Similarity-aware CNN for Efficient Video Recognition at the Edge

Amin Sabet*, Jonathon Hare*, Bashir Al-Hashimi† and Geoff V. Merrett*
* School of Electronics and Computer Science, University of Southampton, UK
† Faculty of Natural and Mathematical Sciences, King's College London, UK
Email: ms4r18@ecs.soton.ac.uk, jsh2@ecs.soton.ac.uk, bashir.al-hashimi@kcl.ac.uk, gvm@ecs.soton.ac.uk

***Abstract*—Convolutional neural networks (CNNs) often extract similar features from successive video frames due to having identical appearances. In contrast, conventional CNNs for video recognition process individual frames with a fixed computational effort. Each video frame is independently processed, resulting in numerous redundant computations and an inefficient use of limited energy resources, particularly for edge computing applications. To alleviate the high energy requirements associated with video frame processing, this paper presented similarity-aware CNNs that recognise similar feature pixels across frames and avoid computations on them. First, with a loss of less than 1% in recognition accuracy, a proposed similarity aware quantization technique increases the average number of unchanged feature pixels across frame pairs by up to 85%. Then, a proposed similarity-aware dataflow improves energy consumption by minimising redundant computations and memory accesses across frame pairs. According to simulation experiments, the proposed dataflow decreases the energy consumed by video frame processing by up to 30%.**

***Index Terms*—Deep neural networks, Quantization, Object Detection, Video Recognition**

I. Introduction

In recent years, deep convolutional neural networks (CNNs) have provided superior performance in computer vision [1], [2] and natural language processing [3]. The superior accuracy of CNNs, however, comes at a high computational complexity and energy cost. Although CNN training usually runs exclusively on powerful server farms, concerns about privacy, reliability, and connectivity have introduced an increasing push to transition inference to the edge [4]. While advantageous, inference at the edge is challenging because devices are usually constrained in terms of computational and energy resources. Therefore, to enable inference at the edge, various energy-efficient and high-performance CNN accelerators have been developed over the years, which are often optimised for image data [5]. However, a particular challenge is that input data streams at the edge are often continuous (e.g. a continuous feed of video from a smart security camera) rather than non-continuous (e.g. an intelligent disease detection system using MRI images) and there is a genuine need for optimising CNNs and CNN accelerators for video data.

Between consecutive frames, the data rate is typically much higher than the real information rate. For example, as shown in figure 1, a home security doorbell may generate many megabytes of video each second [6]; however, the actual information is substantially lower because objects of interest are not at the door for the majority of the time. Similarity between successive video frames occurs even more frequently in applications such as facility monitoring, CCTV surveillance, and portable wildlife surveillance systems, where the camera is stationary and there are fewer moving objects in the videos. However, due to sources of environmental 'noise' e.g., changing lighting conditions, most pixel values change between consecutive frames, despite the high degree of similarity that may exist between the content of video frames. Therefore, the computational results from processing a video frame cannot be stored and reused for successive frames, even when the content of frames is the same. Accordingly, inference must be performed on each frame, individually and independently, leading to an inferior use of limited energy resources. Figure 2 indicates that, during the processing of two consecutive video frames by a state-of-the-art video recognition CNN [7], less than 11% of input data to the CNN layers remains unchanged.

To overcome the energy bottleneck associated with performing the same computations for each video frame individually, this paper introduces similarity-aware CNNs, in which the required computation effort for each frame is proportional to the change in the feature pixels from the previous frame. Similarity-aware CNN is comprised of two main components. The first component is a proposed quantization approach (SQS) for reducing differences between feature pixel values across consecutive frames by abandoning floating-point representation and adopting a lighter fixed-point. The second component is a similarity-aware dataflow (SRS) that subtracts features from subsequent frames to identify pixels that remain unchanged. Computational results from the previous frame are reused for such pixels, and multiply and accumulation (MAC) operations, as well as memory accesses associated with such pixels, are avoided to reduce energy consumption of the processing frame while maintaining inference time.

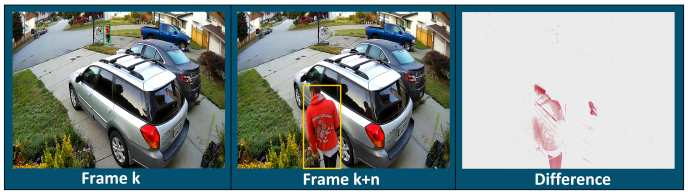


Fig. 1: Content similarity map between two video frames

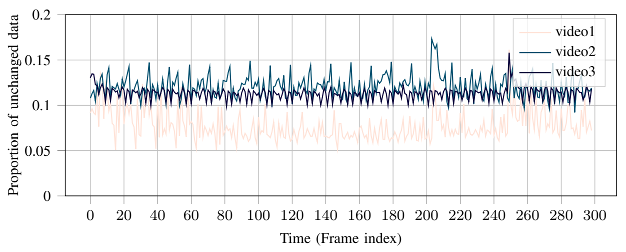


Fig. 2: The averaged of unchanged pixels into all convolutional layers between consecutive frames when processing Video1, video2 and video3. We employed Yolo-V3 [7] as the object detection model and videos are sampled from VIRAT Ground dataset [8]. [1]

SQS preserves up to 85% of feature pixels over succeeding frames, and by avoiding computations for unchanged pixels, SRS reduces video recognition's energy consumption by up to 30%.

The contributions of this paper are as follows:

- A Similarity-aware Quantization Scheme (SQS) enables the maximum possible reuse of previous video frame processing computational results. SQS minimises the difference in feature pixel values between consecutive frames by optimising the granularity with which CNNs are quantized for a given hardware architecture and accuracy tolerance. The evaluation of SQS for video object detection networks demonstrates that the proposed approach increases unchanged feature pixels in CNNs by up to 85% on average while sacrificing less than 1% of recognition accuracy.
- An energy-efficient Similarity-aware Row Stationary dataflow (SRS) optimizes data movements and operations for processing successive data. To improve energy efficiency, SRS subtracts features from consecutive frames and identifies pixels with unchanged feature values, skipping redundant computations/memory accesses for such pixels. SRS is evaluated using the Timeloop simulator [9], and the results indicate that SRS reduces the energy consumption of processing consecutive frames by 30% for various CNNs, including Resnet [10] and Darknet [7].

The remainder of this paper is organised as follows. Previous work is reviewed in section II . Section III introduces the proposed Similarity-Aware convolutional neural networks. Evaluation results are discussed in section IV, and the paper is concluded in section V

II. Related Work

Our work improves the prior art aspects, as discussed in the next sub-sections. SQS minimizes the required computations for video recognition by maximizing unchanged successive data into CNN layers. SRS reduces the energy consumption of video recognition by skipping the redundant computations and memory accesses.

A. Computation Reduction

Various approaches are intended to exploit temporal information between the video frames to reduce the computation complexity of CNNs [11] . Riera et al. [12] proposed applying a uniform distributed linear quantizer on the feature maps to increase unchanged pixels between consecutive frames. Since the feature maps in CNNs are not uniformly distributed, the proposed uniform quantization introduces significant and unacceptable quantization error. Consequently, the proposed approach is only applicable for small and shallow neural networks.

Sigma-Delta [13] computes the difference between the consecutive input data into CNN layers. Then, the different values are rounded to integers and processed by the CNN. Output values from network layers are added to their values from previous executions of the network. SigmaDelta was designed and evaluated only for classification tasks that are less sensitive to temporal information between frames. Since per-frame video classification is less sensitive to spatial changes between consecutive frames, whether SigmaDelat can achieve the same accuracy for real-world video recognition applications, e.g., object detection, is questionable.

In a different line of work to ours, CBinfer [14] uses a software-level solution to increase unchanged pixels between consecutive frames by comparing and filtering the difference between pixels with a threshold value. To achieve considerable computation reduction, threshold values need to be chosen as large numbers, which subsequently leads to a significant accuracy loss in recognition. DeepCache [15] pursues a similar direction to CBinfer. However, to detect blocks of the image that are matched to blocks in the previous frame, this method requires a high tolerance when comparing image tiles to be able to find matches, thereby introducing considerable accuracy losses. Unlike mentioned approaches, our proposed SQS method re-adjusts model parameters with the effects of similarity-aware quantization errors and recovers accuracy degradation. Therefore, SQS outperforms those approaches and achieves higher accuracy for the stat-of-the-art deep CNN for video recognition applications.

B. CNN Dataflows for Video Recognition

A CNN's dataflow defines how data is stored in different levels of the memory hierarchy of a CNN accelerator. Dataflow also schedules, partitions and parallelises operations across computation units. While the topic of CNN dataflows is widely studied for image data [16]–[19], less attention has been paid to specialised dataflows for consecutive executions of CNNs. In a recent work, Zhe et al. [20] proposes a hybrid precision accelerator for video processing that employs inter-frame data re-use. In the proposed approach, only difference frame data is processed for sequential frames. To process difference frames, they are divided into a low precision 4b tensor and a sparse 8b tensor. These tensors are processed

[1] Video1, video2 and video3 are named VIRAT.S.010003.09.000779.000861, VIRAT.S.010207.07.001195.001260, and VIRAT.S.010107.02.000282 in VIRAT dataset, respectively.

separately and added together for final results. The accelerator improves power/speedup by providing extra on-chip SRAM memory to store the latest intermediate feature maps and avoid off-chip accesses. Therefore, the proposed approach is applicable to a limited class of CNNs (e.g., MobileNet [21]) with small feature maps that can be fitted into on-chip SRAM memory.

