AIIA.CVML instructions for Illustrative Programming Projects

Instructions version 1.0

Department of Informatics, Aristotle University of Thessaloniki
Thessaloniki, Greece, 54124

June 9, 2021

Chapter 1

Guidelines for writing an Illustrative Programming Project

1.1 Purpose

The purpose of these projects is a. Programming Algorithms and b. Presentation of their execution and results with an interactive way. This can be done in the following way:

- Visualize the steps of the algorithm based on the changes of parameters from the user e.g., using a GUI interface.
- Visualize the steps and results of the algorithm e.g., with animated GIF.

The presentation of the process is an important part of the project, therefore, any interesting ways of visualizing your work are acceptable.

1.2 Methods

The programming language that is recommended for these projects is Python, as the tools provided for them are written in Python.

Structure:

Many of the illustrative programming topics have various applications each, e.g., five different projects for the different methods for region segmentation.

Each project will be delivered as a **separate project** including the following:

1. A Directory Exercise with sub-directories of Definitions (define the application, e.g., region split-merge segmentation, type of input data, input and output of each routine, expected output), this can be delivered as a power point presentation (with the corresponding name in the file) and Input Data (data given as input).

2. Directory Help: Provide help to the programmer (if needed). Help can be given in the form of written guidelines in a PDF file and also pseudo-code can be provided to help the user follow the steps of the task (as a .py file).

3. Directory Solution with sub-directories of Code (a Python file .py) and Results (results from execution).

A Graphical User Interface (GUI) can have the form depicted in Figure 1.1. There should be a button to choose an image and show it to the user after the processing. For the processing several buttons with the equivalent filters can be created.

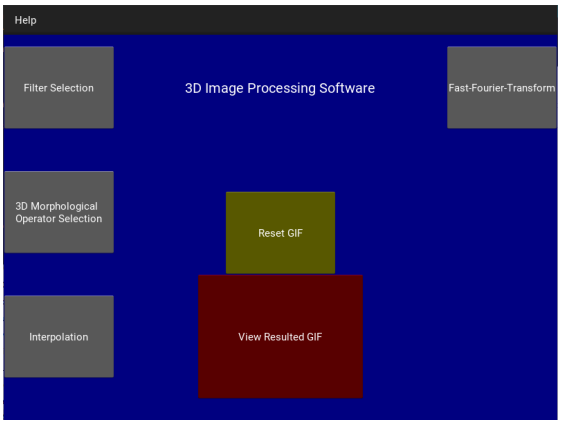

Figure 1.1: Graphical User Interface.

Data and examples can be found in Google drive in the following link:
`https://drive.google.com/drive/folders/1-8PM3lYqbvaXponr6sGbWAGXEWvKF6_w`

ATTENTION!!! Make sure in case you use personal images that are copyright free and suitable for academic purposes.

You can also make the presentation of the task in the form of a notebook, but a power point is also necessary to have a mutual format.

1.3 Tools

The basic library that will be used for these projects is OpenCV. You can simply import it with this command: `import cv2`. The data that will be used for the projects should follow a specific outline. Data will be treated as tensors.

1.3.1 Tensors

A tensor can be represented as a multidimensional array. First, we have 1 Dimensional tensors. A 1D tensor can be considered as 1D array, e.g., `[4,3,7,1]` and can be seen in Figure 1.2 on the left.

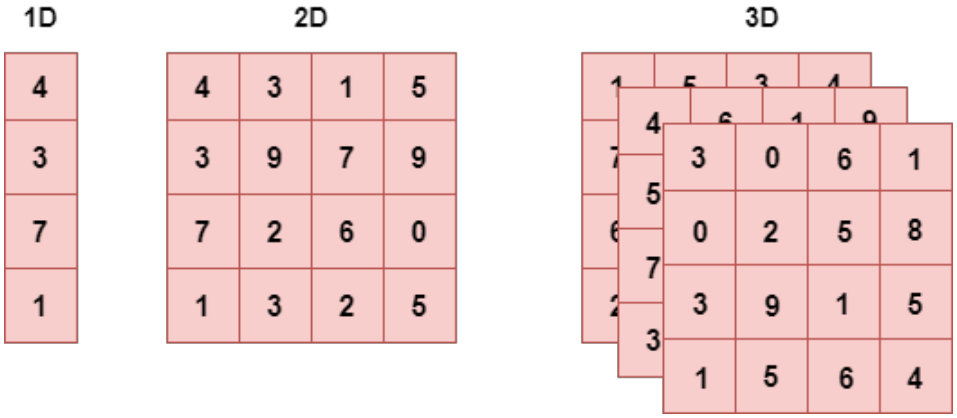

Figure 1.2: Tensors.

In the case of 2D tensors we can consider them as matrices depicting with i the number of rows and j the number of columns. In Figure 1.2 we have an example of 2D tensor (in the middle).

A 3D tensor is consisted of multiple matrices stacked one on top of the other. There are again i rows and j columns for each matrix and the depth represents the number of matrices. In Figure 1.2 (on the rigth) we have an example of 3D tensor.

In Figure 1.3 we have the equivalent example of a 4D tensor.

