MotionDeltaCNN: Sparse CNN Inference of Frame Differences in Moving Camera Videos with Spherical Buffers and Padded Convolutions

Mathias Parger[1]   Chengcheng Tang[2]   Thomas Neff[1]   Christopher D. Twigg[2]

Cem Keskin[2]   Robert Wang[2]   Markus Steinberger[1]

[1]Graz University of Technology, [2]Meta Reality Labs

[1]{mathias.parger, thomas.neff, steinberger}@icg.tugraz.at

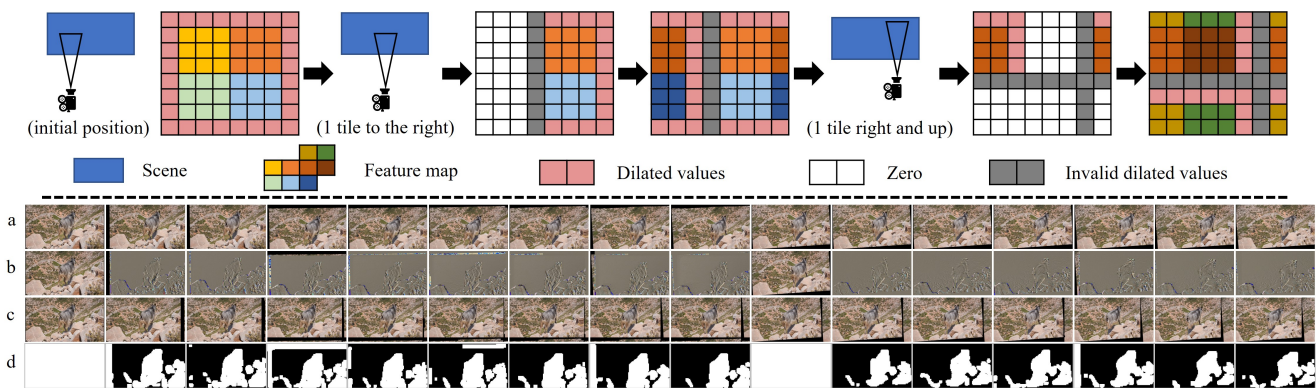[2]{chengcheng.tang, cdtwigg, cemkeskin, rywang}@meta.com

Figure 1: MotionDeltaCNN leverages temporal continuity to accelerate CNN inference for videos with *moving* cameras (bottom) by processing only the sparse frame differences based on a two-dimensional ring buffer, *spherical buffer* (top). Instead of computing each frame individually, we process only the changes (row b) between the aligned current (row a) and previous frames. Our padded convolution creates dilated values (top) which allow for seamless attachment of newly unveiled tiles onto the existing spherical buffer (row c). With these concepts, MotionDeltaCNN speeds up inference by processing only new regions or updated pixels (white areas in row d) without additional memory allocation.

Abstract

Convolutional neural network inference on video input is computationally expensive and requires high memory bandwidth. Recently, DeltaCNN[26] managed to reduce the cost by only processing pixels with significant updates over the previous frame. However, DeltaCNN relies on static camera input. Moving cameras add new challenges in how to fuse newly unveiled image regions with already processed regions efficiently to minimize the update rate - without increasing memory overhead and without knowing the camera extrinsics of future frames. In this work, we propose Motion-DeltaCNN, a sparse CNN inference framework that supports moving cameras. We introduce spherical buffers and padded convolutions to enable seamless fusion of newly unveiled regions and previously processed regions – without increasing memory footprint. Our evaluation shows that we outperform DeltaCNN by up to 90% for moving camera videos.

1. Introduction

Real-time inference of convolutional neural networks (CNN) with video streams remains a power-consuming task and is often infeasible on mobile devices due to hardware and thermal limitations, despite recent efforts aiming at efficient CNN inference through pruning [19, 14], quantization [16, 24, 22], specialized hardware [6, 13, 5] or network optimization [28, 31] Video input allows for performance optimization through temporal similarity between frames. One common method is to use large, slow networks for accurate predictions at key frames, updated with small, fast networks at intermediate frames [34, 33, 20, 18, 9, 21, 12]. However, this approach requires special network design and training and is not suitable for significant frame changes.

Sparse convolutions, on the other hand, can be used with existing pre-trained models, accelerating the large network to the speed of a smaller network without impacting prediction accuracy significantly [25, 26]. The linearity of convolutions
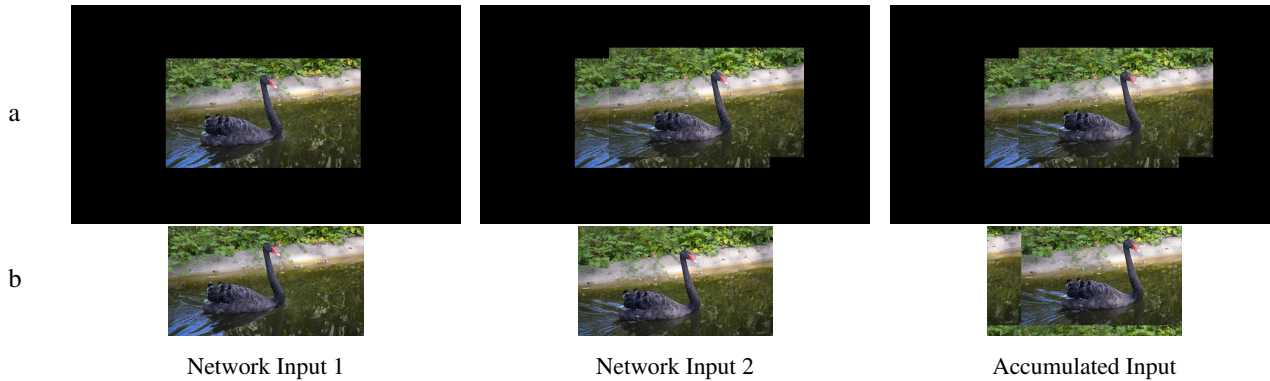
Figure 2: To support moving camera input in sparse inference of frame differences, a naïve approach (a) is to embed the input onto a larger image and only overwrite pixels that are covered in the next frame. This approach comes with large overhead in memory consumption (storing the additional pixels of the embedded frame) and computational cost (checking all pixels for updates). MotionDeltaCNN (b) uses original shape buffers and feature maps – avoiding the memory overhead and reducing the number of pixels that have to be checked for updates.