Riera et al. [12] proposed an accelerator architecture with a large 35MB on-chip memory to store all weight parameters in the on-chip memory to avoid off-chip accesses. A 35MB on-chip memory, on the other hand, is not feasible or cost-effective for use in real-world CNN accelerators.Additionally, the proposed architecture was examined for small CNNs with a maximum of eight convolutional layers. Therefore, for a state-of-the-art video recognition network, e.g., Darknet53, the proposed architecture requires hundreds of megabytes of on-chip memory. However, we propose SRS as a new dataflow for repeated execution of a CNN. By skipping redundant computations and memory accesses, the proposed dataflow reduces the energy consumption of conventional dataflows. Moreover, unlike the mentioned dataflows, our proposed approach does not require additional large on-chip memory to store the latest feature maps and weight parameters.

III. Proposed Method: Similarity-Aware Convolutional Neural Networks

We introduce similarity-aware convolutional neural networks in this part as a novel approach for reducing the energy consumption of video recognition applications by leveraging similarity between features in consecutive frames. The proposed approach involves increasing the number of unchanged feature pixels across successive frames using a novel quantization technique (SQS), followed by avoiding computation on unchanged pixels via a newly introduced dataflow (SRS). We discuss the fundamental concepts of Similarity-Aware Quantization Scheme (SQS) and Similarity-Aware Dataflow (SRS), as well as their major components and thorough implementation.

A. Similarity-aware CNNs

Deep convolutional neural networks are multi-layered networks, consisting of convolution, pooling, fully connected and batch normalization layers. Convolutional layers are extremely computation and energy-intensive, accounting for more than 90% of overall computations in CNNs and generating a large amount of data movement [22]. As illustrated in figure 3, we represent the convolutional layer l as $\left(\mathbf{X}_l, \mathbf{Y}_l, \mathbf{W}_l, \mathbf{b}_l\right)$, where $\mathbf{X}_l \in \mathbb{R}^{C\times(Q+S-1)\times(P+R-1)}$ is the input feature map (*ifmap*), $Y_l \in \mathbb{R}^{N\times K\times P\times Q}$ is output feature map (*ofmap*), $\mathbf{W}_l \in \mathbb{R}^{K\times C\times R\times S}$ filter weight and $\mathbf{b}_l \in \mathbb{R}^{C}$ represents bias parameters, respectively. The convolutional layer computes:

$$\mathbf{Y}_l = \mathbf{X}_l * \mathbf{W}_l + \mathbf{b}_l, \tag{1}$$

Where * is the convolution operation.

```
for (n=0;n<N;n++):
  for(k=0;k<K;k++):
    for(p=0;p<P;p++):
      for(q=0;q<Q;q++):
        for(c=0;c<C;c++):
          for(r=0;r<R;r++):
            for(s=0;s<S;s++):
              ofmap[n][k][p][q] += weight[k][c][p][q]*ifmap[n][c][p+r][q+s]
```

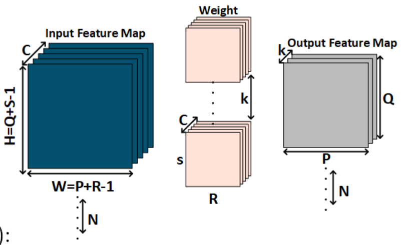


Fig. 3: The representation and computational loop of a Convolutional layer

However, instead of processing *ifmaps* of individual frames independently, a convolutional layer in the similarity-aware CNNs processes the difference in textitifmaps between consecutive frames. Then, the convolutional results are accumulated with the *ofmaps* from previous frame. Therefore, for similarity-aware-CNN, equation (1) can be re-written as,

$$\mathbf{Y}_l^t = ((\mathbf{X}^{(t)} - \mathbf{X}^{(t-1)}) * \mathbf{W}_l + \mathbf{b}_l) + \mathbf{Y}_l^{(t-1)}, \tag{2}$$

where, t represents the order of consecutive *ifmaps*/*ofmaps*. The key idea of similarity-aware CNNs is to update ofmap pixels incrementally based on the difference of *ifmap* pixel values relative to the previous frames. However, regarding similarity-aware CNNs, the main question is how can the incremental update of *ofmap* be exploited to reduce the energy consumption of video recognition? Section III-B addresses this question by proposing SQS, which minimizes incremental updates through consecutive *ifmap*. Then, section III-C demonstrates how SRS reduces the energy consumption of conventional dataflows by leveraging small incremental updates.

B. Similarity-Aware Quantization Scheme

The use of network quantization for model compression has been widely reported [23] [24]. CNN quantization attempts to approximate floating-point data with fixed-point precision in order to minimise model size without sacrificing detection accuracy. Quantization, on the other hand, quantifies the degree of change in the values of feature pixels across consecutive frames when a continuous data stream is present [12]. This section discusses SQS, an approach for minimising changes in feature pixels between consecutive frames.

Let $\mathbf{f}$ be the floating-point data to be quantized (feature maps or parameters), We define a quantizer as,

$$\mathbf{q} = clip(\left[round(\frac{\mathbf{f}}{s}) + z\right], q_n, q_p) \tag{3}$$

Where $\mathbf{q}$ is the quantized data type. z denotes zero-point, an integer to ensure that zero is quantized with no error. This is important to ensure that common operations like zero padding do not cause quantization error. s is the floating-point valued step-size that defines the difference between two representable numbers in quantized data type. The $round(m)$ rounds m to nearest integer. Given a hardware architecture with bw, $q_n = -2^{bw-1}$ and $q_p = 2^{bw-1} - 1$. Function clip(.) is defined as,

$$clip\left(k, r_{\text{min}}, r_{\text{max}}\right) = \begin{cases} r_{\text{min}} & k \leq r_{\text{min}} \\ k & r_{\text{min}} < k < r_{\text{max}} \\ r_{\text{max}} & k \geq r_{\text{max}}. \end{cases} \tag{4}$$

Following equation (3), the quantization can be performed either symmetric or asymmetric. Symmetric quantization partitions the range of floating-point values using a symmetric range. However, symmetric quantization, simplifies the quantization function in equation (3) by replacing the zero point with $z = 0$. For symmetric quantization, the step-size s is calculated as follows.

$$s = 2 \ \text{max}\left(\left|\mathbf{f}_{\text{min}}\right|, \mathbf{f}_{\text{max}}\right) / \left(q_p + q_n\right), \tag{5}$$

where $\mathbf{f}_{\text{min}}$ and $\mathbf{f}_{\text{max}}$ defines the range of floating-point data. The symmetric quantization has the advantage of easier implementation, as it leads to $z = 0$. However, it is sub-optimal for cases where the range of floating-point data could be significantly skewed and not symmetric. For such cases, asymmetric quantization is preferred. For the asymmetric quantization scaling factor and zero point are calculated as follows.

$$s = \left(\mathbf{f}_{\text{max}} - \mathbf{f}_{\text{ min}}\right) / \left(q_{\text{p}} - q_{\text{n}}\right), \tag{6}$$

$$z = q_{min} - round\left(\frac{\mathbf{f}_{min}}{s}\right), \tag{7}$$

where q_{min} is the minimum of quantized data type.

Figure 4 visualises an overview of symmetric and asymmetric quantization. While asymmetric quantization of network parameters (weights and biases) achieves slightly higher accuracy, it requires additional subtraction or linear linearoperation (due to the zero-point value) prior to multiplication, which consumes an additional 10%–20% of energy [25]. As a result, symmetrical quantization of parameters is compatible with current fixed-point CNN accelerators and has been widely implemented in chip design [26]. As a result, we employ symetric quantization to achieve parameter quantization. However, we examine asymmetric and symmetric modes for *ifmap* quantization.

To obtain the scaling factor, s in equation (3), and quantize data the key challenge lies in determining the optimal range of floating-point data ($\mathbf{f}_{min}$ $\mathbf{f}_{max}$) in order to minimizing quantization error. The next sections explain how SQS obtains the scaling factor for quantizing parameters and *ifmap* by searching for the optimal range of floating-point data.

1) Parameter Quantization: To symmetrically quantize weights, $\mathbf{W}$, in layer l, the step-size, s_w^l, is calculated using equation (5). The optimal range of floating-point weights ($\mathbf{W}_{\text{min}}$ and $\mathbf{W}_{max}$) are calculated as the average minimum and maximum of kernel (filter) tensors, K_l, as follows,

$$\left(\mathbf{W}_{\text{min}}^l, \mathbf{W}_{\text{max}}^l\right) = \frac{1}{N_k^l} \sum_{i \in N_k^l} \left(\min\left(K_i^l\right), \max\left(K_i^l\right)\right). \tag{8}$$

Where N_k^l and K_i^l denote the number of kernels and the i^{th} kernel in layer l, respectively. The same procedure is also employed to determine the optimum step-size for biases. While determining the step-size for a specific layer, other layers keep the floating-point precision to prevent the influence of other quantization noise on optimization.

2) Activation Quantization: Whereas parameter quantization primarily affects model accuracy and size, *ifmap* quantization has an effect on model accuracy and the degree of change between *ifmap* pixels across successive frames in a continuous data stream. More specifically, a bigger step-size minimises the difference between pixel values and results in fewer changes between successive *ifmap* pixels. Nevertheless, the reduced data resolution caused by a very large step-size results in a considerable rise in model accuracy loss. The step-size ,s_x^l , for symmetric and asymmetric quantization of *ifmap* $\mathbf{X}$ in layer l are calculated using equations (5) and (6), respectively.

Unlike parameter quantization where the range of the floating-point weights can be computed ahead of time, the range of *ifmap* values ($\mathbf{X}_{min}^l$ and $\mathbf{X}_{max}^l$) is determined by the input data to network. Hence, we propose the range of *ifmap* into each layer to be derived from the floating-point precision histogram of *ifmap* pixel values. To this end, we introduce the cost function f_l in equation (9) as the weighted sum of quantization error (mean square error - (MSE)) and similarity of *ifmap* values in layer l, denoted by SIM. The aim of the proposed cost function is to identify the optimal range of *ifmaps* to co-optimize the quantization error and degree of changes between *ifmap* pixels across frames. During the optimization introduced in algorithm 1, we look for the optimum X_{min} and X_{max} to minimize f_l,

$$f_l = \underset{X_{\text{min}}, X_{\text{max}}}{\operatorname{argmin}} \left(\text{MSE} + \gamma.\beta / \text{SIM}\right), \tag{9}$$

In the following, we will describe each term in more details.

