Computer Vision - Autumn 2021 - Home Assignment 2

Ondřej Schejbal & Jan Vladár

In our assigned we deal with problem of image detection and segmentation. We apply different approaches for template matching, descriptor detection and image segmentation of selected sea plants.

For our project we decided to work with these 2 plant species:

- Zostera
- Mytilus

In [ ]:
```
import os
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt

class1FolderName = './Mytilus_original'
class2FolderName = './Zostera_original'
```

In [ ]:
```
# make the output images larger
plt.rcParams['figure.dpi'] = 120
plt.rcParams['savefig.dpi'] = 300
```

In [ ]:
```
def getFolderNameAndEtalonFolderName(classFolderName, print_it=False):
    folderName = classFolderName[:-9]
    etalonFolderName = folderName + 'Etalons'
    if print_it:
        print('Folder name:', folderName, '\r\nEtalon folder name:', etalonFolderNam
    return folderName, etalonFolderName
```

Task 1

Collect a set of images suitable for the tasks below of at least 2 species. Write code to preprocess the images of plants into a uniform size of your choice, e.g. 1024x1024 pixels.

We have searched for relevant **Zostera** and **Mytilus** images on the web, but most of the pictures contained either some form of watermark, or some additional image distortion. Because of that we were left with only limited number of images for our task. This most likely had significant impact, especially on the deep neural network solution described in part 4 and that's why we decided to use data agmentation for that part of our assignment.

In the code below we are transforming the images into recommended dimensions 1024x1024 while also converting them to the *.png* format.

In [ ]:
```
def transformImagesInDirectory(folderName):
    list_of_files = os.listdir(folderName)
```

```
    targetFolderName = folderName[:-9]
    for idx, file in enumerate(list_of_files):
        image_file_name = os.path.join(folderName, file)
        img = Image.open(image_file_name)  # .convert("L")
        img = img.resize((1024, 1024))
        if not os.path.exists(targetFolderName):
            os.mkdir(targetFolderName)
        img.save(targetFolderName + '/' + str(idx) + ".png")
        # os.remove(image_file_name)
```

In [ ]:

```
def task1():
    transformImagesInDirectory(class1FolderName)
    transformImagesInDirectory(class2FolderName)
# task1()
```

Task 2

Select a set of etalons (e.g. small images containing a sample of some distinctive features) from the an image to be used for matching similar objects. Aim for a set that would be representative on at least 50% of the images of the particular species. Think how to deal with rotations.

In this task we have manually created 9 etalons for each selected sea plant. We have selected the etalons as small as possible, but still containing representative and dominant part of given plants visual.

For Mytilus this was quite simple, but for Zostera it was not. Zostera is basically a sea grass and selecting only one grass stalk was sometimes very challenging. We have also decided to create Zostera etalons from different parts of the grass stalks - from the top, middle and bottom of the stalk.

When selecting the etalons our goal was to have a selection of small pieces from our images, which would have enough representative characteristic needed for succesfull template matching with at least 50 % accuracy for the respective plant species.

In the cells below you can see us matching the created etalons one by one to each image in our dataset. We have used the opencv2 matchTemplate function. This function has many matching methods and after some experiments and studying the differences between them we have decided to use the **TM_CCOEFF_NORMED** match method for this task, which represents the normalized value of correlation coefficient between the images.

Resolution

We were able to achieve ~60 % of accuracy for Zostera and ~55 % accuracy for Mytilus.

We have also experimented with etalons. We observed that modifying the size significantly affects the accuracy. When the etalons were much bigger, the accuracy became significantly lower.

In [ ]:

```
def match_image(img, template):
    w, h = template.shape[::-1]
    match_method = cv2.TM_CCOEFF_NORMED
    res = cv2.matchTemplate(img, template, match_method)
    _, maxval, _, maxloc = cv2.minMaxLoc(res)
```

```
    btm_right = (maxloc[0] + w, maxloc[1] + h)
    cv2.rectangle(img, maxloc, btm_right, 255, 2)
    return maxval
```

In [ ]:

```
def matchEtalonToEachImage(etalons, images):
    # matching_results = []
    matching_results_by_etalon = []
    for etalon_name in os.listdir(etalons):
        etalon_path = etalons + "/" + etalon_name
        # print("etalon:", etalon_path)
        img_template = cv2.imread(etalon_path, 0)
        matching_results = []
        for image_name in os.listdir(images):
            img_path = images + "/" + image_name
            # print("\t img", img_path)
            img = cv2.imread(img_path, 0)
            match_res = match_image(img, img_template)
            matching_results.append(match_res)
        matching_results_by_etalon.append(matching_results)

    averages = []

    for val in matching_results_by_etalon:
        averages.append(np.average(val))
    print("Precision:", np.average(averages) * 100)
    return averages
```

In [ ]:

```
def task2(classFolderName):
    folderName, etalonFolderName = getFolderNameAndEtalonFolderName(classFolderName)
    precisionForEachEtalonList = matchEtalonToEachImage(etalonFolderName, folderName
    # print("Precisions for each etalon:", precisionForEachEtalonList)

print("Mytilus")
task2(class1FolderName)

print("Zostera")
task2(class2FolderName)
```

```
Mytilus
Precision: 54.667162322081055
Zostera
Precision: 59.799787082842414
```

Task 3

Use at least 3 different existing conventional feature detectors provided by OpenCV to find matches of the etalons in the image. NB! Take into account overlaps and subtract the appropriate numbers from total scores.

Evaluate on two different images (called task3a.tiff and task3b.tiff) how well the approach works and which feature detector performs best.

