# ReMove:Leveraging Motion Estimation for Computation Reuse in CNN-Based Video Processing

Masoumeh Khodarahmi
*School of ECE, College of Eng., University of Tehran*
Tehran, Iran
khodarahmi.m@ut.ac.ir

Mehdi Modarressi
*School of ECE, College of Eng., University of Tehran*
*School of CS, Institute for Research in Fundamental Sciences (IPM)*
Tehran, Iran
modarressi@ut.ac.ir

Ardavan Elahi
*Faculty of EE and IT, Vienna University of Technology*
Vienna, Austria
ardavan.elahi@tuwien.ac.at

Farhad Pakdaman
*Faculty of IT and Comm. Sci., Tampere University*
Tampere, Finland
farhad.pakdaman@tuni.fi

***Abstract*—In this paper, we propose a method to reduce the computational load of Convolutional Neural Networks (CNNs) when processing video frames by exploiting computation reuse based on input similarity. Specifically, our approach leverages the temporal redundancy present in video sequences. In the existing computation reuse methods, if certain pixels of two consecutive frames are similar, the computations related to those pixels are skipped, and the results are reused from the previous frame. While pixel-wise comparison between consecutive frames can introduce overhead and partially offset computation reduction, we mitigate this by utilizing motion estimation information inherent in coded video frames. Motion estimation indicates whether a current block of the frame has already appeared in previous frames, allowing for direct reuse of computations without additional comparison overhead. Furthermore, we optimize by fusing CNN layers until the block size becomes smaller than the filter size, ensuring that not only the first layer's computations but also multiple CNN layers' computations are skipped. The experimental results demonstrate an average reduction of 35.4% in the computation amount of the VGG-16 CNN model with no significant loss in accuracy.**

***Index Terms*—CNN, computation reuse, input similarity, motion vector, computation reduction**

## I. Introduction

According to Juniper Networks, global IoT connections are expected to reach billions by 2024, indicating significant growth in IoT deployment across various sectors [1]. Convolutional Neural Networks (CNNs) play a crucial role in such devices due to their effectiveness. In computer vision, CNNs have enabled advancements previously thought impossible, including facial recognition, self-driving cars, self-checkout systems, and advanced medical diagnostics [2]. As deep learning technology rapidly progresses, the structure of CNNs is becoming increasingly complex and diverse [3]. The large amount of computation in convolutional neural networks results from extensive matrix multiplications and convolutions required to process high-dimensional input data across numerous layers. This issue can be seen in Figure 1, which shows the structure of a neural network as an example.

As CNNs become integral to real-time applications, there is an increasing need to accelerate their performance, particularly for embedded systems, which are often employed in mobile and edge devices. Efficient computation reduction is crucial for enabling real-time performance and ensuring the feasibility of deploying applications on embedded systems. By significantly reducing the computational load, it becomes possible to process data more quickly, thereby meeting the low-latency requirements essential for real-time applications. This reduction not only speeds up processing times but also decreases power consumption, which is a critical factor for embedded systems. Computation reuse is one of the most effective strategies for reducing neural network computations. The importance of computation reuse in Convolutional Neural Networks (CNNs) cannot be overstated, as it is vital for optimizing performance and alleviating the substantial memory bandwidth demands inherent in these networks. CNNs involve multiple layers of computations where each layer's output serves as the input for the next, creating significant opportunities for data reuse [4]. This includes several strategies such as filter reuse, where filters from previous layers are