3) Quantization Error (MSE): Given the floating-point precision histogram of *ifmap* $\mathbf{X}$, quantization error is measured as the ℓ_2 norm of the difference between the floating-point and quantized value of *ifmap* [27]. Figure 5 visualizes a histogram of *ifmap* and the quantization intervals for a simple INT3 quantizer. Let z_i and $h(z_i)$ be i-th bin value and its corresponding frequency, $\forall i \in \{1, \ldots, n\}$, where n denotes the number of bins , then the MSE is defined as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} h\left(z_i\right) \times \left(z_i - z_i^q\right)^2 \tag{10}$$

where z_i^q is the quantized value of z_i, driven from equation (3). We set n to 2048 in our experiments.

4) Similarity (SIM): In equation (8), SIM defines similarity as the probability that the difference between two consecutive *ifmap* pixels is less than the quantization step-size of that *ifmap*. In other words, similarity can be interpreted as the probability that a floating point pixel value in two consecutive executions of the CNN falls in the same quantization interval i.e., a quantization interval is equal to step-size value which

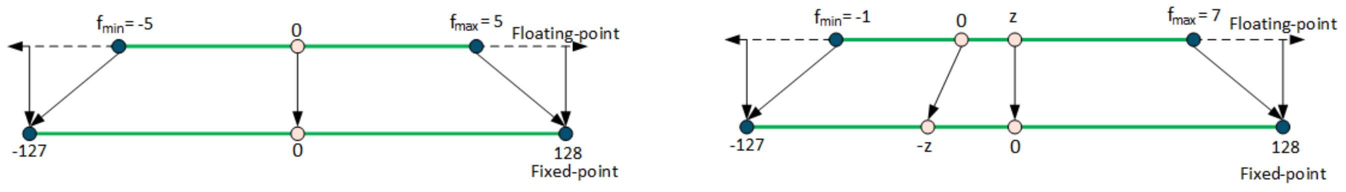


Fig. 4: Illustration of symmetric quantization (left) and asymmetric quantization (right).

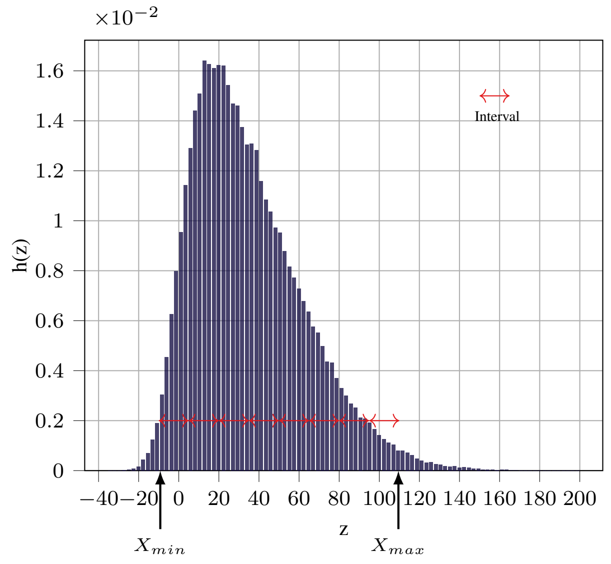


Fig. 5: Histogram of a floating point precision ifmap. Quantization intervals ($\leftrightarrow$) for a INT3 quantizer are superimposed.

is shown as $\leftrightarrow$ in figure 5. Given k quantization intervals, similarity is computed as follows,

$$\mathrm{SIM}(\mathbf{X}) = \sum_{i=1}^{k} P\left(\mathbf{X}_t \in \mathrm{Inv}_i\right) * P\left(\mathbf{X}_{t-1} \in \mathrm{Inv}_i\right), \quad (11)$$

where Inv_i is the i-th quantization interval and $k = |q_p| + |q_n|$. Given the floating-point histogram of $\mathbf{X}$, $P\left(\mathbf{X}_{t/t-1} \in \mathrm{Inv}_i\right)$ can be estimated as the proportion of data in histogram of X between Inv_i and Inv_{i+1}. Therefore, given X_{min}, X_{max}, k, and histogram of $\mathbf{X}$, with z_i and $h(z_i)$, bin values and frequencies, similarity is computed as follows.

$$\mathrm{SIM}(\mathbf{X}) = \frac{1}{H} \sum_{i=0}^{k-1} \left(\sum_{\forall z_i \in [X_{min}+i*s, X_{min}+(i+1)*s]} h\left(z_i\right) \right)^2, \quad (12)$$

Where H is total number of data in histogram of X and is computed as follows.

$$H = \sum_{i=1}^{n} h\left(z_i\right). \quad (13)$$

5) Hyper-parameters (β and γ): β in (9) is a hyper-parameter which normalize SIM and MSE values. γ is a hyper-parameter that controls the strength of our preference for similarity. $\gamma = 0$ imposes no preference on the degree of similarity, and larger γ values force the step-size to become larger. The γ parameter can vary for different network layers and be chosen as a function of the layer's computational complexity and depth. Hence, γ gives us the flexibility to change the degree of computational complexity and power consumption reduction of video recognition.

6) Optimization: Contrary to parameter quantization, where the weights and biases are quantized ahead of time, the step-size for *ifmaps* are computed based on observing the behaviour of the CNN during a calibration process. In the calibration, random images are fed into the network, and the histograms of floating point *ifmaps* into convolutional layers are recorded. Specifically, this is done by inserting observer modules before each convolutional layer to record that data. Then, the optimization algorithm 1 performs a deep search in the histogram bins to choose the clip threshold bins, X_{min} and X_{max}, to minimize quantization error and maximize the similarity (or minimize changes in data) in *ifmap*.

In algorithm 1, $cSum$ is the cumulative histogram of X. Lb and Ub are the lower and upper bounds parameters with an initial value of 0 and 1, respectively. *sBin*, *eBin* and *nBin* indicate the start bin, end bin and number of bins, respectively. In our experiments, *nBin* is set to 2048. *minCost* stores the minimum cost function value so far, during the optimization. minCost has a large value of (infinite) at the beginning. *total* is defined as the total number of data in histogram of X and is computed in equation (13). std is the search stride parameter that defines the granularity of search in histogram and is set to 1×10^{-5} in our experiments. In algorithm 1 the clipping bins (*sBin* and *eBin*) are initially set to the first (zero) and last bins (*nBin*) in the histogram, respectively. Such a clipping setting provides maximum similarity (largest quantization step-size) with large quantization error at the beginning of search. Then, the right and left clipping bins (*eBin* and *sBin*) are gradually moved toward the centre of the histogram to optimize the quantizaton error. This process continues till it reaches the optimum clipping bins which minimize the cost function f_l. Finally, the optimum clipping bins are returned as the X_{min} and X_{max} parameters for activation quantization. The algorithm refalg:1 is an offline iterative process which only takes a few seconds on a regular CPU.

7) Model Fine-tuning: To recover the quantization error, once the optimum step-sizes are determined, they are frozen to retrain the network. We employed the training approach proposed by Jacob et.al. [28] to fine-tune the network. This approach simulates the quantization effects during the forward pass, while the back propagation is still performed using full

Algorithm 1: Identifying optimum clipping bounds

```
Data: Histogram(X)
Result: $X_{min}$, $X_{max}$
while $Lb < Ub$ do
    nLb ← Lb + std # next Lower bound
    nUb ← Ub - std # next Upper bound
    l ← startBin
    r ← endBin
    while $l < eBin$ and $cSum[l] < nLb*total$ do
        l = l+1
    end
    while $r > sBin$ and $cSum[r] < nUb*total$ do
        r ← r-1
    end
    nSBin ← sBin # next start bin
    nEBin ← eBin # next end bin
    if (l-sBin) > (eBin-r) then
        nSBin ← l
        Ub ← nUb
    else
        nEBin ← r
        Lb ← nLb
    end
    $f_l$ ← MSE(Histogram, nSBin, nEBin) + $\gamma \cdot \beta$/SIM(Histogram, nSBin, nEBin)
    if $f_l < minCost$ then
        minCost ← $f_l$ sBin ← nSBin eBin ← nEbin
    end
end
$X_{min}$ ← sBin , $X_{max}$ ← eBin
```

precision.

C. Similarity-Aware Dataflow - SRS

By quantizing CNNs through SQS, the majority of *ifmap* pixel values remain unchanged between consecutive video frames. Therefore, the computational results (*ofmap*) from processing a video frame can be stored and reused for subsequent frames. Accordingly, computations and memory accesses associated with the unchanged pixels can be eliminated. However, to efficiently reuse computation results and eliminate unnecessary computations, a specialized dataflow is required for CNN accelerators. Existing dataflows aim to maximize data reuse and minimize accesses to high energy consumption memory hierarchy levels like DRAM. However, they are not designed to skip redundant computation and memory accesses when input data remains unchanged between consecutive frames. Therefore, it is desirable for the dataflow to be aware of the similarity between *ifmaps*, and hence optimize computation and data movement energy costs. We achieve this goal by proposing a novel dataflow, called similarity-aware dataflow (SRS). SRS adapts row-stationary (RS) dataflow for similarity-aware CNNs, and leverages unchanged pixels between successive *ifmap* (or incremental update of ofmap) to reduce the power consumption of video recognition tasks. This section introduces row-stationary dataflow and shows how SRS adapts it for similarity-aware CNNs.

