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

Летняя школа ННГУ по компьютерному зрению

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

Васильев Е.П.

Нижний Новгород
2022

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:

- Настроить рабочее окружение.
- Установить Intel Distribution of OpenVINO Toolkit.
- 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.

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.

2.1 Настройка рабочего окружения

Скачайте и установите Python с официального сайта (версии 3.6-3.9): https://www.python.org/ftp/python/3.9.13/python-3.9.13-embed-amd64.zip

Создайте и активируйте виртуальное окружение. Если у вас сохранилось окружение с первой практики, создавать новое окружение не нужно, активируйте существующее, и обновите pip до последней версии.

```
mkdir openvino-virtual-environments && cd openvino-virtual-environments
python -m venv openvinoenv
openvino-virtual-environments\bin\activate.bat
python –m pip install --upgrade pip
```

2.2 Установка Intel Distribution of OpenVINO Toolkit

Установка OpenVINO возможна двумя путями:

- Через скачивание инсталлятора с официального сайта [3];
- Через скачивание pip пакета с https://pypi.org/.

Для компиляции приложений на C и C++ поддержкой OpenVINO подходит только первый вариант, для использования OpenVINO на языке Python подходят оба варианта. Мы воспользуемся вторым. Первый вариант установки подробно разобран в практике Летней школы по компьютерному зрению 2021 года [4].

При установке через скачивание pip пакета также можно указать фреймворки, которые мы будем устанавливать вместе с

Для установки с onnx и pytorch:

```
pip install openvino-dev[onnx,pytorch]
```
Для установки вместе с TensorFlow2, MXNet, ONNX, PyTorch:

```
pip install openvino-dev[tensorflow2,mxnet,onnx,pytorch]
```

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 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`**.

```
omz_downloader --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`**:

```
omz_downloader --print_all
```

Для решения задачи классификации неплохой моделью, являющейся хорошим компромиссом между производительностью и качеством, является модель mobilenet-v2.

3.2 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.

```
omz_converter --name <model_name> --download_dir <destination_folder>
```

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

3.3 Запуск примеров для классификации изображений

Скачайте и разархивируйте репозиторий Open Model Zoo, содержащий исходный код всех примеров для моделей из состава OpenVINO: https://github.com/openvinotoolkit/open_model_zoo/archive/refs/heads/master.zip.

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.

```
omz_downloader --name mobilenet-v2-pytorch --output_dir
<destination_folder>
```

```
omz_converter --name mobilenet-v2-pytorch --download_dir
<destination_folder>
python hello_classification.py <destination_folder>\public\mobilenet-v2-
pytorch\FP16\mobilenet-v2-pytorch.xml <image_folder>\dog.jpg CPU
```

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

```
[ INFO ] Creating OpenVINO Runtime Core
[ INFO ] Reading the model: mobilenet-v2-pytorch.xml
[ INFO ] Loading the model to the plugin
[ INFO ] Starting inference in synchronous mode
[ INFO ] Image path: doge.jpg
[ INFO ] Top 10 results:
[ INFO ] class_id probability
[ INFO ] --------------------
[ INFO ] 273      10.3775501
[ INFO ] 151      8.6661396
[ INFO ] 263      8.0712891
[ INFO ] 259      7.8583994
[ INFO ] 174      7.8489938
[ INFO ] 248      7.8138361
[ INFO ] 253      7.6007485
[ INFO ] 260      7.5041566
[ INFO ] 750      7.2182503
[ INFO ] 478      7.0947642
[ INFO ]
[ 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).

```
from openvino.runtime import Core

class InferenceEngineClassifier:
    def __init__(self, model_path = 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:

- **model_path** – путь до xml-файла с описанием модели.
- **classes_path** – путь до файла с именами классов.

Конструктор выполняет следующие действия:

- Инициализация OpenVINO через создание объекта класса **Core**.
- Загрузка модели с диска с помощью функции **read_model** класса **Core.**
- Подготовка модели для оптимизации под конкретное устройство с помощью функции **compile_model**, что соответствует загрузке модели в плагин.
- Загрузка перечня классов изображений из файла.

```
self.core = Core()
self.model = self.core.read_model(model=model_path)
self.exec_model = self.core.compile_model(model=self.model,
device_name=device)
if classes_path:
    self.classes = [line.rstrip('\n') for line in open(classes_path)]
```

4.3 Загрузка и предобработка изображения

Следующим этапом является реализация метода подготовки изображения **_prepare_image**.

В первую очередь необходимо уменьшить или увеличить размер изображения до размера входа сети.

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

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))
```

В общем случае на вход должен подаваться 4-мерный тензор, например, [1,3,224,224], где первая координата – количество изображений в пачке; 3 – количество цветовых каналов изображения; 224, 224 – ширина и высота изображения. Добавим еще одну размерность в наше изображение функцией **expand_dims**.

```
image = np.expand_dims(image, axis = 0)
```

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. Для моделей с одним входом и выходом это выглядит так:

```
input_layer = self.exec_model.input(0)
output_layer = self.exec_model.output(0)
```

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

```
n, c, h, w = input_layer.shape
```

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

```
# Стандартный способ

request = self.exec_model.create_infer_request()
request.infer(inputs={input_layer.any_name: image})
result = request.get_output_tensor(output_layer.index).data

# Либо в одну строку

result = self.exec_model([image])[output_layer]
```

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('-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(model_path=args.model,
        classes_path=args.classes)

    img = cv2.imread(args.input)

    prob = ie_classifier.classify(im
    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 mobilenet-v2-pytorch.xml -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: [['n02115641 dingo, warrigal, warragal, Canis
dingo', 10.388964], ['n02085620 Chihuahua', 8.647619], ['n02113023
Pembroke, Pembroke Welsh corgi', 8.002596], ['n02091467 Norwegian
elkhound, elkhound', 7.9609537], ['n02109961 Eskimo dog, husky',
7.910527]]
```

А вот так будет выглядеть результат работы модели ResNet-50.

```
[ 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. Классификация изображений с большим числом категорий с использованием методов глубокого обучения (2021) [https://github.com/itlab-vision/CV-SUMMER-CAMP-2021/blob/main/practice/2_classification/2_Classification.pdf].
5. Страница скачивания Python 3.9.13 [https://www.python.org/downloads/release/python-3913].
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].
8. OpenVINO API 2.0 tutorial [https://github.com/openvinotoolkit/openvino_notebooks/tree/main/notebooks/002-openvino-api]