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

Educational course «Introduction to deep learning using the Intel® neon™ Framework»

Practice №0 Preprocessing and converting data to HDF5 format for the Intel® neon™ Framework

Supported by Intel

Zhiltsov Maxim, Kustikova Valentina

Nizhny Novgorod
2018

Content

1 Introduction

This practice is a preparatory one and allows you to create an infrastructure for the future work. Here is a description of the installing the Intel® neon™ Framework (neon) [4] and setting up the environment to train and test deep neural networks.

The implementation of all practices is demonstrated by the example of the problem of classifying a person's sex (male, female) from a photo. Thus, further the problem of classification with two categories is solved. As a train and test datasets, the IMDB-WIKI set [5] is used. For the specified set, the images are preprocessed, as well as the conversion of data and labels to the HDF5 format accepted by the neon is implemented. In the subsequent practices it is assumed that the dataset has been prepared and you have to load it. The procedure of data preprocessing and converting to HDF5 format can be transferred to the case of another task and correspondingly other data.

2 Guidelines

2.1 Goals and tasks

The goal of this practice is to represent a general concept of working with the Intel® neon™ Framework and prepare an environment for subsequent practices.

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

1. Install the Intel® neon™ Framework and its dependencies.
2. Configure the environment variables.
3. Study the structure of the Intel® neon™ Framework.
4. Prepare data for the further practice.

2.2 Practice structure

The practice provides guidance on installing the neon framework and the required dependencies. The tutorial contains a description of the command sequence to be performed from the command line for installing neon and setting up the environment for conducting experiments on training and testing deep models. Further, a source code is developed that provides to load IMDB-WIKI dataset [5] for solving the problem of classifying a person's sex from a photo.

2.3 Recommended study sequence

The recommended study sequence is as follows:

1. Install the Intel® neon™ Framework and its dependencies.
2. Configure the environment variables.
3. Study the structure of the Intel® neon™ Framework and the typical workflow for a deep model, based on the course's lecture material and additional sources.
4. Develop scripts for the preparing the data for the further practice, verify its correctness.

3 Manual

3.1 Installing the Intel® neon™ Framework and its dependencies

3.1.1 List of dependencies

Intel® neon™ Framework is developed for Python 2 and 3, runs on Linux and Mac OS platforms. Allows to work on the CPU with the possible use of the Intel® Math Kernel Library and GPU using NVIDIA CUDA.

The list of framework dependencies includes the following libraries and modules:

- Python 2.7+/3.4+ [6],
- Python-pip [7],
- Python-virtualenv [8],
- libhdf5-dev [9],
- (optional) NVIDIA CUDA (8.0) [10],

− (optional) Intel® Math Kernel Library (Intel® MKL) [11].

Further, there is the command sequence to install neon under Linux. Additional installation instructions are represented in the official framework documentation [12].

3.1.2 Python 3

To install and build Python 3, you will need to download the source code of the interpreter [13]. Building Python 3 is performed by a standard set of steps.

```
./configure
make
make install
```

You can see the various build parameters by calling the command:

```
./configure --help
```

You also need to update the **pip** module and install the **virtualenv** module. You can use the following commands:

```
pip install --upgrade pip
pip install virtualenv
```

3.1.3 Intel® neon™ Framework

The first step to install neon framework is to download and build its sources. You can use the following commands.

```
git clone https://github.com/NervanaSystems/neon.git
cd neon; git checkout latest; make python3
```

After that in the build directory, a directory of the virtual environment **.venv** will be created, containing all the necessary Python modules for the library. For convenience in the future work, it is recommended to create a copy or a symbolic link for the virtual environment directory in the directory of practice.

```
cp neon_source_path/.venv .venv # to create a copy
ln -s neon_source_path/.venv .venv # to create a symbolic link
```

To use the virtual environment, you must activate it. To do this, follow the command below.

```
. .venv/bin/activate
```