enables the accumulation of updates over consecutive frames. After processing the first frame densely, upcoming frames can be processed using the difference between the current and the previous frame, the *Delta*, as the network input. Assuming a static camera scenario, this results in large image regions with no frame-to-frame difference. Static elements like background or stationary objects do not require updates after the initial frame. Researchers exploited this sparsity in previous work to reduce the number of FLOPs required for CNN inference [25, 4, 11, 1, 8], but were mostly unable to accelerate inference in practice. With the recent progress in DeltaCNN [26], theoretical FLOP reductions were translated into practical speedups on GPUs, using a custom sparse CNN implementation to exploit sparsity in all layers of the CNN.

DeltaCNN achieves speedups on multiple tasks and datasets, but like prior work, it is optimized for static camera inputs. Even small frame-to-frame camera motion necessitates reprocessing large portions, or the entire image. Spatially aligning consecutive frames, e.g., by leveraging camera extrinsics from IMUs or SLAM on mobile devices, can increase per-pixel similarity and thus update sparsity. Due to the change of camera location and orientation, an aligned new frame may "grow" beyond the initial field of view – out of the previous-results buffers. One way to solve this issue is to embed inputs in a larger frame by padding the input image and drawing updates on top of it. However, the downside of this approach is that padded input images increase the memory consumption and computational cost significantly, as shown in Figure 2.

In this work, we propose MotionDeltaCNN, a sparse CNN inference framework that allows moving camera input with marginal memory overhead. Compared to previous work, we achieve up to 90% higher frame rates in videos with moving cameras, indicating a new perspective for CNN

inference optimization on low-power devices, such as surveillance cameras, smartphones, or VR headsets.

Our main contributions are:

- We propose MotionDeltaCNN, the first framework that leverages temporal continuity to accelerate CNN inference for videos with *moving* cameras by processing only the sparse frame differences.

- At the core of this work, we propose *padded convolutions* for seamless integration of newly unveiled pixels without the need of reprocessing seen pixels.

- We design a two-dimensional ring buffer, a *spherical buffer*, with wrapped coordinates. Our buffer allows partial growth, reset and initialization of new tiles without additional memory allocation.

- We show how MotionDeltaCNN can also be used for speeding up applications with static cameras when only parts of the image require processing.

2. Related Work

The idea of only processing updated pixels in a convolutional layer has been proposed in various ways, with *Recurrent Residual Module (RRM)* [25] being the first work to apply this concept to videos. RRM uses the difference between current and previous input, the *Delta*, as input to convolutional layers. RRM demonstrates large theoretical reductions in FLOPs compared to dense inference resulting from skipping computations involving (nearly) zero-valued entries in feature maps. In practice, skipping individual values is infeasible on most inference hardware like GPUs due to their single instruction, multiple data (SIMD) design.
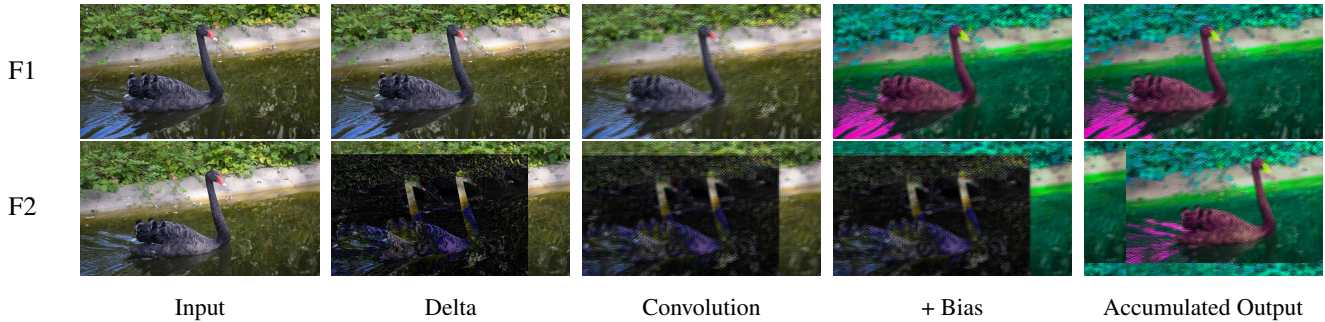
Figure 3: The main concepts of MotionDeltaCNN. We align frame 2 (F2) with frame 1 (F1) using homography matrices. For the intersecting region, we propagate only the aligned frame-to-frame differences (Delta). The newly uncovered regions are propagated directly. The regions in which the buffers store the uncovered regions are reset to zero before processing the current Delta input. After convolving frame 2, we add the bias to all previously unseen regions, and accumulate the result onto our spherical buffer using the offset coordinates for the current frame.

