



Circuits and Systems

Mekelweg 4,
2628 CD Delft
The Netherlands

<http://ens.ewi.tudelft.nl/>

CAS-2021-5081378

M.Sc. Thesis

Temporal Delta Layer: Training Towards Brain Inspired Temporal Sparsity

Preetha Vijayan - 5081378

Abstract

In the recent past, real-time video processing using state-of-the-art deep neural networks (DNN) has achieved human-like accuracy but at the cost of high energy consumption, making them infeasible for edge device deployment. The energy consumed by running DNNs on hardware accelerators is dominated by the number of memory read/writes and multiply-accumulate (MAC) operations required. As a potential solution, this work explores the role of activation sparsity in efficient DNN inference. As the predominant operation in DNNs is matrix-vector multiplication of weights with activations, skipping operations and memory fetches where (at least) one of them is zero can make inference more energy efficient. Although spatial sparsification of activations is researched extensively, introducing and exploiting temporal sparsity is much less explored in DNN literature. This work presents a new DNN layer (called temporal delta layer) whose primary objective is to induce temporal activation sparsity during training. The temporal delta layer promotes activation sparsity by performing delta operation facilitated by activation quantization and l_1 norm based penalty to the cost function. During inference, the resulting model acts as a conventional quantized DNN with high temporal activation sparsity. The new layer was incorporated as a part of the standard ResNet50 architecture to be trained and tested on the popular human action recognition dataset (UCF101). The method caused 2x improvement in activation sparsity, with 5% accuracy loss.

Temporal Delta Layer: Training Towards Brain Inspired Temporal Sparsity for Energy Efficient Deep Neural Networks

MASTER THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

by

Preetha Vijayan - 5081378
born in Kerala, India

Thesis committee:	Prof. Dr. Ir. Rene Van Leuken	TU Delft, CAS
	Prof. Dr. Ir. Zaid Al Ars	TU Delft, CE
	Dr. Ir. Amirreza Yousefzadeh	IMEC
	Dr. Ir. Manolis Sifalakis	IMEC

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology



Delft University of Technology

Copyright © 2021 Circuits and Systems Group
All rights reserved.

Abstract

In the recent past, real-time video processing using state-of-the-art deep neural networks (DNN) has achieved human-like accuracy but at the cost of high energy consumption, making them infeasible for edge device deployment. The energy consumed by running DNNs on hardware accelerators is dominated by the number of memory read/writes and multiply-accumulate (MAC) operations required. As a potential solution, this work explores the role of activation sparsity in efficient DNN inference. As the predominant operation in DNNs is matrix-vector multiplication of weights with activations, skipping operations and memory fetches where (at least) one of them is zero can make inference more energy efficient. Although spatial sparsification of activations is researched extensively, introducing and exploiting temporal sparsity is much less explored in DNN literature. This work presents a new DNN layer (called temporal delta layer) whose primary objective is to induce temporal activation sparsity during training. The temporal delta layer promotes activation sparsity by performing delta operation facilitated by activation quantization and l_1 norm based penalty to the cost function. During inference, the resulting model acts as a conventional quantized DNN with high temporal activation sparsity. The new layer was incorporated as a part of the standard ResNet50 architecture to be trained and tested on the popular human action recognition dataset (UCF101). The method caused 2x improvement in activation sparsity, with 5% accuracy loss.

Acknowledgments

First and foremost, I would like to thank Dr. Rene van Leuken for his invaluable guidance throughout this work. His inputs and suggestions have not only improved the quality of this thesis but also has made me a better researcher. I also genuinely appreciate the warmth with which he inquired about my health and well-being during our calls. I would also like to thank Dr. Amirreza Yousefzadeh and Dr. Manolis Sifalakis for entrusting me with this project. They have always made me aware of the bigger picture of the work I was doing. Also, they have inspired me to focus on the learning rather than worry about the result, which I hope to adapt into my future endeavors as well. Additionally, I extend my gratitude to Dr. Zaid Al Ars for being a part of my thesis committee and taking time to review and assess my work.

I am extremely thankful for my friends Saran, Adeep, and Vishal for supporting me through the good, bad and ugly days. Special thanks to fellow master student, Shreya, for the monthly check-in calls, which made me feel a little less alone on this journey. Also, I am eternally grateful for my friend, Ashu, whose kindness, support (and memes) have sustained me for most of my adult life.

Finally, I am indebted to my brother, Praveen, for being the wind under my wings for as long as I can remember. Last but not the least, I take this opportunity to thank my Amma (who was my first teacher) and Achan, for keeping me grounded during the good days, and for not letting me be miserable during the bad days.

To the Class of 2021 ...!

Preetha Vijayan - 5081378
Delft, The Netherlands
25 August, 2021

Contents

Abstract	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation	1
1.2 Problem Description	1
1.3 Objective	2
1.4 Contributions	3
1.5 Outline	4
I Literature Review	5
2 Energy Efficient Deep Neural Networks	7
2.1 Introduction	7
2.2 Neural Networks - Basics	7
2.2.1 Perceptron	7
2.2.2 Multi Layer Perceptrons	8
2.3 Deep Neural Networks	8
2.3.1 Training	9
2.3.2 Inference	10
2.3.3 Activation Functions	10
2.3.4 Convolutional Neural Networks	12
2.4 Energy Consumption in DNN	14
2.5 Energy Efficient Solutions	14
2.5.1 Quantization	14
2.5.2 Pruning	16
2.5.3 Neural Architecture Search	17
2.5.4 Knowledge Distillation	18
2.6 Conclusion	19
3 Neuromorphic Deep Neural Networks	21
3.1 Sparsity in the Biological Neural Network	21
3.2 Sparsity and Energy Consumption	21
3.3 Sparsity - Related Works	22
3.4 Conclusion	26
II Proposed Methodology	27
4 Temporal Delta Layer	29
4.1 Temporal Redundancy in Feature Maps	29
4.2 Temporal Delta Layer	29

4.2.1	Delta Inference	29
4.2.2	Activation Quantization to Induce Sparsity	31
4.2.3	Sparsity Penalty	37
4.3	Proposed Algorithms	39
4.3.1	Temporal Delta Layer with Fixed Point Quantization	39
4.3.2	Temporal Delta Layer with Learned Step Size Quantization	40
4.4	Conclusion	41
III Results		43
5	Experiments and Results	45
5.1	Experiment Setup	45
5.1.1	Application: Human Action Recognition	45
5.1.2	Dataset - UCF101	45
5.1.3	Architecture	45
5.1.4	Pre-processing - Frame Generation	48
5.1.5	Evaluation Metrics	49
5.1.6	Baseline	49
5.2	Experiments	49
5.2.1	Accuracy v/s Activation Sparsity	51
5.3	Result Analysis	53
5.3.1	Importance of Step Size Initialization	55
5.3.2	Importance of Penalty Co-efficient	55
5.3.3	Layer-wise Sparsity Pattern	57
5.3.4	Static v/s Moving Camera Input	58
5.3.5	Estimated Reduction in Computation and Memory Access	58
5.4	Conclusion	60
6	Conclusions and Future Work	61
6.1	Conclusion	61
6.2	Future Work	62

List of Figures

1.1	(a) Standard DNN performing frame based processing, and (b) delta based model that processes only data that has changed between frames [21]	2
1.2	(a) Standard DNN, and (b) DNN with temporal delta layer	3
1.3	Outline of the thesis	4
2.1	Structure of a perceptron [22]	7
2.2	Multi-layer perceptron with input layer, output layer and 3 hidden layers [24]	8
2.3	Hierarchical feature learning in DNN [26]	9
2.4	Training v/s inference [29]	10
2.5	Sigmoid function and its derivative. Sigmoid ranges between 0 and 1. The maximum value of its derivative is 0.25. [30]	11
2.6	ReLU function [33]	11
2.7	Softmax function - largest value in the output layer has the largest probability [35]	12
2.8	Demonstration of convolution operation [41]	13
2.9	Average and Maximum pooling [42]	13
2.10	Energy consumption of multiply-accumulations in 45nm 0.9V [48]	15
2.11	Pareto analysis accessing the effect of quantization on DNNs classifying MNIST dataset [49]	15
2.12	Neurons and synapses pruning [68]	17
2.13	Neural Architecture Search [73]	18
2.14	Knowledge distillation	19
3.1	Sparsity in Δx can save multiplications between Δx and columns of W that correspond to zero [20]	22
3.2	Types of sparsity available in DNNs	23
3.3	Contours of l_1 norm and the Hoyer (the ratio between l_1 and l_2 norms) regulariser of a 2D vector [10]	23
3.4	Natural spatial sparsity within the activation maps due to ReLU [98]	24
3.5	A sample video sequence demonstrating the fact that moving objects make up only a small part of the overall scene. [99]	25
4.1	Evidence of temporal redundancy in video inputs and consequent feature maps.	30
4.2	Demonstration of two consecutive activation maps leading to near zero deltas	32
4.3	(left) Unquantized delta calculation of two similar vectors causing near zero values, (middle) a uniform quantizer, (right) quantized delta calculation leading to absolute zeroes.	33
4.4	Demonstration of fixed point quantization inducing temporal sparsity. Bits stored in memory are shown in yellow and green. Sign bit is shown in blue. Unused bits are shown in gray.	34
4.5	Importance of step size in quantization: on the right side, in all three cases, the data is quantized to five bins with different uniform step sizes, but without optimum step size value, the quantization can alter the range and resolution of the original data	35

4.6	Modified LSQ - forward pass including scaling, rounding and de-scaling. Quantization step-size considered is 2.263	37
4.7	Activation density v/s number of trials [107]. As the learning increases, the number of active neurons decreases. V1 represents the primary visual cortex and AL represents the anterolateral region of the specimen’s visual processing system.	38
4.8	l_1 norm generating sparser solution better than l_2 . Line H0 represents the task, disc (pink) represents the l_2 ball, diamond (blue) represents the l_1 ball.	39
4.9	Methodology flow of temporal delta layer with fixed point quantization	40
4.10	Methodology flow of temporal delta layer with learned step size quantization	42
5.1	Samples of action categories from UCF101 dataset [110]	46
5.2	Two stream architecture for action recognition - original [112]	47
5.3	Possible input modalities for action recognition: (from left to right) RGB, RGB temporal difference, optical flow - U channel, optical flow - V channel [2]	47
5.4	ResNet architecture: (a) Identity block, (b) convolution block, and (c) ResNet50 block diagram [114]	48
5.5	Baseline: two stream network with single frame RGB images for spatial stream and stack of RGB difference frames for temporal stream. The model achieved 82% validation accuracy.	50
5.6	The distribution of fractional bits required over all the layers of ResNet50. The activations in all layers is quantized to 8 bits (blue). However, each layer selects its number of fractional bits (red) depending on the maximum value in the layer to maintain the range.	53
5.7	Effect of quantization on sparsity: in (a), and (c), the unquantized feature maps has a lot of near zero values, which in turn leads to near zero values in their delta map - (e), deterring the increase of sparsity. Whereas, quantizing the corresponding activations, (b) and (d), causes its delta map, (f), to have more sparsity.	54
5.8	Effect of step size initialisation on accuracy and activation sparsity: '3-3-4-4-4-3' gives a balanced result of accuracy and sparsity.	56
5.9	Evolution of step size from initialization to convergence. As step-size is a learnable parameter, it gets re-adjusted during training to cause minimum information loss in each layer.	56
5.10	Effect of penalty co-efficient: (a) accuracy v/s epoch and (b) activation sparsity v/s epoch. Based on these, a penalty co-efficient of $1e^{-4}$ (red) was chosen for this work.	57
5.11	Comparison between layer-wise activation sparsity of baseline and proposed model - spatial stream. The proposed method results in $> 90\%$ sparsity in more than half of the layers.	57
5.12	Comparison between layer-wise activation sparsity of baseline and proposed model - temporal stream.	58
5.13	Activation sparsity of 'TaiChi' (green) and 'Rafting' (yellow) class in spatial stream. Blue dashed line is the average sparsity over all 101 classes.	59

List of Tables

5.1	Spatial stream - comparison of accuracy and activation sparsity obtained through the proposed scenarios against the benchmark. In the case of fixed point quantization, the reported results are for a bitwidth of 6 bits.	51
5.2	Temporal stream - comparison of accuracy and activation sparsity obtained through the proposed scenarios against the benchmark. In the case of fixed point quantization, the reported results are for a bitwidth of 7 bits.	51
5.3	Result of decreasing activation bitwidth in an attempt to increase activation sparsity while maintaining accuracy. For spatial stream, decreasing below 6 bits caused the accuracy to drop considerably. For temporal stream, the same happened below 7 bits.	53
5.4	Final results on 2 stream network after average fusing the spatial and temporal stream weights. With 5% accuracy loss, the proposed method almost doubles the activation sparsity available in comparison to the baseline.	55

Video processing in real-time is still a challenging task. However, the domain has garnered interest in a wide variety of real-world applications like medical diagnostics, video surveillance, robotics, agriculture, and driver assistance systems. In recent years, the algorithms for video processing has shifted to machine learning based multi-layered, complex, trained feature extractors called deep neural networks (DNNs) [1]. DNNs have lately managed to successfully analyze video data to perform action recognition [2], object tracking [3], object detection [4], etc., with human-like accuracy and robustness.