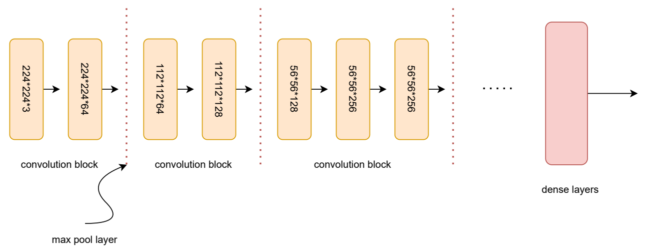


Fig. 1: CNN structure

reused to reduce computation costs [5]; feature map reuse, which involves sharing intermediate feature maps between layers to save memory and computation time [6]; weight reuse, which leverages weights from previous training iterations to improve convergence and model performance [7], [8]. While these approaches achieve significant improvements in speed with negligible loss of accuracy, similarities in the input data can be exploited to further improve the computation reuse. Especially in video applications, as the video data is quite redundant, it leads to significantly high computational load. Previous works such as [9], [10] tried to use the spatial and temporal similarities in video data for computation reuse. Some previous works focus on extracting input similarity to reduce the complexity of deep neural networks (DNNs) by skipping computations related to inputs with negligible changes and reusing results from previous executions, rather than computing the entire DNN [9]–[11]. However, these methods involve some run-time overhead for detecting input similarity, as they typically require pixel-wise comparisons of each frame with the previous one to identify identical inputs and skip the associated computations. To address this issue, we explored methods to reduce the overhead associated with these comparisons. In the input reuse method, the information embedded in coded videos was not utilized to reuse input data within the network.

Video compression is an integral part of many real-worlds vision applications, as it facilitates transmission of bulky video content. Compressed video includes important information, such as motion vectors, which provide useful description of video content. Motion estimation and compensation is a fundamental component in modern video compression standards such as H.264/AVC and H.265/HEVC [12], [13], facilitating efficient video data compression, by accounting for motion between frames. This technique significantly reduces redundancy by finding motion vectors, estimating the movement of scene content (such as objects) from one frame to another, enabling effective data compression and transmission. Inspired by the fact that motion information arrays provide ready-to-use information regarding input data similarities, we propose to exploit them to facilitate computation reuse in CNNs for computer vision based task.

In this paper, we first utilize a fused-layer CNN, as introduced in [14], to reduce feature map data transfer. Then we propose an algorithm to find video blocks with potential for computation reuse, and means to approximate their computations, based on decoded motion vectors. Furthermore, we propose to use motion estimation errors (derived from the compressed video) to control the risk of approximation. Experimental results validate that this method can save significant computations, while leading to negligible loss of the final feature map error.

Based on the above discussion, the contributions of this paper are as follows:

- A method for computation reuse in video applications, based on extracting motion vectors and similarity percentage. The proposed method is used to accelerate an example network VGG-16, and is used for input reuse operations at different layers of the network.
- Implementation of the entire system on publicly available and custom design tools and investigating the accuracy loss and the reduction in computations within the network.

The rest of the paper is organized as follows. Section 2 introduces some preliminary concepts related to our contributions. Section 3 reviews some related work. In Sections 4 and 5 the proposed method and its challenges are presented. Section 6 presents the evaluation results, followed by the conclusion of the paper in Section 7.

## II. Preliminaries

### *A. Motion Estimation*

Video compression aims to reduce bit rate by removing redundancies in raw video data, primarily through motion estimation. Typically, video streams are decoded to extract frames to be processed by Deep Neural Network (DNN), ignoring the valuable similarity information in the encoded frames. We propose using this existing motion information to identify frame similarities, thus avoiding the costly process of detecting similarities for each frame.

Consecutive frames in a video sequence include redundant information, as adjacent frames in natural video scenes are fairly similar. Most of the differences among successive frames are due to movement of objects within the frame, known as motion. Motion estimation is the process of determining these movements. The block matching algorithm (BMA) is the most commonly used motion estimation technique, dividing the current frame into blocks of size N×N and finding a motion for each block to represent its motion w.r.t a previously decoded frame. Choosing the block size, N, for motion estimation

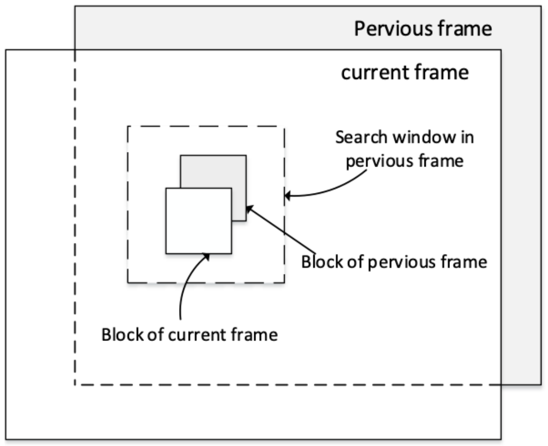


Fig. 2: Block matching algorithm

involves balancing accuracy and processing overhead. Smaller blocks reduce error but increase processing time and power, while too small blocks lose structure, making motion tracking difficult. Each block in the current frame is matched with corresponding blocks in the previous frame and within a surrounding window (N+2w) × (N+2w), where w is the maximum motion displacement. Larger w increases computational load. The best match is often identified through comparing a matching criterion such as the mean of absolute difference (MAD), leading to a motion vector with two parameters (a, b) for each block.The MAD is defined as:

$$\text{MAD}(i,j) = \frac{1}{N^2}\sum_{m=1}^{N}\sum_{n=1}^{N}|f(m,n) - g(m+i,n+j)| \quad (1)$$

where $(-w \leq i,j \leq w)$.

where $f$(m, n) represents pixels of the current block of $N^2$ pixels at the coordinate (m, n) and g(m+i, n+j) represents the corresponding pixel in the previous frame at the new coordinate (m+i, n+j). At the best-matched position of i = a and j = b, the motion vector, MV(a, b), represents the displacement of all the pixels within the block. Figure 3 shows the search window for a single block in the algorithm. The dark square is the block in position (m, n). The first point in the search window is (m-w, n-w) and the last point to be processed is (m+w, n+w). Between the best-matched block in the reference frame and the current block, there is the minimum MAD called minMAD.

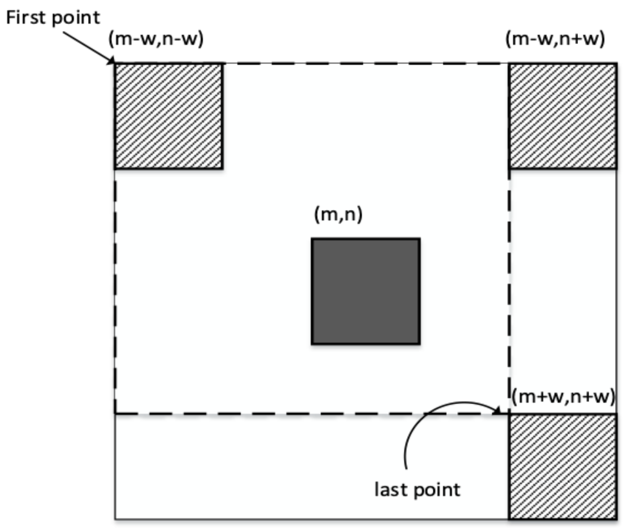


Fig. 3: The search window for block matching

### B. Layer Fusion

According to [14], layer fusion in convolutional neural networks (CNNs) is a technique aimed at optimizing data flow across multiple convolutional layers by processing them concurrently rather than sequentially. Traditional CNN implementations process each layer independently, resulting in large amounts of intermediate data that need to be stored and retrieved from off-chip memory. This approach is inefficient and incurs significant data transfer overhead as CNNs become deeper and more complex.

The layer fusion technique addresses this inefficiency by restructuring the CNN computation so that multiple layers are computed together. This is accomplished by modifying the order in which input data is brought on-chip, creating a pyramid-shaped multi-layer sliding window. This approach allows intermediate values to be computed on-chip and passed directly between layers, thereby eliminating the need for repeated off-chip memory access. By leveraging the spatial locality of convolution operations, layer fusion enables effective on-chip caching and significantly reduces off-chip memory bandwidth requirements. Overall, layer fusion enhances the performance and efficiency of CNN accelerators by minimizing data movement and maximizing data reuse on-chip. Figure 4 illustrates an example of fusing two convolutional layers.

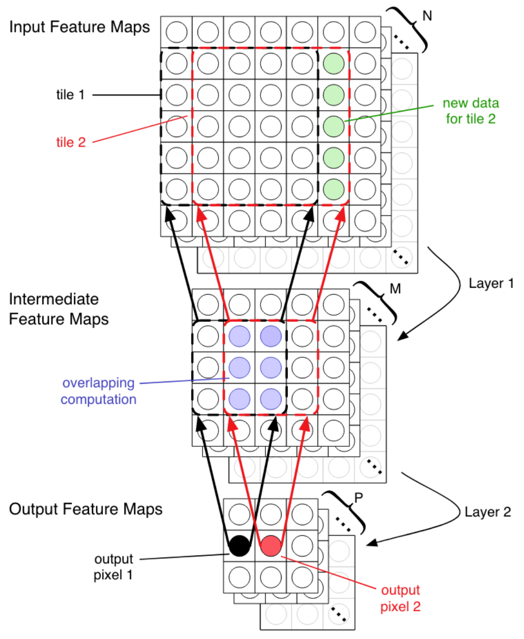


Fig. 4: The example of fusing two convolutional layers [14]