Furthermore, RRM only processes convolutional layers on sparse Delta input. All remaining layers are processed on the full, dense feature maps and therefore require expensive value-conversions before and after each convolutional layer.

Skip-Convolution [11] and *CBInfer* [4] improve the concept of RRM using a per-pixel sparse mask instead of per-value. Like RRM, CBInfer uses a threshold to truncate insignificant updates in the input feature map. This threshold can be tuned for each layer for maximum sparsity while keeping the accuracy at a desired level. In contrast, Skip-Convolution decides per *output* pixel whether an update is required or can be skipped. In both cases, only the convolutional layers (and pooling layers in case of CBInfer) are processed sparsely, requiring expensive conversions between dense and sparse features before and after each of these layers, leaving large performance potential unused.

DeltaCNN [26] propagates sparse Delta features end-to-end using a sparse Delta feature map together with an update mask. This greatly reduces the memory bandwidth compared to previous work. Propagating sparse updates end-to-end eliminates the necessity of converting the feature maps from dense accumulated values to sparse Deltas, and it speeds up other layers bottlenecked by memory bandwidth like activation, batch normalization, upsampling, etc. Using a custom CUDA implementation, DeltaCNN outperforms cuDNN and CBInfer many times on video input. However, DeltaCNN, as well as all other previous work, assumes static camera input. Even single pixel camera motion between two frames can result in nearly dense updates when the image contains high frequency features.

Event Neural Networks show that computational savings are strongly impacted by the intensity of camera motion[8]. With slightly shaking cameras, theoretical FLOP savings were reduced by 40% compared to static camera videos, while moving cameras reduced them by 60%. Depending on the distribution of the pixel updates, the impact on real hardware is likely even higher.

Incremental Sparse Convolution uses sparse convolutions to incrementally update 3D segmentation masks online [23]. They solve a similar problem of reusing results from previously segmented regions, while allowing for new regions to be attached on the fly. Due to their reliance on non-dilating 3D convolutions [10], attaching new regions leads to artifacts along the region borders. We solve this issue using a standard, dilating convolution and by processing all outputs that are affected by the current input.

We propose MotionDeltaCNN: building upon DeltaCNN, we design and implement a sparse inference extension that supports moving cameras. Compared to DeltaCNN, we add the support for variable resolution inputs, spherical buffers, dynamic buffer allocation & initialization and padded convolutions for seamless attachment of newly unveiled regions.

3. Method

MotionDeltaCNN relies on the following concepts:

Frame Alignment The current frame is aligned with the initial frame of the sequence to maximize the overlap of consistent features (see Figure 3 *Delta*).

Spherical Buffers We use wrapped buffer offset coordinates in all non-linear layers to align them with the input (see Figure 3 *Accumulated Output*).

Dynamic Initialization When a new region is first unveiled due to camera motion, MotionDeltaCNN adds biases onto the feature maps on the fly to all newly unveiled pixels to allow for seamless integration with previously processed pixels (see Figure 3 *Bias*).

Padded Convolutions We add additional padding to convolutional layers to process all pixels that are affected by the kernel. These *dilated* pixels are stored in truncated values buffers to enable seamless connection of potentially unveiled neighbor regions in upcoming frames (see Figure 5).

3.1. Background: DeltaCNN

For a better understanding of our contributions, we first summarize the core concept of DeltaCNN, which accelerates CNN inference by processing only the first frame densely and applying sparse updates to changed pixels in subsequent frames. A sparse input *Delta* is calculated by subtracting the previously processed input from the current input frame. Pixels whose Delta is smaller than a threshold are truncated to zero to increase sparsity.

Because of linearity, the output of any convolution layer can be updated by convolving the sparse input Delta, $c(x+\delta) = c(x)+c(\delta)$. This does not hold for non-linear layers like activation (*e.g.*, ReLU) and pooling, which therefore require a separate *accumulated values buffer* to accumulate the inputs to the respective layers across all previously processed frames. The buffers contain the full, dense layer input of the previous frame, including biases from previous layers.

Convolutions with kernel sizes larger than 1x1 pixels dilate the updates across the feature map. A single pixel update in the center of the feature map might span over the entire feature map a few convolutional layers later. This can lead to nearly dense updates for the larger part of the network. DeltaCNN solves this by truncating insignificant updates not only on the input image, but at every activation layer. To avoid accumulating errors indefinitely, the truncated values are accumulated in a dedicated *truncated values buffer*. Once an accumulated, truncated value exceeds a significance threshold, the value is propagated to the next layer.

3.2. Frame Alignment

Robust frame alignment (or video stabilization) is a fundamental requirement for MotionDeltaCNN to achieve high coherence across consecutive frames. For best alignment between two frames in a 3D space, we use homography matrices with 8 degrees of freedom. Homography matrices align two images in 3D to match a selected set of key features of the initial frame - typically the static background.

We embed the image in a grid-aligned frame to ensure a pixel-perfect downscale to all layers. For example, in HRNet [29], the highest resolution feature map spans 384x384 pixels, whereas the lowest resolution feature map spans only 12x12 pixels, resulting in a scaling factor of 32x32. To ensure that all pixels align on all scaling levels, the input to HRNet must be aligned onto a grid of 32x32 pixel tiles. *E.g.*, if a frame has a 20-pixel horizontal offset relative to the first frame, we allocate 32 additional pixels, paste the original frame into the grid-aligned frame and set the 20 left-most pixels as well as the 12 right-most pixels to zero.