Unfortunately, the high accuracy of DNNs comes with high compute and memory costs, making them highly costly in terms of energy consumption. This makes them infeasible for always-on edge devices. Thus, to combat this, algorithms has been developed to design tailor-made hardware architectures [5] [6] [7], as well as to adapt the neural network itself to avoid expensive arithmetic operations by reducing bit precision, exploiting sparsity, and developing more compact architectures [8] [9]. However, they either (a) do not result in high enough energy cost reduction, (b) are inflexible to be integrated to existing networks or (c) suffer a considerable accuracy loss.

This work attempts to propose a new method to efficiently perform inference on DNNs for video-based applications.

1.1 Motivation

Over the years, several methods has been considered to reduce the computations and memory accesses required while executing a DNN. Techniques like network pruning, quantization, regularization using sparsity inducing priors and knowledge distillation [10] [11] [12] has helped in reducing the model size resulting in lesser computation and memory consumption. These methods are particularly popular not only because they are efficient but also due to their inconsequential accuracy loss.

In all the above-mentioned solutions, sparsity is an underlying feature. This is notable as sparse tensors provides the potential to skip computations that involve multiplication with zeroes. Also, they are easier to store and access in memory.

Interestingly, the research involving sparsity induction in DNNs is facilitated by corresponding accelerators and architectures that can exploit the available sparsity in memory and compute operations on the hardware side, e.g. [13] [14] [15]. Besides academic research, several start-ups are also developing commercial versions of sparsity exploiting neural network inference processors [16] [17].

1.2 Problem Description

Structural sparsity (of weights) and spatial sparsity (of activations), are well-researched topics in DNN literature [18]. However, temporal activation sparsity is comparatively less explored

in the context of DNN, although it is a popular concept in neuromorphic computing. Fundamentally, temporal sparsity exists in a signal which is stable over time.

As conventional sensors capture signals with a fixed sampling rate, natural data (e.g. video, audio) contains a great deal of temporal redundancy. In 1991, arguably for the first time, [19] demonstrated that by mimicking the human retina, a silicon retina can reduce acquisition bandwidth by reporting only the spatial and temporal changes. This thesis takes the concept mentioned above of change based processing from data acquisition to the training and inference phases of deep neural networks. As shown in Figure 1.1.(a), DNN inference which process each frame separately with no regard to the temporal correlation is dense and obscenely wasteful. Whereas processing only the changes in the network (see Figure 1.1.(b)) can lead to zero-skipping in sparse tensor operations reducing redundant operations and memory accesses.

Additionally, in practice, performing sparse operations on DNN accelerators with parallel cores often sustain an extra cost per operation because the computations involving irregular or non-structured sparse tensors is difficult to be optimally distributed across the cores. In this case, if the sparsity level is not sufficiently high, most of the off-the-shelf accelerators performing sparse operations cannot decrease the energy cost [20].

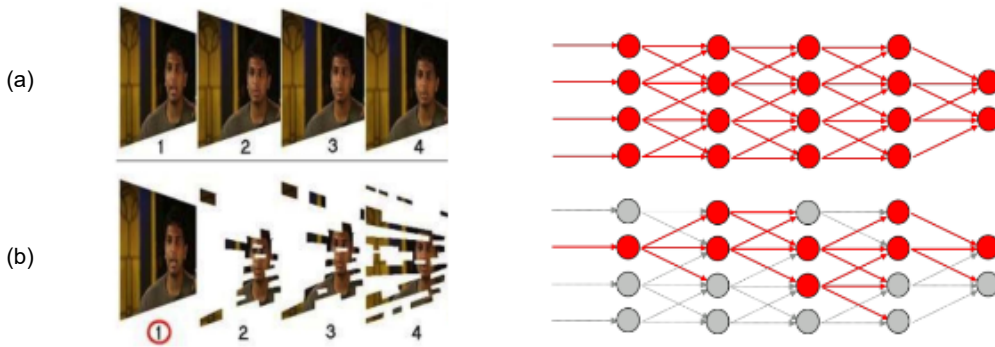


Figure 1.1: (a) Standard DNN performing frame based processing, and (b) delta based model that processes only data that has changed between frames [21]

1.3 Objective

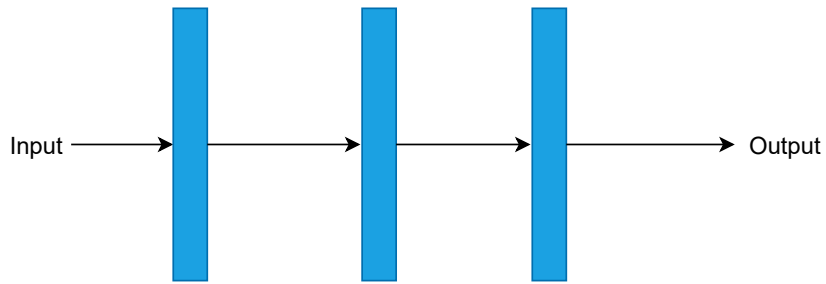
The main objectives of this work are,

1. To induce sufficiently high temporal sparsity without suffering too much accuracy loss.
2. To make the method flexible enough to be integrated with existing architectures.
3. To study whether spatial sparsification methods can facilitate the induction of temporal sparsity.
4. To investigate the consequences of the method.

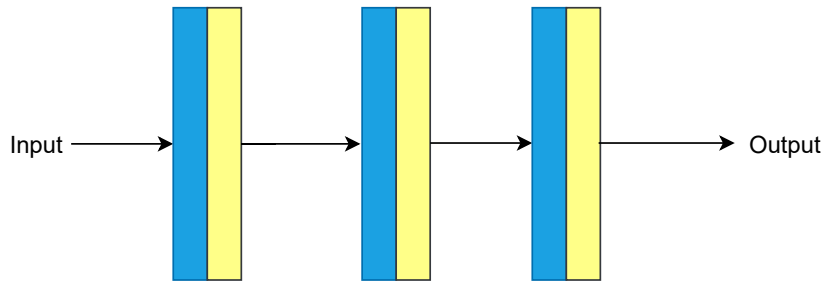
1.4 Contributions

The main contributions of this thesis are as follows:


- A methodology which induces temporal sparsity to potentially any DNN, by means of a new layer (called Temporal Delta Layer), which can be introduced in a DNN at any phase (training, refinement, or inference only).
- The new layer can be integrated into an existing architecture by placing it after all or some of the ReLU activation layer as deemed beneficial. The inclusion of this layer does not require any change to the preceding and following layers (see Figure 1.2).



(a) Standard DNN



(b) Proposed methodology


Conv layer with ReLU
activation



Temporal delta layer

Figure 1.2: (a) Standard DNN, and (b) DNN with temporal delta layer

- During the training phase, the new layer adds a novel sparsity penalty to the overall cost function of the DNN. This l_1 norm based penalty minimizes the activation density of the delta maps (i.e., temporal difference between two consecutive feature maps).
- Two activation quantization methods, namely fixed-point quantization (FXP) and learned step-size quantization (LSQ), are compared in conjunction with the new layer. Insights are provided regarding the consequence of both the methods.

- During training, in the case where the new layer is combined with LSQ, the method considers quantization step-size as a trainable parameter and is optimized by the sparsity penalty. The trainable step-size helps the optimizer to converge with the highest possible sparsity (minimizing the sparsity penalty) and the highest possible accuracy (minimizing the accuracy loss).
- At the end of training, the method results in a conventional quantized DNN. This means that during inference, the activations are already quantized, and the layer only does the delta operation.
- The new layer can be trained using two different input modalities: RGB and RGB difference, both of which were found to be compatible with the layer.

1.5 Outline

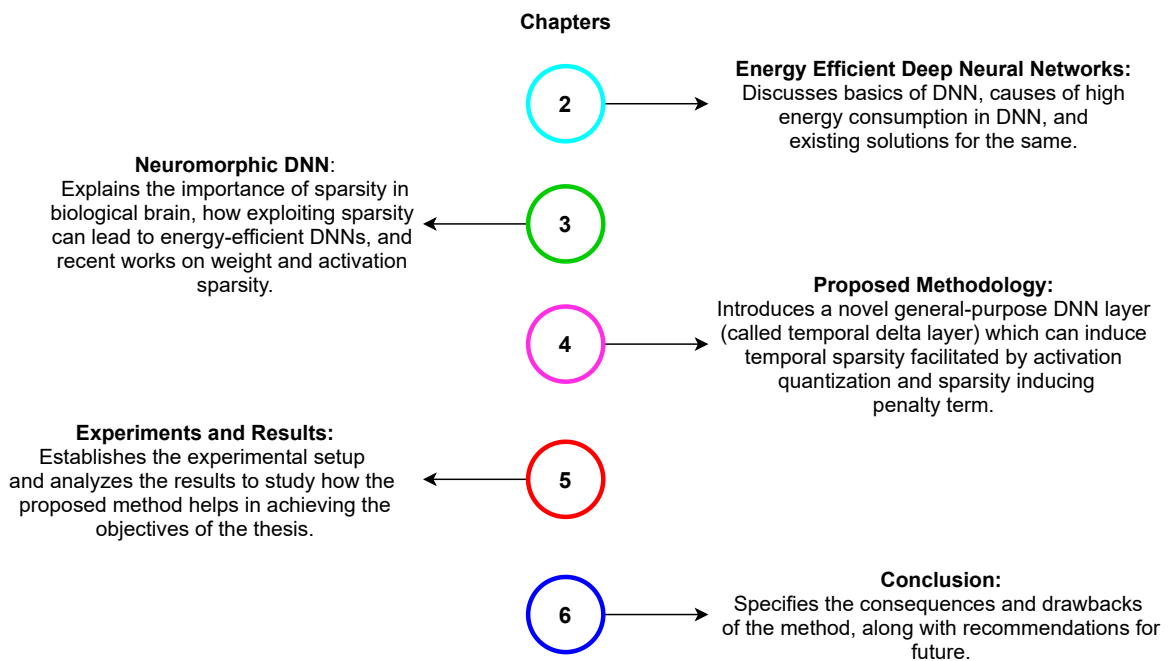


Figure 1.3: Outline of the thesis

Part I
Literature Review

2.1 Introduction

This chapter can be conceptually divided into three parts: (a) basics of deep neural network (DNN) and convolutional neural networks (CNN) as a subset of DNN, (b) what causes high energy consumption during DNN inference, and (c) what are the existing solutions which combat this problem.

2.2 Neural Networks - Basics

2.2.1 Perceptron

Perceptron is the fundamental building block of a neural network, and it represents an abstract model of the biological neuron [22]. As shown in Figure 2.1, (x_1, x_2, \dots, x_n) are the inputs, $(w_{1j}, w_{2j}, \dots, w_{nj})$ are the weights that can be learned during training, and o_j is the output. Mathematically, net_j is the scalar product between the weight vector and input vector (Eq. 2.1). Then, the value of net_j is passed through an activation function, φ , which activates the input if a threshold, θ_j , is fulfilled (Eq. 2.2). Basically, the perceptron can linearly split the solution space. Unfortunately, this indicates that a basic perceptron cannot learn problems that are not linearly separable. However, multi-layer perceptrons can solve non-linear problems if the activation, φ , is non-linear.

$$net_j = \sum_{i=0}^n x_i \cdot w_i \quad (2.1)$$

$$o_j = \varphi(net_j) = \begin{cases} 0, & \text{if } net_j < \theta_j \\ 1, & \text{if } net_j \geq \theta_j \end{cases} \quad (2.2)$$

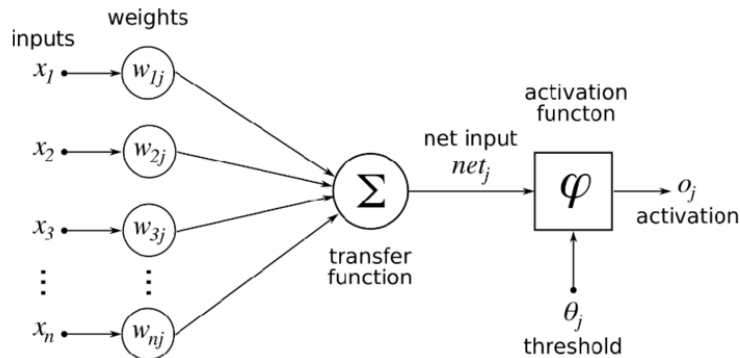


Figure 2.1: Structure of a perceptron [22]

2.2.2 Multi Layer Perceptrons

Multi-layer perceptron (MLP), also known as feedforward networks, consists of networks of perceptrons connected in series [23]. As shown in Figure 2.2, the first layer is the *input* layer, the last layer is the *output* layer, and the layers in between are the *hidden* layers. The number of hidden layers in a network represents its *depth*, and the number of perceptrons (or units) within a layer represents the *width* of that layer.

