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 №1  
Построение алгоритма классификации ЭКГ**

*При поддержке компании Intel*

*Васильев Е.П.*

Нижний Новгород  
2020

**Содержание**

[1 Введение 4](#_Toc55808914)

[2 Методические указания 4](#_Toc55808915)

[2.1 Цели и задачи работы 4](#_Toc55808916)

[2.2 Структура работы 4](#_Toc55808917)

[2.3 Рекомендации по проведению занятий 4](#_Toc55808918)

[3 Установка Intel Distribution of OpenVINO Toolkit и его зависимостей для Python 4](#_Toc55808919)

[3.1 Установка Python 3 4](#_Toc55808920)

[3.2 Создание виртуальной среды Python 5](#_Toc55808921)

[3.3 Установка Intel Distribution of OpenVINO Toolkit 5](#_Toc55808922)

[3.4 Установка дополнительных модулей Python 5](#_Toc55808923)

[4 Запуск примеров и демо-приложений OpenVINO на языке Python 6](#_Toc55808924)

[4.1 Настройка окружения OpenVINO 6](#_Toc55808925)

[4.2 Скачивание моделей 6](#_Toc55808926)

[4.3 Конвертация моделей 6](#_Toc55808927)

[4.4 Запуск примеров для классификации изображений 7](#_Toc55808928)

[5 Разработка приложения для классификации изображений с использованием OpenVINO 8](#_Toc55808929)

[5.1 Рабочие скрипты 8](#_Toc55808930)

[5.2 Загрузка модели 8](#_Toc55808931)

[5.3 Загрузка и предобработка изображения 9](#_Toc55808932)

[5.4 Вывод модели 9](#_Toc55808933)

[5.5 Обработка выхода модели 10](#_Toc55808934)

[5.6 Создание тестирующего примера 10](#_Toc55808935)

[5.6.1 Разбор параметров командной строки 10](#_Toc55808936)

[5.6.2 Создание основной функции 10](#_Toc55808937)

[5.7 Запуск приложения 11](#_Toc55808938)

[6 Разработка приложения для классификации ЭКГ с использованием OpenVINO 11](#_Toc55808939)

[6.1 Конвертация модели 11](#_Toc55808940)

[6.2 Рабочие скрипты 12](#_Toc55808941)

[6.3 Загрузка модели 12](#_Toc55808942)

[6.4 Загрузка и предобработка данных 13](#_Toc55808943)

[6.5 Вывод модели 13](#_Toc55808944)

[6.6 Создание тестирующего примера 13](#_Toc55808945)

[6.6.1 Разбор параметров командной строки 13](#_Toc55808946)

[6.6.2 Создание основной функции 14](#_Toc55808947)

[6.7 Запуск приложения 14](#_Toc55808948)

[7 Дополнительные задания 15](#_Toc55808949)

[8 Литература 15](#_Toc55808950)

[8.1 Основная литература 15](#_Toc55808951)

[8.2 Дополнительная литература 15](#_Toc55808952)

[8.3 Ресурсы сети Интернет 15](#_Toc55808953)

# 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 изучении глубоких моделей для решения задачи классификации изображений 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.
* Скачать и конвертировать глубокую классификационную модель.
* 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.

# Установка Intel Distribution of OpenVINO Toolkit и его зависимостей для Python

## Установка Python 3

Для работы OpenVINO 2021.1 лучше всего использовать последнюю версию Python 3.8 (также можно использовать Python 3.6 или 3.7), которую можно скачать с официального сайта [5].

Информация, приведенная ниже, актуальна для пользователей операционной системы Windows. Если при установке путь до бинарных файлов Python не добавлен в переменную окружения PATH, то для работы с Python из командной строки требуется выполнить следующую команду, чтобы сделать его доступным из командной строки:

set PATH="C:\Users\<USERNAME>\AppData\Local\Programs\Python\Python38;%PATH%"

## Создание виртуальной среды Python

Библиотеки на языке Python устанавливаются в систему подобно тому, как устанавливаются программы в операционную систему. При этом может потребоваться несколько разных версий одной библиотеки. Для этого можно создать несколько окружений – виртуальных сред Python, в которых будут установлены разные версии библиотеки.