3.3. Spherical Buffers

MotionDeltaCNN enables support for moving cameras by introducing 2D ring buffers, *spherical buffers*. These buffers align previously processed pixels with pixels of the current
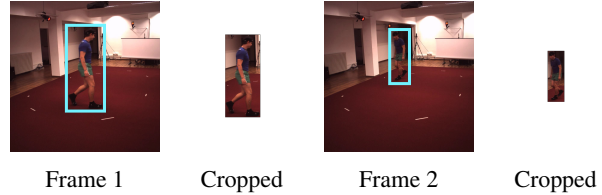

| Frame 1 | Cropped | Frame 2 | Cropped |

Figure 4: MotionDeltaCNN can handle sparse Delta inputs of different crop sizes, and focus only on a fraction of the image, thanks to the concept of spherical buffers. This is not possible for DeltaCNN which always processes sparse Delta inputs of the entire frame.

frame by translating feature map coordinates to buffer coordinates. Pixel coordinates exceeding image borders are wrapped around the buffer, *i.e.*, mapped to the opposite border using the modulo operator. This enables the reuse of previous results for moving cameras without increasing the memory overhead over DeltaCNN (see Figure 5). Our buffers support 2D offsets relative to the initial frame as well as cropped inputs (see Figure 4). Frame scales can be helpful when only a part of the image is updated in the current frame - for example, when a frame is cropped to the bounding box of a region of interest. Tile offset indices are provided during inference to align features in buffers with the current input. By multiplying tile offset indices with a layer's tile size (in the example above 32x32 pixels in highest resolution feature maps, 1x1 pixel in the lowest resolution), we get the pixel offset coordinates for buffer access in non-linear layers.

3.4. Dynamic Initialization

When the input frame covers a previously unseen area, new tiles are allocated in all buffers before inferring the frame. A new tile is allocated by deleting a previously stored tile at the desired location and resetting respective values to zero. In doing so, we lose information about the accumulated biases inferred in the initial frame. We compensate for this by adding the biases onto reset regions of the Delta feature map during inference.

3.5. Padded Convolutions

After the initial frame, MotionDeltaCNN propagates only Delta updates for the remaining sequence. This can cause inconsistencies when mixing previously seen and newly unveiled pixels. Consider a pair of *adjacent* pixels $\mathbf{p}_{seen}$ and $\mathbf{p}_{new}$ across the boundary, the resulting output at $\mathbf{p}_{new}$ may "miss" previously accumulated updates from pixel $\mathbf{p}_{seen}$ (see Figure 5), since they were "outside".

MotionDeltaCNN addresses this *prospectively* by increasing the padding of convolutional layers to include all pixels that could be affected by convolutions on the current input. This leads to an increased output size compared to the input

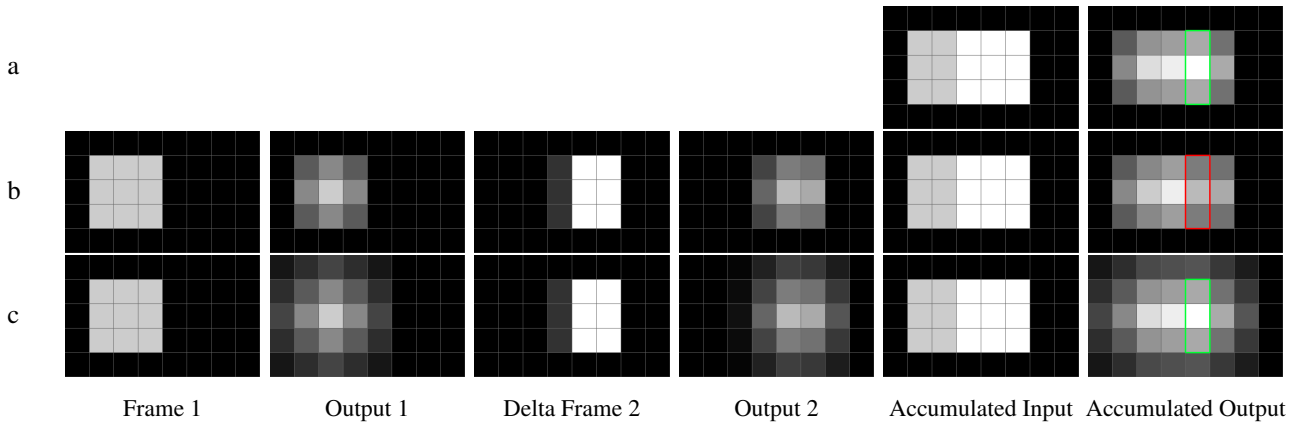| Frame 1 | Output 1 | Delta Frame 2 | Output 2 | Accumulated Input | Accumulated Output |

Figure 5: Padded convolutions enable fusing images recorded at different perspectives. Naïvely applying convolutions separately to the initial frame and the Delta of Frame 2, and aggregating the outputs (row b) does not achieve the same results as one would get if the convolution is directly applied to a large fused input buffer (row a). This is because of how the boundary pixels are treated and dropped when applying convolutions and border cropping. To address this issue, we need to cache and pad the output features, as shown in row c. See Section 3.5 for details regarding how to handle this in a practical manner.

when the kernel size is larger than 1x1. For example, a 64x64 pixel input and a 3x3 kernel result in a 66x66 pixel output.

With modern CNNs consisting of hundreds of convolutional layers, growing the feature map at each layer could quickly exceed our buffers and computational budget. To prevent this, in practice, we move and stash these *output* values at *dilated* pixels, and crop the output feature map of the current buffer to its original shape. Since MotionDeltaCNN already uses truncated values buffers for storing values that were not yet propagated through the activation layer, we only need to increase the size of the buffer slightly to include the *dilated* values and accumulate them there for later use. Combined with spherical buffers, this allows us to fuse new image regions from all 4 directions seamlessly at the cost of memory and compute overhead for the *dilated* border pixels. Since most convolutions use kernels of size 3x3 and 1x1, the overhead is typically low.

