Nizhny Novgorod State University
Institute of Information Technologies, Mathematics and Mechanics
Department of Computer Software and Supercomputer Technologies

Educational course «Modern methods and technologies of deep learning in computer vision»

Practice №1
Image classification with a large number of categories using deep learning

Supported by Intel

Vasiliev E.P.

Nizhny Novgorod
2020

Content

1 Introduction

This practice is an introductory one and aimed at the creating infrastructure for all practices. In this practice, the installation procedure for the Intel Distribution of OpenVINO Toolkit [3] is given and the environment setup for working with the toolkit is described.

Implementation of the solutions is performed using Python 3. Models of the Open Model Zoo are used as pre-trained deep learning models [3]. In this practice, the problem of image classification is described and a scheme for solving it using the Intel Distribution of OpenVINO Toolkit is proposed.

2 Guidelines

2.1 Goals and tasks

The goal of this practice is to study deep models for solving the problem of image classification using the Intel Distribution of OpenVINO Toolkit.

To achieve this goal, it is necessary to solve the following tasks:

- Install the Intel Distribution of OpenVINO Toolkit.
- Setup software environment.
- Learn the structure and modules of the Intel Distribution of OpenVINO Toolkit.
- Download and convert deep classification model.
- Perform image classification.

2.2 Practice structure

The guide describes installation of the Intel Distribution of OpenVINO Toolkit and its dependencies. This guide includes a list of commands to configure the software environment and to execute classification samples using various deep models. Further, source code for solving the image classification problem is developed step-by-step.

2.3 Recommended study sequence

The recommended study sequence is as follows:

- Install the Intel Distribution of OpenVINO Toolkit and its dependencies.
- Setup the software environment.
- Learn the structure and modules of the Intel Distribution of OpenVINO Toolkit using the corresponding lecture and documentation of the OpenVINO Toolkit.
- Develop source code for solving image classification problem using the Inference Engine component, which is part of the Intel Distribution of OpenVINO Toolkit, and verify the classification result.

3 Installing the Intel the Distribution of OpenVINO Toolkit and its dependencies

3.1 Installing Python 3

To operate with OpenVINO 2019 R3.1, we recommend to use the latest version of Python 3.7, which can be downloaded from the official web-site [4].

The information below is relevant for users of the Windows operating system. If during installation the path to the Python binaries was not added to the PATH environment variable, then you need to run the following command to make Python accessible from the command:

```
set PATH="C:\Program Files (x86)\Python3.7\bin;%PATH%"
```

3.2 Creating a Python environment

Python libraries are installed on the operating system in the same way as software packages. It may require several different versions of the same library. However, while developing applications, you may need several different versions of the same Python library. To solve this problem, you can create several virtual environments in which different versions of the library will be installed.

To create a new Python virtual environment named **openvinoenv**, execute the following commands.

```
mkdir openvino-virtual-environments && cd openvino-virtual-environments
python -m venv openvinoenv
```

To activate and use the virtual environment, use the commands represented below.

For Windows:

```
openvino-virtual-environments\bin\activate.bat
```

For Linux:

```
source openvino-virtual-environments/bin/activate
```

Further, you need to activate the existing virtual environment every time. For convenience, you can save all the commands in a text file so that at the next time you do not type them manually, or create a script that activates Python, a virtual environment and OpenVINO.

3.3 Installing the Intel the Distribution of OpenVINO Toolkit

To install the OpenVINO Toolkit, you need to download the installer from the official web-site [5]. Download is free and requires registration. The registration key will be available after downloading. It is not necessary to save this key; the software works without activation.

3.4 Installing the additional Python modules

To infer deep models using the OpenVINO Toolkit, you need to convert the model from the original framework to the OpenVINO intermediate representation (IR). It requires to install the current version of the training framework. You are able to install the framework you are interested in or all frameworks at once using one of the following commands.

Installing all frameworks:

```
pip install -r "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\model_optimizer\requirements.
txt"
```

Only Caffe:

```
pip install -r "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\model_optimizer\requirements_
caffe.txt"
```

To download pre-trained models from the Open Model Zoo repository, you need to install dependencies required for network manipulations from Python.

```
pip install -r "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\open_model_zoo\tools\download
er\requirements.in"
```

After these steps, the OpenVINO is ready for the further use.

4 Executing the OpenVINO samples and demos in Python

4.1 Configuring the OpenVINO environment

When the OpenVINO Toolkit is installed and the Python virtual environment is created and activated, it is required to add the OpenVINO Python libraries to the **PATH** environment variable using one of the following commands.