For this task we were supposed to use 3 feature detectors to find matches of the etalons in the image. We have decided to use 3 different approaches:

1. SIFT with FlannBased matching
2. ORB with BruteForce matching

3. SIFT with BruteForce matching

Our implementation is in the *hw2_3.py* file.

Our results can be seen in the *Task3Results* file. Here we can observe the difference between selected feature detectors.

We have observed that ORB does not produce as good results as SIFT. It matches some features correctly but it's mostly able to match only few descriptors correctly.

When using the SIFT feature detector we have decided that we will only consider descriptors that are close to each other. With this we wanted to prevent matching points which are out of our range of interest. For this we have used the Lowe's ratio test.

BruteForce vs. FlannBased matcher comparison

BruteForce matcher is going to try all the possibilities (which is the meaning of "Brute Force") and hence it will find the best matches. FLANN, meaning "Fast Library for Approximate Nearest Neighbors", will be much faster but will find an approximate nearest neighbors. It will find a good matching, but not necessarily the best possible one. We have experimented with the FLANN's parameters in order to increase the precision ("quality" of the matchings), and we have observed that in some cases it was negatively affecting the speed of the algorithm.

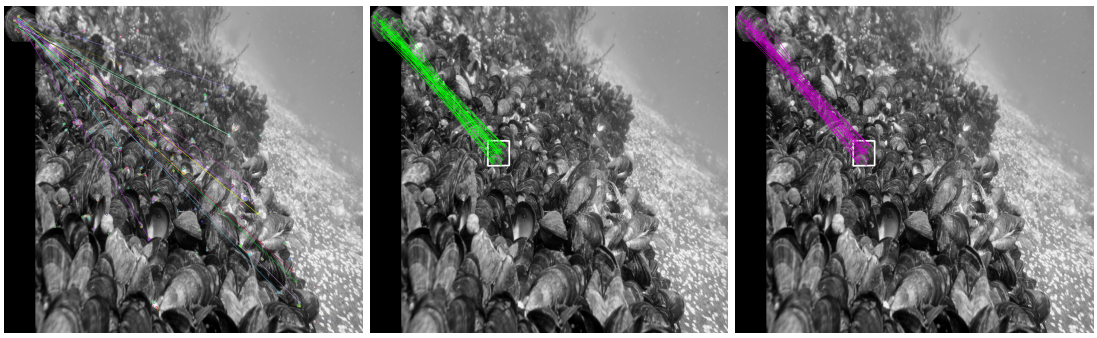
Conclusion

When comparing the ORB and both SIFT detectors we were able to see that the SIFT detectors perform much better. When we compared the SIFT matching techniques we have come to conclusion that FLANN is much faster than BFMatcher but it only finds an approximate nearest neighbor, which is a good matching but not necessarily the best. Anyway none of the selected detectors performed well when matching etalons with an image which was not containing the etalon. Our results on selected 2 images can be seen in the *Task3Results* for each 3 feature detectors.

We have also observed that if we rotate the etalon, the approach used in task 2 is not able to find the etalon in the image. But here, using the methods mentioned above we are able to find the etalons even if they were rotated.

On the first image you can see ORB feature matcher where we want to find etalon on the image, reasult is not so good because there are a lot of match out of correct range. Next example of ORB matching is in Task3Results folder where we were comparing the same etalons but with another image.

On the second and third images are results of SIFT with Flann based & SIFT with BruteForce, those results are great because we exactly found etalons on the images, this etalons were also rotated and our matching functions didnt have any troubles with correct matching. Next example of SIFT with Flann based $ SIFT with BruteForce are in Task3Results folder where we were comparing the same etalons but with another images.



Task 4

Improve the baseline by applying deep learning.

Our model implementation can be found in the root of this project's directory and consists of these files:

- dataset.py
- mask_creator.py
- model.py
- train.py
- utils.py

First we have annotated our data using the CVAT annotation tool. We have distinguished 3 classes: Zostera, Mytilus and the background. We have then exported the annotations in the COCO format.

Then we have used the **COCO** (Common Objects in Context) library to load the exported annotations and we have prepared relevant mask and frame (original image with highlighted mask) images using it's build in functions.

Masks and Frames are saved and can be seen in folders with the same name.

At first we have decided to approach the problem using prebuild models in TensorFlow. We prepared data generator functions and also applied the concept of data augmentation to prepare augmented data generator. Then we have tried to run the MobileNetV2 network on our prepared data, but at this point we have occured a problem with our data. We have tried a lot of different modification, but in each case we were either getting bad results from the model or not providing the model with correct data format.

Our implementation can be found in the *hw2_4.py*.

Because this approach was not successfull we have decided to try using the *pytorch* library instead.

After some research we have found an interesting article which presented cropped version of the UNET model's architecture. Paper with the models architecture can be seen in **U-Net_Convolutional_Networks.pdf**.

We have used our masks created in the previous steps and prepared *train.py* script which allowed us to easily modifythe model's parameters. We have experimented with different values of the epochs count, batch size and different resizing of the input images in order to get the

best segmentation accuracy. We have decided to split the dataset into train and validation sets with the ration 5:3.

Our best result was achieved after 56 epochs with accuracy around 40 %.

Conclusion

In the end we were able to successfully prepare working model for our plant segmentation task. The accuracy of the model was not really good, but that was expected given the low number of images in our dataset.

Since in task 3 we have used feature detection and in task 4 we were performing image segmentation there is not a clear way how to compare these two approaches. We have observed that the feature detection is not working very well, since it finds only exact matches of our etalons. The neural network provides much better results since it is able to detect the plants on it's own without the need of etalons. On the other hand it's need good data annotations.

Example of our original image, it's mask and the prediction.

In the prediction the more dark parts of the image signalizes that the model was mor sure with it's prediction.