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

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.

2 Методические указания

2.1 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.

2.2 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.

2.3 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.

3 Запуск примеров и демо-приложений OpenVINO на языке Python

3.1 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
```

3.2 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\download
er.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\download
er.py" --print_all
```

Для решения задачи сегментации можно использовать модель DeepLabv3.

3.3 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\converte
r.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.

3.4 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\download
er.py" --name deeplabv3 --output_dir <destination_folder>
python "C:\Program Files
(x86)\Intel\openvino_2021\deployment_tools\tools\model_downloader\download
er.py" --name deeplabv3 --download_dir <destination_folder>
python "C:\Program Files
(x86)\Intel\openvino_2021.1.110\deployment_tools\open_model_zoo\demos\pyth
```

```
on_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.

4 Разработка приложения для классификации ЭКГ с использованием OpenVINO

4.1 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.

4.2 Рабочие скрипты

Для выполнения первой практической работы необходимо создать файл **ecg_segmentation_sample.py**, содержащий класс классификатора изображений **ECGSegmenter** с 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 ECGClassifier:
    def __init__(self, configPath = None, weightsPath = None,
                 device = 'CPU'):
        pass

    def _prepare_data(self, data):
        pass

    def classify(self, data):
        pass

    def process_output(self, data, results):
        pass
```

4.3 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()
```

2. 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)
```

3. 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)
```

4.4 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
```

4.5 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))
```

2. 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
```

3. С помощью функции **_prepare_image** подготовить изображение.
4. Вызвать функцию синхронного исполнения модели.

```
output = self.exec_net.infer(inputs = {input_blob: blob})
```

5. Из выхода модели получить тензор с результатом классификации.

```
output = output[out_blob]
```

4.6 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')
```

4.7 Создание тестирующего примера

4.7.1 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
```

4.7.2 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())
```

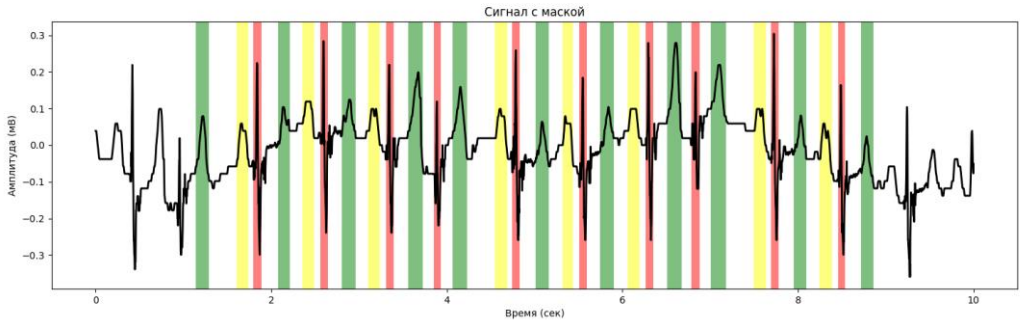
4.8 Запуск приложения

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.



5 Литература

5.1 Основная литература

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

5.2 Дополнительная литература

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

5.3 Ресурсы сети Интернет

3. Страница репозитория Open Model Zoo [https://github.com/openvinotoolkit/open_model_zoo].

4. Страница скачивания Intel Distribution of OpenVINO Toolkit [https://software.intel.com/en-us/openvino-toolkit/choose-download].
5. Страница скачивания Python 3.8.6 [https://www.python.org/downloads/release/python-386/].
6. Документация Intel Distribution of OpenVINO Toolkit [https://docs.openvinotoolkit.org/latest/index.html].
7. OpenVINO classification sample [https://docs.openvinotoolkit.org/latest/_inference_engine_ie_bridges_python_sample_classification_sample_README.html].