Для того, чтобы создать новую виртуальную среду Python, выполните команды, показанные ниже.

mkdir openvino-virtual-environments && cd openvino-virtual-environments

python -m venv openvinoenv

Чтобы активировать виртуальную среду, выполните одну из следующих команд.

В Windows:

openvino-virtual-environments\bin\activate.bat

В Linux:

source openvino-virtual-environments/bin/activate

В дальнейшем при работе с Python необходимо активировать существующую виртуальную среду Python. Для удобства можно сохранить все команды в текстовый файл, чтобы в следующий раз не набирать их вручную, либо создать скрипт, активирующий Python, виртуальную среду и OpenVINO.

## Установка Intel Distribution of OpenVINO Toolkit

Для установки OpenVINO необходимо скачать инсталлятор с официального сайта [3]. Для скачивания потребуется бесплатная регистрация. Во время скачивания вам станет доступен ключ. Сохранять ключ не обязательно, ПО работает без активации.

## Установка дополнительных модулей Python

В лабораторной работе все пути представлены в Windows формате (обратные слеши в путях, пути с пробелами как Program Files должны быть в кавычках). При работе в Linux и Mac пути будут аналогичными до уровня папки установки OpenVINO.

Для работы с OpenVINO необходимо конвертировать модель из оригинального фреймворка модели в промежуточный формат (Intermediate representation, IR) OpenVINO. Для конвертации требуется установить актуальную версию тренировочного фреймворка. Можно установить один интересующий фреймворк или все сразу, используя одну из следующих команд.

Для всех фреймворков:

pip install -r "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\model\_optimizer\requirements.txt"

Только для Caffe:

pip install -r "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\model\_optimizer\requirements\_caffe.txt"

Для скачивания моделей из зоопарка моделей Open Model Zoo требуется установить зависимости для работы с сетью из Python.

pip install -r "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\open\_model\_zoo\tools\downloader\requirements.in"

После данных действий ваш виртуальный энвайрмент полностью готов к разработке на языке Python с использованием OpenVINO.

# 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 классификации неплохой моделью, являющейся хорошим компромиссом между производительностью и качеством, является модель Squeezenet1.1.

## 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 классификации изображений

В пакете OpenVINO содержится файл classification\_sample.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 squeezenet1.1 --output\_dir <destination\_folder>

python "C:\Program Files (x86)\Intel\openvino\_2021\deployment\_tools\tools\model\_downloader\downloader.py" --name squeezenet1.1 --download\_dir <destination\_folder>

python "C:\Program Files (x86)\Intel\openvino\_2021\inference\_engine\samples\python\_samples\classification\_sample\classification\_sample.py" -i <path\_to\_image> -m <path\_to\_model>\squeezenet1.1.xml

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

[ INFO ] Creating Inference Engine

[ INFO ] Loading network files:

squeezenet1.0.xml

squeezenet1.0.bin

[ INFO ] Preparing input blobs

[ WARNING ] Image dog.jpg is resized from (486, 729) to (227, 227)

[ INFO ] Batch size is 1

[ INFO ] Loading model to the plugin

[ INFO ] Starting inference in synchronous mode

[ INFO ] Processing output blob

[ INFO ] Top 10 results:

Image dog.jpg

classid probability

------- -----------

208 0.6787384

243 0.1161512

207 0.0784803

247 0.0298401

167 0.0120248

222 0.0113272

246 0.0085536

159 0.0085039

212 0.0081170

242 0.0065376

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

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

# Разработка приложения для классификации изображений с использованием OpenVINO

## Рабочие скрипты

Для выполнения первой практической работы необходимо создать два файла: файл ie\_classifier.py, содержащий класс классификатора изображений InferenceEngineClassifierс методами \_prepare\_image, classify, get\_top, и файл classification\_sample.py, содержащий тестирующий код для запуска InferenceEngineClassifier.