For Windows:

```
"C:\Program Files (x86)\IntelSWTools\openvino\bin\setupvars.bat"
```

For Linux:

```
source opt/intel/openvino/bin/setupvars.sh
```

4.2 Downloading model

Open Model Zoo [3] is a repository containing a large number of pre-trained deep models that can be executed using the OpenVINO Toolkit. This repository stores models and parameters for converting models from different frameworks into the intermediate representation.

To download pre-trained models from the Open Model Zoo repository, you need to use the Model Downloader tool. Execute the **download.py** script to use this tool.

```
python "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\tools\model_downloader\downlo
ader.py" --name <model_name> --output_dir <destination_folder>
```

where **<model_name>** is the name of the model to download, and **<destination_folder>** is the directory where to download the model.

The list of all models available for downloading can be obtained using the key **--print_all** when executing the script:

```
python "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\tools\model_downloader\downlo
ader.py" --print_all
```

To solve the classification problem, you can use the ResNet-50 model, which is a good compromise between performance and quality.

4.3 Converting model

To convert downloaded models, it is required to execute the Model Optimizer tool using the **converter.py** script.

```
python "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\tools\model_downloader\conver
ter.py" --name <model_name> --download_dir <destination_folder>
```

To convert your own models, you need to add additional converter parameters and use the **mo.py** module, which will be described later.

4.4 Executing image classification samples

The OpenVINO Toolkit contains the **classification_sample.py** file, which performs classification of any image using a deep neural network. A full description of this sample and user guide are available on the official web-site [7].

To execute the sample, please, download and convert the ResNet-50 model, the sequence of commands is given below, you only need to replace the paths in angle brackets with the real paths of your computer.

```
python "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\tools\model_downloader\downlo
ader.py" --name resnet-50 --output_dir <destination_folder>
python "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\tools\model_downloader\conver
ter.py" --name resnet-50 --download_dir <destination_folder>
python "C:\Program Files
(x86)\IntelSWTools\openvino\inference_engine\samples\python_samples\classi
fication_sample\classification_sample.py" -i <path_to_image> -m
<path_to_model>\resnet-50.xml
```

After starting the sample, the following output should appear in the console.

```
[ INFO ] Creating Inference Engine
[ INFO ] Loading network files:
        resnet-50.xml
        resnet-50.bin
[ INFO ] Preparing input blobs
[ WARNING ] Image dog.jpg is resized from (718, 937) to (224, 224)
[ 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
------- -----------
  250    0.3644813
  248    0.1609627
  267    0.1092821
  249    0.0567346
  222    0.0396488
  247    0.0295657
  233    0.0204531
  228    0.0150599
  188    0.0118753
  174    0.0112558

[ INFO ] This sample is an API example, for any performance measurements
please
```

Source code of samples can be used to learn the programming interface of the Inference Engine component. The next section provides the step-by-step tutorial for developing your own application for image classification.

5 Developing the image classification application using the OpenVINO Toolkit

5.1 File structure

For the first practice, please, create two files**:** **`ie_classifier.py`** is a file containing the **`InferenceEngineClassifier`** class with **`_prepare_image`**, **`classify`**, **`get_top`** methods to implement image classification, and **`classification_sample.py`** is a file containing the testing code for the **`InferenceEngineClassifier`**.

Methods of the **`InferenceEngineClassifier`** class:

- **_init_** is a constructor, it initializes the Inference Engine and loads the model from file.
- **_prepare_image** is a method to convert the image into the deep model input array.
- **classify** is a method for image classification using the deep model.
- **get_top** is a method to select N best classification results (with maximum confidence).

```
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
```

We separate the class and the testing code for this class, since the **InferenceEngineClassifier** class will be needed as part of the following practices. Further, we consider the implementation of each method of the specified class and testing code.

5.2 Loading model

In order to load the model, we need to implement the constructor of the **InferenceEngineClassifier** class placed in the **ie_classifier.py** file. The constructor receives the following required and optional parameters:

- **configPath** is a path to the .xml file of the model description.
- **weightsPath** is a path to the .bin file of the model weights.
- **classesPath** is a path to the file containing class names for the given classification model.

The constructor performs the following actions:

- Creating an object of the class **IECore**.
- Creating an object of the class **IENetwork** with parameters corresponding to the model paths.
- Loading the created object of the **IENetwork** class into **IECore**, this means loading the model into the plugin.
- Loading the class names from the file located at path **classesPath**.