The units are fully connected between the layers. The output of every unit in layer l is connected to the input of every unit in layer $l + 1$, and the input of every unit in layer l is connected to the output of every unit in layer $l - 1$. Unlike a single perceptron, MLP can solve non-linear problems because of its multi-layered structure, given the activation function used is non-linear.

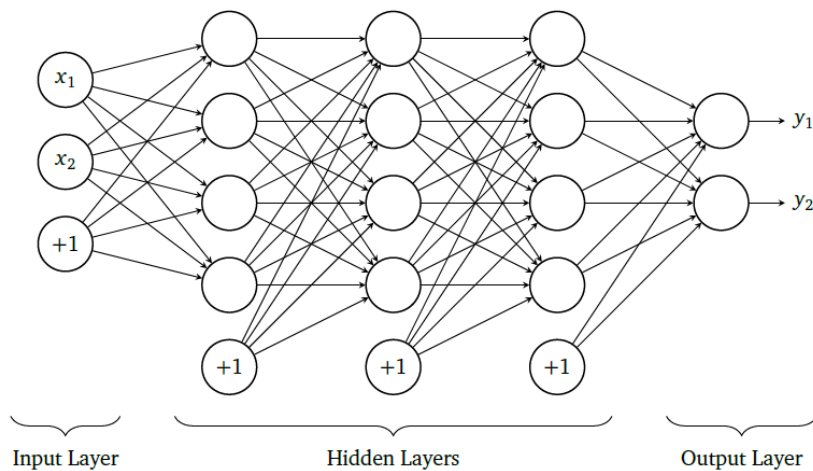


Figure 2.2: Multi-layer perceptron with input layer, output layer and 3 hidden layers [24]

Theoretically, a single-layered network can mathematically represent any function. However, practically, the training algorithm might not find the right weight parameters for the network, or generalise well. Nevertheless, it was empirically proved that deeper neural networks have a better chance at finding the right parameters and reducing the generalisation error.

The next section (2.3) mainly references the book by Ian Goodfellow [1].

2.3 Deep Neural Networks

A multi-layered network with more than three hidden layers can be termed a deep neural network. Similar to the biological brain, deep neural networks use hierarchical learning [25]. For example, in face recognition, lower layers of a DNN learn edges, curves, and corners. Middle layers compose these low-level features to form complex structures like eyes, nose, lips, etc., and the top layers learn the actual faces (see Figure 2.3).

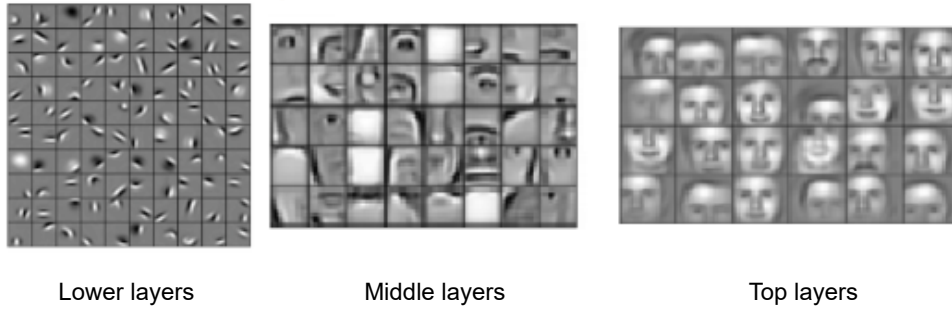


Figure 2.3: Hierarchical feature learning in DNN [26]

2.3.1 Training

Note: For explanation, only supervised learning is considered. This means that every example in the training set is a pair of input data and labelled output data. The objective is to find a function that approximately maps the input to the output and is generalizable enough to map unseen data to the correct output label.

Consider a training set with input vectors, x_n and output vector, y_n where $n = 1, 2, \dots, N$. The goal is to train the function, $f(x; \Theta) = y$, where Θ is the learnable parameter. In most networks, Θ is trained to minimize the error between *predicted* y and *true* y . This loss function is decided according to the learning problem. For simplicity, let loss function, $J(\Theta)$, be *mean square error*, the squared difference between $f(x; \Theta)$ and y . Furthermore, the loss function is calculated as the average over the complete training set (Eq. 2.3).

$$J(\Theta) = \frac{1}{2N} \sum_{n=1}^N \|f(x_n; \Theta) - y_n\|^2 \quad (2.3)$$

The loss function is optimized using the gradient descent method [27]. It is an iterative optimization algorithm that updates the weights with the help of the gradient of $J(\Theta)$ with respect to Θ and tries to find the global minima of the loss function. The gradient provides two crucial pieces of information. One is the direction in which Θ is to be moved to find the global minima, and the other is how big of a step to take to reach that minimum. Gradient descent can be expressed as,

$$\Theta \leftarrow \Theta - \eta \frac{\partial J(\Theta)}{\partial \Theta} \quad (2.4)$$

The partial derivatives of the gradient are computed using backpropagation. Backpropagation is a computation method based on the chain rule of calculus and operates by passing values backwards through the network to compute how each weight affects the loss. Also, in Eq. 2.4, η represents the learning rate, which intuitively describes the step size to take in the steepest descent direction. If η is too small, the convergence of the network to a global minimum tends to take a long time, and if η is too large, the algorithm might jump over the best solution. Therefore, η is an important hyperparameter to consider while training a network.

Furthermore, DNN optimization has been an active research field. Stochastic gradient descent, Nesterov accelerated gradient, Adagrad, Adam, RMSprop are a few popular optimizers in use [28].

2.3.2 Inference

Inference is the process of inferring or predicting against unseen data while using the trained DNN model. Unlike training, inference does not include a backward pass to compute the error and update weights (see Figure 2.4).

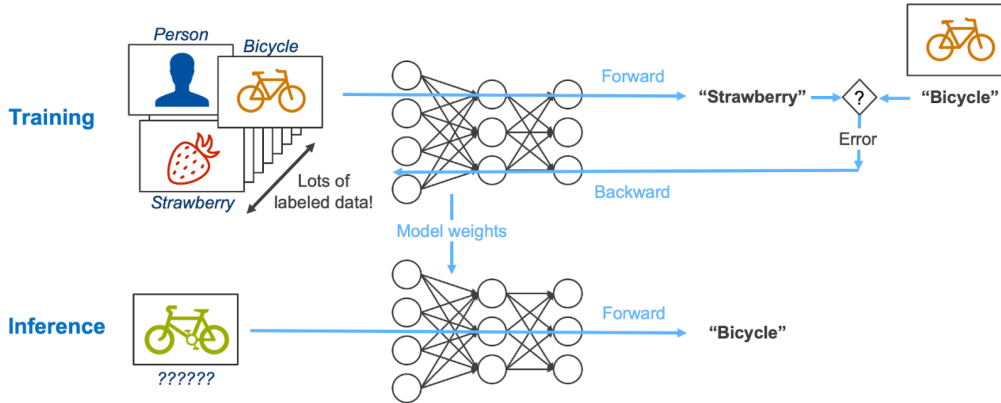


Figure 2.4: Training v/s inference [29]

2.3.3 Activation Functions

The activation function is an essential component of the DNN as it gives the network the capability to solve non-linear problems. In a linear activation or no activation, the network can only solve linearly separable problems, no matter how deep the network is. Apart from the non-linearity, activation functions must be continuously differentiable to compute useful gradients.

Traditionally, saturated activation functions, like *sigmoid* and *tanh* were mostly used. They were limited to $(0, 1)$ or $(-1, 1)$. This led to the vanishing gradient problem. For example, as shown in Figure 2.5, the maximum value of the sigmoid derivative is 0.25 (i.e. less than 1). Therefore, the gradients of the lower layer weights will involve the product of many numbers less than 1, leading to the gradient itself becoming very small (close to zero). This leads to the stagnation of learning and the delaying of convergence of the network.

In the recent past, new activations have replaced the saturated ones [31]. Two of them, *ReLU* and *Softmax*, which are used in this thesis, are explained below.

ReLU - Rectified Linear Unit ReLU, as shown in Figure 2.6, is arguably the most popular activation function in use for DNN. It can be expressed as,

$$\sigma(x) = \max(0, x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (2.5)$$

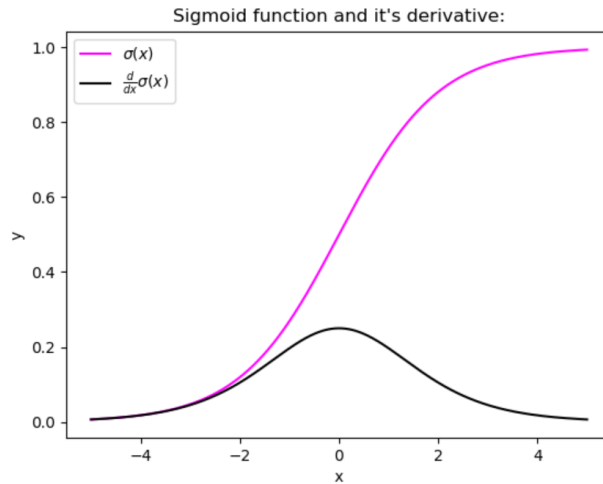


Figure 2.5: Sigmoid function and its derivative. Sigmoid ranges between 0 and 1. The maximum value of its derivative is 0.25. [30]

Eq. 2.5 indicates that if $x < 0$, the activation output is 0 otherwise, the output is x . An interesting property of the ReLU function is its derivative, which is either 0 (for negative x) or 1 (for positive x). This speeds up the backpropagation and solves the vanishing gradient problem because multiplication with one would not change the gradient. Also, [32] [33] has proved that using ReLU as the activation function in DNN leads to better results and faster convergence than using the saturated activation function.

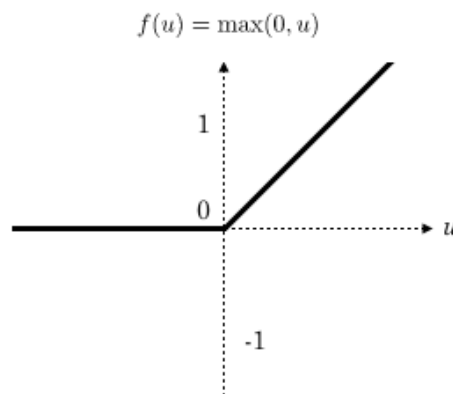


Figure 2.6: ReLU function [33]

Softmax The softmax function converts a real-valued vector with K elements into a real-valued vector that sums to 1. The input vector can have positive, negative or zero values, but the softmax turns it into a value between 0 and 1, so that the output can be interpreted as a probability. These probabilities can be considered as the network's "confidence score" indicating how sure it is about the input. If the softmax input is small or negative, it translates to low probability, and if the input is large, it turns into high probability (see Figure 2.7).

Softmax is also called multi-class logistic regression as it can represent the probability distribution of input over K classes. Therefore, it is mainly used in the last layer of the network [34].

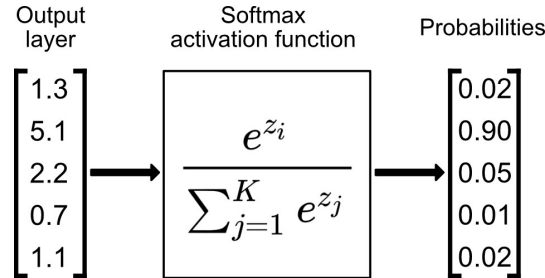


Figure 2.7: Softmax function - largest value in the output layer has the largest probability [35]

2.3.4 Convolutional Neural Networks

Convolutional neural networks (CNN) are a subset of DNN that is popularly used to process visual imagery-based applications (although not limited to it). These applications include image classification [36], image segmentation [37], pose estimation [38], face recognition [39], etc. CNN mainly consists of a convolutional layer, pooling layer, and fully connected layer [40].

2.3.4.1 Convolution Layer

The convolution operation is the fundamental building block of a CNN. The kernels or filters are the learnable parameters of this layer. Filters are composed of synapses or weights. Kernels are spatially small compared to their inputs but extends depth-wise through the input volume. Typically used kernel sizes are 3x3, 5x5, and 7x7. The third dimension of the kernel corresponds to the number of input channels.

During the forward pass, the entries of each filter are convoluted with the input volume. This involves computing the dot products between the kernel and the input at any position, as shown in Figure 2.8. Convolution is typically succeeded by a non-linear activation function (e.g. ReLU). The resulting outputs are called *activation maps* or *feature maps*. Then, these activation maps are stacked depth-wise to produce output volume. The output size of these maps depends on three hyperparameters: *padding*, *stride* and *depth*.

- The convolution operation on an input matrix reduces the output size, which leads to information loss. To avoid that, the input volume is padded with zeros around the border. The two most common choices for padding are *valid* and *same*. The valid convolution means no padding, and the same convolution means that the output size and input size are the same.
- The stride indicates the number of pixels that the filter is slid on the input image/feature map. Larger strides reduce the size of output volumes spatially.
- The depth represents the number of filters that are used in the convolution operation. Each filter is learning a different aspect in the input like edges, blobs, colours, etc.

