CV Lab 05: Image Segmentation

1 Introduction

The aim of this lab was to utilise the classical Mean Shift algorithm, as well as modern Conv-Deconv CNNs to perform image segmentation.

2 Mean Shift

2.1 Computing distances in colour space

The distance function is used in the Mean Shift algorithm to determine the similarity or dissimilarity between data points in the colour or LAB space. Given a point x and the set of all points X, the Euclidean distance $d(x, X_{i,:})$ to each other point is calculated as follows:

$$d(x, X_{i,:}) = \|x - X_{i,:}\|_2 = \sqrt{\sum_{j=1}^{3}(x_j - X_{i,j})^2}$$

Note when implementing in NumPy, `x.shape` is `(3)` and `X.shape` is `(N,3)`, where `N` is the number of points, $H \times W$. We vectorize our implementation using NumPy to efficiently calculate the L2 norm of each distance vector: `np.linalg.norm(X - x, axis=1)`, ensuring to take the norm over the colour dimensions. Points closer together in colour space will contribute more significantly, when updating their values in `update_point()`.

2.2 Gaussian weighting

We use a zero-mean Gaussian distribution, $N \sim (0, \sigma^2)$ where σ is the `bandwidth` parameter, to compute the weights of points from X when calculating the average for x. This is simply implemented as `np.exp(-0.5 * (dist / bandwidth) ** 2)`. Lower distance values correspond to a higher probability density, resulting in points close in colour space having higher weighting when calculating the average.

2.3 Updating the points

To update the point x's position in colour space, we take a weighted average over all the other points within the specified radius, using the weights calculated with the `distance()` and `gaussian()` functions. Since our radius is $+\inf$, the average is calculated over all points in X. This can be done as follows:

```
def update_point(weight, X):
    return np.sum(weight[:,np.newaxis] * X, axis=0) / np.sum(weight)
```

`weight` is a size `(N)` array so `weight[:,np.newaxis]` broadcasts it to the same dimensions as `X`, `(N,3)`, so that they can be multiplied element-wise. We divide the sum of the weights to normalise the average.

2.4 Experimenting with different bandwidths

The `bandwidth` parameter corresponds to the standard deviation of the Gaussian kernel we use. Larger values mean that weightings will be more uniform, whereas small bandwidths correspond to only weighting points that are very close in colour space to x.

The original image is shown in Fig. 1.

The segmentation results of Mean Shift for varying values of `bandwidth` are shown in Fig. 2.

We observe that lower bandwidths correspond to segmentation with higher resolutions, whereas higher corresponds result in coarser segmentation. This is again due to the higher bandwidths resulting in greater Gaussian smoothing in the weighting, meaning that the scale of clustering in colour space is also greater. This also means that the number of different segmented classes, or clusters in colour space, is greater for larger bandwidths. We can see this in the larger numbers of colours in Fig. 2a.

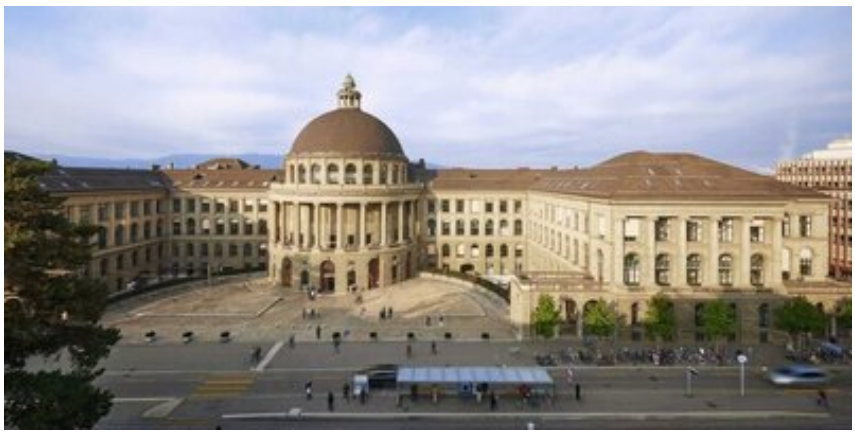

Figure 1: Image to be segmented.

(a) bandwidth=2.5

(b) bandwidth=3.0

(c) bandwidth=5.0

(d) bandwidth=7.0

Figure 2: Segmentation results for varying *bandwidth* values

When setting `bandwidth=1.0`, the number of clusters becomes larger than the number of colours available in `colors` array, with a size of `(24,3)` indicating 24 distinct colours. This results in an `IndexError` since we try to index outside of our `colors` array. To avoid this, we can either use a larger colour map, or we can assign a random RGB colour to each cluster center. For example, some pseudocode:

```
for each centroid:
    assign centroid to np.random.rand(3)  # random RGB array
```

Setting `bandwidth=2.5` seems to be a good tradeoff between too many cluster centroids, leading to spurious segments, whilst maintaining the fine details of structures.