5.3 Loading and preprocessing image

The next step is to implement the **_prepare_image** method. Deep models require images in a per-channel format, and not pixel-by-pixel format, input images have to be converted from the format RGBRGBRG... to the format RRRGGGBBB... You can use the **transpose** function to do this.

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

It is also necessary to resize the image to the size of the network input.

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

In common, a 4-dimensional tensor should be set to the model input, for example, tensor [1,3,227,227], where the first coordinate is the number of images in a batch (subset of images processed simultaneously); 3 is the number of color channels of the image; 227, 227 are width and height of the image. However, if a 3-dimensional tensor [3,227,227] is set to the network input, then the OpenVINO Toolkit will automatically add the fourth dimension.

It is also worth remembering one fact about the OpenCV library. OpenCV stores images in a BGR format, not RGB. If the model is loaded from the Open Model Zoo and converted with default parameters, then this moment is already taken into account the model. However, if the model is not used from the Open Model Zoo, then the red and blue channels of the image have to be swapped.

5.4 Inferring model

The next step is the implementation of the **classify** method, which launches the deep model inference on the device specified in the constructor. The sequence of operations for the **classify** method is as follows:

1. Get information about the model input and output.

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

2. From the model input, obtain the input dimension required by the model for the image.

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

3. Preprocess image using the function **_prepare_image**.
4. Infer the model in synchronous mode.

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

5. Extract the tensor with the classification result from the model output.

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

5.5 Processing model output

To process the output, it is necessary to implement the **_get_top** function in order to extract the most probable N classes predicted by the neural network. To derive top-N probabilities, probabilities can be sorted in ascending order. You can use the **numpy.argsort** function for implementation. It is worth noting that the **argsort** function receives a one-dimensional tensor as an input. If the input tensor has a size [1,1000], then it is necessary to convert it to a tensor of the size [1000]. It is required to establish compliance with the list of classes contained in the file, the path to which is passed as an input parameter to the constructor.

5.6 Implementing sample

5.6.1 Parsing command line arguments

When working with Python programs, it is the best practice to use the command line and execute scripts with named arguments. In Python, the **argrparse** package supports named arguments processing. This package allows you to describe the name, type, and other parameters for each argument. We recommend to implement the **build_argparser** function that will create an **ArgumentParser** object to process command line arguments.

In this practice, the following command line arguments will be required:

- Path to the input image (required).
- Path to the model weights file (required).
- Path to the model configuration file (required).
- Path to the file containing class names (optional).

```
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
```

5.6.2 Implementing main function

In the file **classification_sample.py** create a function **main** that implements the following steps:

1. Parsing command line arguments.
2. Creating an object of the **InferenceEngineClassifier** class with the necessary parameters.
3. Reading the image.
4. Classifying the image.
5. Displaying of the classification result to the screen.

To output logs, use the **logging** package.

```
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())
```

6 Executing developed sample

The easiest way to execute your sample is the command line.

```
python ie_classification_sample.py -i image.jpg -m resnet-50.xml \
    -w resnet-50.bin -c imagenet_synset_words.txt
```

The **-i** argument specifies the path to the image, the **-m** argument specifies the model configuration path, the **-w** argument specifies the model weights path, the **-c** argument specifies the file containing a list of classes for the model.

The result of application execution is as follows. A message about the start of the application is displayed, then a list of classes and their scores are represented.

```
[ 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]]
```

7 Additional tasks

The developed classification sample contains the minimum required functionality. As additional tasks, it is proposed to provide support for the following features:

1. Support for the classification of not only one image, but also several images.
2. Support for the output of deep models, not only on the CPU, but also on the Intel Processor Graphics or Neural Compute Stick (if it is available).
3. Support for asynchronous inference mode of deep models.

It is proposed to solve these tasks independently using the documentation and examples included in the OpenVINO Toolkit.

8 Literature

8.1 Books

1. Chollet. F. Deep Learning with Python. – Manning Publications Co, NY, USA, – 2017.

8.2 Further reading

2. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming. – O'Reilly Media, Inc., CA, USA, 2015.

8.3 References

3. Open Model Zoo home page [https://github.com/opencv/open_model_zoo].
4. Python 3.7.5 download page [https://www.python.org/downloads/release/python-375].
5. Intel Distribution of OpenVINO Toolkit download page [https://software.intel.com/en-us/openvino-toolkit/choose-download].
6. Intel Distribution of OpenVINO Toolkit documentation [https://docs.openvinotoolkit.org/2019_R3.1/index.html].
7. OpenVINO classification sample [https://docs.openvinotoolkit.org/latest/_inference_engine_ie_bridges_python_sample_classification_sample_README.html].