приведена ниже, только ***нужно заменить пути*** в угловых скобках на реальные пути в вашем компьютере.

python "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\tools\model\_downloader\downloader.py" --name deeplabv3 --output\_dir <destination\_folder>

python "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\tools\model\_downloader\downloader.py" --name deeplabv3 --download\_dir <destination\_folder>

python "C:\Program Files (x86)\Intel\openvino\_2021.1.110\deployment\_tools\open\_model\_zoo\demos\python\_demos\segmentation\_demo\segmentation\_demo.py" -i <path\_to\_image> -m <path\_to\_model>\deeplabv3.xml

После запуска данного кода в консоли должен появиться выход работы данного примера.

[ INFO ] Creating Inference Engine

[ INFO ] Loading network

[ INFO ] Preparing input blobs

[ WARNING ] Image 1.png is resized from (473, 653) to (513, 513)

[ INFO ] Batch size is 1

[ INFO ] Loading model to the plugin

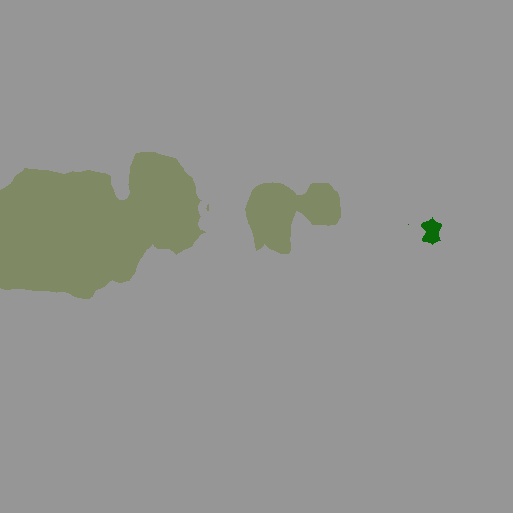
[ INFO ] Starting inference

[ INFO ] Processing output blob

[ INFO ] Result image was saved to out\_0.bmp

[ INFO ] This demo is an API example, for any performance measurements please use the dedicated benchmark\_app tool from the openVINO toolkit

Ниже приводится пример сегментации изображения с помощью глубокой модели.



Код данного и остальных примеров можно использовать для изучения программного интерфейса компонента Inference Engine. Далее приводится последовательность разработки аналогичного приложения.

# Разработка приложения для сегментации ЭКГ с использованием OpenVINO

## Конвертация модели

Для конвертации загруженных моделей нужно воспользоваться инструментом Model Optimizer, который располагается по пути: openvino\_2021.1.110\deployment\_tools\model\_optimizer\mo.py. Данный модуль анализирует архитектуру глубоких моделей. Вызовите скрипт mo.py с одним параметром – путем до вашей модели классификации ЭКГ.

python "C:\Program Files (x86)\Intel\openvino\_2021\ deployment\_tools\model\_optimizer\mo.py" --input\_model <path\_to\_your\_model>

Если все слои поддерживаются в Model Optimizer, то вы увидите следующий текст в консоли:

Model Optimizer arguments:

Common parameters:

- Path to the Input Model: C:\\_dev\dl\_ecg\openvino\ecg\_segm.onnx

- Path for generated IR: C:\\_dev\dl\_ecg\openvino\.

- IR output name: ecg\_segm

- Log level: ERROR

- Batch: Not specified, inherited from the model

- Input layers: Not specified, inherited from the model

- Output layers: Not specified, inherited from the model

- Input shapes: Not specified, inherited from the model

- Mean values: Not specified

- Scale values: Not specified

- Scale factor: Not specified

- Precision of IR: FP32

- Enable fusing: True

- Enable grouped convolutions fusing: True

- Move mean values to preprocess section: None

- Reverse input channels: False

ONNX specific parameters:

Model Optimizer version: 2021.1.0-1237-bece22ac675-releases/2021/1

[ SUCCESS ] Generated IR version 10 model.