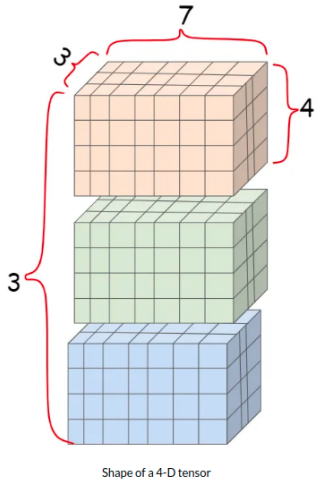

Figure 1.3: 4D tensor.

1.4 Input Data

1.4.1 1D signal

The first type of input is a 1D signal that is simply in the form of a 1 Dimensional array and can be treated as such.

1.4.2 2D Image

Another type of input is a 2D image. We can read an image with the command `image = cv.imread('img')`, where img is the image we insert as an input. The shape of an image can be derived with the command `(h, w, d) = image.shape`, where h is the height, w is the width and d is the depth of the image. In particular the depth depicts the Red-Green-Blue (RGB) channels of the image.

An image can be either RGB (Figure 1.4) or Grayscale (Figure 1.5), in the first condition the channel is equal to 3, whereas in the second the channel is equal to 1.

Figure 1.4: RGB image.

Figure 1.5: Grayscale image.

1.4.3 3D Video

In the case of a video we can read it using the command `video = skvideo.io.vread('video')`. The shape of the video can be derived again with the command `(f, h, w, d) = video.shape`, where f is the number of frames, h is the height, w is the width and d is the Red-Green-Blue (RGB) channel of the video. Each frame of a video can be considered as a single image.

A video can be considered as a sequence of images in time like in Figure 1.6.

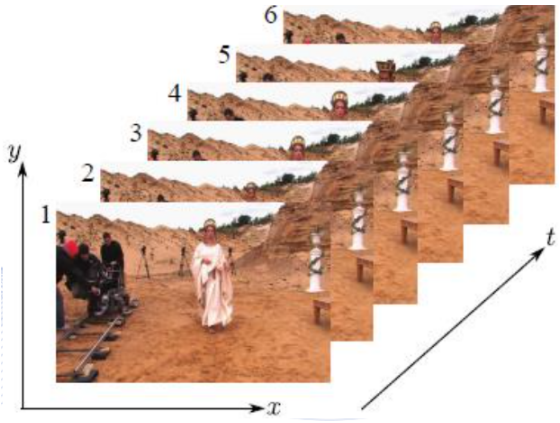

Figure 1.6: Rigid body cube.

In order to have a general format in the projects all tensors should be in the form of a 4D tensor. In the case of 1D signal the tensor will be in the form `(1,h,1,1)`, where the elements in the extra dimensions will be zero. For images, if we have a RGB image the tensor will be `(1, h, w, c)`, while for grayscale images it will be `(1, h, w, 1)`. For the video there wont be any changes as it is already a 4D tensor `(f, h, w, c)`, unless the video is grayscale and it will be written as `(f, h, w, 1)`.

1.4.4 Classification-Clustering

In the case of classification and clustering labels should also be written in a common way. The labels should be written as integers and also provide the number of classes.

E.g., labels from Cifar-10:

```
labels = {'airplane', 'automobile', 'bird', 'cat',
          'deer', 'dog', 'frog', 'horse', 'ship',
          'truck'}.
```

We want the labels to be represented as integers:

```
labels = {0:'airplane',1:'automobile', 2:'bird',
          3:'cat', 4:'deer', 5:'dog', 6:'frog',
          7:'horse', 8:'ship', 9:'truck'}
```

1.4.5 Regression

In the case of regression the labels will be float numbers and as there are no classes we will pass 0 or -1 for the input.

1.5 Processing

The process of images and videos will be done using the tensors. Therefore, the functions can be used for neural networks with Pytorch or TensorFlow. The functions written should be reusable from other users or give the ability to be updated with a new method.

1.5.1 Tensors and Arrays

Pytorch:

In case you are working with Pytorch the commands used to convert a tensor to a numpy array and vice versa are the following.

Array to tensor:

```
tensor = torch.from_numpy(array)
```

Tensor to array:

```
array  = tensor.numpy(tensor)
```

TensorFlow:

If you are working with Tensorflow the equivalent commands are written below.

Array to tensor:

```
tensor = tf.convert_to_tensor(array,
                              dtype=tf.float32)
```

Tensor to array:

```
array = tf.make_ndarray(tensor)
```

1.6 Output Data

After processing all data should be available in their original form. Therefore, 4D tensors should be turned into 1D signals, images or videos accordingly.

In order to load tensors in their initial dimension you can use the following command:

```
np.reshape(array, dimensions)
```

An example of how to use this command is given below.

```
a = np.array([[1,2,3], [4,5,6]])
np.reshape(a, 6)
array([1, 2, 3, 4, 5, 6])
```

To view an image you can use the following command:

```
cv2.imshow(image)
```

To turn an array to a video use the following command (for small videos):

```
skvideo.io.vwrite("output_1.mp4", array)
```