ENACT: ENTROPY-BASED CLUSTERING OF ATTENTION INPUT FOR REDUCING THE COMPUTATIONAL RESOURCES OF OBJECT DETECTION TRANSFORMERS - SUPPLEMENTARY MATERIAL

1. CLUSTERING DESCRIPTION

After calculating the Keys' entropy and identifying its concave and convex regions, we must reduce the Keys and Values by regions of same curvature sign. Additionally, we want the values of the entropy to be normalized so that their sum within a region equals to 1. To do that we run a softmax function on each separate region whose indices correspond to regions of the same sign in the result of the convolution of the entropy with the 2d Sobel kernel. For example, assume that at one point the output of the step function is $[...1, 1, 1, -1, -1, 1, 1, 1, 1, ...]$ and the indices are $[...i, i+1, ..., i+8, ...]$. In the respective indices of the entropy the softmax function will be run on three separate regions, which are from i to $i+2$, $i+3$ to $i+4$ and $i+5$ to $i+8$. The exact computation is shown in Eq. 1, where $\mathcal{H}_\mathrm{j}$ is the self-information of pixel j.

$$
\begin{aligned}
\mathcal{H}_\mathrm{j} &= \frac{\exp(\mathcal{H}_\mathrm{j})}{\sum_{\mathrm{k=i}}^{\mathrm{i+2}} \exp(\mathcal{H}_\mathrm{k})} \text{ if } \mathrm{j} \in [\mathrm{i}, \mathrm{i}+2] \\
\mathcal{H}_\mathrm{j} &= \frac{\exp(\mathcal{H}_\mathrm{j})}{\sum_{\mathrm{k=i+3}}^{\mathrm{i+4}} \exp(\mathcal{H}_\mathrm{k})} \text{ if } \mathrm{j} \in [\mathrm{i}+3, \mathrm{i}+4] \\
\mathcal{H}_\mathrm{j} &= \frac{\exp(\mathcal{H}_\mathrm{j})}{\sum_{\mathrm{k=i+5}}^{\mathrm{i+8}} \exp(\mathcal{H}_\mathrm{k})} \text{ if } \mathrm{j} \in [\mathrm{i}+5, \mathrm{i}+8]
\end{aligned}
\tag{1}
$$

Let $\mathcal{H}$s be the signs of the entropy curvature, $\mathcal{H}$ the self-information values, and K and V be the Keys and Values respectively. Initially, $\mathrm{K}, \mathrm{V} \in \mathbb{R}^{\mathrm{N} \times \mathrm{HW} \times \mathrm{d}}$, while $\mathcal{H} \in \mathbb{R}^{\mathrm{N} \times \mathrm{HW}}$ and the same goes for $\mathcal{H}$s, where N is the batch size, HW the total number of pixels in the feature map, i.e. the spatial dimensions, and d the dimensions of the feature vector. Firstly, we flatten them all along the batch size, making $\mathcal{H}$ and $\mathcal{H}$s 1-dimensional and K, V 2-dimensional. Subsequently, we locate the regions of same curvature sign by keeping the indices where the sign changes, as well as the length of the regions by subtracting consecutive region start indices. By default we consider that integer multiples of the spatial dimensions are also region start indices, because they are the starting points of a different feature map.

Algorithm 1 Cluster Algorithm

```
K, V: Unclustered Keys and Values
K', V': Clustered Keys and Values
H: Entropy Values
Hs: Entropy Curvature signs
RSI: Region Start Indices
RL: Region Lengths
N: Number of Regions
d: Feature Vector Dimension
ThIdx: Thread Id along the x axis
ThIdy: Thread Id along the y axis
if ThIdx < N and ThIdy < d then
    sK ← 0
    sV ← 0
    sExp ← 0
    for RSI[ThIdx] ≤ i < RSI[ThIdx] + RL[ThIdx] do
        sK ← sK − exp(H[i]) * Hs[i] * K[i * d + ThIdy]
        sV ← sV − exp(H[i]) * Hs[i] * V[i * d + ThIdy]
        sExp ← sExp + exp(H[i])
    end for
    K'[ThIdx * d + ThIdy] ← sK/sExp
    V'[ThIdx * d + ThIdy] ← sV/sExp
end if
```

Our clustering function takes K, V, $\mathcal{H}$, $\mathcal{H}$s, the region start indices and the region lengths as input, and outputs K', V', which are the clustered Keys and Values respectively. In order to implement the pixel grouping, regardless of the region size, in an optimal manner, we create a CUDA kernel which makes use of 2D thread blocks. Along one dimension, the number of threads is the same as the number of regions, and the other the feature vector dimension. The detailed pseudocode is shown in Algorithm 1 and we also provide a visualization of the process in Figure 1. The newly created $\mathrm{K}', \mathrm{V}' \in \mathbb{R}^{\mathrm{L} \times \mathrm{d}}$ where L is the total number of pixels of the clustered Keys and Values and is much smaller than NHW.