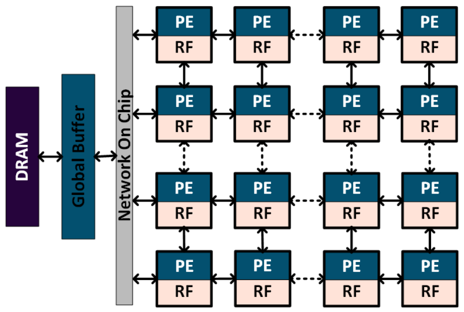


Fig. 6: An abstract on CNN accelerator consists of an array of processing elements, each equipped with a register buffer, a global on-chip buffer and an off-chip DRAM memory

1) Row-Stationary Dataflow : Figure 6 shows an abstract view of a CNN accelerator architecture, pervasive in many state-of-the-art CNN accelerators [29]–[31]. CNN accelerators are primarily composed of off-chip DRAM, a global buffer (GB) and an array of processing element (PE) components. Each PE is equipped with a local register file (RF).

Row-stationary dataflow divides the 7D convolution operation, shown in figure 3, into 1D simple convolution primitives. Each primitive includes one row of filter weights and one row of *ifmap* pixels, which are used to generate one row of partial sum (*psum*). The *psum*s from different primitives accumulate together to generate one row the *ofmap* pixels [32]. The computations for each primitive are mapped to one PE. Row-stationary dataflow reads a row of *ifmap* pixel from DRAM and passes it to the GB and then from GB to RFs inside PEs. A row of weights is read from the DRAM and it is directly passed to the RFs. The computation process inside each PE starts by sending each *ifmap* pixel from the RF to the MAC engine. Each *ifmap* pixel is multiplied by a row of filter weights and generates one *psum* pixel. Once a row of *psum* pixels is computed, it is forwarded to the neighbour PEs to compute one row of ofmap. The computed *ofmap* is sent back to GB and ultimately will be stored in DRAM.

2) Energy Consumption Analysis of row-stationary Dataflows : The energy consumption of a convolutional layer can be estimated (with 95% accuracy [9]) by accounting for the total number of required MAC operations and the number of accesses to each level of memory hierarchy for all data types (*ifmap*, weights, *psums*/*ofmap*), and weighing them with the energy cost of MAC operation and the energy cost of accessing that specific memory level [9], [32]. Figure 7 shows the energy breakdown of a row-stationary dataflow for five convolutional layers. The energy consumption of convolutional layers in row-stationary is dominated by RF accesses, because the dataflow fully exploits different types of data movement in the local RFs and between PEs to minimize accesses to DRAM and GB (which have a higher access energy cost).

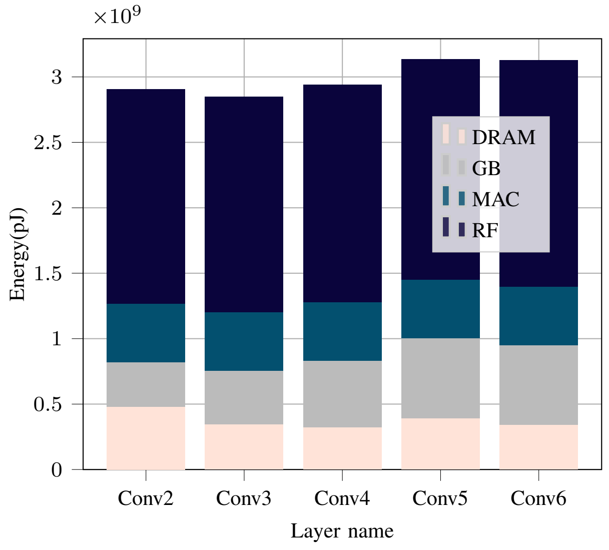


Fig. 7: Energy breakdown of RS measured by Timeloop [9] for five different convolution layers in table III onto Eyeriss architecture [32].

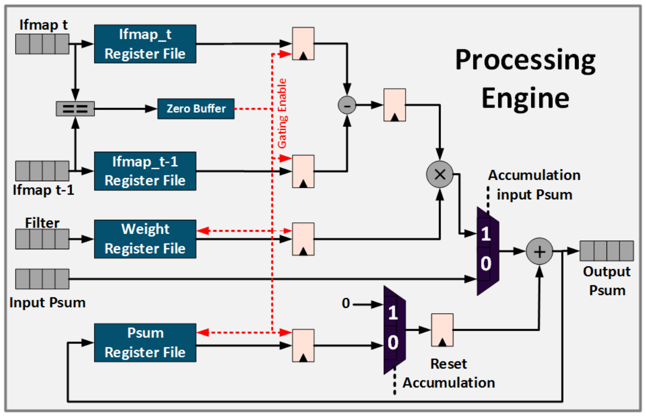


Fig. 8: Computational flow of a PE in SRS

3) Similarity-Aware Dataflow (SRS): Similar to row-stationary dataflow, SRS operates on 1D primitives. A 1D primitive in SRS consists of one row of *ifmap* from the current frame (*ifmap*$_t$), one row of *ifmap* from the previous frame (*ifmap*$_{t-1}$), and one row of filter weight. After the computed rows of *psum* from different primitives are accumulated together, the results are added to the row of *ofmap* from the previous frame.

SRS maps each primitive into one PE to provide maximum *ifmap* and filter reuse at the RF level. Figure 8 shows the abstract of computation flow of a PE in SRS to process a 1D primitives. Initially, SRS detects the unchanged ifmap pixels (between *ifmap*$_t$ and *ifmap*$_{t-1}$) coming to the PE. As shown with dotted red lines in figure 8, for the unchanged pixels, SRS gates accesses to weight and *Psum* RF. Accordingly, MAC operations are eliminated and *psum* pixel remains unchanged. Since most of the *ifmap* pixels remain unchanged between consecutive video frames, SRS skips numerous redundant MAC operations and RF accesses. On the other hand, when a pixel changes between *ifmap*$_t$ and *ifmap*$_{t-1}$, *ifmap* pixels are subtracted from each other and the MAC operations are performed between subtraction result and a row of filter weights. The computation result is accumulated to the input *psum* pixel values. The SRS reduces the number of MAC operations and RF accesses for weight and *ofmap*. However, SRS increases RF accesses for *ifmap*. The next section shows that SRS reduces the overall energy consummation of convolutional layers by eliminating MAC operations and RF accesses for *ofmap* and filter weights. It is notable that SRS uses the similar data movement scheme between PEs as row-stationary dataflow.

4) Energy Consumption Analysis of SRS : The energy consumption of SRS evaluated by accounting for the number of required MAC operations and the number of accesses to each level of the memory hierarchy. This section shows that in the presence of similarity between *ifmap* pixels, SRS improves the energy consumption of original row-stationary dataflow by reducing the number of accesses to RFs and eliminating numerous MAC operations.

RF Accesses: As shown in figure 7, the energy consumption of convolutional layers in row-stationary dataflow is dominated by the RF accesses. This is because of numerous accesses to RF to read/write *ifmap*, filter weight and *psum* pixels. However, SRS skips the majority of RF accesses for the unchanged *ifmap* pixels. By quantizing CNNs through SQS, most of the *ifmap* pixels remain unchanged between consecutive frames; therefore, SRS eliminates accesses to RF for fetching filter weights and *psum*, resulting in an energy consumption reduction of convolutional layers. Nonetheless, compared to a row-stationary dataflow, SRS requires more accesses to the *ifmap* RF to read *ifmap* pixels(for processing each *ifmap* pixel two accesses are required to read *ifmap*$_{t-1}$ and *ifmap*$_t$). However, since each *ifmap* pixel shared between a row of filter weights, the increased number of accesses to *ifmap* RF to read *ifmap* pixels is significantly less than the eliminated RF accesses by SRS.

GB and DRAM accesses The general flow of data in SRS is similar to the original row-stationary dataflow. However, SRS stores and reuses *ifmap/ofmap* from previous frames to reduce computational complexity. Therefore, SRS increases access to DRAM and GB for both *ifmap* and *ofmap* tensors. However, SRS doesn't increase the number of DRAM and GB accesses for filter weights.

Computational Operations SRS eliminates the MAC operation for unchanged *ifmap* pixels. However, Compare to row-stationary dataflow SRS requires an additional subtraction operation is required for the changed *ifmap* pixels. Since a MAC operation consumes 10$\times$ more energy than a subtraction [33], the energy overhead of extra subtraction operations is small compared to the saved energy by eliminated MAC operations.

IV. Results and Discussion

A. Evaluation

We evaluate SQS on YOLOv3 network [7], as the widely used per-frame video object detection model for mobile devices, trained on the MS COCO dataset [34]. We evaluate the accuracy of SQS on the test set of MS COCO. We use a small subset (5k random images) of COCO test set for SQS calibration. We implemented SQS using the PyTorch framework. To evaluate the energy consumption of SRS, we employed and adapted the Timeloop engine [9], a powerful and accurate infrastructure for modelling and evaluating the design space of DNN accelerator architectures. We evaluate and compare the power consumption of SRS with the row-stationary dataflow on the Eyeriss architecture [32], which uses 168 PEs (each with a 224b ×16-b SRAM filter scratch pad, 16b×12 and 16b×24 register files for *ifmap* and *ofmap* scratch pads, respectively), a single shared 108KB global buffer, and a backing DRAM. We provide videos for evaluating the efficiency of SRS from VIRAT Ground dataset [8]. VIRAT Ground consists of videos from stationary cameras in urban streets. Videos are sampled sampled at 15 fps.

Metrics: We use accuracy, computational complexity, and energy consumption to evaluate the performance of our framework. To report the accuracy results, we use different metrics. We report the top-k accuracy for classifier models and mAP (Mean Average Precision) as the accuracy for object detection.

B. Accuracy and Computation Complexity

We evaluate SQS's efficiency in reducing the computational complexity of processing consecutive video frames, as well as the impact of quantization on the accuracy of object detection results. We investigated symmetric and asymmetric quantizations of *ifmaps*.