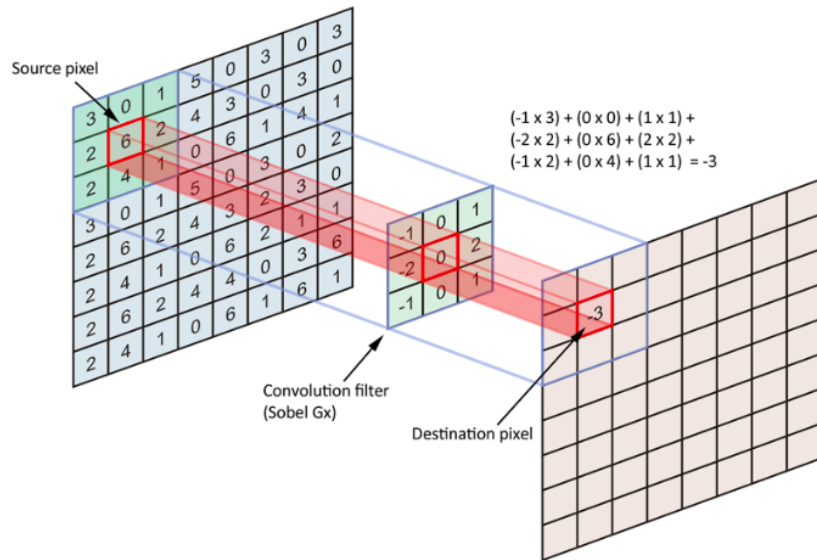


Figure 2.8: Demonstration of convolution operation [41]

2.3.4.2 Pooling Layer

The pooling layer is often placed after the convolution layer, and its function is to reduce the spatial dimension of the incoming matrix. The process is also referred to as sub-sampling or down-sampling. The main parameters of the pooling layer are the stride and kernel size. Two common types of pooling are average pooling and maximum pooling, where the average and maximum value within a kernel is chosen to be propagated to further layers, respectively (see Figure 2.9). Max pooling is usually preferred over average pooling. The intuition for this preference is that the maximum or large number indicates that an important feature has been detected.

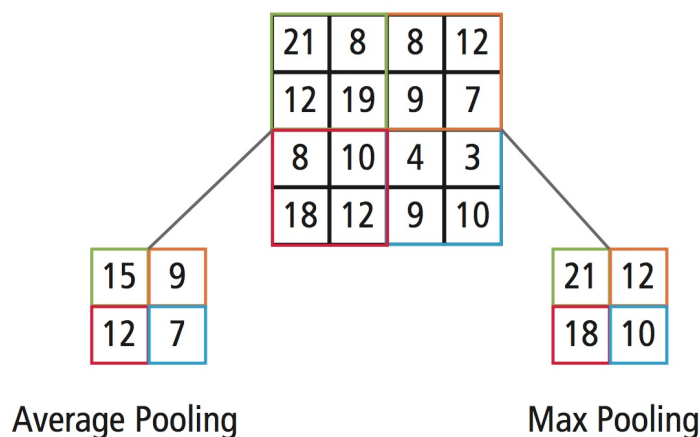


Figure 2.9: Average and Maximum pooling [42]

2.3.4.3 Fully Connected Layer

The fully connected layers are multi-layer perceptron (generally a two or three-layer MLP) that aims to cast the activation maps generated by the feature extractor into a class probability distribution, usually with the help of softmax function.

Note: There are other layers typically used in a CNN, like batch normalization layer and dropout layer. Interested reader can refer [43] [44].

2.4 Energy Consumption in DNN

Typically, a DNN accelerator has the following components [45]:

- An off-chip memory or DRAM to store the feature maps and weights of the DNN network. This component can contain several GBs of data. Using DRAM is an economic strategy to extend memory hierarchy as they are cost-effective.
- An on-chip global buffer (GLB), which holds the inputs and weights before it is fed into all the processing elements (PEs). The energy cost of accessing GLB is two orders of magnitude lower than that of the off-chip memory [46].
- A collection of processing elements (PEs), each with its own ALU, to simultaneously perform multiply-and-accumulate (MAC) operations. The PEs also contain register files (RFs) to store parameters locally, and these registers have lower energy costs than that of the on-chip buffer.
- A Network-on-Chip (NoC) that connects the PEs with each other and to the GLB. The NoC can assume various configurations depending on how the operations are spatially distributed and temporally scheduled on the PEs.

The computations carried out in the DNNs are mostly MACs, but typically they are performed on a large set of weights and inputs. Therefore, the bottleneck is not caused by the arithmetic computations but by the memory accesses. Every MAC requires three reads from memory, i.e. RF, SRAM or DRAM (input pixel, weight and partial sum) and one write (updated partial sum). As shown in Figure 2.10, memory access from DRAM has an energy cost ~ 2 orders of magnitude higher than a MAC operation [47]. Therefore, to achieve energy-efficient DNN processing without compromising performance: (a) the hardware design should develop data-flows that support parallel processing with minimal data movement, and (b) algorithm design should strive to decrease the computations and memory accesses significantly.

2.5 Energy Efficient Solutions

2.5.1 Quantization

As shown in Figure 2.11, energy cost can be reduced by lowering the precision of arithmetic operations, making it computationally efficient and compressing the size of DNNs, decreasing the memory accesses. One of the methods to accomplish this is quantization. It refers to the method of mapping values from a higher bit precision to a lower precision format.

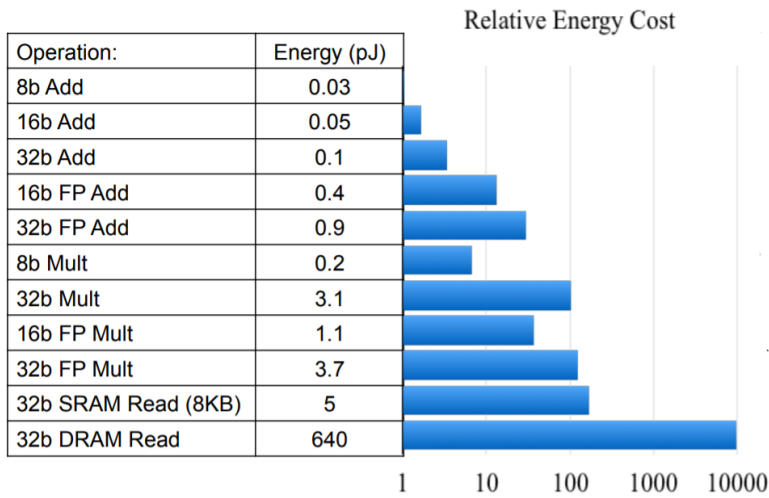


Figure 2.10: Energy consumption of multiply-accumulations in 45nm 0.9V [48]

The quantization technique can be broadly categorised into *post-training quantization*, where a full precision model is quantized after training, and *quantization aware training*, where quantization is a part of training, with the effect of it taken into consideration within the main training loop.

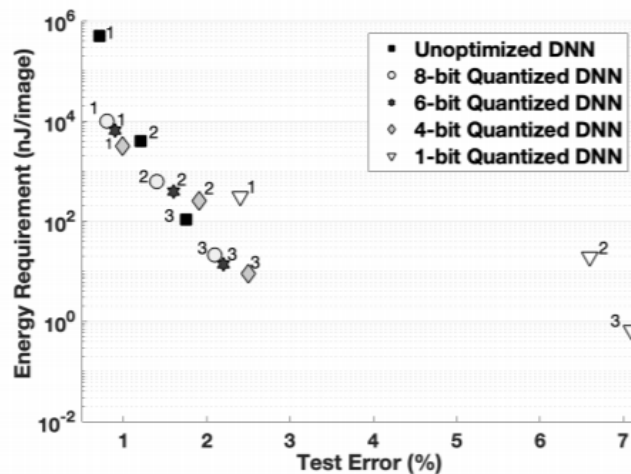


Figure 2.11: Pareto analysis accessing the effect of quantization on DNNs classifying MNIST dataset [49]

Post-training quantization schemes are relatively simple in terms of implementation. For example, ACIQ [50] calculates the best clipping value by analysing the weight distribution, which is typically Laplacian or Gaussian. Nvidia’s TensorRT [51] measures the relative entropy between quantized and full-precision weights using KL divergence to find the optimum clipping value. However, these methods fail to reach baseline accuracy because of the lack of re-training.

Another set of approaches involve quantization aware training where the objective is to

minimize the quantization error. For instance, BNN [52] quantizes the weights and activations of the network to binary values of -1 and +1. XNOR-net [8] extends this idea by quantizing them into binary converting a convolution operation to an XNOR shift operation. Later, WRPN [53] and DoReFa [54] used multiple bits to represent weights and activations unlike bipolar quantization. PACT [55] suggest learning the clipping value during training itself for each layer. Techniques, so, far, use uniform quantization step size. However, QIL [56] and APoT [57] propose that non-uniform quantization during training provides better accuracy.

Furthermore, there have been improvements aiming for *mixed-precision quantization* where each network layer has a different bitwidth. HAQ [58] and ReLeQ [59] use reinforcement learning to learn different bit precision for different layers. Another work, HAWQ [60], uses the second order hessian to decide which layers to quantize and how much to quantize.

Advantages

- Quantization helps reduce the circuit area, memory footprint, memory access, and energy requirement.
- Active research on non-uniform and mixed-precision quantization is helping in achieving lower bit precision while maintaining accuracy.

Limitations

- Most of the quantization techniques mentioned above, which results in at par baseline accuracy, works under the condition that the first and last layer remains unquantized (or quantized to a higher precision). It remains an unsolved problem.
- In most cases, special hardware is required to resourcefully tap into the benefits of quantization, like dynamic precision architectures.

2.5.2 Pruning

Removing or pruning the unimportant connections and parameters from DNNs, as shown in Figure 2.12, can reduce the computations and the number of memory accesses. Research in pruning mainly focuses on two objectives: identifying the least promising neurons or parameters that can be pruned and training / fine-tuning the pruned model to recover the baseline accuracy [61].

Earlier works in this field centred around data-agnostic criteria to identify the unimportant weights. [62] used second order Hessian matrix to analyse the weights to be pruned without compromising model accuracy. As an extension, [63] suggested that pruning harmless weights iteratively could be more beneficial than pruning them all at once. [64] considered the l_1 -norm of filters as the pruning criteria. [65] used the scaling factor of the batch normalization layer as the threshold. Apart from the data-agnostic methods, another approach is to exploit the training data to measure saliency, which can be termed data-aware pruning. [66] proposed to activation entropy to identify the channels to be pruned. [67] argued the use of gradient magnitude to prune the model during training. [10] performs pruning, quantization, and encoding to reduce the model size by 95%.

Advantages

- Pruning leverages the redundant information carried in a DNN model thus reducing model size.
- Pruning methods are typically model agnostic and can be applied to any given network with minor modifications.

Limitations

- A typical pruning algorithm requires iterative model training, significantly increasing the time required to build these models. [49] surveys that using quantization and pruning together can increase the training time up to 600%.
- Taking advantage of pruning requires custom hardware or special compression techniques to represent sparse matrices [18].

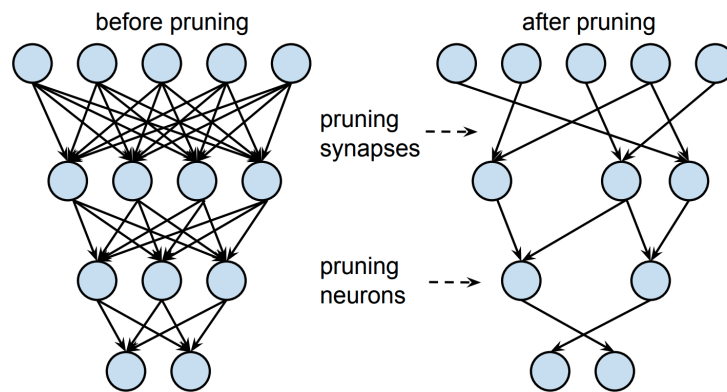


Figure 2.12: Neurons and synapses pruning [68]

2.5.3 Neural Architecture Search

Manually finding the best network or architecture for a particular application is difficult with many architectural possibilities. Network Architecture Search (NAS) is a method that automates the DNN architecture design without manual efforts. NAS suggests candidate architectures from the search space, which are then trained and validated. The validation accuracy is used for reinforcement learning to propose the next candidate architecture (see Figure 2.13). NASNet [69], AmoebaNet [70] and MNasNet [71] are examples of NAS which obtain state-of-the-art accuracy. MNasNet especially uses a multi-objective reward mechanism to search for a DNN architecture that matches both latency and accuracy requirements. NAS algorithms are extremely time and resource consuming. A more recent algorithm called Single-Path-NAS [72] reduces the architecture search time on ImageNet but comes at the cost of lesser accuracy.

Advantages

- NAS systematically finds the balance between accuracy, latency, and memory by searching through the solution space of all the candidate architectures without any human intervention.

Limitations

- The computational and resource demands of most of the NAS algorithms are prohibitively intensive, making it difficult to find trained and tested architectures for large datasets.