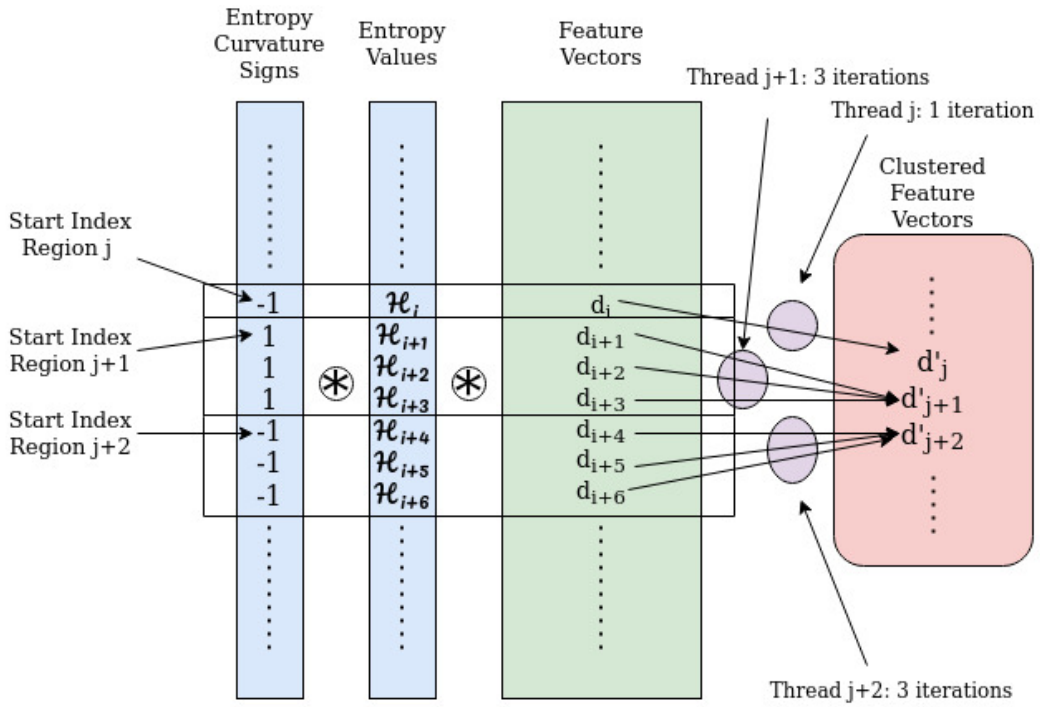

Fig. 1. Visualization of the ENACT clustering process. The weighted entropy values are multiplied with the curvature signs, and then with the feature vectors of the Keys or Values. Each region with same curvature sign corresponds to a thread in the CUDA kernel. The grouping is done by summing the elements in the same region, with as many iterations as the number of pixels corresponding to that thread. Therefore, the number of rows in the clustered output will be the number of threads.

2. GRADIENT CALCULATION

In order to correctly calculate the gradients of K and V that pass through the clustering algorithm, we need to consider the connections each variable has with each of the resulting outputs.

$$
\begin{aligned}
\mathrm{K}'_{\mathrm{j}} &= \sum_{\mathrm{i=n}}^{\mathrm{n+m}} \frac{\exp(\mathcal{H}_{\mathrm{i}})}{\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}})} \mathrm{K_i} \\
\mathrm{V}'_{\mathrm{j}} &= \sum_{\mathrm{i=n}}^{\mathrm{n+m}} \frac{\exp(\mathcal{H}_{\mathrm{i}})}{\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}})} \mathrm{V_i} \\
\frac{\partial \mathcal{L}}{\partial \mathrm{K}_{\mathrm{i}}} &= \frac{\exp(\mathcal{H}_{\mathrm{i}})}{\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}})} \frac{\partial \mathcal{L}}{\partial \mathrm{K}'_{\mathrm{j}}} \\
\frac{\partial \mathcal{L}}{\partial \mathrm{V}_{\mathrm{i}}} &= \frac{\exp(\mathcal{H}_{\mathrm{i}})}{\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}})} \frac{\partial \mathcal{L}}{\partial \mathrm{V}'_{\mathrm{j}}}
\end{aligned}
\tag{2}
$$

For K and V it is more straightforward since they are not interconnected with K', V', namely K is used only for the calculation of K' and V for V'. Starting therefore, with the gradients of K and V, let $\mathcal{L}$ denote the loss, and $\frac{\partial \mathcal{L}}{\partial \mathrm{K}'}$, $\frac{\partial \mathcal{L}}{\partial \mathrm{V}'}$ the gradients of the clustered Keys and Values respectively. Each index of the original unclustered input is used to compute only one index of the clustered output, which simplifies the calculations. Let i denote indices of the original input, and j those of the clustered output. Therefore, $0 \leq \mathrm{i} < \mathrm{NHW}$, and $0 \leq \mathrm{j} < \mathrm{L}$. In Equation 2 we show the mathematical formula of the clustering's forward pass and resulting backward pass for the computation of the gradients of K and V. The values n and m correspond to the starting index of the original input and region length respectively that correspond to the region j.

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathcal{H}_{\mathrm{i}}} &= \frac{\partial \mathcal{L}}{\partial \mathrm{K}'_{\mathrm{j}}} \frac{\partial \mathrm{K}'}{\partial \mathcal{H}_{\mathrm{i}}} + \frac{\partial \mathcal{L}}{\partial \mathrm{V}'_{\mathrm{j}}} \frac{\partial \mathrm{V}'}{\partial \mathcal{H}_{\mathrm{i}}} \\
\frac{\partial \mathrm{K}'}{\partial \mathcal{H}_{\mathrm{i}}} &= -\frac{\exp(\mathcal{H}_{\mathrm{i}})}{\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}})} \Big(\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \frac{\exp(\mathcal{H}_{\mathrm{k}})}{(\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}}))^2} \mathrm{K_k} + \mathrm{K_i} \Big) \\
\frac{\partial \mathrm{V}'}{\partial \mathcal{H}_{\mathrm{i}}} &= -\frac{\exp(\mathcal{H}_{\mathrm{i}})}{\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}})} \Big(\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \frac{\exp(\mathcal{H}_{\mathrm{k}})}{(\sum_{\mathrm{k=n}}^{\mathrm{n+m}} \exp(\mathcal{H}_{\mathrm{k}}))^2} \mathrm{V_k} + \mathrm{V_i} \Big)
\end{aligned}
\tag{3}
$$

For the gradient of the entropy however, the process is more complicated because it is used for the calculation of both K' and V'. Therefore, its gradient will be given by the chain rule consisting of both $\frac{\partial \mathcal{L}}{\partial \mathrm{K}'}$ and $\frac{\partial \mathcal{L}}{\partial \mathrm{V}'}$. The exact formula for calculating the gradient of the entropy is shown in Equation 3.

3. SELF ATTENTION ADJUSTMENTS

The original self attention module in detection transformers takes Queries (Q) and Keys (K) as input where both $\mathrm{Q}, \mathrm{K} \in \mathbb{R}^{\mathrm{N} \times \mathrm{HW} \times \mathrm{d}}$, which means that the attention weights have shape $\mathrm{N} \times \mathrm{HW} \times \mathrm{HW}$. With our module, $\mathrm{K}' \in \mathbb{R}^{\mathrm{L} \times \mathrm{d}}$ and the number of pixels L consists of all the feature maps. Therefore, we have to ensure that each Query feature map in the batch is connected with its correct clustered counterpart of the Keys. To that end, we keep the indices where each feature map begins in the clustered Keys K'. Again we try to take advantage of GPU parallelism, by implementing a CUDA kernel with 2d blocks, where the first axis of threads corresponds to the Query pixels, and the second to the clustered Key ones. Q threads that belong to the nth feature map, are enforced by condition to be matched with K' threads whose id lies between the nth and nth plus 1 starting index. Therefore, the attention weights dot product is computed between equivalent feature maps. In a similar manner we compute the attention map which is the product of the attention weights and Values.