Accuracy: Figure 9 compares the mAP of SQS implementation of Yolo-V3 [7] against 32-bit floating-point precision (FP32), conventional quantization approach [28] (INT8), CBinfer [14] , and DeepCach [15] implementations. The SQS is repeated for $\gamma = 0.1$, $\gamma = 0.3$, and $\gamma = 0.5$, using both symmetric, SQS(sym), and asymmetric, SQS(asym), quantization. While CBinfer and DeepCach use the mean square error (MSE) of objectness score to evaluate the accuracy of object detection, we use the mAP as the standard detection metric because the objectness score doesn't reflect the accuracy of predicted bounding boxes. Figure 9 shows that SQS reduces the mAP only by 0.52% and 0.3% compared to FP32 and INT8, respectively. However, the SQS outperforms CBinfer and DeepCach because they impose errors introduced by thresholding differences between frames. However, unlike these approaches, our proposed approach fine-tunes network parameters to recover. We evaluate the impact of γ value in accuracy of similarity-aware CNN. Increasing γ value leads to mAP reduction because a large γ value forces the quantizer to select a larger step-size that consequently leads to more quantization error. However, larger γ reduces the computational complexity of video recognition, as we will

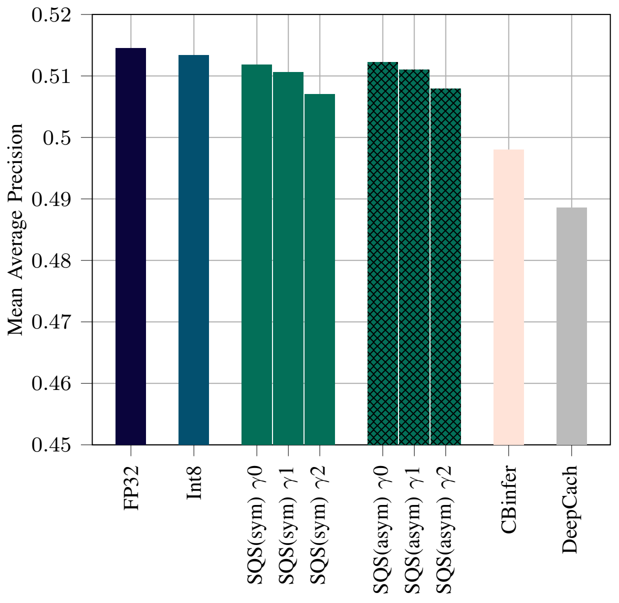


Fig. 9: Comparing mean average precision of video object detection of floating-point precision (FP32), INT8, CBinfer, DeepCach with SQS implementation of YOLOv3. Both asymmetric and symmetric SQS are repeated for $\gamma_0 = 0.1$, $\gamma_0 = 0.3$ $\gamma_0 = 0.5$ [7].

discuss in the next section. According to the observations, asymmetric quantization has a marginally greater accuracy than symmetric quantization. The rule of Batch normalisation, on the other hand, causes *ifmaps* to be symmetrical in most cases. Aside from that, network parameters are trained to be zero centred in order to avoid gradients that explode or vanish [35]. As a result, the detection accuracy of symmetric and asymmetric quantizations is nearly identical in both cases, as both *ifmaps* and parameters are symmetrically distributed. Due to the fact that symmetric quantization has nearly the accuracy of asymmetric quantization and a low computational cost during inference, we use it in the building of SRS.

Table I shows the Top1 and Top-5 errors of SQS, INT8 [28] and SigmaDelta [13] compared to the FP32 accuracy of the VGG-16 [36] classifier network. The symmetric and asymmetric quantizations are examined for SQS. The SQS achieves classification accuracy close to INT8. However, the proposed SQS outperforms the SigmaDelta in both Top-1 and Top-5 accuracy. Moreover, asymmetric quantization of SQS has slightly lower quantization error than symmetric mode. Notably, we did not involve the proposed approach by Riera et al. [12] in our evaluations because it introduces significant accuracy loss for state-of-the-art CNN. The proposed approach can examined only for the shallow CNN.

Figure 10 visualizes further evaluation results. Figure 10 compares the detection result and unchanged *ifmap* pixels while processing the security camera videos from VIRAT. Instead of processing consecutive frames, we process video frames with five and three time-step distances. The top row in figure 10 shows the detection results of processing video frames with the FP32 network. The horizontal arrows between the two frames indicate the proportion of data that remains unchanged between frames. The vertical arrows show the average of intersection over union (IoU) of bounding boxes predicted by FP32 and similarity-aware CNN. Using the FP32 network, only 11%-13% of *ifmaps* remains unchanged between frames with five and three time-step distances. The bottom row shows the detection results of processing the same video frames as in the top row, processed by the SQS quantized network. While getting the high detection accuracy as FP32 CNN, SQS increases unchanged *ifmaps* between frames up to 67%- 76%.

Computational Complexity Reduction: Previous works such as CBinfer [14], DeepCach [15], and SigmaDelta [13] aim to identify unchanged pixels and skip computations at the software level. Therefore, they use general-purpose GPU/CPU platforms to execute their approach. For example, DeepCach and CBinfer use Nexus 6 and Jetson TX1, respectively, that consume tens of Joules of energy. Yet, we propose SRS as a novel hardware-level solution to detect unchanged pixels and skip redundant computations. We use the Eyeriss [32] accelerator platform that consumes tenths Joules of energy. Therefore, comparing the energy consumption of our proposed SRS with software-level approaches does not provide a fair comparison. Hence, we use the required number of MAC operations to compare the efficiency of SRS with those approaches. Figure 11 compares the average required MAC operation for our proposed similarity-aware CNN (SRS+SQS) for both symmetric and asymmetric quantization, CBinfer, DeepCach and FP32 to process a series of random video sequences. Figure 11 also shows the required MAC operations when running our proposed similarity-aware CNN but using the conventional quantization approach (SRS+INT8). We measured the number of MAC operations of proposed similarity aware CNN for $\gamma = 0.1$, $\gamma = 0.3$ $\gamma = 0.5$. Figure 11 reveals that the proposed similarity aware CNN requires fewer MAC operations compare to other approaches because SRS skips MAC operations associated with individual unchanged pixels. Furthermore, asymmetric quantization requires slightly more MAC operations than symmetric quantization. This is due to the fact that asymmetric quantization often results in a smaller clipping range. Due to the lower clipping range, the step-size factor is reduced, resulting in an increase in *ifmap* pixels changes between subsequent frames.

However, the throughput of CBinfer and DeepCach is highly dependent on the pattern of changes between consecutive frames i.e., when a row of *ifmap* pixels changes between two consecutive frames, CBifer loses all its performance. On the other hand, the computational complexity reduction of our proposed similarity-aware CNN is independent of the pattern of similarity between consecutive frames. Notably, SRS+INT8 outperforms CBinfer and DeepCach. However, figure 11 shows that SRS+SQS outperforms SRS+INT8 as SQS increases the similarity between consecutive pixels. More computations can be skips with larger γ values; however, larger γ reduces

TABLE I: The classification error of VGG-16 using SQS, conventional quantization, and SigmaDelta network compared to full precision FP32 network. SQS is repeated for $\gamma_0 = 0.1$, $\gamma_0 = 0.3$ $\gamma_0 = 0.5$ [7].

	Error (Top-1)	Error (Top5)	MACs (Gops)
INT8 [28]	2.22%	1.65%	5.1
$\text{SQS}_{\gamma_0}^{sym}$	2.61%	1.72%	3.14
$\text{SQS}_{\gamma_1}^{sym}$	2.83%	1.77%	3.01
$\text{SQS}_{\gamma_2}^{sym}$	3.01%	1.83%	2.94
$\text{SQS}_{\gamma_0}^{asym}$	2.56%	1.69%	3.18
$\text{SQS}_{\gamma_1}^{asym}$	2.81%	1.76%	3.01
$\text{SQS}_{\gamma_2}^{asym}$	2.92%	1.81%	2.98
SigmaDelta	2.93%	1.76%	3.11

TABLE II: Normalized energy cost relative to a MAC operation [33]

	ADD	MAC	Register File	Global Buffer	DRAM
Norm Energy	0.1×	1×	1×	6×	200×

the detection accuracy. The fourth column of table I compares the averaged required mac operations for performing image classification on a series of random sequences of video frames. Table I shows that our proposed approach outperforms the SigmaDelta and INT8 in the required number of MAC operations. However, asymmetric SQS requires more MAC operations compared to symmetric SQS. Notably that in this paper we only aim to imrove the energy consumption of video recognition and we don't aim to improve the inference time and we leave it as feature work.

Notably, SRS does not attempt to improve execution time by utilising unchanged pixels, as SRS does not leverage the MAC engine's idle cycle for *ifmap* pixels. As a result, the number of cycles required by SRS and RS for each convolutional layer is identical, and the runtime of similarity-aware CNNs and conventional CNNs is similar to that of the other layers, i.e., activation is identical between similarity-aware and traditional CNNs.

C. Energy consumption

We evaluate the energy consumption of the proposed similarity-aware CNN by calculating the energy consumption of convolutional layers, as they are responsible for 90% of the overall required energy for CNNs [9]. Energy consumption of a convolutional layer is estimated by accounting for the total MAC operations alongside with the number memory accesses, weighting them with their energy costs. Table II shows the normalized energy consumption of accessing data from different storage levels relative to the computation of a MAC. These numbers are extracted from a commercial 45-nm process, and we used them in our final experiments [33].