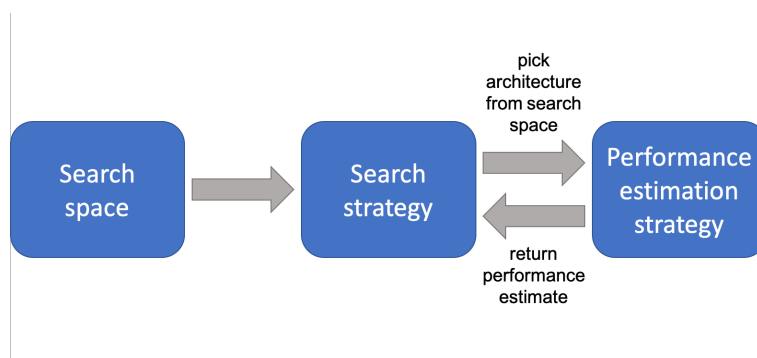


Figure 2.13: Neural Architecture Search [73]

2.5.4 Knowledge Distillation

Typically, larger, more complex DNN models which are computationally intensive leads to better performances. To leverage this, [74] proposed a model "compression" method where a pre-trained larger complex network is used to train a smaller DNN model. Larger networks have more capacity, thus enabling them to learn complex training data features, which are difficult for a smaller network to learn from scratch. In this method, the larger network is called the *teacher* and the smaller network is called the *student*.

Knowledge is transferred to the student model by learning the class distribution of the softmax layer outputs (soft labels) from the teacher model. To tap better into the 'dark knowledge' available in trained teacher models, [12] introduced the softmax temperature (T). As T increases, the target probabilities of the teacher and student are softened, providing more information as to which classes the teacher found more similar to the predicted class (see Figure 2.14). A recent work [75] also discusses the idea of distilling knowledge from feature maps of the teacher model directly.

Advantages

- It is computationally efficient to use the distilled smaller network for inference.
- Training is faster.
- It can be combined with other energy-efficient methods like quantization [76] [77] and neural architecture search [78].

Limitations

- There is no method to quantify the quality of the knowledge obtained from the teacher, which makes the generalisability of the approach questionable.
- Highly dependent on the structure of teacher and student.

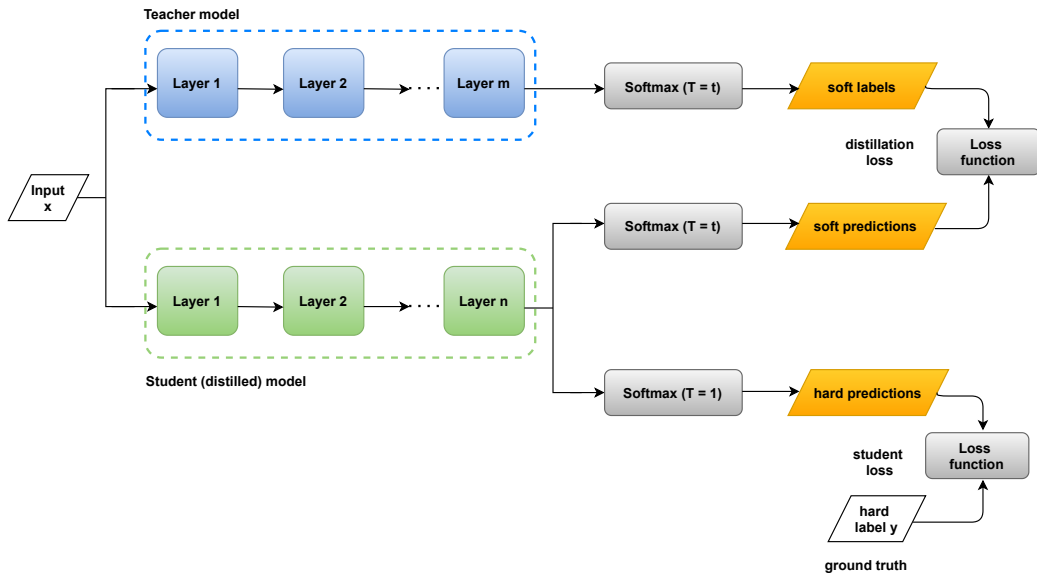


Figure 2.14: Knowledge distillation

2.6 Conclusion

The chapter discussed the basics of DNN architecture including its most commonly used layers. Then, the main causes of energy consumption in DNN processing were explained. Finally, existing techniques from the recent literature were discussed along with their advantages and disadvantages, which pursues the concept of energy efficient DNNs.

Firstly, this chapter discusses the importance of sparsity in the biological brain and how it contributes to making the brain an energy-efficient system. Then, it is explained how sparsity translates to less energy consumption in the case of a DNN. Finally, recent works dealing with both weight and activation sparsity are surveyed.

3.1 Sparsity in the Biological Neural Network

Although DNNs, as mentioned in Section 2.3, are bio-inspired, they have not been able to find the balance between power consumption and accuracy yet, especially while dealing with computationally heavy streaming signals. On the other hand, the brain's neocortex handles complex tasks like sensory perception, planning, attention, and motor control while consuming less than 20 W [79]. Therefore, the brain's functioning can be reverse engineered to bridge the energy consumption gap between the brain and DNNs.

Scalable architecture, in-memory computation, parallel processing, communication using spikes, low precision computation, sparse distributed representation, asynchronous execution, and fault tolerance are some of the characteristics of the biological neural networks that can be leveraged [80]. Among these, the thesis focuses on the viability of using sparsity within DNNs to achieve energy efficiency. *A sparse vector indicates that most of its elements are zero.* The human brain is considered a sparse model because approximately out of the 86 billion neurons available, only 1% to 10% of neurons are active at any given time [81] [82].

Specifically, while processing streaming signals, neurons detect the changes in their input and let it accumulate past some threshold, upon which they send a discrete “spike” signal notifying the downstream neurons of this detected change. Therefore, when extracting features from temporally correlated signals like videos, the biological brain opts for a change based (*or event-based*) processing approach rather than a frame-based one. This spatio-temporal sparse representation within the neurons also contributes to making the system noise resilient and robust.

3.2 Sparsity and Energy Consumption

Sparsity in weights or activations of DNNs is a feature that can be used to achieve energy efficiency. During a matrix-vector multiplication between a weight matrix and an activation vector, zero elements in the tensor can be skipped as they do not contribute to the final result. Therefore, the multiplications between zero-valued activations and their corresponding weight columns can be skipped, leading to computational as well as memory access reduction (see Figure 3.1). Moreover, it is also possible to skip multiplications between zero-valued weight elements and non-zero input activation elements as well to reduce energy cost, though this aspect has not been considered in this work. [20] approximates the theoretical computation and memory access reduction (in delta based networks) to be the inverse of activation density,

which is the ratio of non-zero elements to all elements in an activation vector (Eq. 3.1 and 3.2).

$$\text{Computation reduction} = \frac{\text{Cost}_{\text{comp,dense}}}{\text{Cost}_{\text{comp,sparse}}} \approx \frac{1}{\text{activation density}} \quad (3.1)$$

$$\text{Memory access reduction} = \frac{\text{Cost}_{\text{mem,dense}}}{\text{Cost}_{\text{mem,sparse}}} \approx \frac{1}{\text{activation density}} \quad (3.2)$$

Although sparsity has the potential to decrease energy consumption, there are two main pitfalls to consider. (a) If the sparsity is non-structured, it might not contribute to an energy-efficient system due to the overhead of irregular execution paths and memory access, (b) the non-structured sparsity needs to be significantly high to see a realistic decrease in energy cost [83].

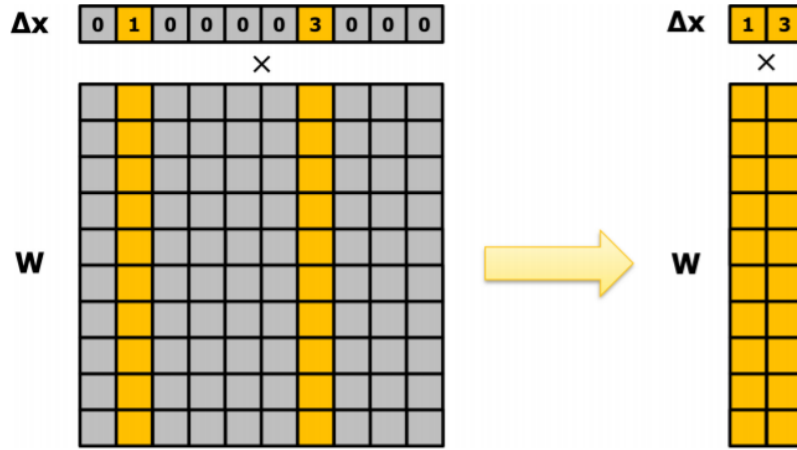


Figure 3.1: Sparsity in Δx can save multiplications between Δx and columns of W that correspond to zero [20]

3.3 Sparsity - Related Works

There are broadly two types of sparsity available in DNNs: weight sparsity (related to the interconnect between neurons) and activation sparsity (related to the number of neurons). Furthermore, activation sparsity can be categorised into spatial and temporal sparsity, which exploits the spatial and temporal correlation within the activations, respectively, [16] as shown in Figure 3.2.

Weight sparsity: The concept of weight sparsity in the biological brain indicates that if a group or layer of cells projects to another layer, only 1% - 5% of the possible neuron to neuron connections exist [84]. [85] takes this idea and hypothesises that every DNN contains sub-networks (called winning tickets) that, when trained in isolation can reach test accuracy comparable to the original network.

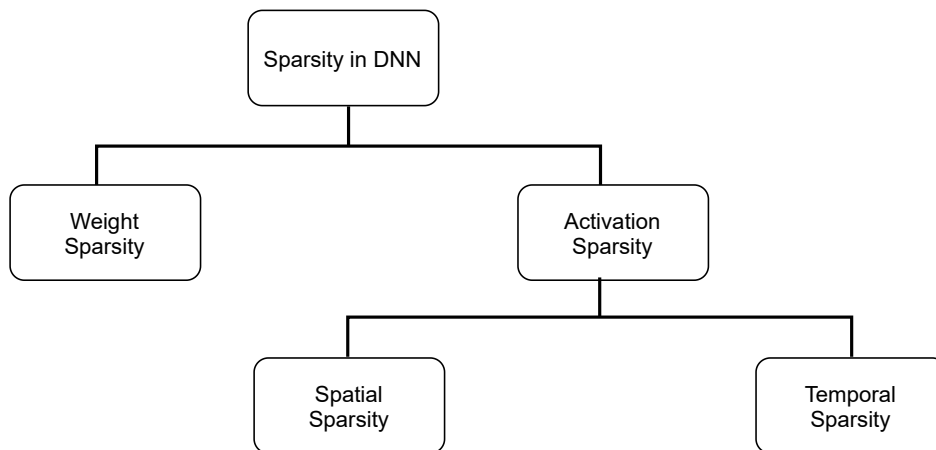


Figure 3.2: Types of sparsity available in DNNs

In this effort, inspired by the Hoyer measure [86] used in traditional compressed sensing problems (see Figure 3.3), [87] present DeepHoyer, a set of sparsity-inducing weight regularizers that leads to sparser solutions than traditional regularization methods like LASSO and Ridge. [88] uses Taylor score to remove more than 98% of weights and quantizes the remaining weights into powers of two to enable the use of shifters instead of multipliers on the hardware level, dramatically reducing hardware complexity. Similarly, [89] introduce a novel penalty called Hierarchical Adaptive Lasso (HALO), which adaptively learns to sparsify over-parameterized networks through adaptive shrinkage. Furthermore, [90] proposes Soft Threshold Reparameterization (STR), which in turn obtains a non-uniform sparsity budget across layers. Whereas, [91] suggests an energy-aware pruning algorithm for DNNs that directly uses the energy consumption of a DNN to guide the sparsification process.

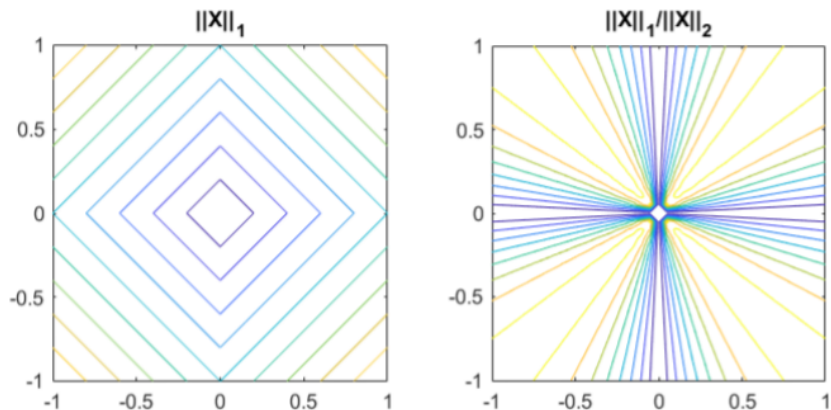


Figure 3.3: Contours of l_1 norm and the Hoyer (the ratio between l_1 and l_2 norms) regulariser of a 2D vector [10]

Spatial sparsity: As a natural consequence of ReLU function (Section 2.3.3), a non-trivial portion of the activations in a DNN are already sparse, as shown in Figure 3.4. [92] observes

that AlexNet exhibits an average of 49.4% activation sparsity across the entire network. To further this, [93] explored l_1 regularization to induce sparsity in the activation maps, showing that sparsity can be increased by up to 60% for image classification models. [94] provides an activation density (or the number of non-zero activations) driven pruning approach that provides a heuristic way of obtaining the optimal number of neurons for each layer of a network during training. [95] proves that spatial sparsity can be significantly increased via Hoyer regularization and thresholding, with acceptable accuracy loss, beyond l_1 regularization. Similarly, to take advantage of the spatial correlation in activation maps, [96] introduces differential convolution, which uses the differences (or the deltas) between the columns of activation maps rather than on their absolute values. This also reduces the effectual bits, directly reducing the DRAM memory access. [97] explores a dynamic winners-take-all (WTA) dropout mechanism where only those activations with high magnitude are allowed to propagate to the following layers, thus inducing structured spatial sparsity.

