# Computer Vision – Image Segmentation

Emilia Szymańska

November 2022

## 1 Mean shift algorithm

The mean shift algorithm consists of three steps: calculating the distances between each point and all points, applying a Gaussian kernel function and updating the analyzed point position. To get the distances, I pass the difference between the point and the vector of all other points as the argument to *torch.norm* function. Then I utilize the formula $k(x) = exp(\frac{-x^2}{2h^2})$ (where h is the bandwidth and x is the distance between points) to apply the Gaussian function. Finally, I compute the weighted mean $\sum_i w_i x_i / \sum_i w_i$ for $i$ points, where the weights correspond to the distances between points and $x$ is the feature we are considering (in our case we needed to compute the weighted mean three times as we have 3 elements to describe a point). This weighted mean is the updated point position. Code implementation is presented below.

The normalization term of Gaussian distribution does not affect the final output, as in the weighted mean calculation it would cancel out anyway (we could take it out before the sums in both the denominator and numerator).

I did not implement the vectorization of inputs – it is supposed to be simple, but I do not know how to do it in PyTorch (unless I find the solution before the deadline and manage to upload it).

```
def distance(x, X):
    return torch.norm(x[None, :]-X, p=2, dim=1)

def gaussian(dist, bandwidth):
    return torch.exp(-torch.square(dist)/(2*bandwidth**2))

def update_point(weight, X):
    weights_sum = torch.sum(weight)
    wx_sum1 = torch.sum(weight * X[:,0])
    wx_sum2 = torch.sum(weight * X[:,1])
    wx_sum3 = torch.sum(weight * X[:,2])
    x1 = wx_sum1 / weights_sum
    x2 = wx_sum2 / weights_sum
    x3 = wx_sum3 / weights_sum
    return torch.Tensor([x1, x2, x3])
```