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

Летняя школа Intel – ННГУ по компьютерному зрению

Практическая работа №2
Классификация изображений с большим числом категорий с использованием методов глубокого обучения

Васильев Е.П.

Нижний Новгород
2021

1 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.
- Разработать программный код для решения задачи классификации с применением компонента Inference Engine, входящего в состав инструмента Intel Distribution of OpenVINO Toolkit, проверить его работоспособность.

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

Перед установкой дополнительных модулей в Python активируйте виртуальную рабочую среду, которую вы создали в прошлый раз.

2.1 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.

2.2 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.tx
t"
```

Только для Caffe:

```
pip install -r "C:\Program Files
(x86)\Intel\openvino_2021\deployment_tools\model_optimizer\requirements_ca
ffe.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.

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

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 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\download
er.py" --name squeezenet1.1 --output_dir <destination_folder>
python "C:\Program Files
(x86)\Intel\openvino_2021\deployment_tools\tools\model_downloader\converte
r.py" --name squeezenet1.1 --download_dir <destination_folder>
python "C:\Program Files
(x86)\Intel\openvino_2021\inference_engine\samples\python
\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.

4 Разработка приложения для классификации изображений с использованием OpenVINO

4.1 Рабочие скрипты

Шаблон для выполнения практической работы расположен в файле **doge_classifier.py**, содержащий класс классификатора изображений **InferenceEngineClassifier** с методами **_prepare_image**, **classify**, **get_top**.

Методы класса **InferenceEngineClassifier**:

- **_init_** – конструктор класса, инициализирует Inference Engine и загружает модель;
- **_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,
            classesPath = None):
        pass

    def get_top(self, prob, topN = 1):
        pass

    def _prepare_image(self, image, h, w):
        pass

    def classify(self, image):
        pass
```

4.2 Загрузка модели

Для того, чтобы выполнить загрузку модели, в файле **ie_classifier.py** необходимо реализовать конструктор класса **InferenceEngineClassifier**. 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.

Конструктор выполняет следующие действия:

- Создание объекта класса **IECore**.
- Создание объекта класса **IENetwork** с параметрами – путями до модели.
- Загрузка созданного объекта класса **IENetwork** в **IECore**, что соответствует загрузке модели в плагин.
- Загрузка перечня классов изображений из файла.

4.3 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.

4.4 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.inputs))
out_blob = next(iter(self.net.outputs))
```

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

```
n, c, h, w = self.net.inputs[input_blob].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]
```

4.5 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], то необходимо его преобразовать в тензор размера [1000]. При этом необходимо установить соответствие с перечнем классов, содержащемся в файле, путь до которого передан в качестве входного параметра конструктора.

4.6 Запуск программы

4.6.1 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 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
```

4.6.2 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())
```

5 Запуск приложения

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

```
python doge_classifier.py -i doge.jpg -m resnet-50.xml -w resnet-50.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: [['n02110185 Siberian husky', 36.448127031326294],
['n02109961 Eskimo dog, husky', 16.096267104148865], ['n02113799 standard
poodle', 10.928214341402054], ['n02110063 malamute, malemute, Alaskan
malamute', 5.673458427190781], ['n02104029 kuvasz', 3.9648767560720444]]
```

6 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).

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

7 Литература

1. Шолле Ф. Глубокое обучение на Python. – СПб.: Питер. – 2018. – 400с.
2. Рамальо Л. Python. К вершинам мастерства / Пер. с англ. Слинкин А.А. – М.: ДМК Пресс. – 2016. – 768с.
3. Страница репозитория Open Model Zoo [https://github.com/opencv/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 3.8.9 [https://www.python.org/downloads/release/python-389].
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].