## III. Related Work

**Input induced computation reuse** aims to reduce calculations related to input data. Several works, such as [9], [15], [16], have addressed this area. In [9], the similarity of inputs is discussed for the first time as a means to reduce computations. In [15], a similar procedure is employed for the computation reuse aspect of the paper. Inspired by the method in [9], [16] reduces the amount of network computations by using the incremental operation method and then applying the Winograd algorithm [17] instead of the original convolution.

However, this basic method involves comparing a block of the current frame with the same block in the source frame. If they are equal, the results of the source block replace the results of the current block. This process occurs in every convolutional layer, resulting in the overhead of comparison for all blocks and all layers.

**Use motion estimation for inference** Motion vectors have been effectively utilized in recent research to enhance object detection and tracking in video analysis. For example, the Alchemist accelerator design leverages built-in motion vectors from compressed video streams, significantly reducing decoding latency and computational overhead [18]. Similarly, the "Online MOT Tracker in Compressed Domain (OTCD)" employs a combination of key and non-key frames, using restored RGB images for detection in key frames and motion vectors for propagation in non-key frames. This approach leads to substantial speedup in multi-object tracking [19]. Both methods highlight the effectiveness of using motion vectors directly from compressed video for efficient real-time processing in object detection.

However, both methods also show potential drawbacks, such as missed detections or false positives. The Alchemist method [18], while energy-efficient, may overlook finer details that are not represented in the motion vectors. Similarly, the OTCD method struggles with handling scale variations in bounding boxes, as the motion vectors used for displacement prediction do not account for changes in object size [19].

## IV. Method

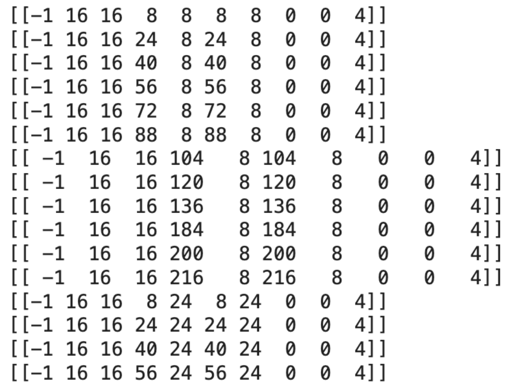
### A. Motion Estimation Analysis

According to [19], frames in a compressed video are organized into Groups of Pictures (GOP), which consist of three general types of frames: I-frames (intra-coded frames), P-frames (predictive frames), and B-frames (bi-directional frames). I-frames are not dependant on other frames, while P-frames and B-frames are encoded using motion vectors (MVs) and residuals (the remaining error from motion estimation). For simplicity, we adopt the same assumption as [19], [20] that the compressed video contains only I-frames and P-frames.

Initially, we extracted motion vectors by combining tools such as MVS [21] with custom designs. This process yielded the motion vectors for each block, as well as the motion estimation error. We then selected those blocks based on corresponding motion vectors where the Mean Absolute Difference (MAD) was below a predetermined threshold, which we set to 5. At this stage, we gathered a number of suitable motion information arrays. Each motion information array includes information such as the block size, the starting points of the blocks in both frames, the motion vector and the scale of the motion vector. Given the appropriate motion vectors and their information, we proceeded to reuse the computations. Figure 5 exemplifies motion information for a few blocks. This process occurs on the encoder side during compression and does not add any time overhead. For each block where the MAD is less than the threshold, we assign a tag of 1, and for the remaining blocks, we assign a tag of 0. These tags are sent along with the supplementary information.