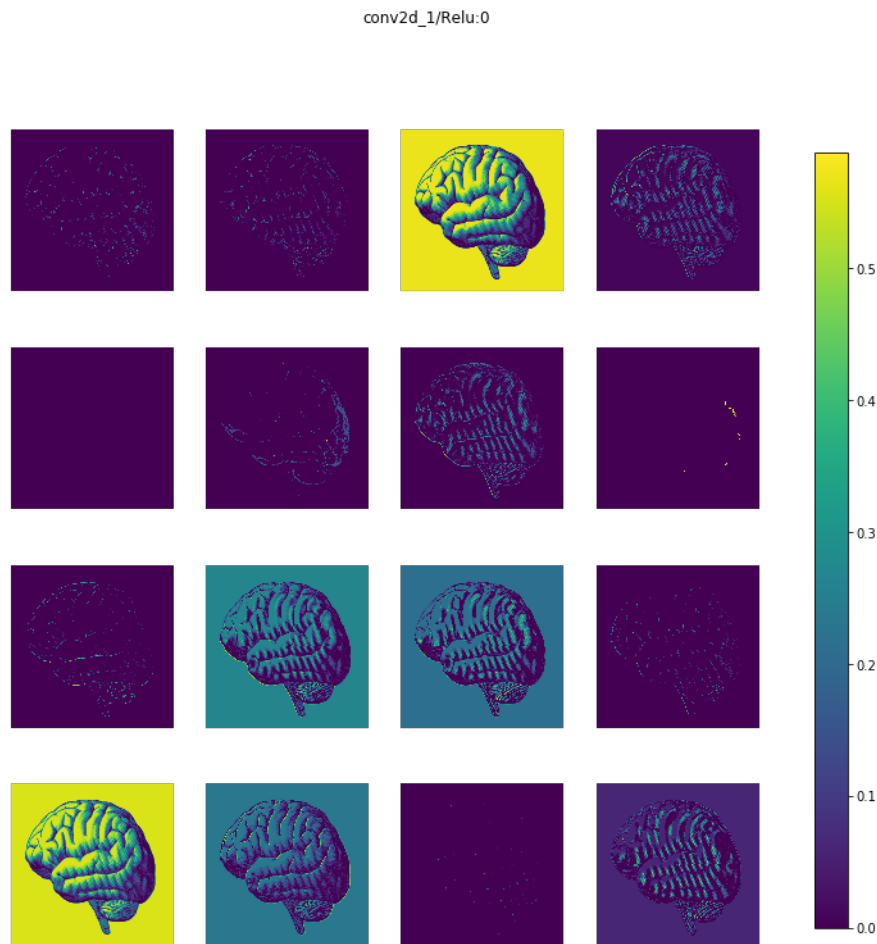


Figure 3.4: Natural spatial sparsity within the activation maps due to ReLU [98]

Temporal sparsity: Unlike weight and spatial sparsity, exploiting the temporal redundancy of DNNs while processing streaming data as a means to reduce energy consumption is a relatively less explored idea. Exploiting temporal sparsity translates to skipping re-

calculation of a function when its input remains unchanged since the last update (see Figure 3.5).



Figure 3.5: A sample video sequence demonstrating the fact that moving objects make up only a small part of the overall scene. [99]

Few methods exploited the compressed representation (like H.264, MPEG-4, etc.) of videos at the input stage itself as an approach. These compression techniques only retain a few key-frames completely and reconstruct others using motion vectors and residual error, thus using temporal redundancy. Considering this, [100] proposes using multiple CNNs that directly operate on the motion vectors and residuals, in addition to a small number of complete images. Similarly, [101] introduces activation motion compensation (AMC), which is based on video compression coding. AMC reduces the size of CNN layers enough for them to be stored on-chip, reducing the chances of off-chip memory access, which costs more energy.

Another path includes finding a neuron model which is somewhere in between “frame-based DNN” and “event-based spiking neural networks”. This thesis is an attempt in the aforementioned direction. A similar work, CBIInfer [99], proposes replacing all spatial convolution layers in a network with change-based temporal convolution layers (or CBconv layers). In this, a signal change is propagated forward only when a certain threshold is exceeded. Likewise, [102] tapped into temporal sparsity by introducing Sigma-Delta Networks, where neurons in one layer communicated with neurons in the next layer through discretized delta activations. An issue when it comes to CBIInfer [99] is the potential error accumulation over time as the method is threshold-based. If the neuron states are not reset periodically, this threshold can cause drift in the approximation of the activation signal and degrade the accuracy. Whereas, sigma-delta [102] scheme experiments on smaller datasets like temporal MNIST, which might not be a reliable confirmation of the method’s effectiveness.

One of the arguable downsides of change based processing is that temporally sparse computations need remembering the states from the last update and re-using them efficiently. Therefore, it requires more memory than frame-based processing. However, it can be countered with the fact that in the biological brain as well, there is an evident well-proved trade for less power consumption with the help of more memory [80]. Therefore, it can be argued that brain-inspired DNN inference has to be supported with new memory technologies as well.

3.4 Conclusion

This chapter, firstly, discussed how the biological neural networks uses features like sparse representation to maintain an energy conservative system. Then, it is explained how sparsity helps in reducing the energy consumption in DNNs. Finally, after establishing the merit of sparsity in DNNs, existing works spanning weight and activation sparsity are surveyed.

Part II
Proposed Methodology

Temporal Delta Layer

In this chapter, a general-purpose novel DNN layer is introduced, called temporal delta layer, which can increase the temporal activation sparsity of the model by exploiting the temporal redundancy within the network. This is achieved by applying delta operator and sparsity inducing penalty on layer activations during training. The chapter also explains how existing techniques like quantization helps the new layer in promoting temporal sparsity.

4.1 Temporal Redundancy in Feature Maps

In video-based applications, traditional deep neural networks rely on frame-based processing. That is, each frame is processed entirely through all the layers of the model. However, there is very little change in going from one frame to the next through time, which is called temporal locality. Therefore, it is wasteful to perform computations to extract the features of the non-changing parts of the individual frame. Taking that concept deeper into the network, if feature maps of two consecutive frames are inspected after every activation layer throughout the model, this temporal overlap can be observed (see Figure 4.1). Therefore, this work postulates that temporal sparsity can be significantly increased by focusing the inference of the model only on the changing pixels of the feature maps (or deltas). The obtained temporal sparsity reduces the energy cost of the inference by skipping the computations and memory accesses corresponding to the zero-valued delta activations.

4.2 Temporal Delta Layer

4.2.1 Delta Inference

As seen in Figure 4.1, there is untapped potential in processing only the changing pixels of temporally continuous feature maps and still not losing any (or very little) of the critical information in the scene. This work introduces a new layer that calculates the delta (or difference) between two temporally consecutive feature maps and quantifies the degree of these changes at only relevant locations in the frame. Since zero changes are not propagated through the layer, the role of this layer may be perceived as "analog event propagation". It is considered an "analog event" as it is not the presence of change, but the magnitude of change that is propagated through.

To better understand it mathematically, in a standard DNN layer, the output activation is related to its weights and input vector through Eq. 4.1 and 4.2.

$$Y_t = WX_t + B \quad (4.1)$$

$$Z_t = \sigma(Y_t) \quad (4.2)$$

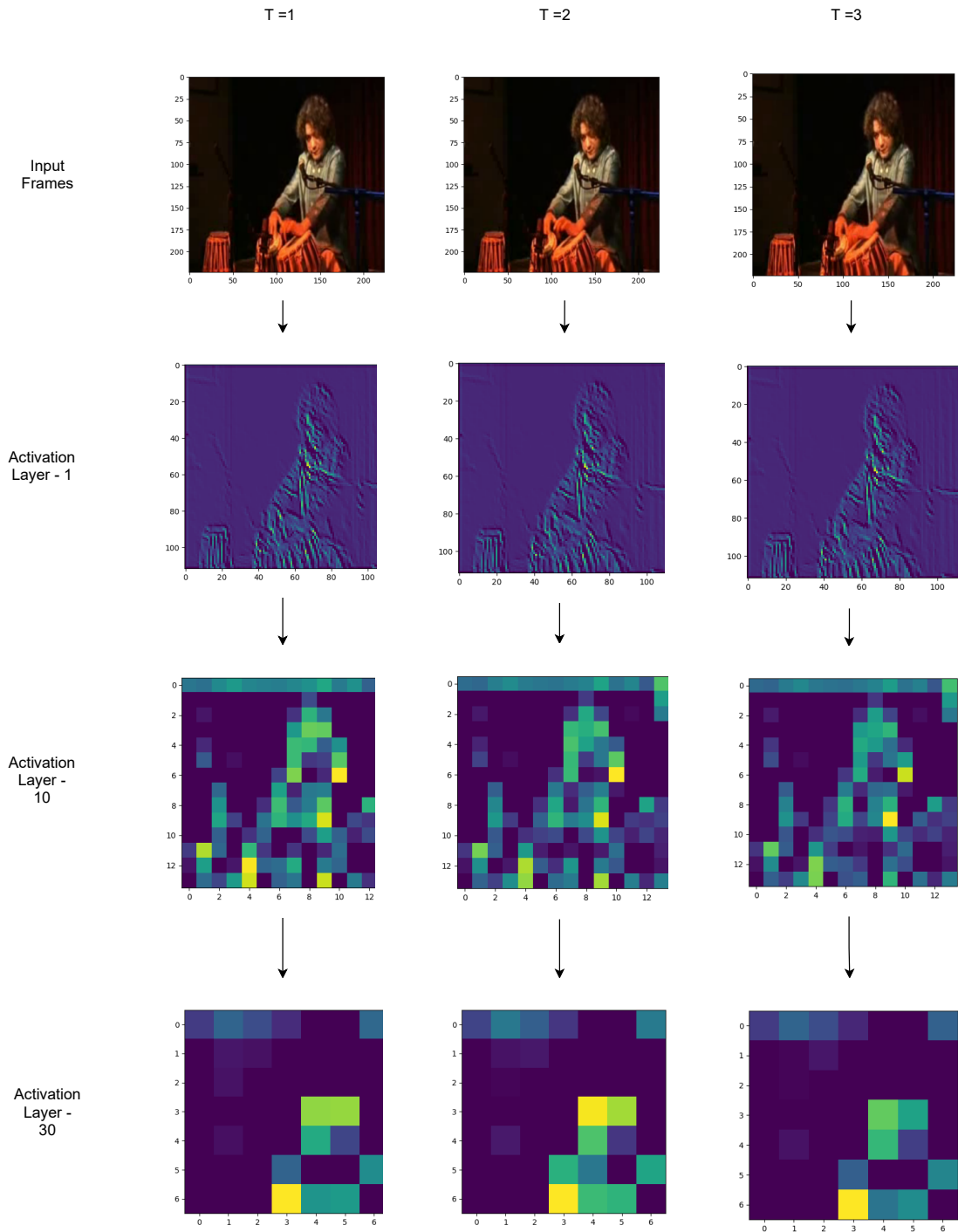


Figure 4.1: Evidence of temporal redundancy in video inputs and consequent feature maps.

where W and B represents the weights and bias parameters, X_t represents the input vector, and Y_t represents the transitional state. Then, Z_t is the output vector which is the result of $\sigma(\cdot)$ - a non-linear activation function. t indicates that the tensor has a temporal

dimension. However, in the temporal delta layer, weight-input multiplication transforms into,

$$\Delta Y_t = W \Delta X_t = W(X_t - X_{t-1}) \quad (4.3)$$

$$\begin{aligned} Y_t &= \Delta Y_t + Y_{t-1} \\ &= W(X_t - X_{t-1}) + W(X_{t-1} - X_{t-2}) + \dots + Y_0, \quad \text{where } Y_0 = B \\ &= WX_t + B, \end{aligned} \quad (4.4)$$

$$\Delta Z_t = Z_t - Z_{t-1} = \sigma(Y_t) - \sigma(Y_{t-1}), \quad \text{where } \sigma(Y_0) = 0 \quad (4.5)$$

In Eq. 4.3, instead of using X_t directly, only changes or ΔX_t are multiplied with W . Using the resulting ΔY_t , the corresponding Y_t can be recursively calculated with Eq. 4.4, where Y_{t-1} is the transitional state obtained from the previous calculation. Eq. 4.5 is the final delta activation output that is passed onto the next layer.