Методы класса InferenceEngineClassifier:

* \_init\_ – конструктор класса, инициализирует Inference Engine и загружает модель (ключевое слово pass добавлено чтобы код компилировался даже когда тело функции пусто);
* \_prepare\_image – метод, который преобразует изображение в формат входа нейронной сети;
* classify – метод классификации изображения при помощи нейронной сети;
* get\_top – метод для выбора первых N наилучших результатов классификации (с максимальной достоверностью).

class InferenceEngineClassifier:

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

device = 'CPU', classesPath = None):

pass

def get\_top(self, prob, topN = 1):

pass

def \_prepare\_image(self, image, h, w):

pass

def classify(self, image):

pass

Мы осознанно разделяем класс и тестирующий код для этого класса, поскольку класс InferenceEngineClassifier понадобится как составная часть для следующих практических работ. Далее последовательно рассмотрим реализацию каждого метода указанного класса и тестирующего кода.

## Загрузка модели

Для того, чтобы выполнить загрузку модели, в файле ie\_classifier.py необходимо реализовать конструктор класса InferenceEngineClassifier. 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.
* classesPath – путь до файла с именами классов (текстовый файл, где на каждой строке имя класса).

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)

1. Загрузка перечня классов изображений из файла.

if classesPath:

self.classes = [line.rstrip('\n') for line in open(classesPath)]

## Загрузка и предобработка изображения

Следующим этапом является реализация метода подготовки изображения \_prepare\_image. Обработка изображений глубокими моделями отличается от обработки изображений классическими алгоритмами тем, что сети принимают изображения поканально, а не попиксельно, изображения в подаваемой пачке необходимо преобразовать из формата RGBRGBRG… в формат RRRGGGBBB... Для этого можно воспользоваться функцией transpose.

image = image.transpose((2, 0, 1))

Также необходимо уменьшить или увеличить размер изображения до размера входа сети.

image = cv2.resize(image, (w, h))

В общем случае на вход должен подаваться 4-мерный тензор, например, [1,3,227,227], где первая координата – количество изображений в пачке; 3 – количество цветовых каналов изображения; 227, 227 – ширина и высота изображения. Однако, если на вход сети подать трехмерный тензор [3,227,227], то OpenVINO автоматически добавит четвертую размерность.

Также стоит помнить особенность работы библиотеки OpenVINO. Ядро библиотеки хранит изображения в последовательности BGR, а не RGB. Если модель загружается из Open Model Zoo и конвертируется с параметрами по умолчанию, то тогда данный момент уже учтен, однако если используется модель не из Open Model Zoo, то необходимо поменять красный и синий каналы изображения местами.

## Вывод модели

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

n, c, h, w = 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]

## Обработка выхода модели

Для обработки выхода необходимо реализовать функцию \_get\_top, для того чтобы получить первые N предсказанных нейросетью классов. Чтобы вывести N наибольших вероятностей, номера вероятностей можно отсортировать по возрастанию. Для этого можно воспользоваться функцией numpy.argsort. Стоит отметить, что функция argsort получает на вход одномерный тензор. Если на входе тензор размера [1,1000,1,1], то необходимо его преобразовать в тензор размера [1000], для этого можно использовать функцию np.squeeze(). При этом необходимо установить соответствие с перечнем классов, содержащемся в файле, путь до которого передан в качестве входного параметра конструктора.