### B. Computation Reuse

To reuse the computations from the reference frame in the current frame, we utilize the motion vectors extracted in the previous step. This approach allows us to directly place the block results from the network layers, corresponding to the selected motion vectors, into the feature map of the current frame, bypassing the need for comparison overhead.



```
[[-1 16 16  8  8  8  8  0  0  4]]
[[-1 16 16 24  8 24  8  0  0  4]]
[[-1 16 16 40  8 40  8  0  0  4]]
[[-1 16 16 56  8 56  8  0  0  4]]
[[-1 16 16 72  8 72  8  0  0  4]]
[[-1 16 16 88  8 88  8  0  0  4]]
[[ -1  16  16 104   8 104   8   0   0   4]]
[[ -1  16  16 120   8 120   8   0   0   4]]
[[ -1  16  16 136   8 136   8   0   0   4]]
[[ -1  16  16 184   8 184   8   0   0   4]]
[[ -1  16  16 200   8 200   8   0   0   4]]
[[ -1  16  16 216   8 216   8   0   0   4]]
[[-1 16 16  8 24  8 24  0  0  4]]
[[-1 16 16 24 24 24 24  0  0  4]]
[[-1 16 16 40 24 40 24  0  0  4]]
[[-1 16 16 56 24 56 24  0  0  4]]
```

Fig. 5: Motion information array structure

However, we faced several challenges, such as the presence of padding and strides, which is discussed in detail in the challenges section. The algorithm is shown in 1.

To implement this method in a real network, we need to access the output of the layers to modify them and compare the final output of the model; First output is the actual output and the other is the reused output. Finally, to evaluate the accuracy, we compare these two outputs.

The most suitable network for this reuse method is the VGG [22] network (VGG-16) because it is an efficient network with stride = 1 and 3×3 filters. In this network, due to stride = 1, the dimensions of the blocks are reduced by two units by passing through each convolutional layer. For example, $16 \times 16$ blocks become $14 \times 14$ blocks after passing through the first layer. In this research, we changed the output of the 7th layer and checked the results. In fact we ignored the effect of paddings and strides. Therefore, in the 7th layer, we reused the 4x4 blocks.

## V. Challenges

In this step, we selected only those vectors whose minMAD was below a certain threshold for extracting the motion vector phase. As a result, the entire feature map could not be reused and for example, only 60% of an image could be reused. Another challenge was padding. Although the feature map size remains unchanged as it passes through each convolutional layer, the size of the blocks is reduced, leading to a decrease in the extent of reuse. Consequently, the actual reuse fell below 60%. To address this issue, instead of replacing $2 \times 2$ blocks in the 7th layer, we used $4 \times 4$ blocks for replacement, which resulted in a slight increase in MSE error. However, this method is content-dependent, so different video types resulted in varying levels of computation reduction and MSE error. Figure 6 shows how this approximation is done in the first two layers to clarify this issue. In this case, two rows or columns of pixels are approximated from each side of the feature map. This approximation continues to the next layers until the approximation can no longer be applied from the motion vector. Figure 7 shows the procedure.

**Algorithm 1** Computation reuse based on motion information array

```
Require: $mv\_arr$: List of motion vectors
Require: $frames\_arr$: List of video frames
 1: for all $f \in frames\_arr$ do
 2:     if $f$ is I-frame then
 3:         Apply full computations
 4:     else
 5:         while True do
 6:             if End of frame then
 7:                 return
 8:             end if
 9:             if The tag of the block's start point is 0 then
10:                 Apply full computations to the end of the block with fused architecture
11:             else
12:                 Copy the corresponding block in the reference feature map to the current feature map based on $mv\_arr$
13:             end if
14:         end while
15:     end if
16: end for
```

