Classification of Zooplankton Images Using Deep Learning

In this section, we provide more details on the classification of zooplankton images using deep learning and we explain how to train and use the models.

The models considered so far are InceptionV3 (see the following paper) und ResNet101 (see this paper).

Model Training under Linux

The source code for the deep learning models can be checked out from the internal gitlab repository. Under Linux you have to run:

```
$ git clone https://github.com/o2a-data/o2a-data-ml.git
```

The models are provided both in the form of Jupyter notebooks (to be found in the notebooks subfolder) and as Python scripts (example-scripts subfolder).

Preparation of the Anaconda Environment

In order to work with the provided models, you need a suitable Anaconda environment. We assume you have already installed anaconda (see Anaconda documentation), e.g. under /opt/anaconda. Open a shell and change your working directory to the path containing the project, e.g.

```
$ cd /home/user/machineLearningExample
```

Activate anaconda (in case it is not automatically activated by your shell) and create the new environment with:

```
$ source /opt/miniconda/bin/activate
$ conda env create --name ml --file conf/environment.yml
```

where lokiml is the name of a new conda environment to be used by your project. After successfully creating the new environment, activate it each time you want to work with this project in the Linux terminal:

```
$ conda activate ml
```

Training the Models using Jupyter Notebooks

There are three notebooks that are used for data pre-processing, and several other notebooks that provide variations of the InceptionV3 and ResNet101 models:

```
preprocessing_lokiDataset_deepLearning_step1_classSize.ipynb
preprocessing_lokiDataset_deepLearning_step2_imagePreparation.ipynb
preprocessing_lokiDataset_deepLearning_step3_train_valid_test_split.ipynb
classification_loki_deepLearning_InceptionV3-preTrained-rmsprop-1000.ipynb
classification_loki_deepLearning_InceptionV3-preTrained-rmsprop.ipynb
classification_loki_deepLearning_InceptionV3-selfTrained-rmsprop-1000.ipynb
classification_loki_deepLearning_resNet-preTrained-101.ipynb
classification_loki_deepLearning_resNet-selfTrained-101.ipynb
```

At the beginning of each notebook, a few variables specify where the notebook looks for input data and where it stores its results. You only need to set the variable

```
basePath = '/path/to/folder/containing/dataset/'
```

to point at the folder where you have your dataset and all other paths will be set accordingly. In order to obtain a dataset you can use for training, you can contact the maintainers of this documentation.

Besides setting the basePath variable in each notebook, make sure that the notebook is using the environment (kernel) you have created in the previous section. You can now run the notebooks as usual.

Training the Models from the Linux Terminal

For each Jupyter notebook we provide a corresponding Python script that can be run directly from the terminal. These scripts must also be configured by setting the basePath variable. After activating the conda environment with

```
$ conda activate ml
```

you can start the scripts with

```
$ python example-scripts/preprocessing_lokiDataset_deepLearning_step1_classSize.py
$ python example-scripts/preprocessing_lokiDataset_deepLearning_step2_imagePreparation.py
$ python example-scripts/preprocessing_lokiDataset_deepLearning_step3_train_valid_test_split.py
$ python example-scripts/classification_loki_deepLearning_InceptionV3-preTrained-rmsprop-1000.py
```

for the InceptionV3 pre-trained model and

```
$ python example-scripts/preprocessing_lokiDataset_deepLearning_step1_classSize.py
$ python example-scripts/preprocessing_lokiDataset_deepLearning_step2_imagePreparation.py
$ python example-scripts/preprocessing_lokiDataset_deepLearning_step3_train_valid_test_split.py
$ python example-scripts/classification_loki_deepLearning_resNet-preTrained-101.py
```

for the resNet101 pre-trained model.

Methods

In this section, we consider the implementation of the deep learning models from a programming point of view. For general information about the project please refer to Stella Mahler's bachelor's thesis. You can find more detailed information to the methods used for data preparation and for the training in the annotated notebooks:

```
preprocessing_lokiDataset_deepLearning_step1_classSize-annotated.ipynb
preprocessing_lokiDataset_deepLearning_step2_imagePreparation-annotated.ipynb
preprocessing_lokiDataset_deepLearning_step3_train_valid_test_split-annotated.ipynb
classification_loki_deepLearning_InceptionV3-annotated-de.ipynb
classification_loki_deepLearning_InceptionV3-annotated-en.ipynb
```

Input data and preprocessing

The images to be processed must be stored as png or jpeg files. The metadata must be stored as an Ecotaxa tsv file. During preprocessing, the images are distributed into classes. Classes that contain too many samples are cut while classes that contain too few samples are augmented. Images are then scaled to the same size and centred. Finally, the remaining images are split into a training and a test set.

Models

In this section, we sketch how a CNN model is customized so that the reader can have a starting point for adapting a model to their own needs. The complete source code is found in the project itself.

InceptionV3

InceptionV3 is provided by Keras (see the Keras documentation). You can import it in your code with

```
from keras.applications.inception_v3 import InceptionV3
```

This model can be used in different ways. First of all, it can be used with pre-trained weights:

```
base_model = InceptionV3(weights='imagenet', include_top=False)
```

In this case, the model is pre-trained with images from the ImageNet project, a large collection of annotated images. As an alternative, the model can be constructed without pre-trained weights like so

```
base_model = InceptionV3(weights=None, include_top=False)
```

In both cases, we set include_top=False to indicate that we want to add a classification layer of our own.

We then add a few more layer (including a classification layer) and embed the result in the final model:

```
output = base_model.output
pooling = GlobalAveragePooling2D(name='avg_pool')(output)
dropout = Dropout(0.4)(pooling)
predictions = Dense(num_classes, activation='softmax')(dropout)
model = Model(inputs=base_model.input, outputs=predictions
```

ResNet101

This model is also provided by Keras (see the Keras documentation). You can import it in your code with

from keras_applications.resnet import ResNet101, preprocess_input

As with the InceptionV3 model, we can use a model with pretrained weights or train the mode ourselves, e.g.

```
base_model = ResNet101(weights='imagenet', include_top=False, input_shape=(1000,1000,3)) # pre-trained
base_model = ResNet101(weights=None, include_top=False, input_shape=(1000,1000,3)) # self-trained
```

Again, we can add the missing extra layers that are used for the classification, e.g.

```
output = base_model.output
pooling = GlobalAveragePooling2D(name='avg_pool')(output)
dropout = Dropout(0.4)(pooling)
predictions = Dense(num_classes, activation='softmax')(dropout)
model = Model(inputs=base_model.input, outputs=predictions)
```

For the chosen application (classification of plankton images), the ResNet101 model turned out to be a good compromise between accuracy and training time.