Computer Vision Assignment 2

Saurabh Vaishampayan
svaishampaya@student.ethz.ch

November 2021

1 Mean Shift Algorithm

We were asked to perform mean shift implementation using two methods: performing mean shift updates for one point at a time iteratively (Naive implementation) and performing mean shift updates for a batch of points iteratively (batch processing). The implementation we followed is very similar for both the approaches, because the naive implementation can be thought of as a special case of batch processing with batchsize 1. Of course while doing this, one has to be careful to squeeze and unsqueeze the tensors generated during naive implementation to relevant dimensions at appropriate steps. Below, we give the implementation for batch processing and wherever required we state the modifications for the case of Naive implementation.

1. X is the input image, a tensor of shape $N \times C$, where N is the number of pixels and C is number of channels.

2. - Naive implementation: For each point x in X we first convert it to a two dimensional tensor by torch.unsqueeze(1), followed by torch.transpose. This makes $x \sim 1 \times C$.
   - Batch implementation: Using DataLoader() class from PyTorch, we make batches x of batchsize K from X. Here x is of dimensions $K \times C$.

   Now the pairwise distance is computed using torch.cdist function. Note that $K = 1$ for Naive implementation.
   $$d = \text{torch.cdist(x, X)}; \quad d \sim K \times N$$

3. Weights are computed according to RBF kernel using torch.exp function. Note that $K = 1$ for Naive implementation.
   $$w = e^{-\frac{d^2}{2\sigma^2}}; \quad w \sim K \times N$$

4. Each point is replaced by weighted mean:
   $$x_i^{new} = \frac{\sum_j w_{ij} x_j^{old}}{\sum_j w_{ij}}$$

   This can be implemented as a matrix multiplication of tensors $w \sim K \times N$ and $X \sim N \times C$ (implemented as torch.matmul). This is followed by a transpose to make it of shape $C \times K$). This is divided by summation of w along second index (torch.sum(w,axis=1)) which will give an output of $C \times K$.

   - In the case of Naive implementation the output will be $C \times 1$. This has to be transposed (torch.transpose) and also squeezed (tensor.squeeze(1)) to get a one dimensional tensor as required in the update step.
   - In the case of batch processing implementation the output will be $C \times K$. This has to be transposed (torch.transpose) and then returned as the updated means for all the points in the batch.

5. This is repeated for all points for multiple iterations until desired accuracy.

1.1 Comparison and inferences

1. The following table lists the exact timings versus batch sizes:

Batch size	Computation Time
1	26.04
2	14.20
4	9.62
8	6.64
16	4.54
32	2.85
64	4.79
128	4.58
256	4.17

For Naive Implementation, Computation time was 24.803 seconds.

2. We know that batch processing involves parallel updates of multiple points at same time, and batch processing with batch size equal to 1 involves updates of a single point at a time. Hence we expect the time for naive implementation and batch processing with batchsize 1 to be roughly equal, which is what we observe.

3. When implementing batch processing, we perform updates of all points in the batch in parallel. So we expect that batch processing will give lesser computation time as batch size increases. Infact we expect the time to have an inverse relation with increasing batch size. This, however, may not be true always because of:

   - Constant overhead due to other tasks not related to batch processing.
   - Depending on memory available, the computer may switch to a mix of iterative and parallel implementation internally after a large batch size. e.g. If the batch size is too big, then the computer will run out of memory to process the tensor operations in parallel and will instead switch to a mix of iterative and parallel processing.

4. We can observe the above in the plot below. The plot is a log-log plot of batch sizes vs computation times. When the log(computation time) decreases linearly with increase in log(batch size), the inverse relation is said to hold. i.e. we are in the region where batch processing is done in parallel. When the batch size becomes large, the tensor operations do not have enough memory and are hence are implemented as a hybrid of iterative and parallel computation, we see deviation in the plot and the plot starts to become flat.

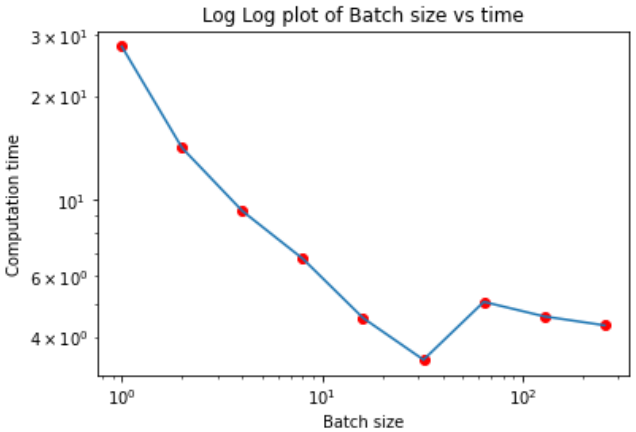

Figure 1: Log log plot of batch size vs computation time