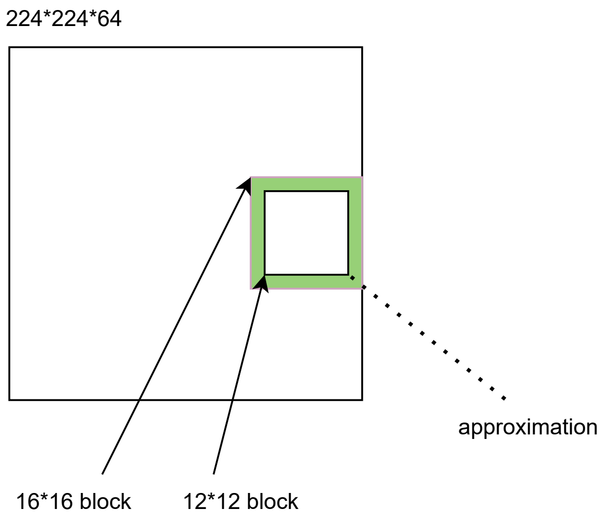


Fig. 6: Approximating a 12x12 block with a 16x16 block

## VI. Evaluation And Results

The study presents an analysis of accuracy and computational volume reduction in quantized networks, specifically focusing on uint8 quantization. Quantized networks offer a balance between accuracy and computational efficiency, making them a promising approach for optimizing performance in various applications. To evaluate the performance of this reuse method, the amount of computation reduction should first be measured.

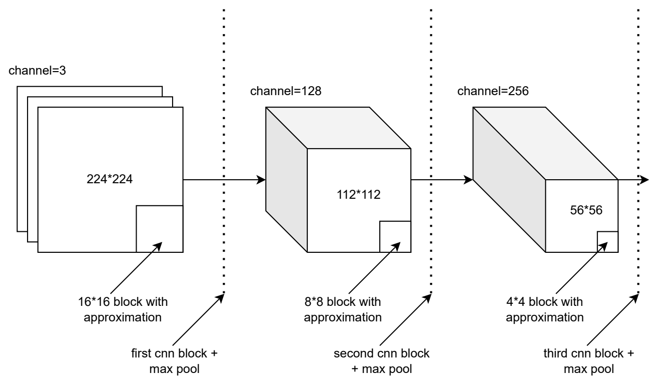

channel=3
224*224
16*16 block with approximation
first cnn block + max pool
channel=128
112*112
8*8 block with approximation
second cnn block + max pool
channel=256
56*56
4*4 block with approximation
third cnn block + max pool

Fig. 7: Approximation and reuse effect in diiferent CNN blocks

### A. Computation Reduction

To calculate the amount of computations each layer in the VGG-16 architecture has to perform, we need to consider the number of parameters (weights) in each layer and the number of operations required to process the input data through that layer.

- Convolutional Layers: For each convolutional layer, the number of computations can be calculated using the formula:

$$\begin{aligned}\text{Number of computations} &= \text{input channel}\times \\ &\text{output channels} \times \text{filter height} \times \text{filter width} \\ &\times \text{output height} \times \text{output width}\end{aligned}$$

- Max Pooling Layers: Max pooling layers simply select the maximum value from a set of values. They don't involve any additional computations beyond selecting the maximum.
- Fully Connected Layers: The number of computations in a fully connected layer can be calculated as:

$$\text{Number of computations} = \text{input width} \times \text{output width}$$

By evaluating all the layers of this network, we found that for an input image with dimensions of $224 \times 224 \times 3$, approximately 15.7 billion computations are needed. About 12% of the computations are related to the first two layers, around 30% are related to the first four layers, and approximately 59% are related to the first seven convolutional layers, and so on. Considering that more than half of the blocks have a minMAD lower than or equal to the specified value(5), a large part of these layers are reused and skipped. In Figure 8 and Figure 9, the reduction percentage for each video can be seen in the seven-layer-reuse method. Since the number of motion vectors that can be used depends on the video content, for each video, according to the diagram, we have a reuse count based on the selected motion vectors, all of which are numbers less than or equal to 59%. In this network, we were able to reuse up to 7 layers (by approximating border point computations).

