Assignment 2

Computer Vision ETH, Fall 2021
Quentin Guignard
5th November, 2021

Mean shift implementation

In this assignment we were asked to performed the mean shift algorithm on a cow image. We implemented successfully the first version by taking advantage of pytorch's vectorization feature that allowed us to perform mathematics operations along the axis of our tensors. We proceeded as described in the hand out. For each point, we computed the distance to all other points. We then applied the Gaussian RBF on these distances to finally update the new point with a weighted mean of the tensor. Unfortunately, we couldn't find the syntactic way of implementing the batching. Therefore, we will describe the idea we were trying to implement. After that we will still highlight some results of our algorithm running on the CPU and compare it with the GPU.

The main idea we had for batching was, that given N 3D points, to stack the initial tensor N times leading to the tensor $[X_1, ..., X_N]$. We would have computed the distance by taking expanding each 3D point p_i of X to the form $X' = [p_{1,1}, ..., p_{1,N}, ..., p_{i,1}, ..., p_{1,N}, ..., p_{N,1}, ..., p_{N,N}]$ and performing $D = \|X - X'\| = [d_{1,1}, ..., d_{1,N}, ..., d_{i,1}, ..., d_{1,N}, ..., d_{N,1}, ..., d_{N,N}]$ that would contains the distances from a single point to all the others in X. We would then compute the RBF of these in the same way as we did in the code (see source) leading to $G = RBF(D) = [g_{1,1}, ..., g_{1,N}, ..., g_{i,1}, ..., g_{1,N}, ..., g_{N,1}, ..., g_{N,N}]$ (the formula is not represented here but found in the source). Finally, we would have updated the points as $X = [\sum_j X_1(j) * g_j, ..., \sum_j X_i(j) * g_j, ..., \sum_j X_N(j) * g_j]$. Where X is now the next stacked tensors containing the next values.

Finally, let's quickly presents our results (non-batch only). On the CPU (i7-7700HQ) we obtained 16.147 seconds against 11.672 seconds on the GPU (GTX 1050 4GB). The CPU has a throughput of 384 GFLOPS and the GPU has a throughput of 1862 GFLOPS. The GPU has a throughout that is therefore 4.85 times higher than the CPU however we observed a speed up of only 1.38. This unexpected difference might due to the problem being small implying that the time was dominated by data transfer to the GPU (we loop over each pixel and the data has to be probably resend redundantly). We also think that it could be due to non-batched architecture. Indeed, its also greatly possible that the time was dominated by the Python interpretation which could explain such results.

Figure 1: target image to obtain after applying the mean shift algorithm

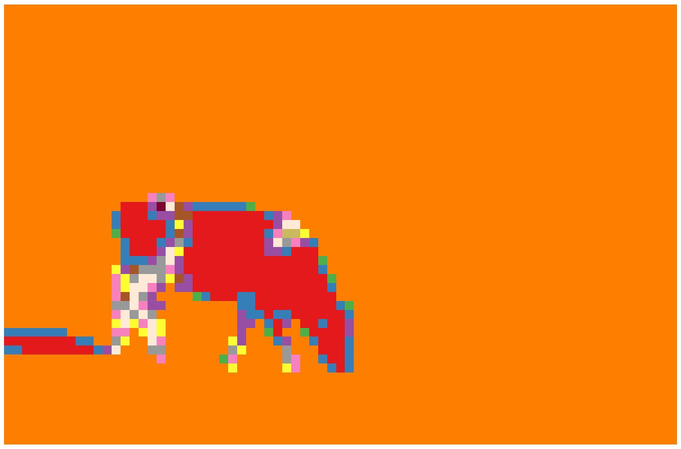

Figure 2: Our actual result obtained after applying the mean shift algorithm