[ SUCCESS ] XML file: C:\\_dev\dl\_ecg\openvino\.\ecg\_segm.xml

[ SUCCESS ] BIN file: C:\\_dev\dl\_ecg\openvino\.\ecg\_segm.bin

[ SUCCESS ] Total execution time: 16.81 seconds.

В выходной папке появятся файлы с расширением .xml и .bin – модель в формате OpenVINO. Данные модели готовы к запуску в OpenVINO.

## Рабочие скрипты

Для выполнения первой практической работы необходимо создать файл ecg\_segmentation\_sample.py, содержащий класс классификатора изображений ECGSegmenterс методами \_prepare\_data, segment, process\_output, и содержащий тестирующий код для запуска ECGSegmenter.

Методы класса ECGSegmenter:

* \_init\_ – конструктор класса, инициализирует Inference Engine и загружает модель (ключевое слово pass добавлено чтобы код компилировался даже когда тело функции пусто);
* \_prepare\_data – метод, который преобразует ЭКГ в формат входа нейронной сети;
* segment – метод классификации ЭКГ при помощи нейронной сети;
* process\_output – постпроцессинг результата;

class ECGSegmenter:

def \_\_init\_\_(self, configPath = None, weightsPath = None,

device = 'CPU'):

pass

def \_prepare\_data(self, data):

pass

def segment(self, data):

pass

def process\_output(self, data, results):

pass

## Загрузка модели

Для того, чтобы выполнить загрузку модели, в файле ECGClassifier.py необходимо реализовать конструктор класса ECGSegmenter. Подробно прочитать про загрузку модели в конечное устройство можно в лекции №3 «Обзор инструмента Intel Distribution of OpenVINO Toolkit». Конструктор получает следующие обязательные и необязательные параметры:

* configPath – путь до xml-файла с описанием модели.
* weightsPath – путь до bin-файла с весами модели.

Конструктор выполняет следующие действия:

1. Создание объекта класса IECore.

self.ie = IECore()

1. Создание объекта IENetwork загруженной с диска модели с помощью функции ieCore.read\_network, параметрами этой функции являются пути до модели.

self.net = self.ie.read\_network(model=configPath, weights=weightsPath)

1. Создание объекта ExecutableNetwork объекта с помощью функции ieCore.load\_network, параметрами этой функции загруженная модель, устройство для инференса и параметры запуска.

self.exec\_net = self.ie.load\_network(network=self.net,

                                             device\_name=device)

## Загрузка и предобработка данных

В реальных приложениях вам может понадобиться предобработать ваши данные – заполнить неизвестные значения, привести к входному формату модели. Мы же проверим что наш входной вектор такого же размера, что и вход модели.

def \_prepare\_data(self, df, k, n, l):

data = df.astype(np.float)

if data.shape[-1] != l:

raise RuntimeError('Input data is not acceptable to model')

return data

## Вывод модели

Следующий этап – реализация метода классификации изображения segment, который запускает вывод глубокой модели на устройстве, указанном в конструкторе. Логика работы метода segment следующая:

1. Получить данные о входе и выходе нейронной сети.

input\_blob = next(iter(self.net.input\_info))

out\_blob = next(iter(self.net.outputs))

1. Из данных о входе нейронной сети получить требуемые нейросетью размеры для входного изображения.

k, n, l = self.net.input\_info[input\_blob].input\_data.shape

1. С помощью функции \_prepare\_image подготовить изображение.
2. Вызвать функцию синхронного исполнения модели.

output = self.exec\_net.infer(inputs = {input\_blob: blob})

1. Из выхода модели получить тензор с результатом классификации.

output = output[out\_blob]

## Постпроцессинг

В качестве результата работа классификационной модели мы получили вектор размера 4\*5000 чисел, который содержит вероятности принадлежности каждого отрезка ЭКГ к одному из классов. Необходимо написать код, который сначала переведет вероятности в класс

    def process\_output(self, data, results):

        # Code for plot

        v\_to\_del = {1:'p', 2:'qrs', 3:'t'}

        sample\_rate = 500

        def remove\_small(signal):

            max\_dist = 12

            last\_zero = 0

            for i in range(len(signal)):

                if signal[i] == 0:

                    if i - last\_zero < max\_dist:

                        signal[last\_zero:i] = 0

                    last\_zero = i

        def merge\_small(signal):

            max\_dist = 12

            lasts = np.full(signal.max() + 1, -(max\_dist+1))

            for i in range(len(signal)):

                m = signal[i]

                if i - lasts[m] < max\_dist and m > 0:

                    signal[lasts[m]:i] = m

                lasts[m] = i

        def mask\_to\_delineation(mask):

            merge\_small(mask)

            remove\_small(mask)

            delineation = {'p':[], 'qrs':[], 't':[]}

            i = 0

            mask\_length = len(mask)

            while i < mask\_length:

                v = mask[i]

                if v > 0:

                    delineation[v\_to\_del[v]].append([i, 0])

                    while i < mask\_length and mask[i] == v:

                        delineation[v\_to\_del[v]][-1][1] = i

                        i += 1

                    t = delineation[v\_to\_del[v]][-1]

                i += 1

            return delineation

        wave\_type\_to\_color = {

                    "p": "yellow",

                    "qrs": "red",

                    "t": "green"

                    }

        def plot\_signal\_with\_mask(signal, mask):

            plt.figure(figsize=(18, 5))

            plt.title("Сигнал с маской")

            plt.xlabel("Время (сек)")

            plt.ylabel("Амплитуда (мВ)")

            x\_axis\_values = np.linspace(0, len(signal) / sample\_rate, len(signal))

            plt.plot(x\_axis\_values, signal, linewidth=2, color="black")

            delineation = mask\_to\_delineation(mask)

            for wave\_type in ["p", "qrs", "t"]:

                color = wave\_type\_to\_color[wave\_type]

                for begin, end in delineation[wave\_type]:

                    begin /= sample\_rate

                    end /= sample\_rate

                    plt.axvspan(begin, end, facecolor=color, alpha=0.5)

        # Process output

        results = results[0]

        print(results, results.shape)

        results = results.argmax(axis=0)

        results[:500] = 0

        results[-500:] = 0

        # Make plot

        plot\_signal\_with\_mask(data, results)

        plt.savefig('plot.png')

## Создание тестирующего примера

### Разбор параметров командной строки

В работе очень удобно пользоваться командной строкой и запускать программы с именованными аргументами. В языке Python для этого используется пакет argrparse, который позволяет описать имя, тип и другие параметры для каждого аргумента. Создайте функцию build\_argparser, которая будет создавать объект ArgumentParser для работы с аргументами командной строки.

В данной лабораторной работе потребуются следующие аргументы командной строки:

* Путь до входных данных (обязательный аргумент).
* Путь до весов нейронной сети (обязательный аргумент).
* Путь до конфигурации нейронной сети (обязательный аргумент).

def build\_argparser():

parser = argparse.ArgumentParser()

parser.add\_argument('-m', '--model', help = 'Path to an .xml \

file with a trained model.', required = True, type = str)

parser.add\_argument('-w', '--weights', help = 'Path to an .bin file \

with a trained weights.', required = True, type = str)

parser.add\_argument('-i', '--input', help = 'Path to \

image file', required = True, type = str)

return parser

### Создание основной функции

Создайте функцию main, которая выполняет следующие действия:

1. Разбор аргументов командной строки.
2. Создание объекта класса ECGSegmenter с необходимыми параметрами.
3. Чтение ЭКГ.
4. Классификация ЭКГ.
5. Вывод результата классификации на экран.