## B. Accuracy loss

We performed this experiment by estimating the border points and replacing the $4 \times 4$ blocks. Instead of using the $2 \times 2$ blocks, we opted for $4 \times 4$ blocks in 7-layers reuse. These adjustments led to a reduction in MSE of the last layer of CNN layers. Accuracy is defined as the ratio of correctly predicted frames that have been reused according to the reference frame, to the total number of frames. Based on [9], quantized network has many similar calculations despite its acceptable performance, so we also used the VGG quantized model in this experiment. Because the accuracy of the quantized model is lower than the accuracy of the original model, sometimes both sets of results classify the images incorrectly; therefore, most of the obtained classification accuracies are 100% (Figure 8). According to [23], CNNs are sensitive to changes in the high frequencies of an image but resistant to changes in low frequencies, which are coarse structures. This robustness in deeper layers is due to their dealing with more abstract, high-level features, which makes them less susceptible to fine-grained noise. Therefore, the more we reuse computations in deeper layers, the better the final classification accuracy of the model becomes, eventually reaching 100% accuracy. This balance between computation reduction and accuracy is crucial for applications requiring efficient processing without compromising performance. But comparing the feature maps in the last convolution layer and the final vector, which consists of 1000 numbers after the dense layers, in order to make our method universal and apply it to other networks with different applications, is the best option to report the difference. The appropriate comparison criterion is mean squared error (MSE) for reporting the difference between two feature maps. In Figure 9, the MSE error for 5 different videos can be seen. This figure shows the impact of video content on computation reduction and MSE error in our method. To show the effect of different thresholds for motion vector selection, we compared two $1 \times 1000$ feature maps and then drew a digram of MSE for 5 different threshold values (5, 7, 8, 10, 15). Figure 10 shows the low impact of threshold on the MSE of the last video, while they have a significant impact on reducing computations.

TABLE I: Details of Selected Videos [24], [25]

| Video Label | Video Name | Frame Count | Rate(fps) |
|---|---|---|---|
| Video1 | moving_bikes.mp4 | 67 | 50 |
| Video2 | woman.mp4 | 88 | 25 |
| Video3 | Sponge_Bob.mp4 | 43 | 25 |
| Video4 | people_near_desk.mp4 | 66 | 25 |
| Video5 | people_on_the_grass.mp4 | 67 | 50 |

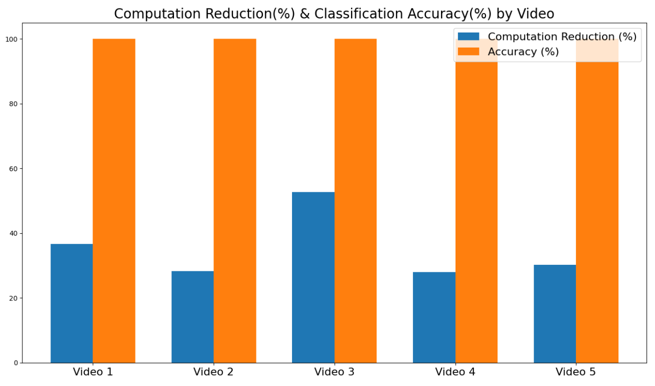


Fig. 8: Bar chart of accuracy percentage and computation reduction percentage in different 5 videos for a fully quantized uint8 network

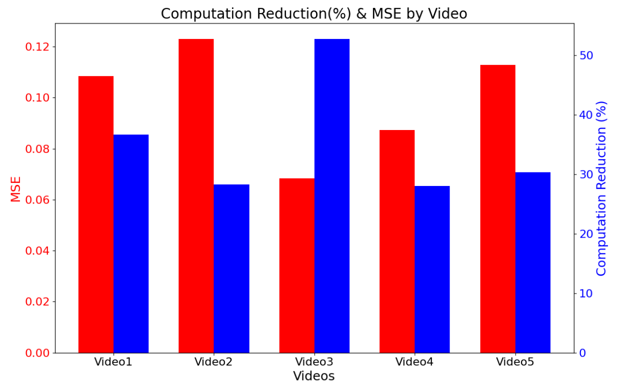


Fig. 9: Bar chart of MSE and computations reduction in different 5 videos for a fully quantized uint8

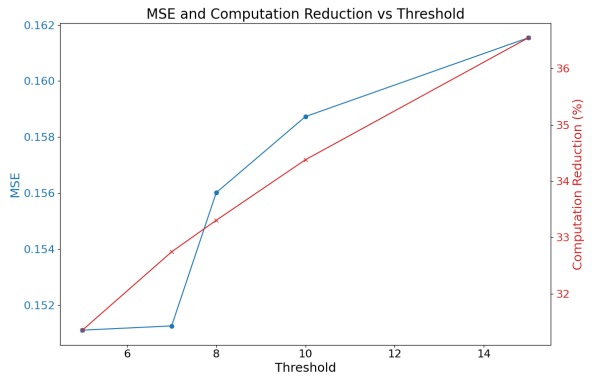


Fig. 10: MSE and computation reduction vs threshold

## C. dataset and implementation

We selected 5 different types of videos from [24], [25] datasets ensuring they cover most types of motions. For our CNN model, we utilized the TensorFlow quantization library

[26] to enhance computational efficiency through post-training quantization techniques.