## 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 входного изображения (обязательный аргумент).
* Путь до весов нейронной сети (обязательный аргумент).
* Путь до конфигурации нейронной сети (обязательный аргумент).
* Путь до файла с именами классов (необязательный 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)

parser.add\_argument('-c', '--classes', help = 'File containing \

classnames', type = str, default = None)

return parser

### Создание основной функции

Создайте функцию main, которая выполняет следующие действия:

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

Для вывода логов в консоль предлагается использовать пакет logging.

import logging as log

def main():

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

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

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

log.info("Start IE classification sample")

ie\_classifier = InferenceEngineClassifier(configPath=args.model,

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

extension=args.cpu\_extension, classesPath=args.classes)

img = cv2.imread(args.input)

prob = ie\_classifier.classify(img)

predictions = ie\_classifier.get\_top(prob, 5)

log.info("Predictions: " + str(predictions))

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 ie\_classification\_sample.py -i image.jpg -m squeezenet1.1.xml \

-w squeezenet1.1.bin -c imagenet\_synset\_words.txt

Аргумент -i задает путь к изображению, аргумент -m задает путь к конфигурации модели, аргумент -w задает путь к весам модели, аргумент -c задает путь к файлу с именами классов для модели.

Результат запуска приложения должен выглядеть следующим образом. Выводится сообщение о старте приложения, затем выводится список классов и их вероятность.

[ INFO ] Start IE classification sample

[ INFO ] Predictions: [['n02099712 Labrador retriever', 67.87383556365967], ['n02108422 bull mastiff', 11.615122854709625], ['n02099601 golden retriever', 7.8480251133441925], ['n02109525 Saint Bernard, St Bernard', 2.984011545777321], ['n02089973 English foxhound', 1.2024826370179653]]

# Разработка приложения для классификации 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:\Users\eugene\Desktop\2035\arrhythmia.onnx

- Path for generated IR: C:\Users\eugene\Desktop\2035\.

- IR output name: arrhythmia

- 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:\Users\eugene\Desktop\2035\.\arrhythmia.xml

[ SUCCESS ] BIN file: C:\Users\eugene\Desktop\2035\.\arrhythmia.bin

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 классификации ЭКГ необходимо создать файл: файл ie\_ecg\_classifier.py, содержащий класс классификатора изображений ECGClassifierс методами \_prepare\_data, classify и тестирующий код для запуска ECGClassifier.

Методы класса ECGClassifier:

* \_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;
* classify – метод классификации ЭКГ при помощи нейронной сети;

class ECGClassifier:

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

device = 'CPU'):

pass

def \_prepare\_data(self, data):

pass

def classify(self, data):

pass

## Загрузка модели

Для того, чтобы выполнить загрузку модели, в файле ECGClassifier.py необходимо реализовать конструктор класса ECGClassifier. 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, 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 classify, который запускает вывод глубокой модели на устройстве, указанном в конструкторе. Логика работы метода classify 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.

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]

## 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 ECGClassifier 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 classification sample")

ie\_classifier = ECGClassifier(configPath=args.model,

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

# Read data

data = pd.read\_csv(args.input, header=None, na\_values="?")

# Start inference

prob = ie\_classifier.classify(data)

log.info("Predictions: " + str(prob))

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 ECGclassification.py -i test1.txt -m ECG.xml -w ECG.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. Выводится сообщение о старте приложения, затем выводится список классов и их вероятность.

[ INFO ] Start ECG classification sample

[ INFO ] Predictions: [[-223.05115]]

# Дополнительные задания

Созданный пример классификации содержит минимально необходимый функционал. В качестве дополнительных заданий предлагается обеспечить поддержку следующих возможностей:

1. Поддержка классификации не только одной картинки, но и набора из нескольких изображений.
2. Поддержка выполнения вывода глубоких моделей не только на CPU, но и на Intel Processor Graphics или Neural Compute Stick (при наличии).
3. Поддержка асинхронного вывода глубоких моделей.

Данные задания предлагается выполнить самостоятельно, опираясь на документацию и примеры, входящие состав пакета OpenVINO.

# 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].