3.6. Buffer management

We use a buffer manager that is responsible for: keeping track of each tile's state, allocating new regions in the spherical buffer, initializing newly allocated tiles and adding biases to feature maps during inference. As mentioned before, spherical buffers wrap coordinates around image borders to allow for infinite panning. One downside of this approach is that when borders grow in one direction, we overwrite the *dilated* pixels stored for growing in the opposite direction (see Figure 1). Restoring *dilated* pixels is non-trivial and would require inferring the negative accumulated values of the deleted tiles - which adds high overhead. Instead, we simply delete the *dilated* pixels and restrict the growth in the opposite direction. Once the camera moves back over a restricted region, all buffers are reset entirely, and the cur-

rent frame has to be inferred densely - becoming the new reference until the next buffer reset. Restoring the truncated values by only resetting the tiles next to the invalid truncated values does not work, as they would again lack valid truncated values required for seamless attachment to persistent tiles. While a full reset reduces the reuse of previously processed features, in our evaluations, the number of resets is fairly small as camera pans typically last many frames without changing directions. Our buffer manager stores for each tile of the grid the currently held global coordinates and the directions in which the growth is restricted. Every time the camera moves, the buffer manager checks whether the tiles already hold a value from the corresponding coordinates. If not, it resets the respective tiles if possible or requests a buffer reset if growth is restricted.

3.7. Optimizations

While the above-mentioned features can already lead to a large speedup in moving camera scenarios, the speedup can be increased significantly with some small optimizations. For the DAVIS 2017 dataset [27], we noticed strong salt-and-pepper style noise in the update masks. Since video stabilization is not perfect, some pixels align incorrectly, especially in high frequency details like leaves of a tree. This leads to a strong color change for some isolated pixels in the image, resulting in a large compute overhead when processing the image in tiles. We mitigate this issue using region-of-interest (ROI) focused sensing and noise suppression. When we know the ROI from previous frames, *e.g.*, in the case of object detection or object segmentation, we can increase the sensitivity around these regions and decrease the sensitivity further away. Additionally, we suppress noise coming from imperfect image alignment by downscaling

| Frame 1 | Frame 2 | T | T+S | T+N | T+S+N |

Figure 6: We increase update sparsity using noise suppression and region of interest scaling. T uses the original truncation implementation of DeltaCNN. Scaling the Delta further away from the region of interest $(T + S)$ slightly increases sparsity. Using noise suppression $(T + N)$ to eliminate single pixel updates increases sparsity even further. Combining scaling and noise suppression $(T + S + N)$ achieves the highest sparsity, capturing only the most important updates of the moving camel. Note that in all cases, we dilate the update mask by 10 pixels; explaining the low sparsity in case T. Source: DAVIS 2017

the Delta, processing which pixels exceed the thresholds and suppressing single pixel updates using average pooling. Overall, these steps lead to a significant speedup with only minor impact on accuracy (see Figure 6). Further optimizations can be found in the supplemental material.

4. Evaluation

We evaluate MotionDeltaCNN in two scenarios: a) using video stabilization to align subsequent frames with moving cameras and b) using bounding boxes in a static camera scenario to infer only regions of interest while retaining out-of-interest features from previous frames.

Video Object Segmentation In task a), we evaluate the video object segmentation model BMVOS [7] on the DAVIS 2017 dataset [27]. BMVOS uses the DenseNet [15] backend for feature encoding, the most computationally expensive part of the model. We use weights pretrained on the DAVIS dataset, which were provided by the authors of BMVOS. The DAVIS 2017 dataset consists of a set of sequences with one or more segmented objects of which the masks are provided only for the first frame - and the model needs to paint the segmentation mask for the remaining frames in the video. Most of the videos are recorded using moving cameras, and are therefore well suited for evaluating MotionDeltaCNN. The DAVIS 2017 dataset does not provide camera extrinsics. Instead, we use OpenCV [2] to estimate homography matrices between two consecutive frames. This simplistic method fails in some cases, *e.g.*, when it does not find good features due to motion blur or with motion parallax. We see the tasks of video stabilization as orthogonal and therefore only evaluate on the set of videos (14 sequences) for which this simple video stabilization approach works reliably.

In video object segmentation, the segmentation mask from the previous frame can be used to suppress noise in the upcoming frame. We use this mask to lower the update sensitivity further away from the ROI - leading to an increase in sparsity. We deliberately tuned the thresholds for DeltaCNN with a lower accuracy target than for MotionDeltaCNN to show that we achieve higher frame rates even when targeting higher accuracy. The details about threshold tuning and

ROI-focused sensing are found in the supplemental material.

Human Pose Estimation In task b), we perform human pose estimation using *HRNet* [30, 32] on the *Human3.6M*[1] dataset [3, 17]. Human3.6M uses static cameras and is recorded in a static studio with only a single moving actor in the room. We use this dataset to evaluate a second scenario for MotionDeltaCNN: top-down pose estimation. With top-down pose estimation, humans in the scene are detected first, and then the pose is predicted for each detected human. Each bounding box is cropped and inferred individually - estimating one set of joints per person. Since Human3.6M provides per frame bounding boxes of the actors as part of their labels, we omit the step of detecting humans in the scene. We snap the given bounding boxes onto a 32x32 pixel grid and then increase them by 32 pixels on each side where possible. The increased bounding boxes ensure that joint positions are not too close to image borders, as this could negatively impact prediction accuracy. We use the same weights and thresholds as were used in the DeltaCNN paper and evaluate on the same data subset (Subject S11).