In this case, the form of the command line should change. From this point, the **python**, **pip**, and other commands relate to an interpreter located in a virtual environment. Python modules will be copied from the source interpreter. The virtual environment is terminated using the deactivation command.

```
deactivate
```

3.1.4 Additional modules

Several additional modules will be required for the further practice. They can be installed by the command shown below. You must first activate the virtual environment.

```
pip install -r Practice/requirements.txt
```

3.2 Configuring the environment variables

3.2.1 Using the Intel® Math Kernel Library

If you want to use the Intel® MKL [11], the following environment variables may be useful for improving performance:

```
export KMP_AFFINITY=compact,1,0,granularity=fine
export OMP_NUM_THREADS=<Number of Physical Cores>
```

The variable **KMP_AFFINITY** specifies how to assign threads of the MKL library on the physical cores of the processor. The variable **OMP_NUM_THREADS** captures the maximum number of threads used by OpenMP. Additional information on these parameters can be found on the Intel® MKL website [14].

3.2.2 Using NVIDIA CUDA

If you want to use NVIDIA CUDA [10] and GPU, then you need to set the following environment variables:

```
export PATH=/usr/local/cuda/bin:$PATH
```

```
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:\
          /usr/local/cuda/lib:/usr/local/lib:$LD_LIBRARY_PATH
```

These variables are responsible for searching the executable files and libraries. For convenience, these variables can be written to the file **`~./bashrc`** so that they are automatically set when the user logs on.

3.2.3 Allow access to the components

To use the components of the practices, you need to add them to the Python search paths.

```
export PYTHONPATH=/<full_path_to_practice>:$PYTHONPATH
```

3.3 Preparing data for the further practice

3.3.1 Preprocessing of visual data and data management in the Intel® neon™ Framework

Preprocessing of visual data may involve the following transforms:

- Convert images to the same size by cutting, zooming or adding borders.
- Subtract the mean value of pixel intensities, dividing by the standard deviation.
- Remove incorrect input and output data.
- Extend the dataset through reflections, turns, shifts, scaling.
- Add noise to images.
- Balance the number of samples in the different classes.

During the data preparation, the set is also divided into train and test sets, if they are not initially represented.

The result of this stage is data in a format that is compatible with the used library. The Intel® neon™ Framework allows you to work with data in several ways [15]:

- Storing all data in the device memory (type **`ArrayIterator`**).
- Storing data in the external memory, loading small pieces of data into the device memory (type **`HDF5Iterator`** and Aeon dataloaders [16]).

Also, for widely known datasets, embedded loaders are provided [17]. Data is provided to neon through iterators. Further, **`HDF5Iterator`** is created.

3.3.2 Preparing the IMDB-WIKI dataset for solving the problem of person's sex classification

In the course practice, the IMDB-WIKI dataset [5] is used. The set contains about 60,000 images. Along with the images, the set contains metadata, which includes various additional information, such as the face position, the person's sex in the photo, and the time of photographing. The image size in the set is not fixed and varies from 32x32 to 512x512 pixels, so additional actions are needed to prepare the images.

1. Download the dataset. It is performed by calling the two commands listed below.

```
# ~1 GB
wget https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/static/wiki_crop.tar
tar -xf wiki_crop.tar
```

2. Convert images to the same size. In this step, a combination of a cutting from the center of the image for large images and an adding the neutral elements for small images is used. The approach does not guarantee that the face will remain in the image, but with a sufficiently large cut the presence of the face can be guaranteed with a high probability. A more correct approach is to use data about the position of a person extracted from the metadata. The **`wiki_crop/wiki.mat`** metadata file is a mat-file and can be opened using the SciPy module [18]. The file contains an associative array of variables, which includes the variable **`wiki`**. A variable is a structure containing fields defined in metadata [5]. For the task of classifying persons' sex, the following fields are of interest:
   - The 2d field contains an array of directories to the input images. The array shape is $1xN$, where N is a number of images.
   - The 3d field contains an array with labels. The values are 0, 1 and **`NaN`** of the **`float`** type. The shape of the array is $1xN$.

   To work with images, the Pillow module is used [19]. The work with the data arrays is carried out through the NumPy module [20].

```
from scipy.io import loadmat
import numpy as np
from PIL import Image
```

Using the description of the metadata [5], load the image and the label:

```
metadata = loadmat(dataset_root + '/wiki_crop.mat')['wiki'][0][0]

i = 0 # index of the element
image = Image.open(dataset_root + '/' + metadata[2][0][i][0])
gender = metadata[3][0][i].astype(int)
```

Convert all data to RGB format:

```
if (len(image.getbands()) != 3):
    image = image.convert("RGB")
```

Cut the image center or append image borders:

```
def center_crop(image, crop_size):
    return image.crop((
        (image.size[0] - crop_size[0]) / 2,
        (image.size[1] - crop_size[1]) / 2,
        (image.size[0] + crop_size[0]) / 2,
        (image.size[1] + crop_size[1]) / 2,
        ))

image_size = (128, 128)
image = center_crop(image, image_size)
```

3. Creating train and test datasets. At this step, the dataset is divided into a train and test subsets. This is done manually, because the authors of the subset set do not regulate it. We will create a set of element indices of the set:

```
genders = metadata[3][0]
indices = []
```

The description of the set indicates that some images may contain undefined gender tags. Remove such records from the dataset:

```
import math

for i, gender in enumerate(genders):
    if (math.isnan(gender) == True):
        continue
    indices.append(i)
```

We interleave the indices at random and separate the datasets:

```
import random

random.shuffle(indices)

train_ratio = 0.66
threshold_index = int(train_ratio * len(indices))
train_indices = indices[: threshold_index]
test_indices = indices[threshold_index :]
```

4. Computing mean pixel intensity. To improve the convergence properties of optimization algorithms, we transform the data to the zero mean value. We compute the mean value over the dataset for each of the image channels.

```
def compute_mean(indices, metadata):
    channel_count = 3 # RGB
    channels_mean = np.zeros(channel_count)

    for pos, i in enumerate(indices):
        image = Image.open(dataset_root + '/' + metadata[2][0][i][0])
```

```
        if (len(image.getbands()) != 3):
            image = image.convert("RGB")
        image = center_crop(image, image_size)

        # convert the representation from (H, W, C) to (C, H, W)
        image_array = np.array(image, dtype=np.int8).transpose((2, 0, 1))
        channels_mean += np.mean(image_array, axis=(1, 2))

    channels_mean = channels_mean / len(indices)
    return channels_mean
```

5. Saving the data. By this time, the data has been preprocessed, but for use during the training of deep models it is necessary to save them in the required format. Consider the option of storing data in external memory and loading fragments as needed. neon supports working with data in HDF5 format, providing the iterator **`HDF5Iterator`**. Convert the dataset to this format using the h5py module. Following the documentation [21], we save each subset of the data in a separate file of the HDF5 format. These files should have the following structure:
   - An input array containing input images in the format $(N, C \times H \times W)$ of the type **`float`** or **`numpy.int8`**, where N is the number of samples in the dataset, C is the number of image channels, $H \times W$ is the height and width of the images.
   - The **`lshape`** attribute of the **`input`** array, which describes the way data is interpreted. It is a tuple (C, H, W).
   - The optional **`mean`** attribute of the **`input`** array containing the average pixel values over the channels. The form of the attribute data is $(C, 1)$.
   - An **`output`** array containing output values. In our case, these are labels of the size $(N, 1)$ and of the type **`numpy.int8`**.
   - The optional **`nclass`** attribute of the **`output`** array containing the number of classes. Used in classification problems.

   Sample code for saving data for train dataset:

```
import h5py as h5

channels_mean = compute_mean(train_indices, metadata)
indices = train_indices
dataset_file = h5.File(save_dir + '/train.h5', 'w')

input_channels_count = 3
input_channel_size = image_size[0] * image_size[1]
input_size = input_channels_count * input_channel_size

dataset_inputs = dataset_file.create_dataset('input',
    (len(indices), input_size), dtype=np.int8)
dataset_inputs.attrs['lshape'] = (input_channels_count,
    image_size[1], image_size[0])
dataset_outputs = dataset_file.create_dataset('output',
    (len(indices), 1), dtype=int)
dataset_outputs.attrs['nclass'] = 2

for pos, i in enumerate(indices):
    image = Image.open(dataset_root + '/' + metadata[2][0][i][0])
    if (len(image.getbands()) != 3):
        image = image.convert("RGB")
    image = center_crop(image, image_size)

    image_array = np.array(image, dtype=np.int8).transpose((2, 0, 1))
    dataset_inputs[pos] = image_array.flatten()
    dataset_outputs[pos] = gender
```

```
dataset_inputs.attrs['mean'] = channels_mean / len(indices)

dataset_file.close()
```

6. Checking correctness of the saved data. By this time, the data has been saved in the correct format. It remains to verify that neon is able to load them.

```
import neon.backends
neon.backends.gen_backend('cpu', batch_size=1) # initialization

from neon.data import HDF5Iterator
train_iter = HDF5Iterator(save_dir + '/train.h5')
```

Try to display the first element of the set:

```
import numpy as np
import PIL.Image

train_it = iter(train_iter) # create the iterator
entry = next(train_it) # the pair (input, target) is in the device memory
image_data = entry[0].get()
# image_data.shape == (C, H, W, batch_size)
gender_data = entry[1].get()
# gender_data.shape == (1, batch_size)

image = PIL.Image.fromarray(image_data.astype('i1') \
   .reshape((3, 128, 128)).transpose((1, 2, 0)), 'RGB')
gender = gender_data[0]

print(gender)
image.show()
```

After a successful check, the data can be considered as prepared for the further practice.

The complete sources for this example is found in course materials:
`Practice0_intro/imdb_wiki_converter.py` и **`datasets/imdb_wiki_face_dataset.py`**.

4 Literature

4.1 Books

1. Haykin S. Neural Networks: A Comprehensive Foundation. – Prentice Hall PTR Upper Saddle River, NJ, USA. – 1998.
2. Osovsky S. Neural networks for information processing. – 2002.
3. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].

4.2 References

4. Overview of the Intel® neon™ Framework [http://neon.nervanasys.com/docs/latest/overview.html].
5. IMDB-WIKI dataset [https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki].
6. Python [https://www.python.org].
7. Python-pip [https://pypi.org/project/pip].
8. Python-virtualenv [https://virtualenv.pypa.io/en/stable].
9. HDF5 [https://support.hdfgroup.org/HDF5].
10. NVIDIA CUDA [https://developer.nvidia.com/cuda-downloads].
11. Intel® Math Kernel Library [https://software.intel.com/en-us/mkl].
12. Installation of the Intel® neon™ Framework [http://neon.nervanasys.com/docs/latest/installation.html].
13. Python 3.5.3 [https://www.python.org/downloads/release/python-353].

14. Intel® Math Kernel Library: KMP_Affinity [https://software.intel.com/en-us/node/522691].
15. Load data in the Intel® neon™ Framework [http://neon.nervanasys.com/docs/latest/loading_data.html].
16. Aeon dataloader [http://neon.nervanasys.com/docs/latest/loading_data.html#aeon-dataloader].
17. Intel® neon™ Framework: load well-known datasets [http://neon.nervanasys.com/docs/latest/datasets.html].
18. SciPy [https://www.scipy.org].
19. Pillow [https://pillow.readthedocs.io/en/3.1.x/index.html].
20. NumPy [http://www.numpy.org].
21. Intel® neon™ Framework: HDF5Iterator type [http://neon.nervanasys.com/docs/latest/generated/neon.data.hdf5iterator.HDF5Iterator.html#neon.data.hdf5iterator.HDF5Iterator].