Supplementary Material

In this document, we provide further details about the implementation, evaluation and optimizations.

Maximum Pooling

In the original DeltaCNN implementation, maximum pooling layers process the result for newly accumulated inputs and for previously accumulated inputs, which is then subtracted to obtain the Delta output of the layer. In MotionDeltaCNN, this approach does not work anymore as soon as regions inside the accumulated input values buffer are reset to zero. When the pooling kernel accesses a pixel's neighboring pixels, with some of them reset and others retained from previous frames, the output for the previous frame cannot be processed correctly anymore - leading to incorrect Delta values. We solve this using an additional buffer storing the previous output of a pixel. Since the Delta value is always relative to the previous output, storing it implicitly solves the dependency on neighboring pixel states. Each pixel was either reset itself in the current frame - in this case, we process the Delta against a zero reference - or it was retained from previous frames - in which case we process the Delta against the previous output.

Video Object Segmentation Evaluation Details

In video object segmentation, we reduce the update rate by scaling updates outside the region of interest, i.e., the segmentation mask of the previous frame. We dilate the previous mask multiple times using maximum pooling with different kernel sizes (10, 20, 40) and compute the average pixel values of the three masks m. The image Delta is then scaled using a factor of $0.4 + 0.6 * m$ before comparing it against the truncation threshold.

The truncation thresholds for BMVOS are auto-tuned on a sequence from the training dataset (tractor-sand) using a threshold of 0.02 as a starting point for all layers but the first. The first layer uses 0.15 and an update mask dilation of 10 pixels (a single pixel update would be dilated over 21x21 pixels). We do not use the same, sensitive thresholds for DeltaCNN since they lead to dense updates nearly every frame due to the lack of image alignment. Instead, we increase the threshold of the first layer to 0.2 and auto tune the remaining thresholds on the same sequence without frame alignment. This leads to a slightly lower accuracy for DeltaCNN than for MotionDeltaCNN. We deliberately target a lower accuracy when tuning the thresholds for DeltaCNN since we aim to show that we outperform DeltaCNN even when they are allowed to truncate updates more aggressively. The auto-tuning procedure is implemented as an iterative front-to-back process as described in DeltaCNN [3].

Further Optimizations

Implicit bias application Adding biases onto newly allocated tiles requires launching a separate kernel on the GPU after every convolutional and batch normalization layer if they use biases. However, layers that are directly followed by activation & truncation layers can skip this step and add the bias implicitly during activation. By initializing the newly allocated tiles of the activation layer's truncated values buffer with the bias instead setting it to zero, the bias is automatically added when applying the activation. This way, we can reduce the overhead of attaching new tiles to existing buffers.

Optimizing memory consumption The memory overhead of MotionDeltaCNN can be a limiting factor on low-end hardware. But in many cases, memory consumption can be reduced by disabling truncation on selected activation layers at the cost of increased sparsity. For example, the DenseNet [2] backbone of BMVOS [1] consists mainly of pairs of 3x3 and 1x1 convolutions. In this network, we only truncate in the activation layer following a 3x3 convolution in which updates are dilated. This way, every second activation layer needs only an accumulated values buffer - saving 50% of the layer's memory at the cost of slightly higher update rates.

Qualitative Results Human Pose Estimation

Figure 1 displays a qualitative comparison between the joint positions predicted using the dense reference, DeltaCNN and MotionDeltaCNN. As can be seen in this example, the predicted joint locations are identical in a all 3 cases for the majority of the joints. However, in some cases, small differences between the approaches can be noticed. The most significant being in Frame 40, where MotionDeltaCNN predicts two joints incorrectly, whereas DeltaCNN and the dense reference each only have one incorrect prediction. At closer inspection, it can be seen that the predicted joint position of the elbow only varies slightly between the dense reference and MotionDeltaCNN, just enough to exceed the accuracy threshold.

References

[1] Suhwan Cho, Heansung Lee, Minjung Kim, Sungjun Jang, and Sangyoun Lee. Pixel-Level Bijective Matching for Video Object Segmentation. *Proceedings - 2022 IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022*, pages 1453–1462, 2022.

[2] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:2261–2269, 2017.

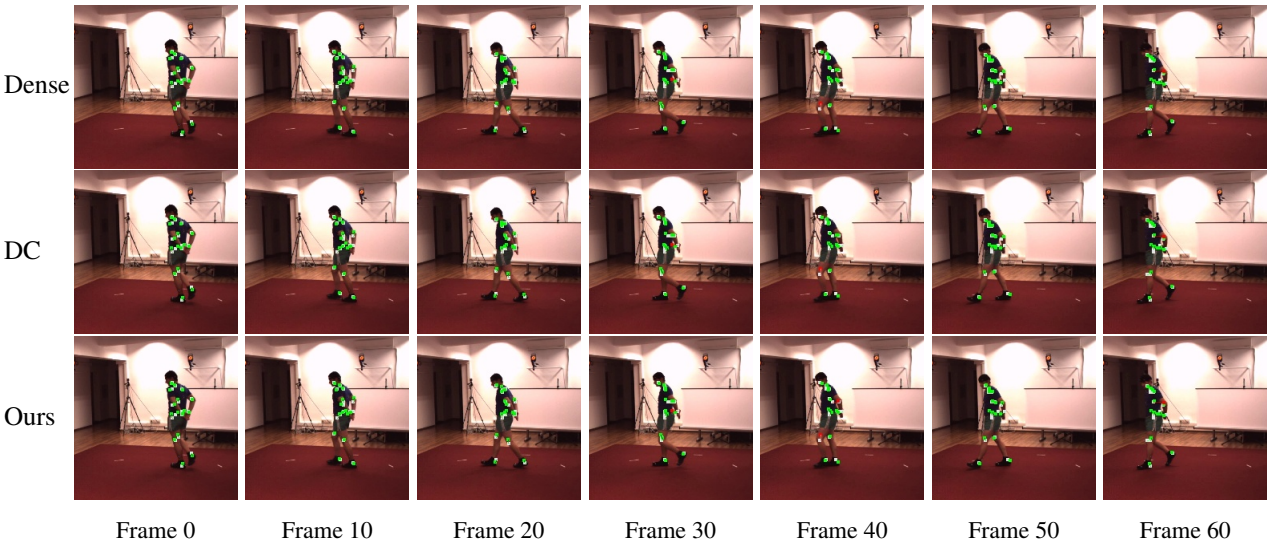

Figure 1: Qualitative comparison between conventional, dense inference, DeltaCNN (DC) and MotionDeltaCNN using HRNet on the Human3.6M dataset. For better visibility, we zoomed in closer to the subject (2x zoom). White markers are the ground truth joint positions. Green and red are the predicted joint positions, with red indicating an incorrect prediction according to PCKh@0.5.

[3] Mathias Parger, Chengcheng Tang, Christopher D. Twigg, Cem Keskin, Robert Wang, and Markus Steinberger. Deltacnn: End-to-end cnn inference of sparse frame differences in videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12497–12506, June 2022.