Another notable difference between the standard DNN layer and proposed layer is the role of bias. In delta based inference, bias is only used as an initialization for the transitional state, Y_0 in Eq. 4.4. However, since bias tensors do not change over time, their temporal difference is zero and is removed from Eq. 4.3.

Now, as the input video is considered temporally correlated, the expectation is that ΔX_t and by association ΔZ_t are also temporally sparse. In essence, the temporal sparsity between consecutive feature maps is cast on the spatial sparsity of the delta map that is propagated. Additionally, Y_t in Eq. 4.1 and 4.4 are always equal. This indicates that as long as the input is the same, both standard DNN and temporal delta layer based DNN provide the same result at any time step.

4.2.2 Activation Quantization to Induce Sparsity

Note: Quantization in the context of this thesis refers to activation quantization only. Weight quantization is not within the scope.

4.2.2.1 How Does Quantization Affect Temporal Sparsity?

The temporal redundancy in feature maps of two consecutive frames is explained in section 4.1. However, if looked closely, it can be observed that these feature maps are similar but not identical as shown in Figure 4.2a and 4.2b. Therefore, if two such consecutive feature maps are subtracted, the resulting delta map has many near zero values, thus restricting the potential increase in temporal sparsity (Figure 4.2c). This is mainly due to the higher precision available in the floating point representation (FP32) of the activations. A plausible solution to decrease the precision of the activations is to use quantization.

Quantization, as mentioned in Section 2.5.1, refers to the method of mapping values from higher precision values (FP32) to a lower precision format. Typical DNNs consider quantization as a means to induce spatial sparsity and to reduce memory footprint. The temporal delta layer proposes to quantize the input activations before delta operation as a means to induce temporal sparsity. The effect of quantization on temporal sparsity has been illustrated with an example in Figure 4.3. The figure explains that the difference between two similar vectors with higher precision tends to hover around zero but not converge to absolute

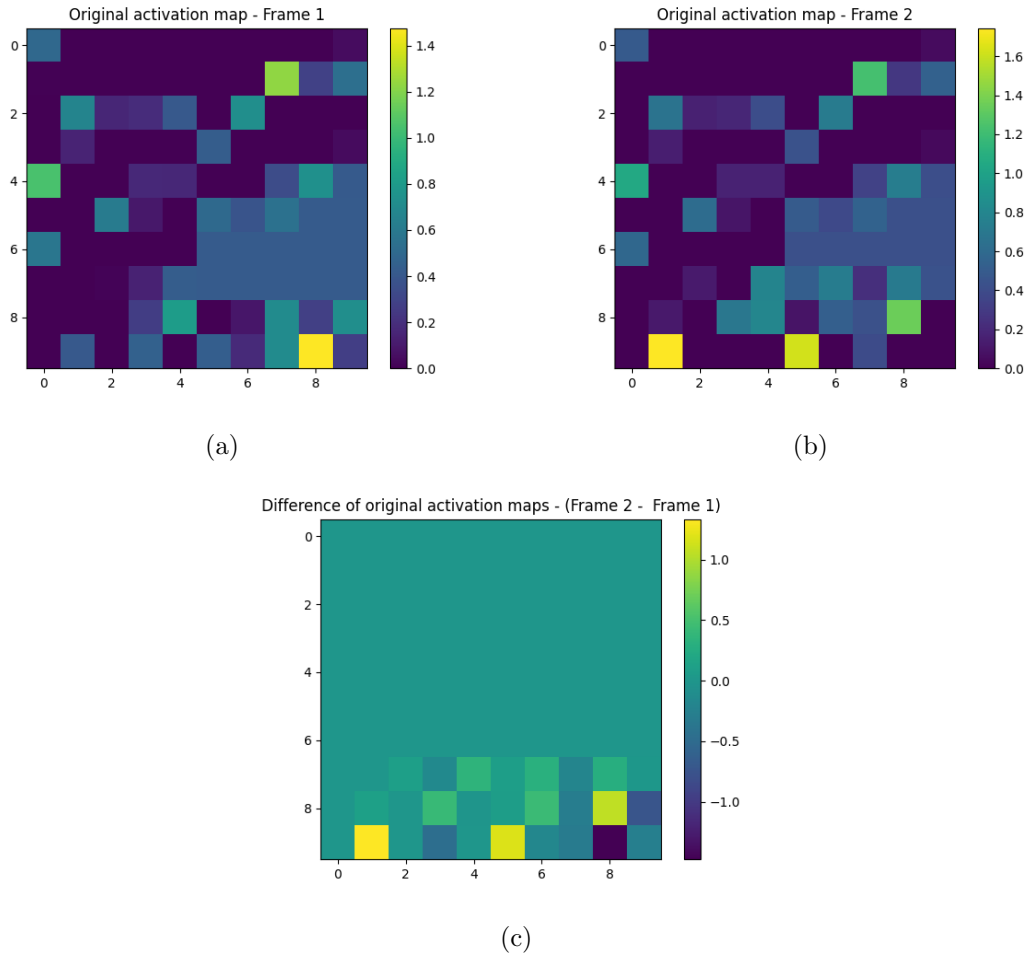


Figure 4.2: Demonstration of two consecutive activation maps leading to near zero deltas

zero in most cases. However, if these vectors are quantized, the probability of two similar nearby values being quantized to the same number increases, leading to their difference going to absolute zero.

In this work, a post-training quantization method (fixed point quantization [103]) and a quantization aware training method (learnable step size quantization [104]) are considered for comparison as a temporal sparsity facilitator for the new layer.

4.2.2.2 Fixed Point Quantization

In this method, the floating point numbers are quantized to integer or fixed point representation [103]. A floating point representation consists of three components: the sign, the significant and the exponent. For example, in IEEE 754 representation, a single-precision 32-bit floating point number has 1 bit for sign, 8 bits for the exponent and 23 bits for the significant. It leads to a very high dynamic range, roughly $[=10^{-38}, 10^{38}]$, far beyond the values usually seen in DNN inference, and increases the resolution or precision for numbers close to 0. The number nearest to 0 is about $\pm 1.4 \times 10^{-45}$. Therefore, due to high resolution, two similar floating point values have difficulty going to absolute zero when subtracted.

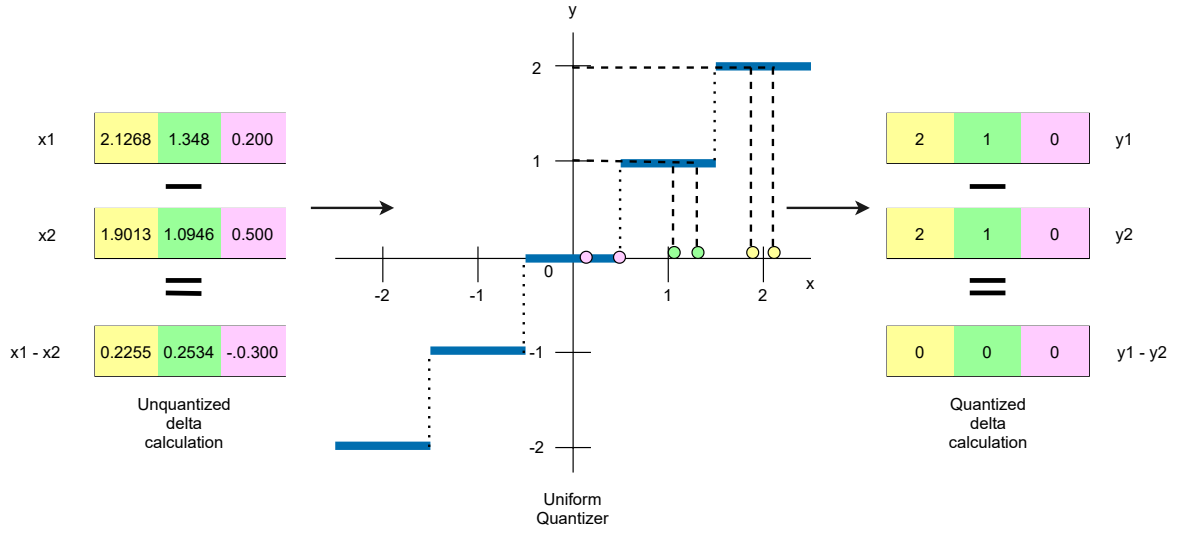


Figure 4.3: (left) Unquantized delta calculation of two similar vectors causing near zero values, (middle) a uniform quantizer, (right) quantized delta calculation leading to absolute zeroes.

On the other hand, in fixed point representation, the integer and the fractional part have fixed length. This limits both range and precision. That is, if more bits are used to represent the integer part, it subsequently decreases the precision and vice versa. For example, in a 32 bit fixed point representation with 16 bits for integer and 16 for fraction, the range is $[-32768, 32768]$, and the resolution is 1.52×10^{-5} .

Method: Firstly, a bitwidth is defined to which the 32-bit floating parameter is to be quantized, BW . Then, the number of bits required to represent the unsigned integer part of the parameter (x) is calculated as shown in Eq. 4.6.

$$I = 1 + \lfloor \log_2 (\max_{1 < i < N} |x|) \rfloor \quad (4.6)$$

A positive value of I means that I bits are required to represent the absolute value of the integer part, while a negative value of I means that the fractional part has I leading unused bits. Now, it is known that 1 bit is for sign, so the number of fractional bits, F , is given by Eq. 4.7.

$$F = BW - I - 1 \quad (4.7)$$

Considering the parameters, BW - bitwidth, F - fractional bits, I - integer bits, and S - sign bit, Eq. 4.8 maps the floating point parameter x to the fixed point by,

$$Q(x) = \frac{C(R(x \cdot 2^F), -t, t)}{2^F} \quad (4.8)$$

where $R(\cdot)$ is the round function, $C(x, a, b)$ is the clipping function, and t is defined as,

$$t = \begin{cases} 2^{BW-S}, & BW > 1 \\ 0 & BW \leq 1 \end{cases}$$

4.2.2.3 Demonstration of Fixed Point Quantization Influencing Temporal Sparsity

Consider frame 1 has an activation value $a_1 = 83.5625$ and frame 2 has an activation value $a_2 = 84.3750$. Without quantization, the delta calculation leads to a non zero value. They require $BW = 12$ bits and 11 bits, respectively, for their precise representation. However, if both values are mapped into lower precision fixed point representation with $BW = 4$ bits, their quantized value becomes 84 , and their difference is driven to zero (See Figure 4.4).

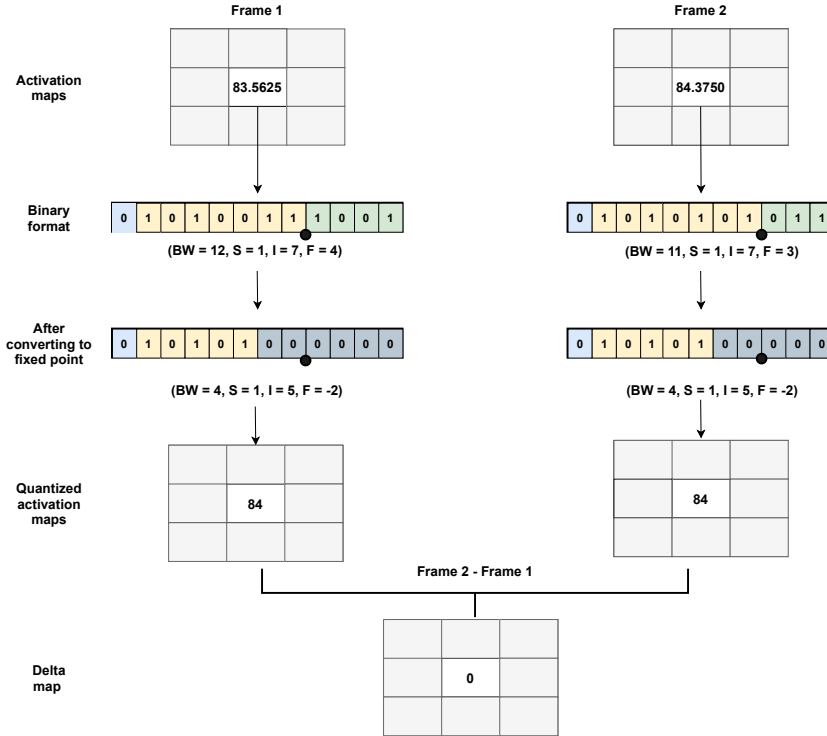


Figure 4.4: Demonstration of fixed point quantization inducing temporal sparsity. Bits stored in memory are shown in yellow and green. Sign bit is shown in blue. Unused bits are shown in gray.

Therefore, in this work, the hypothesis is that, mapping the activations from 32-bit floating point to lower bitwidth fixed point representation can potentially lead to significant temporal sparsity without compromising accuracy.