Both scenarios are compared against dense inference of cuDNN and against the original DeltaCNN without moving camera support. In HRNet, we only measure model inference time, without the overhead of post-processing or accuracy evaluation. The input shape is 384x384 with a tile size of 32x32 pixels unless stated otherwise. In the case of BMVOS, we only measure the duration of the feature encoding model to better highlight the difference of pure CNN inference speeds. For this evaluation, the camera input is embedded into a frame of shape Wx480 pixels, where W depends on the aspect ratio of the video (typically 864px) and is aligned onto a grid with a tile size of 48x48 pixels.

Hardware We conduct our evaluations on two GPUs: Nvidia GTX 1080 Ti and Nvidia RTX 3090 with 3584/10496 CUDA cores and 11GB/24GB of VRAM respectively. The evaluations are performed using 32-bit precision, but our approach can be applied to other data types as well.

---

[1]The Human3.6M dataset was accessed and used only by the co-authors at Graz University of Technology
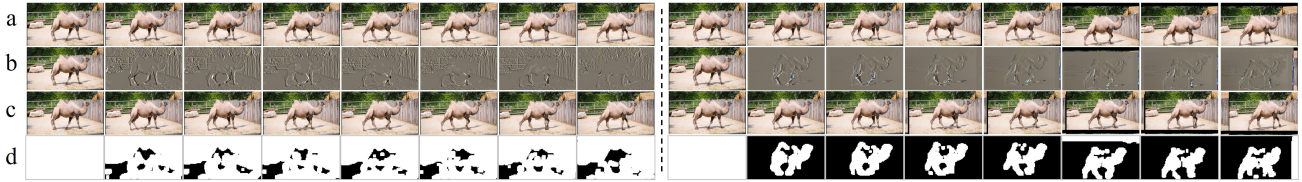
Figure 7: A DAVIS 2017 sequence processed using DeltaCNN (left) and MotionDeltaCNN (right). Row a shows the original input on the left, and the aligned and cropped input on the right. Row b visualizes the Delta between the (aligned) input and the previous input in row c. Row d visualizes the update mask with white pixels indicating an update. MotionDeltaCNN achieves much higher sparsity on the same input by reusing previous results for the aligned, static background.

5. Results

We evaluate speedup and accuracy of MotionDeltaCNN compared to DeltaCNN without explicit moving camera support as well as the dense cuDNN backend. In this section, we report the differences on the GTX 1080 Ti - for results on the RTX 3090, please see the respective tables.

Video Object Segmentation MotionDeltaCNN outperforms state-of-the-art backends in sparse mode by a margin of up to 100% over the cuDNN baseline (see Table 1). The DAVIS 2017 dataset consists mainly of videos with large bounding boxes and often fast camera motion, leading to a high average update rate of 29%.

Excluding videos without camera motion highlights the relative speedup over DeltaCNN without moving camera support with +85% speedup over cuDNN instead of +8%.

MotionDeltaCNN, like DeltaCNN, requires larger workloads with more powerful hardware to show speedups. With 10.496 cores in a RTX 3090, small batch sizes typically do not fully utilize the available hardware resources in most layers - skipping some tiles in this scenario does not affect the overall performance unless all tiles can be skipped altogether. By increasing the batch size, we can fully utilize the GPU - cores that skip a tile can immediately continue processing the next tile, leading to an overall speedup.

Over the 990 frames in the DAVIS dataset, we had to reset the buffers 9 times because of cameras panning over invalid truncated values. This low number of resets has only a small impact on the overall performance.

Human Pose Estimation MotionDeltaCNN can outperform cuDNN by up to 11x in the case of HRNet (see Table 2). Please note that the frame rate of cuDNN is 30% lower than in the DeltaCNN paper since we enforced the use of 32-bit multiplications in all layers - the same accuracy that we also use in our implementation. Furthermore, the accuracies do not match since we always use the full, uncropped images as input. The high sparsity in the Human3.6M dataset together with the efficient implementation allows us to accelerate inference significantly. Compared to DeltaCNN, which does not support varying resolution input or moving cameras, we were able to achieve 13% higher frame rates with similar

accuracy. We level the playing field in comparisons against the dense cuDNN backend by inferring the same cropped images that we use as input for MotionDeltaCNN. Still, we outperform cuDNN by up to 3.2x, and achieve a speedup of up to 3.2x over dense DeltaCNN with full resolution input. For Human3.6M, we know that the actors never cover the entire frame at once. We use this knowledge to lower the size of the buffers by 25% on each axis - reducing the memory footprint of buffers by 44% in total. These savings allow us to increase the batch size even further. When fully utilizing all available memory with an increased batch size, we achieve frame rates up to 3.9x over the cropped cuDNN baseline and 37% over DeltaCNN.

Ablation Study We evaluated the individual impact of our contributions in an ablation study (see Table 3). *Without dynamic initialization*, MotionDeltaCNN always resets the buffers when new image tiles are unveiled. After a reset, the next frame is processed densely. This has a negative impact on the speedup, but slightly increases the accuracy as it regularly flushes accumulated errors. *Padded convolutions* are key in enabling spherical buffers. Without them, new tiles are not fused into existing buffers correctly, leading to a reduction in accuracy. Our evaluation shows that *(ROI) focused sensing* enables great speedups without lowering the accuracy of the segmentation. By *disabling noise suppression* as well, the model becomes more sensitive to errors in image alignment and small changes in the image. To compensate for this, we increase the threshold of the first truncation from 0.15 to 0.35 for this evaluation. However, even with an increased threshold, the speedup is significantly smaller compared to default truncation, while the accuracy is already much lower. Overall, the ablation study shows that all parts of MotionDeltaCNN work in tandem to achieve the best speedup without large impact on accuracy.

