Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
and Geometry Lab

Computer Vision Lecture

Assignment 2 Report

Autumn Term 2022

Author:
Pascal Lieberherr

Contents

1 Report **2**
1.1 Mean Shift 2
1.2 Image Segmentation 3

Chapter 1

Report

1.1 Mean Shift

Loop Implementation

In the loop implementation I simply iterate over every row in the image matrix $X \in \mathbb{R}^{3675 \times 3}$. Then I compute the distance of the i-th pixel w.r.t. every other pixel in the image matrix X. Afterwards I use the gaussian kernel function to weight the computed distance. With the weighted distance for the i-th pixel I compute the weighted mean for the i-th pixel. The weighted mean$\in \mathbb{R}^{1x3}$ is then stored in $X_{new} \in \mathbb{R}^{3675x3}$ at the i-th row. After I iterated over every row in X I repeat the same process 20 times with X_{new} as the new input. Besides the for loop over every row in X I used vectorized operations anywhere else e.g. numpy.linalg.norm().

Batch Implementation

Since for loops are in-efficient especially in python the goal of the batch implementation is to replace the for loop that iterates over every row in X with matrix operations. The run time for the slow (for loop) implementation on a CPU is 21s. For the fast (batch) implementation it's 15s also on the CPU. For both approaches I get the same output. For the fast implementation I first build a huge pytorch tensor X_{batch} by stacking the image matrix X as many times as the columns of X. In order to compute the distance I first compute a pytorch tensor x_{batch} such that the distance for every pixel w.r.t to every other pixel is the 2-norm of $X_{batch} - x_{batch}$. $x_{batch}, X_{batch} \in \mathbb{R}^{3675*3675 \times 3}$. In x_{batch} every pixel $\in \mathbb{R}^{1x3}$ is repeated 3675 times resulting in a shape of $3675 * 3675 \times 3$. On the distance matrix $(X_{batch} - x_{batch})$ I again apply the kernel. As a result, I obtain the weight matrix $\in \mathbb{R}^{3675*3675 \times 1}$ Next, the weighted mean for each pixel needs to be computed. This can again be computed without loops i.e. only matrix operations. For this I need to reshape the weight matrix into a 3675×3675 matrix called W. Then the weighted mean is: see code snippet below. With this approach I could compute X_{new} without loops. I repeat the process 20x with X_{new} as the new input.

```
def update_point_batch(weight, X):

    weight = torch.tensor(weight)
    weight_new = weight.view(-1,X.shape[0])
    nomi = torch.matmul(weight_new,X)
    denomi = torch.sum(weight_new, dim=1, keepdim=True)

    return nomi/denomi
```

Figure 1.1: Code snippet weighted mean batch implementation

1.2 Image Segmentation

Accuracy: 0.870