We examine the SRS energy consumption behaviour on a variety of convolutional layer configurations. We conduct

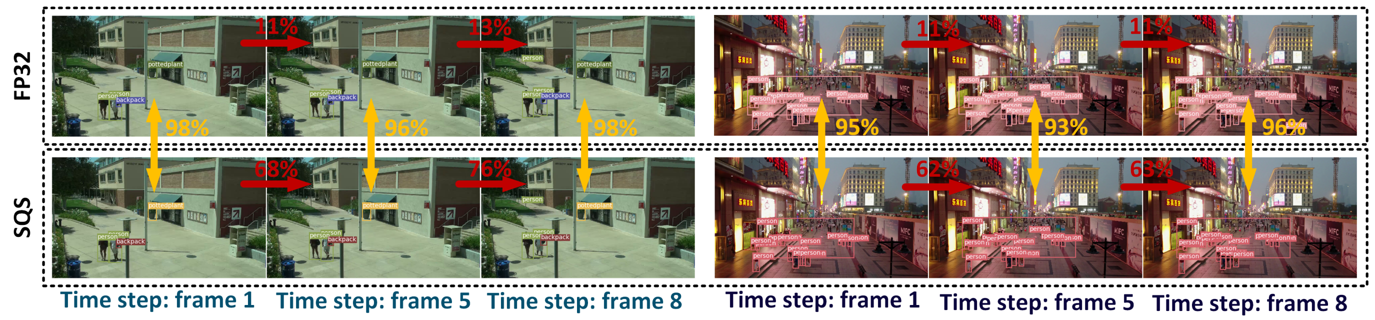


Fig. 10: Detection result (bounding boxes) , average of IoU between bounding boxes (yellow arrows), and unchanged *ifmap* pixels (red arrows) when processing video frames with five and three time step distances with FP32 Yolo-V3 [7] (top row) and SQS Yolo-V3 (Bottom row)

TABLE III: Comparing the configuration of convolutional layers of YOLO and the energy consumption of RS and SRS dataflows for those layers

Layer	ofmap (K,P,Q)	filter (C,S,R)	ofmap (MB)	filter (MB)	*ifmap* (MB)	RS (μJ)	SRS (μJ)	Threshold (%)
Conv1	(16,416,416)	(3,3,3)	0.00864	5.3	0.0037	548	778	100
Conv2	(32, 208, 208)	(16,3,3)	0.018e	2.76	5.3	2904	2266	53
Conv3	(64,104,104)	(32,3,3)	0.073	1.38	2.76	2849	2238	49
Conv4	(256,52,52)	(128,3,3)	0.29	0.69	1.38	2940	2140	47
Conv5	(512,26,26)	(256,3,3)	1.8	0.34	0.69	3135	2616	58
Conv6	(1024,13,13)	(512,3,3)	4.7	0.17	0.34	3127	2176	48
Conv7	(11024,13,13)	(512,1,1)	0.0128	0.17	0.17	709	900	100

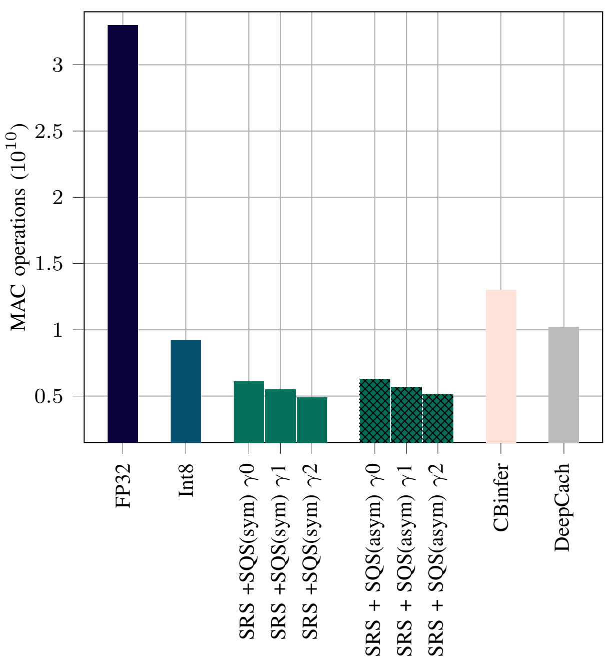


Fig. 11: Comparing the average required number of MAC operation for video object detection of floating-point precision (FP32), CBinfer, DeepCach, and SRS implementations of YOLOv3 [7]. SRS is repeated for symmetric and asymmetric SQS with $\gamma_0 = 0.1$, $\gamma_0 = 0.3$ $\gamma_0 = 0.5$

the energy consumption evaluation on Darknet (Darknet is the backbone network of Yolo-V3) [7] and Resnet-18 [10] workloads as widely used backbone networks for object detection applications. Table III shows the configuration of seven convolutional layers (Conv1 to Conv7) with different configurations (ofmap and filter sizes). These convolutional layers are driven from Resnet [10] and Darknet [7].

Similar to CBinfer [14], DeepCach [15], and SigmaDelta [13], we compare the energy consumption of our proposed similarity-aware CNN with processing individual video frames, independently. Figure 12 compares the energy breakdown of the Conv2 layers processing by row-stationary dataflow (RS) and similarity-aware dataflows (SRS). As can be seen, the energy consumption of the row-stationary dataflow is dominated by RF accesses. Among RF accesses, the weight RF consumes more energy than *ifmap* and *psum* RF accesses because there are more accesses. MAC operations and GB accesses consume more energy than DRAM accesses. Assuming that 80% of *ifmap* pixels remained unchanged compared to the previous frame (the average unchanged data over VIRAT), the energy breakdown of SRS shown in the right columns. Since SRS eliminates all weight and *psum* RF accesses and MAC operations for unchanged *ifmap* pixels, the energy consumption of weight and *psum* RF and MAC operations are reduced by 80%. On the other hand, SRS increases accesses to *ifmap* RF, GB and DR. Consequently, SRS increases the energy consumption of DRAM and GB. Despite increasing *ifmap* energy consumption, due to the significant reduction in weight and *psum* RF energy consumption, SRS decreases the

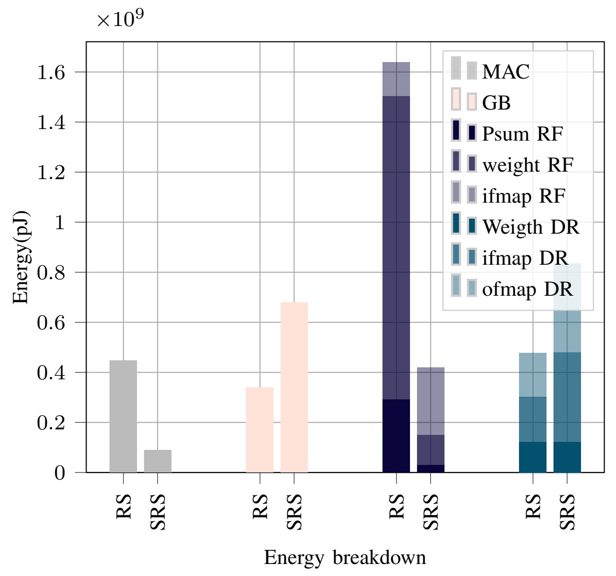


Fig. 12: Energy breakdown of Conv2 for RS and SRS dataflows

overall energy consumption of RF.

Figure 13 compares the energy consumption of processing an *ifmap* by Conv1-Conv7 with SRS and row-stationary dataflows. For SRS, we assume that 80% of *ifmap* pixels remained unchanged compare to the previous *ifmap* pixels. For Conv2 to Conv6, SRS reduces the total energy consumption of processing *ifmap* by 21%, 21%, 27%, 16%, 30%, respectively. Since the energy consumption of row-stationary dataflow in Conv2-Conv6 is dominated by the RF, eliminating 80% of weight/*psum* accesses and MAC operation by SRS leads to a reduction in total energy consumption for these layers.

Figure 13 shows that energy consumption of row-stationary dataflow in Conv1 and Conv7 is dominated by DRAM and GB, respectively. This is because of small filter size of these layers. Consequently, increasing DRAM accesses by SRS leads to small growth in overall energy consumption of Conv1 and Conv7. As the table III shows, the energy consumption of Conv1 and Conv7 are considerably less than Conv2-Conv6. Hence, a small increase in energy consumption of these layers does not significantly increase the total energy consumption of similarity-aware CNNs.

Generally, the energy efficiency of SRS depends on the configuration of the convolutional layers, specifically the size of *ofmap*/weights, and the level of the similarity between *ifmap*. For each convolutional layer when the similarity between consecutive *ifmaps* is less than a threshold value, SRS loses it's performance because the energy overhead of extra GB and DRAM accesses by SRS will be more than the saved energy by eliminating MACs and RF accesses. The ninth column of table III shows the minimum similarity between *ifmap* required by SRS to reduce the energy consumption. It can be seen that with less than 50% similarity between *ifmap*, SRS improves the energy consumption of convolutional layers. The threshold value is smaller for convolutional layers with a large filter size. Threshold value increases with for smaller filter size; therefore, more similarity between *ifmaps* are required for those layers to improve energy consumption of the convolutional layer.

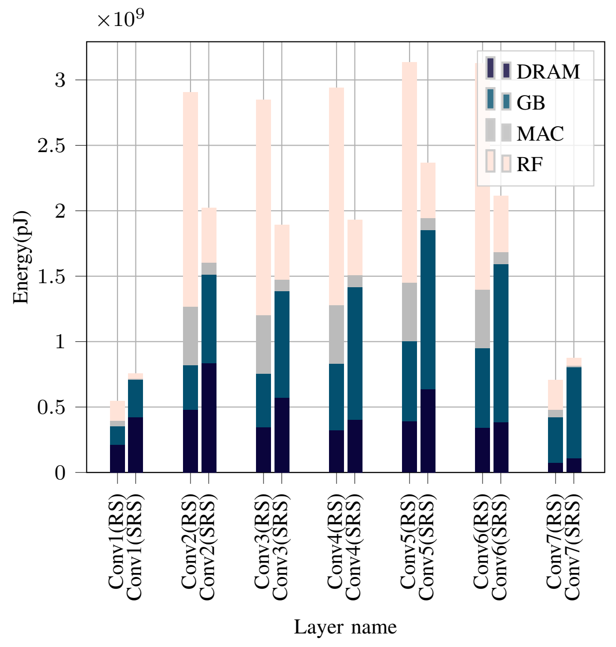


Fig. 13: Energy Consumption breakdown of Convolutional layers in table 3 processed by row-stationary dataflow (RS) and similarity-aware dataflow (SRS).

D. Accuracy-Efficiency Trade-Off

While for some scenarios any drop in accuracy is unacceptable, many applications allow for some trade-off between accuracy and energy consumption [37]. We analyse and demonstrate the trade-off between energy consumption and quantization error in consecutive execution of CNNs for Conv2-Conv5 in figure 15. The γ in equation (9) controls our preference for lower energy consumption over quantization error. Increasing γ factor immediately results a gain in unchanged *ifmap* pixel and consequently reducing power consumption. However, larger γ value increases the quantization step-sizes which consequently increases the quantization error (MSE). It's notable that further increasing of γ value the doesn't significantly improve the energy consumption of CNNs because despite the increased step-size, some *ifmap* pixels changes between consecutive frames. According to our experiments the optimal value for γ is around 0.1 - 0.2.

E. Overhead

Energy overhead: Figure 14 compares the energy breakdown of Conv1-Conv7 for SRS (for ten level of similarity) with RS dataflow. For each convolutional layer, the most left bar shows the energy breakdown of row stationary dataflow. Moving from the second left bar to the most right bar of each convolutional layer, bars show the energy consumption of SRS for 100%, 90%,...,10%, and 0% similarity, respectively.

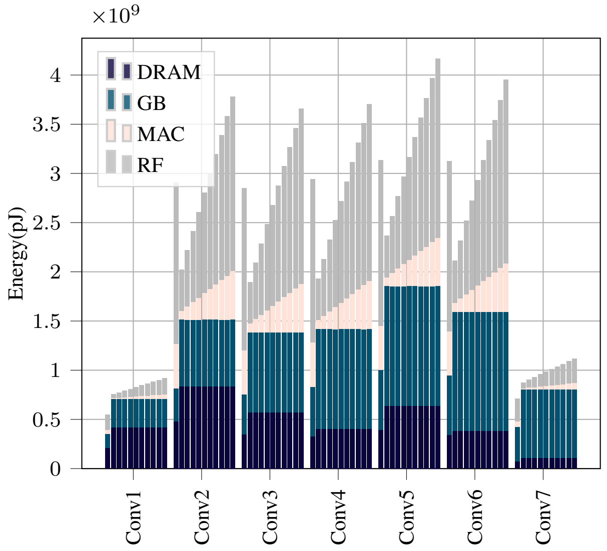


Fig. 14: Energy consumption and energy breakdown of convolutional layer for different level of similarity between *ifmaps*. For each convolutional layer, the bars from left to right show the energy breakdown for RS dataflow, SRS with 100% ,90%, $\rightarrow$, 0% similarity, respectively.

As can be seen, the energy efficiency of SRS reduces with the reduction in the level of similarity between consecutive *ifmaps*. For example in Conv4 bars, for any level of similarity above the 50% SRS outperforms the row-stationary dataflow. When the similarity between *ifmaps* drops below 50%, row-stationary outperforms SRS. For 0% similarity SRS increases the energy consumption by 26%. Our observation shows that the similarity between ifmaps often remains above 70-80% even when there is significant changes between consecutive frames. Therefore, SRS outperforms row-stationary dataflow when processing consecutive video frames. Notably, SRS does not utilize the idle cycle of MAC engines for unchanged *ifmaps* pixels. To avoid idle cycles, unchanged pixels need to be identified *ifmaps* before feeding them to the PEs array. Also, a control unit is required to restructure *ifmaps* by removing and indexing unchanged pixels. Since unchanged pixels are irregular and unstructured, designing an efficient fine-grain irregular sparse accelerator remains a challenging problem in the space of CNN accelerator design.

Memory overhead: Similarity-aware CNN requires more memory, because it needs to store and use additional intermediate *ifmaps* and the previous output. For example in Eyeriss accelerator architecture, SRS requires an extra ifmap scratchpad (12b × 16 register file) in each PE, an extra global buffer space of 30KB, and 40MB extra DRAM memory. Considering that modern CNN accelerators often come with around 0.1-24MB on chip memory [38], a 35KB kilobytes extra memory is acceptable for modern CNN accelerators.

Memory Bandwidth Overhead: The majority of DRAM bandwidth in the CNN accelerators is associated with weight parameters. The bandwidth for weight parameters in Yolo-V3 varies between 0.75 to 9 world/cycle across convolutional layers. However, *ifmaps* only require bandwidth within 0.04 to 0.41 world/cycle. Therefore, the additional bandwidth requires by SRS to read the latest *ifmaps* does not impose significant overhead on total DRAM bandwidth. SRS increases the required bandwidth by 3%-14% for the Yolo-V3 network.

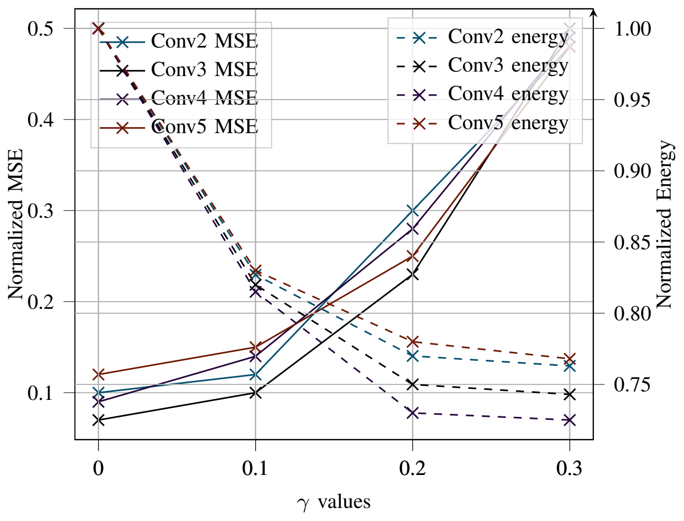


Fig. 15: Trade off between energy consumption and quantization error of convolutinal layer

F. *Similarity-aware CNNs for Stationary and non-stationary cameras*

The energy consumption of a similarity-aware CNN can be precisely approximated by adding the energy consumed by each of its convolutional layers [9], as convolutional layers consume about 90% of total energy in CNNs [22]. To fully investigate the proposed approach, we compare the energy consumption of similarity-aware Resnet18 and Yolov3 networks when the similarity between consecutive frames is between 5% and 95% to the energy consumption of the conventional approach, which processes video frames individually. The figures 17a and 17b compare the energy consumption of conventional CNNs (RS) and similarity-aware CNNs (RSR) approaches for Resnet18 and Yolov3, respectively, at various levels of similarity. It's worth noting that the similarity indicated on the x-axis of figures 17a and 17b refer to the similarity of feature maps pixels between consecutive frames in the first convolutional layer. However, we observed that this similarity varies by 30% in deeper convolutional layers. In the final convolutional layer, this variation reaches a maximum of 50%. While the energy consumption of RS remains constant over all range of similarity between frames, SRS consumes energy dynamically and is dependent on the similarity of video frames. As seen in Yolov3 and Resnet18, conventional RS outperforms RSR when the similarity is less than about 40%-50%, because the overhead associated with fetching additional feature maps from DRAM outweighs the energy saved by avoiding computations and memory accesses. However, as similarity increases between video frames, SRS skips more computations and memory accesses and its energy consumption decreases almost linearly, outperforming RS approach. Given figures (17a and 17b), we can infer whether similarity-aware CNNs (Yolov3 and Resnet) outperform conventional CNNs if we know the degree of similarity between features in successive frames. Therefore, we measured feature similarity between consecutive frames across an entire dataset to demonstrate the application and performance of similarity-aware CNNs. We extended the analysis to include both stationary and non-stationary camera videos. Figure 16b depicts the histogram of similarity between feature maps in ImagenetVID dataset [1], captured by non-stationary cameras. ImagenetVID is a collection of videos captured at a rate of 20-25 fps by dynamic cameras including fast-moving objects. Regardless of the fast-moving senses between frames, the average of similarity between frames remains slightly above 50%; thus, the similarity-aware CNNs are capable of achieving acceptable performance even for dynamic videos. Furthermore, figure 16a illustrates the histogram of VIRAT ground dataset [8] captured by stationary cameras. As can be seen, the average similarity between features in consecutive frames exceeds 80%, implying that similarity-aware CNNs performed particularly well when processing stationary camera videos. We observed that a lower degree of similarity between the features of consecutive frames (as indicated by the green bars in figure 16a) is associated with frames containing fast moving objects that are typically close to the camera; however, when the moving objects are further away from the camera or there are no moving objects in the environment, the degree of similarity between the features of consecutive frames remains greater than 80%. In conclusion, while similarity-aware CNNs perform better with static cameras due to their high degree of similarity, they can also be used with dynamic cameras but perform less well.

V. Conclusions and Future works

Given the computational bottleneck associated with processing a large number of video frames individually by CNNs for video recognition applications, this article proposes a novel similarity-aware CNN that takes advantage of small differences in the feature pixels of consecutive frames to reduce the computational complexity of video recognition. The proposed similarity-aware CNN employs a similarity-aware quantization technique (SQS) to minimise the difference in pixel values between frame pairings. When frames have similar appearances, minimising differences results in feature pixels remaining intact.Similarity-aware CNNs employ a novel similarity-aware computational dataflow (SRS) that detects unchanged feature pixels between successive frames and omits redundant MAC procedures and memory accesses associated with them. SQS increases theunchanged feature pixels between frames by up to 85% while incurring a negligible accuracy loss of $< 1\%$. Additionally, SRS reduces video recognition's energy consumption by up to 30%.

Acknowledgement

This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/S030069/1. Experimental data can be found at https://doi.org/10.5258/SOTON/D2067

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[2] Z. Sabetsarvestani and e. a. Sober, Barak and, "Artificial intelligence for art investigation: Meeting the challenge of separating x-ray images of the ghent altarpiece," *Science advances*, vol. 5, no. 8, 2019.

[3] G. Singh and P. Bhatia, "Relation extraction using explicit context conditioning," *Proc.Conf. NAACL*, 2019.

[4] E. Li, L. Zeng, and Z. Zhou et al., "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans.on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[5] V. Sze, Y.-H. Chen, and T.-J. Yanget al., "Efficient processing of deep neural networks: A tutorial and survey," *Proc. of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[6] "https://support.google.com/googlenest/answer/9243617?hl=en-gbs."

[7] M. J. Shafiee, B. Chywl, and F. Li et al., "Fast yolo: a fast you only look once system for real-time embedded object detection in video," *preprint arXiv:1709.05943*, 2017.

[8] S. Oh, A. Hoogs, and A. Perera et al., "A large-scale benchmark dataset for event recognition in surveillance video," in *Proc. of IEEE CVPR,*, 2011.

[9] A. Parashar, P. Raina, and Y. S. Shao et al., "Timeloop: A systematic approach to dnn accelerator evaluation," in *ISPASS*. IEEE, 2019, pp. 304–315.

[10] K. He, X. Zhang, and S. Ren et al., "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.

[11] A. Sabet, J. Hare, B. Al-Hashimi, and G. V. Merrett, "Temporal early exits for efficient video object detection," *preprint arXiv:2106.11208*, 2021.

[12] M. Riera, J.-M. Arnau, and A. González, "Computation reuse in dnns by exploiting input similarity," in *ACM/IEEE 45th Annu. Int.Symp. on Computer Architecture (ISCA)*. IEEE, 2018, pp. 57–68.

[13] P. O'Connor and M. Welling, "Sigma delta quantized networks," in *ICLR*, 2017.

[14] L. Cavigelli and L. Benini, "Cbinfer: Exploiting frame-to-frame locality for faster convolutional network inference on video streams," *IEEE TCSVT*, 2019.

[15] M. Xu, M. Zhu, and Y. Liu et al., "Deepcache: Principled cache for mobile deep vision," in *Proc. of the 24th Annu. Int.Conf. on Mobile Computing and Networking*, ser. MobiCom '18, 2018, p. 129–144.

[16] M. Sankaradas, V. Jakkula, and S. Cadambi et al., "A massively parallel coprocessor for convolutional neural networks," in *ASAP*. IEEE, 2009, pp. 53–60.

[17] S. Chakradhar, M. Sankaradas, and V. Jakkula et al., "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. of the 37th Annu. Int.Symp. on Computer architecture*, 2010, pp. 247–257.

[18] S. Gupta, A. Agrawal, and K. Gopalakrishnan et al., "Deep learning with limited numerical precision," in *Int.Conf. on Machine Learning*, 2015, pp. 1737–1746.

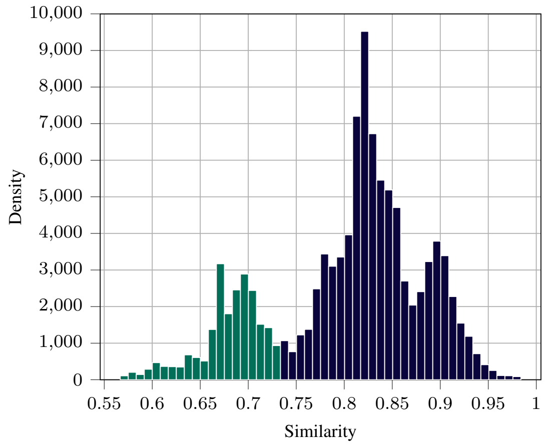
[19] Z. Du, R. Fasthuber, and T. Chen et al., "Shidiannao: Shifting vision processing closer to the sensor," in *Proc. of the 42nd Annu. Int.Symp. on Computer Architecture*, 2015, pp. 92–104.

[20] Z. Yuan, Y. Yang, and Y. et al., "14.2 a 65nm 24.7 μj/frame 12.3 mw activation-similarity-aware convolutional neural network video processor using hybrid precision, inter-frame data reuse and mixed-bit-width difference-frame data codec," in *ISSCC*. IEEE, 2020, pp. 232–234.

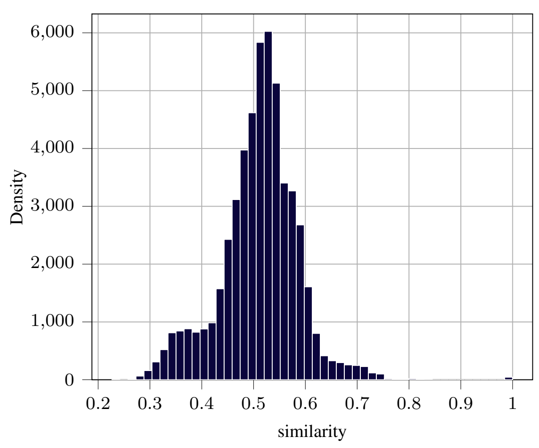
[21] A. G. Howard, M. Zhu, B. Chen, and et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *preprint arXiv:1704.04861*, 2017.

[22] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Int.Conf. on artificial neural networks*. Springer, 2014, pp. 281–290.

[23] S. Shin, Y. Boo, and W. Sung, "Fixed-point optimization of deep neural networks with adaptive step size retraining," in *IEEE ICASSP*, 2017, pp. 1203–1207.

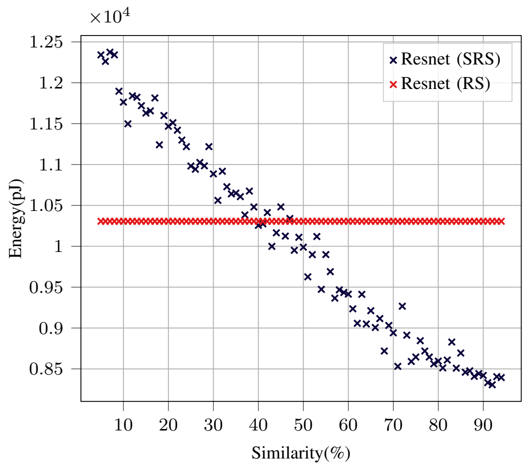


(a) Stationary

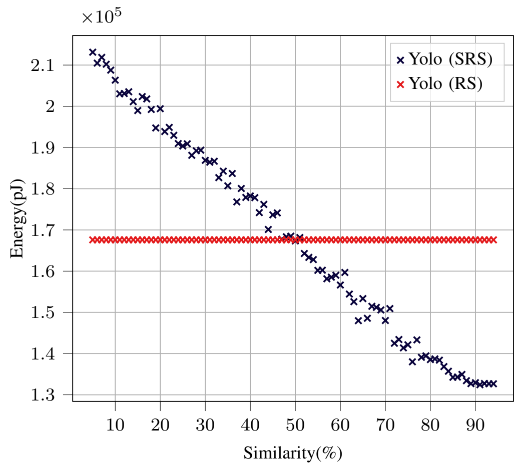


(b) Non-stationary

Fig. 16: Histogram of similarity between the features of consecutive frames from stationary (VIRAT ground) and non-stationary (ImagenetVID dataset) cameras, When frames are processed by Yolov3.



(a) Resnet



(b) Yolo

Fig. 17: Comparing the energy consumption of similarity-aware Yolo and Resnet to that of conventional Yolo and Resnet when frame similarity varies between 10% and 90%.

[24] D. e. a. Lin, "Fixed point quantization of deep convolutional networks," in *ICML*, 2016, pp. 2849–2858.

[25] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *preprint arXiv:1806.08342*, 2018.

[26] X. Zhao, Y. Wang, X. Cai, C. Liu, and L. Zhang, "Linear symmetric quantization of neural networks for low-precision integer hardware," 2020.

[27] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization," *preprint arXiv:1511.06488*, 2015.

[28] B. Jacob, S. Kligys, and B. Chen et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. of the IEEE CVPR*, 2018, pp. 2704–2713.

[29] N. P. Jouppi, C. Young, and N. Patil et al., "In-datacenter performance analysis of a tensor processing unit," in *Int.Symp. on Computer Architecture*, 2017, pp. 1–12.

[30] V. Akhlaghi, A. Yazdanbakhsh, and K. Samadi et al., "Snapea: Predictive early activation for reducing computation in deep convolutional neural networks," in *ISCA*. IEEE, 2018, pp. 662–673.

[31] A. Parashar, M. Rhu, and A. Mukkara et al., "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 27–40, 2017.

[32] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[33] M. Horowitz, "computing's energy problem (and what we can do about it)," in *IEEE ISSCC*, 2014.

[34] T.-Y. e. a. Lin, "Microsoft coco: Common objects in context," in *ECCV*. Springer, 2014, pp. 740–755.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE ICIV*, 2015, pp. 1026–1034.

[36] S. et al., "Very deep convolutional networks for large-scale image recognition," *preprint arXiv:1409.1556*, 2014.

[37] N. K. Jayakodi, S. Belakaria, and A. Deshwal et al., "Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models," *ACM TECS*, vol. 19, no. 1, pp. 1–24, 2020.

[38] K. Siu, D. M. Stuart, and M. Mahmoud et al., "Memory requirements for convolutional neural network hardware accelerators," in *IISWC*, 2018, pp. 111–121.