6. Discussion

Our evaluations prove that MotionDeltaCNN can accelerate CNN video inference with moving cameras where existing sparse methods fail to outperform the dense reference. However, our evaluation also shows that freely moving cameras, especially when combined with dense updates as in

Videos	Backend	IoU	GFLOPs	GTX 1080 Ti b=1		GTX 1080 Ti b=5		RTX 3090 b=1		RTX 3090 b=12	
				FPS	speedup	FPS	speedup	FPS	speedup	FPS	speedup
	cuDNN	76.85	22.1	30.7	1.0	40.9	1.0	45.4	1.0	103	1.0
	DeltaCNN (dense)	76.85	22.1	35.0	1.14	41.2	1.01	66.5	1.46	100	0.97
All	MotionDeltaCNN (dense)	76.76	22.6	33.5	1.09	40.3	0.99	57.7	1.27	93.6	0.91
	DeltaCNN (sparse)	76.18	15.5	42.5	1.38	51.4	1.26	**70.0**	**1.54**	125	1.21
	MotionDeltaCNN (sparse)	76.32	**6.8**	**54.8**	**1.79**	**81.9**	**2.00**	65.2	1.44	**215**	**2.1**
	cuDNN	79.59	22.1	30.7	1.0	40.9	1.0	45.4	1.0	103	1.0
Moving	DeltaCNN (sparse)	79.38	18.8	38.2	1.24	44.1	1.08	**70.2**	**1.55**	106	1.03
	MotionDeltaCNN (sparse)	79.43	**7.3**	**52.3**	**1.70**	**75.1**	**1.84**	65.0	1.43	**201**	**1.95**

Table 1: Results for the task of video object segmentation using BMVOS on the DAVIS dataset. We track only the DenseNet backend for evaluating frames per second (FPS). For accuracy benchmarks, the remaining network is processed densely. GFLOPs are reported as average over the test set. Both devices are evaluated with a batch size b of one, and the maximum batch size each device can hold in memory.

Backend	PCKh@0.5	PCKh@0.2	GFLOPs	GTX 1080 Ti b=1		GTX 1080 Ti b=16		RTX 3090 b=1		RTX 3090 b=40	
				FPS	speedup	FPS	speedup	FPS	speedup	FPS	speedup
cuDNN	96.03%	83.33%	47.1	22.5	1.0	24.8	1.0	12.3	1.0	69.6	1.0
DeltaCNN (dense)				24.2	1.08	41.3	1.7	26.0	2.1	95.8	1.4
MotionDeltaCNN (dense)	96.00%	83.32%	17.7	32.5	1.44	91.4	3.7	22.0	1.8	232	3.3
cuDNN (cropped)	95.92%	83.11%	13.3	24.0	1.07	71.0	2.9	11.8	1.0	209	3.0
DeltaCNN (sparse)	95.95%	82.31%	3.55	**44.9**	**2.0**	202	8.1	**26.2**	**2.1**	547	7.9
MotionDeltaCNN (sparse)	95.95%	82.33%	3.45	41.2	1.83	228	9.2	22.6	1.8	584	8.4
MotionDeltaCNN (sparse †)	95.93%	82.22%	**3.44**	42.5	1.89	**276**	**11.1**	22.5	1.8	**691**	**9.9**

Table 2: Speed and accuracy comparisons of different CNN backends used for pose estimation on the Human3.6M dataset. The same set of auto-tuned thresholds for update truncation is used for all devices and batch sizes b. † limits the size of the input from 384x384 to 288x288 pixels - freeing up 44% of the memory used for buffers. The freed-up memory is used to increase the batch size to fully utilize the available VRAM (b=28 on 1080 Ti and b=66 on 3090). cuDNN cropped uses the same, cropped input as ours, instead of the full camera input.

	FPS	IoU	# resets
cuDNN	40.9	79.59	-
DeltaCNN	44.1	79.38	-
MotionDeltaCNN	**75.1**	**79.43**	**8**
- w/o dynamic init	56.4	79.63	110
- w/o padded conv.	65.3	77.89	8
- w/o ROI scaling	63.6	79.33	8
- w/o ROI & noise*	50.9	79.06	8

Table 3: Ablation study. Evaluating BMVOS on DAVIS2017 dataset using a Nvidia GTX 1080 Ti with a batch size of 5. We evaluate on the subset of videos with significant camera motion. * increased threshold on video input.

the DAVIS dataset, lead to a much smaller speedup than the static camera setup with sparse motion in Human3.6M. Furthermore, the additional padding slightly increases memory and compute overhead over DeltaCNN.

Receptive Field While MotionDeltaCNN does achieve outputs comparable to standard CNN inference, it should be noted that even in dense mode, the results are not identical. This can be observed in Figure 5 where the fused output of MotionDeltaCNN matches the result using the fused input in row a, which includes information from neighbor pixels seen in previous frames. Since we do not actively propagate inverted accumulated values once a previously processed region gets out of view, the receptive field of later layers, also for pixels inside the current view, still contains some information from that now unseen area. This can have positive and negative effects alike. These features might not be valid anymore. At the same time, some out-of-view features can help to make better predictions due to additional context.

Applications MotionDeltaCNN is a general-purpose framework for video inference and can thus be used with all pre-trained CNNs by replacing the original, dense layers with our sparse equivalent. As additional input, MotionDeltaCNN requires camera extrinsics for the frame to be processed. While most publicly available datasets do not

include camera extrinsics and therefore require image stabilization, smartphones, tablets and VR headsets nowadays have built-in support for tracking the camera pose between frames in real-time. Potential applications are not limited to classic computer vision applications like tracking and segmentation, but it could also be used for video up-scaling, filtering, etc. However, applications like image-denoising (diffusion) are likely not a good fit as they provide low levels of spatial update sparsity between iterations and they are very sensitive to noise.

Limitations One of the main limitations of our approach is its dependency on robust video stabilization. Our experiments with the DAVIS dataset showed that simple video stabilization techniques can break when blur or parallax is involved, or when it fails to find robust features. Parallax can greatly reduce the gains of our method, even when the scene itself is mostly static. When foreground and background layers move at different speeds, frames can only be properly aligned to stabilize one of these layers - leading to dense updates in layers moving at different speeds.

MotionDeltaCNN also comes with the trade-off between buffer overallocation and losing information by cropping the input to match the shape of the buffer. In our evaluations, when the camera zooms or rotates around the forward axis (*roll*), the aligned input can exceed the size of the buffers, and therefore requires slight cropping to fit into the target shape (see Figure 7). Alternatively, the buffers could also be increased to allow for larger inputs in upcoming frames - at the cost of higher memory consumption.

7. Conclusion

We propose MotionDeltaCNN, an extension to the sparse video CNN framework DeltaCNN adding support for moving camera input. To the best of our knowledge, we are the first to propose a solution for moving cameras with sparse inference. MotionDeltaCNN uses spherical buffers, padded convolutions and dynamic initialization of newly unveiled regions to support combining previously processed pixels and newly unveiled pixels together with high sparsity. We test our approach in practice by implementing all kernels natively on the GPU to evaluate the real-world performance with modern hardware. Our experiments show that we outperform DeltaCNN by up to 90% on moving camera input and that MotionDeltaCNN can also be beneficial to lower the memory footprint in static camera scenarios.

References

[1] Udari De Alwis and Massimo Alioto. TempDiff: Temporal Difference-Based Feature Map-Level Sparsity Induction in CNNs with <4% Memory Overhead. *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems, AICAS 2021*, pages 1–4, Jun 2021.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Cristian Sminchisescu Catalin Ionescu, Fuxin Li. Latent structured models for human pose estimation. In *International Conference on Computer Vision*, 2011.

[4] Lukas Cavigelli and Luca Benini. CBinfer: Exploiting Frame-to-Frame Locality for Faster Convolutional Network Inference on Video Streams. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(5):1451–1465, May 2020.

[5] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DianNao family: Energy-efficient hardware accelerators for machine learning. *Communications of the ACM*, 59(11):105–112, 2016.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DaDianNao: A Machine-Learning Supercomputer. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2015-January:609–622, Jan 2015.

[7] 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.

[8] Matthew Dutson, Yin Li, and Mohit Gupta. Event neural networks. 2021.

[9] Zhipeng Fan, Jun Liu, and Yao Wang. Adaptive Computationally Efficient Network for Monocular 3D Hand Pose Estimation. Technical report, 2020.

[10] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, Nov 2018.

[11] Amirhossein Habibian, Davide Abati, Taco S. Cohen, and Babak Ehteshami Bejnordi. Skip-Convolutions for Efficient Video Processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2695–2704, Jun 2021.

[12] Amirhossein Habibian, Haitam Ben Yahia, Davide Abati, Efstratios Gavves, and Fatih Porikli. Delta Distillation for Efficient Video Processing. *arXiv (ECCV sub)*, 2022.

[13] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, pages 243–254, 2016.

[14] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, 2015-January:1135–1143, 2015.

[15] 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.

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18:1–30, 2018.

[17] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014.

[18] Samvit Jain, Xin Wang, and Joseph E. Gonzalez. Accel: A corrective fusion network for efficient semantic segmentation on video. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:8858–8867, Jun 2019.

[19] CUN Le. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2:598–605, 1990.

[20] Yule Li, Jianping Shi, and Dahua Lin. Low-Latency Video Semantic Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 5997–6005, 2018.

[21] Feng Liang, Ting-Wu Chin, Yang Zhou, and Diana Marculescu. Ant: Adapt network across time for efficient video processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2603–2608, June 2022.

[22] Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. *33rd International Conference on Machine Learning, ICML 2016*, 6:4166–4175, 2016.

[23] Leyao Liu, Tian Zheng, Yun-jou Lin, Kai Ni, and Lu Fang. INS-Conv : Incremental Sparse Convolution for Online 3D Segmentation. *Cvpr*, pages 18975–18984, 2022.

[24] Bert Moons, Koen Goetschalckx, Nick Van Berckelaer, and Marian Verhelst. Minimum energy quantized neural networks. *Conference Record of 51st Asilomar Conference on Signals, Systems and Computers, ACSSC 2017*, 2017-October:1921–1925, Apr 2018.

[25] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent Residual Module for Fast Inference in Videos. Technical report, 2018.

[26] 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.

[27] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv:1704.00675*, 2017.

[28] Laurent Sifre and Stéphane Mallat. PhD Thesis, Ecole Polytechnique, CMAP Rigid-Motion Scattering For Image Classification. 2014.

[29] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:5686–5696, 2019.

[30] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *CVPR*, 2019.

[31] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. Technical report, 2019.

[32] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *European Conference on Computer Vision (ECCV)*, 2018.

[33] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. Towards High Performance Video Object Detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7210–7218, 2018.

[34] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:4141–4150, 2017.