Для вывода логов в консоль предлагается использовать пакет logging.

def main():

log.basicConfig(format="[ %(levelname)s ] %(message)s",

level=log.INFO, stream=sys.stdout)

args = build\_argparser().parse\_args()

log.info("Start ECG segmentation sample")

ie\_segmenter = ECGSegmenter(configPath=args.model,

weightsPath=args.weights, device=args.device)

data = np.loadtxt(args.input)

segmentation = ie\_segmenter.segment(data)

ie\_segmenter.process\_output(data, segmentation)

return

if \_\_name\_\_ == '\_\_main\_\_':

sys.exit(main())

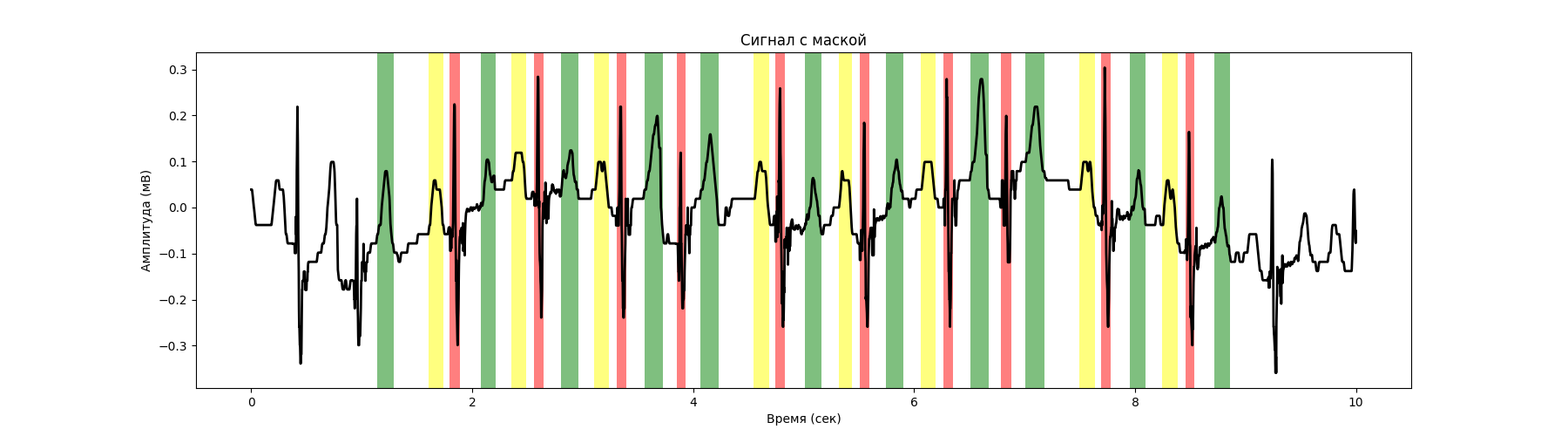
## Запуск приложения

Запуск разработанного приложения удобней всего произвести из командной строки. Для этого необходимо открыть командную строку. Строка запуска будет иметь следующий вид:

python ECGSegmentation.py -i segm\_test1.txt -m ecg\_segm.xml -w ecg\_segm.bin

Аргумент -i задает путь к текстовому файлу, аргумент -m задает путь к конфигурации модели, аргумент -w задает путь к весам модели, аргумент

Результат запуска приложения должен выглядеть следующим образом. В папке с программой появляется изображение “plot.png” с графиком сегментации ЭКГ.



# Литература

## Основная литература

1. Шолле Ф. Глубокое обучение на Python. – СПб.: Питер. – 2018. – 400с.

## Дополнительная литература

1. Рамальо Л. Python. К вершинам мастерства / Пер. с англ. Слинкин А.А. – М.: ДМК Пресс. –2016. – 768с.

## Ресурсы сети Интернет

1. Страница репозитория Open Model Zoo [https://github.com/openvinotoolkit/open\_model\_zoo].
2. Страница скачивания Intel Distribution of OpenVINO Toolkit [https://software.intel.com/en-us/openvino-toolkit/choose-download].
3. Страница скачивания Python 3.8.6 [https://www.python.org/downloads/release/python-386/].
4. Документация Intel Distribution of OpenVINO Toolkit [https://docs.openvinotoolkit.org/latest/index.html].
5. OpenVINO classification sample [https://docs.openvinotoolkit.org/latest/\_inference\_engine\_ie\_bridges\_python\_sample\_classification\_sample\_README.html].