## VII. Conclusion

In this paper, we reused the computations of video blocks that are similar to their corresponding blocks at the reference frame, using the information available from motion estimation. This reuse continues not just for one or two layers but up to 7 layers without significant error. We demonstrated that we could reduce the computations of the VGG-16 convolution neural network by an average of 35.4%.

## References

[1] J. Networks, "Iot connections to reach 83 billion by 2024: Report," CISO MAG. [Online], 2024, [Accessed: Jul. 08, 2024]. [Online]. Available: https://cisomag.com/iot-connections-to-reach-83-billion-by-2024-report/

[2] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022.

[3] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, no. 4, p. 99, 2024. [Online]. Available: https://doi.org/10.1007/s10462-024-10721-6

[4] A. Gondimalla, J. Liu, M. Thottethodi, and T. N. Vijaykumar, "Occam: Optimal data reuse for convolutional neural networks," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 1, dec 2022. [Online]. Available: https://doi.org/10.1145/3566052

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint*, 4 2017.

[8] H. Mahdiani, A. Khadem, A. Ghanbari, M. Modarressi, F. Fattahi-Bayat, and M. Daneshtalab, "$\delta$nn: Power-efficient neural network acceleration using differential weights," *IEEE Micro*, vol. 40, no. 1, pp. 67–74, 2020.

[9] M. Riera, J.-M. Arnau, and A. Gonzalez, "Computation reuse in dnns by exploiting input similarity," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 57–68.

[10] F. Alantali, Y. Halawani, B. Mohammad, and M. Al-Qutayri, "Slid: Exploiting spatial locality in input data as a computational reuse method for efficient cnn," *IEEE Access*, vol. 9, pp. 57 179–57 187, 2021.

[11] V. Janfaza, K. Weston, M. Razavi, S. Mandal, F. Mahmud, A. Hilty, and A. Muzahid, "Mercury: Accelerating dnn training by exploiting input similarity," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2023, pp. 638–650. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HPCA56546.2023.10071051

[12] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[13] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[14] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 10 2016, pp. 1–12.

[15] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 1953–1965, 2020.

[16] S. Li, Q. Wang, J. Jiang, W. Sheng, N. Jing, and Z. Mao, "An efficient cnn accelerator using inter-frame data reuse of videos on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1587–1600, 2022.

[17] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4013–4021.

[18] Y. Wang, Y. Wang, H. Li, Y. Han, and X. Li, "An efficient deep learning accelerator for compressed video analysis," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[19] Q. Liu, B. Liu, Y. Wu, W. Li, and N. Yu, "Real-time online multi-object tracking in compressed domain," *IEEE Access*, vol. 7, p. 76489–76499, 2019. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2019.2921975

[20] S. R. Alvar and I. V. Bajić, "Mv-yolo: Motion vector-aided tracking by semantic object detection," 2018. [Online]. Available: https://arxiv.org/abs/1805.00107

[21] L. Bommes, X. Lin, and J. Zhou, "Mvmed: Fast multi-object tracking in the compressed domain," in *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2020, pp. 1419–1424.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:14124313

[23] J. Grabinski, J. Keuper, and M. Keuper, "Aliasing and adversarial robust generalization of cnns," *Machine Learning*, vol. 111, pp. 1–27, 08 2022.

[24] X. Xu, S. Liu, and Z. Li, "Tencent video dataset (tvd): A video dataset for learning-based visual data compression and analysis," 2021. [Online]. Available: https://arxiv.org/abs/2105.05961

[25] T. Tanaka, H. Choi, and I. V. Bajić, "Sfu-hw-tracks-v1: Object tracking dataset on raw video sequences," 2021. [Online]. Available: https://arxiv.org/abs/2112.14934

[26] TensorFlow, "Post-Training Quantization." [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization