DEEPHEALTH

Tensor Manipulation and Data Augmentation with ECVL

Costantino Grana - UNIMORE

Winter School 25/01/2022

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Contents

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

From ECVL to EDDL

- ECVL allows easy integration and exchange of data with EDDL.
- The ECVL - EDDL interfacing is based on two main functions: *ImageToTensor* and *TensorToImage*.
- *ImageToTensor* transforms an ECVL Image in a EDDL Tensor, converting its data to floating point numbers and rearranging its channels to "xy[c/z/o]", which is how EDDL Tensor handles data in color images. Finally, Image data are copied into Tensor data.

```
import pyecvl.ecvl as ecvl
from pyeddl.tensor import Tensor

image = ecvl.ImRead("/mnt/data/winter_school/lena.png")
t = ecvl.ImageToTensor(image)
```


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

From EDDL to ECVL

- *TensorToImage* creates a float "xyo" raw Image with *none* color space, and copies Tensor data into the Image data.
- *TensorToView* is also available in order to avoid the copy of the data.

```python
import pyecvl.ecvl as ecvl
from pyeddl.tensor import Tensor

image = ecvl.ImRead("/mnt/data/winter_school/lena.png")
t = ecvl.ImageToTensor(image)
image_from_tensor = ecvl.TensorToImage(t)
view_from_tensor = ecvl.TensorToView(t)
```


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Dataset Format

- A simple and flexible YAML syntax to describe a dataset for the DeepHealth libraries (EDDL/ECVL), in order to provide a unified way for loading data.
- Complete description: [DeepHealth-Toolkit-Dataset-Format]()
- Optional elements: *name*, *description*, *classes* (mandatory in case of a classification task), *features* (additional information related to each image), *split*.

```yaml
# Example of DeepHealth toolkit dataset format
# Arrays are always 0 based

# Descriptive string used just for pretty reporting (optional)
name: dataset_name

# Descriptive string to document the file (optional)
description: >
  This is an example of long
  text which describes the use of this dataset and
  whatever I want to annotate.

  You can also write multiple paragraphs with the only
  care of indenting them correctly.

# Array of class names (optional)
classes: [class_a, class_b, class_c]

# Array of features names (optional)
features: [feature_1, feature_2, feature_3, feature_4]

# Split (optional) is a dictionary with a custom number of arrays.
# They list the indexes of the images to be used in different phases.
split:
  training: [0, 1]
  validation: [2]
  test: [3]
```


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Dataset Format

- Mandatory element: the list of the *images* with their *location* and optionally their *label* (a class or a path) and *values*.

```yaml
images:
# label can be a class name (string)...
# values can be an array with a positional correspondence with the features array...
  - location: image_path_and_name_1
    label: class_b
    values: [value_1, null, value_3, null]

# ... or the class index (integer) wrt the classes array
# ... or a dictionary with the name of the feature coupled with its value
  - location: image_path_and_name_2
    label: 2
    values: { feature_1: value_1, feature_3: value_3 }

# In the case of multi class problems, label can be an array of class names (array of strings)...
  - location: image_path_and_name_3
    label: [class_a, class_c]

# ... or an array of class indexes (array of integers)
  - location: image_path_and_name_4
    label: [0, 2]
# label can be a path (string) to an image in case of a segmentation task
  - location: image_path_and_name_5
    label: path_to_ground_truth_image

# Remember that labels are optional
  - location: image_path_and_name_6
  - location: image_path_and_name_7
```

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Dataset

- ECVL also provides a *Generator* which creates a Dataset file from a directory tree (Dataset-Generator)

```
gcd = ecvl.GenerateClassificationDataset(input_directory)
cls_d = gcd.GetDataset()
cls_d.Dump("classification_dataset.yml")
```

- ECVL allows to parse the Dataset YAML file to create a Dataset object.

```
input_dataset = "/mnt/data/winter_school/mnist.yml"
d = ecvl.Dataset(input_dataset)
print("name:", d.name_)
print("classes:", d.classes_)
print("features:", d.features_)
print("n. samples:", len(d.samples_))
```


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

DEEPHEALTH

Augmentation

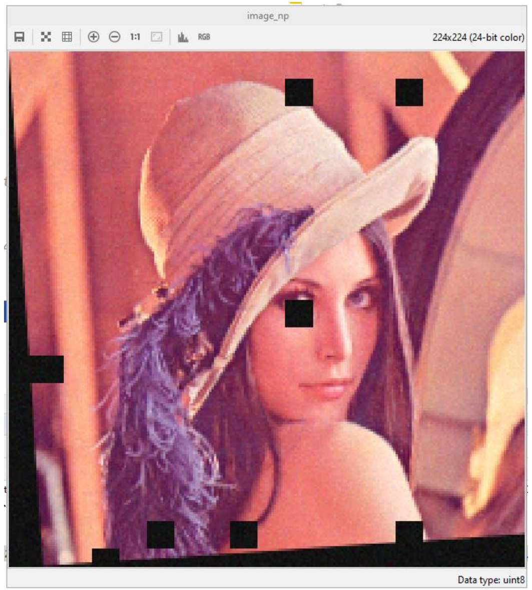
- Data Augmentations can be loaded from text…

```
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("/mnt/data/winter_school/lena.png")

AUG_TXT = '''\
SequentialAugmentationContainer
    AugResizeDim dims=(224,224) interp="linear"
    AugRotate angle=[-5,5] center=(0,0) interp="linear"
    AugAdditiveLaplaceNoise std_dev=[0,0.51]
    AugCoarseDropout p=[0,0.05] drop_size=[0.02,0.1] per_channel=0
    AugAdditivePoissonNoise lambda=[0,40]
end
'''

augs_from_text = ecvl.AugmentationFactory.create(AUG_TXT)
augs_from_text.Apply(image)
```



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Augmentation

- …or directly written in the code

```
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("/mnt/data/winter_school/lena.png")
disp_image = ecvl.Image.empty()

training_augs = ecvl.SequentialAugmentationContainer([
    ecvl.AugResizeDim([224, 224]),
    ecvl.AugMirror(.5),
    ecvl.OneOfAugmentationContainer(
        0.3,
        [ecvl.AugElasticTransform([10, 20], [2, 4]),
        ecvl.AugGridDistortion([2, 5], [-0.3, 0.3]),
        ecvl.AugOpticalDistortion([-0.3, 0.3], [-0.1, 0.1])]
    ),
    ecvl.AugRotate([-30, 30]),
])

training_augs.Apply(image)
```

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Augmentation

- The list of all the augmentations is available at https://deephealthproject.github.io/ecvl/master/classecvl_1_1_augmentation.html
- Different augmentations can be applied to each split defined in the Dataset

```
training_augs = ecvl.SequentialAugmentationContainer([
    ecvl.AugResizeDim([224, 224]),
    ecvl.AugMirror(.5),
    ecvl.OneOfAugmentationContainer(
        0.3,
        [ecvl.AugElasticTransform([10, 20], [2, 4]),
        ecvl.AugGridDistortion([2, 5], [-0.3, 0.3]),
        ecvl.AugOpticalDistortion([-0.3, 0.3], [-0.1, 0.1])]
    ),
    ecvl.AugRotate([-30, 30]),
])

validation_augs = ecvl.SequentialAugmentationContainer([
    ecvl.AugResizeDim([224, 224]),
])

test_augs = ecvl.SequentialAugmentationContainer([
    ecvl.AugResizeDim([224, 224]),
])

ds_augs = ecvl.DatasetAugmentations([training_augs, validation_augs, test_augs])
```


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Deep Learning Dataset

- The Dataset object is extended by DLDataset, which contains information for the deep learning functionalities, specifically for generating data to feed the neural network

```
input_dataset = "/mnt/data/winter_school/mnist.yml"
batch_size = 64
dataset_augmentations = ecvl.DatasetAugmentations([training_augs, validation_augs, test_augs])
d = ecvl.DLDataset(input_dataset, batch_size, dataset_augmentations, ctype=ecvl.ColorType.GRAY,
                   num_workers=6, queue_ratio_size=5, drop_last=[True, False, False])
```

- In order to create samples in a non-blocking scenario, *num_workers* threads work in parallel to:
  - loading the images
  - apply augmentations
  - convert them to Tensors
  - push the Tensors to a queue from which the main thread can pull a batch for the network

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

ProduceImageLabel

- These steps are performed by the function *ProduceImageLabel*, which (for now only in C++) can be overwritten to customize the creation of the batches. For example, for a dataset that consists of 3D volumes which we want to push into the queue slice by slice:

```
class CustomDataset : public DLDataset
{
…
void ProduceImageLabel(DatasetAugmentations& augs, Sample& elem) override
{
    Tensor* label_tensor = nullptr, *image_tensor = nullptr;
    Image img = elem.LoadImage(ctype_, false);
    Image gt = elem.LoadImage(ctype_gt_, true);
    const int slices = img.Channels();

    // Apply chain of augmentations to sample image and corresponding ground truth
    augs.Apply(current_split_, img, gt);

    // Push the slice and its ground truth to the queue
    for (int cur_slice = 0; cur_slice < slices; ++cur_slice) {
        View<DataType::int16> v_volume(img, { 0, 0, cur_slice}, { img.Width(), img.Height(), n_channels_ });
        View<DataType::int16> v_gt(gt, { 0, 0, cur_slice}, { gt.Width(), gt.Height(), n_channels_ });
        ImageToTensor(v_volume, image_tensor);
        ImageToTensor(v_gt, label_tensor);

        queue_.Push(elem, image_tensor, label_tensor);
    }
}
…
}
```


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Generate batches

- With the *Start* method the threads are spawned and begin to fill the queue, until the *Stop* method is called.
- The *GetBatch* function will retrieve a vector of samples, which contains information like the name of the files in the batch, the Tensor containing the batch of the prepared images and the Tensor containing the corresponding labels.

```
d.Start()
for b in range(num_batches_train):
    samples, x, y = d.GetBatch()
    eddl.train_batch(net, [x], [y])
    losses = eddl.get_losses(net)
    metrics = eddl.get_metrics(net)

    print(f'Train - epoch [{e + 1}/{args.epochs}] - batch [{b + 1}/{num_batches_train}]'
          f' - {loss_name}={losses[0]:.3f} - {metric_name}={metrics[0]:.3f}', flush=True)
d.Stop()
```


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

DEEPHEALTH

Thank you!

Costantino Grana
costantino.grana@unimore.it

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.