4.2.2.4 Possible Drawback of Fixed Point Quantization

Fixed point quantization, as shown in 4.2.2.2, is a fairly straightforward mapping scheme and is easy to be included in the model training process during the forward pass before the actual delta calculation. However, it poses a limitation to the extent of quantization possible without sacrificing accuracy. Typically, an 8-bit quantization can sustain floating point accuracy with this method, but if the bitwidth goes below 8 bits, the accuracy starts to deteriorate significantly. This is because, unlike (static) weights, activations are dynamic and activation patterns change from input to input making them more sensitive to harsh quantization [105].

Especially considering the objective of the thesis, where the quantization is proposed as a means to facilitate temporal sparsity while maintaining accuracy, quantizing all layers to a fixed bitwidth of ~ 8 is not an optimal solution. This is because quantizing the layers of a network to the same bitwidth can mean that the inter-channel behaviour of the feature maps are not captured properly. Since the number of fractional bits is usually selected depending on the maximum activation value in a layer, this type of quantization tends to cause excessive information loss in channels with a smaller range.

4.2.2.5 Learned Step-Size Quantization

Quantization aware training is the most logical solution to the drawback mentioned in 4.2.2.4 as it can potentially recover the accuracy in low bit tasks given enough time to train.

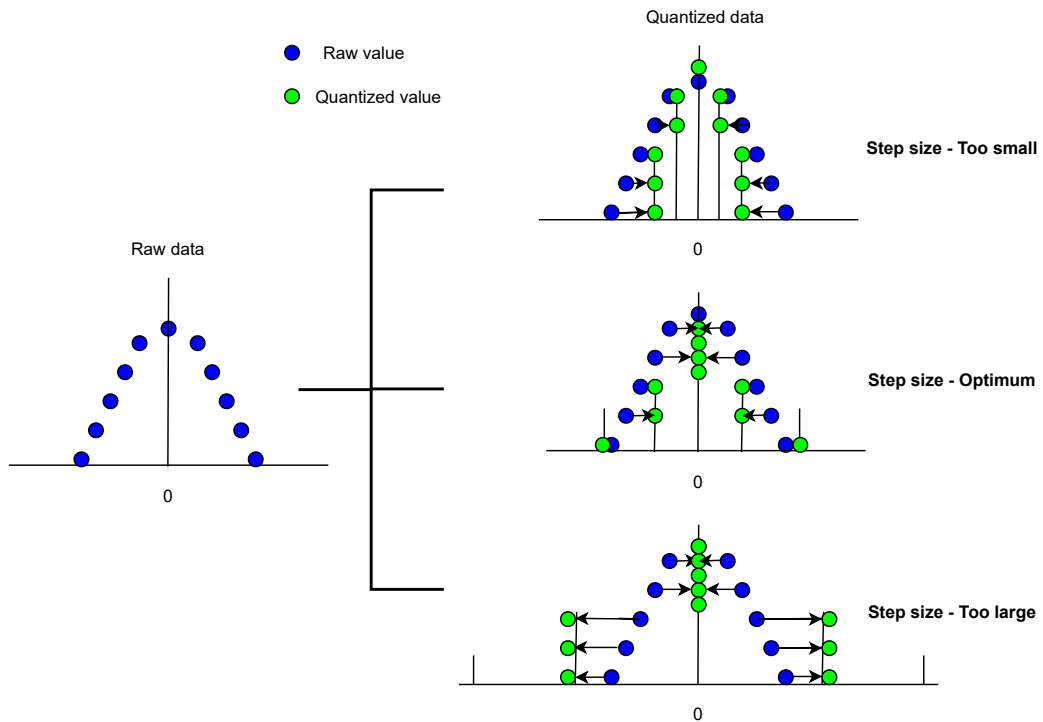


Figure 4.5: Importance of step size in quantization: on the right side, in all three cases, the data is quantized to five bins with different uniform step sizes, but without optimum step size value, the quantization can alter the range and resolution of the original data

Therefore, a symmetric uniform quantization scheme is considered called Learned Step size Quantization (LSQ) [104]. This method considers the quantizer itself as a trainable parameter which is trying to minimize the task loss using backpropagation and stochastic gradient descent. This serves two purposes: (a) step size, which is the width of quantization bins, gets to be adaptive through the training according to the activation distribution. It is vital to find an optimum step size because, as shown in Figure 4.5, if the step size is too small or too large, it can lead to the quantized data being a poor representation of the raw data. (b) as the step size is a model parameter, it is also directly seeking to improve the metric of interest, i.e. accuracy.

Method Given: x - the parameter to be quantized, s - step size, Q_N and Q_P - number of negative and positive quantization levels respectively, \bar{x} - integer scaled and quantized representation of x , \hat{x} - quantized representation with the same scale as x ,

$$\bar{x} = \lfloor \text{clip}(x/s, -Q_N, Q_P) \rfloor \quad (4.9)$$

$$\hat{x} = \bar{x}.s \quad (4.10)$$

where, $\text{clip}(a, r_1, r_2)$ clips any value of a below r_1 to r_1 and above r_2 to r_2 , and, $\lfloor a \rfloor$ rounds the value to the nearest integer. Considering the number of bits, b , to which the data is to be quantized, $Q_N = 0$ for unsigned and $Q_N = 2^{b-1}$ for signed data. Similarly, $Q_P = 2^{b-1}$ for unsigned and $2^{b-1} - 1$ for signed data.

Eq. 4.9 and 4.10 can be re-written as,

$$q(x; s) = \begin{cases} \lfloor \frac{x}{s} \rfloor .s, & \text{if } -Q_N \leq \frac{x}{s} \leq Q_P \\ -Q_N.s, & \text{if } \frac{x}{s} \leq -Q_N \\ Q_P.s, & \text{if } \frac{x}{s} \geq Q_P \end{cases} \quad (4.11)$$

Step size gradient: LSQ provides an opportunity to train the step size, s , by including the gradient of Eq. 4.11 with respect to s in the training loss during back-propagation.

$$\nabla_s q(x; s) = \begin{cases} \lfloor \frac{x}{s} \rfloor - \frac{x}{s}, & \text{if } -Q_N \leq \frac{x}{s} \leq Q_P \\ -Q_N, & \text{if } \frac{x}{s} \leq -Q_N \\ Q_P, & \text{if } \frac{x}{s} \geq Q_P \end{cases} \quad (4.12)$$

The gradient in Eq. 4.12 is derived by considering the differentiation of the *round* function as a Straight Through Estimator (STE). Intuitively, if step size, s , is small compared to the activation x , the width of the quantization bin is more, and thus, the number of quantization levels is less. Similarly, if s is large compared to x , the quantization bin width is small, and thus, the number of quantization levels is more.

4.2.2.6 Modified LSQ

In this work, the LSQ method is slightly modified to remove the clipping function from the equations as (a) the bitwidth, b , required to calculate Q_N and Q_P is not known. This is because the bitwidth is not pre-defined and is determined using the activation statistics of each layer while training which leads to a mixed precision model, which is more advantageous, and (b) clipping leads to accuracy drop as it alters the range of the activation. That is, if activations are clipped during training, there could be a significant difference between the real-valued activation value and the quantized activation value, which in turn affects the gradient calculations and, therefore the SGD optimization.

Thus, in temporal delta layer, the forward pass of the quantization includes only scaling, rounding and de-scaling as shown in Figure 4.6 and can be mathematically expressed as,

$$q(x; s) = \lfloor \frac{x}{s} \rfloor .s \quad (4.13)$$

The gradient of the Eq. 4.13 for backpropagation is given by Eq. 4.14.

$$\nabla_s q(x; s) = \lfloor \frac{x}{s} \rfloor - \frac{x}{s} \quad (4.14)$$

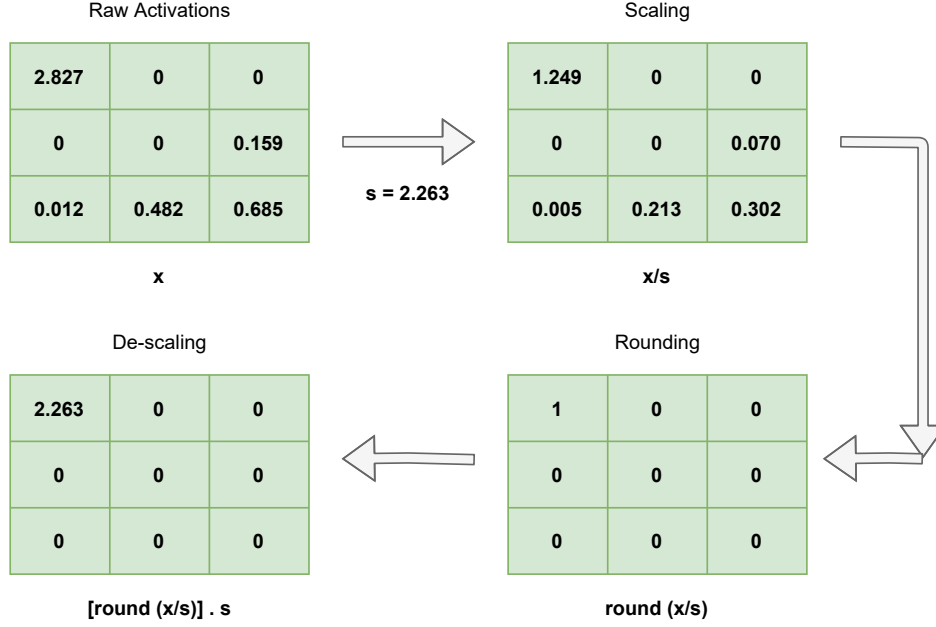


Figure 4.6: Modified LSQ - forward pass including scaling, rounding and de-scaling. Quantization step-size considered is 2.263

4.2.3 Sparsity Penalty

Quantized delta maps, created using the above-mentioned method, in itself has a fair number of absolute zeroes (or sparsity) available. However, like the biological brain, learning can help in increasing this sparsity further. The inspiration for this came from an elegant set of experiments performed by Y. Yu et al. [106]. The experiment showed a particular 30 seconds video to rodent specimens and tracked their activation density during each presentation. As shown in Figure 4.7, activation density decreased as the number of trials increased, i.e as the learning increased, the active neurons required for inference decreases.

Adapting the said concept to this work, the model (including temporal delta layer) is fine-tuned by training over epochs to minimize the total number of non-zero activations in delta maps while maintaining the model accuracy. Decreasing the density of activations inadvertently leads to more sparsity. For this purpose, a l_1 norm based constraint is introduced to the loss function. This is termed as the *sparsity penalty*. Therefore, the new cost function can be mathematically expressed as $cost\ function = task\ loss + sparsity\ penalty$, i.e,

$$Cost\ function = Task\ loss + \lambda \left(\frac{l_1\ norm\ of\ active\ neurons\ in\ delta\ map}{total\ number\ of\ neurons\ in\ delta\ map} \right) \quad (4.15)$$

where task loss minimizes the error between the true value and the predicted value and, sparsity penalty minimizes the overall temporal activation density. The λ mentioned in Eq.

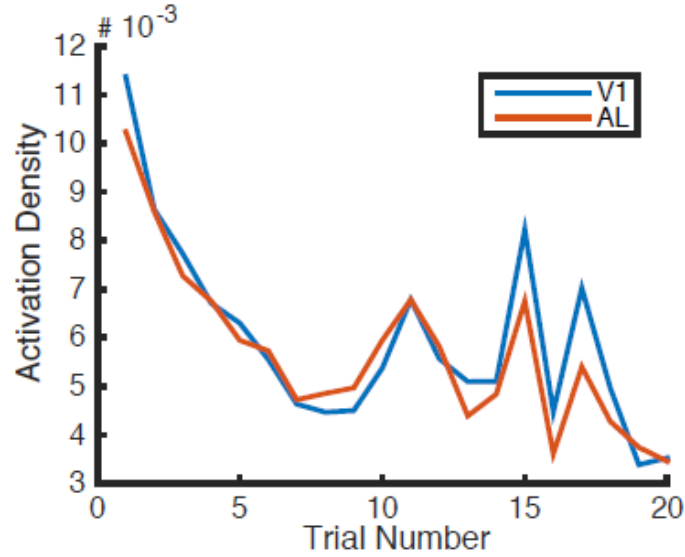


Figure 4.7: Activation density v/s number of trials [107]. As the learning increases, the number of active neurons decreases. V1 represents the primary visual cortex and AL represents the anterolateral region of the specimen’s visual processing system.

4.15 refers to the penalty co-efficient of the cost function. If λ is too small, the sparsity penalty takes little effect and model accuracy is given more priority and if λ is too large, sparsity becomes the priority leading to very sparse models but with unacceptable accuracy. The key is to find the balance between task loss and sparsity penalty.

4.2.3.1 Why l_1 Norm?

The purpose of the sparsity penalty is to minimize the number of non-zero activations in the temporal delta layer, yet Eq. 4.15 uses l_1 norm instead of l_0 norm. This is because although l_0 norm represents the cardinality of the delta maps directly, it is a non-convex problem, so is not preferred as the constraint. The other popular penalties include l_1 , l_2 norm and their combinations. Here, l_1 norm is the sum of absolute delta values (Eq. 4.16) and l_2 norm is the square root of the squared sum of delta values (Eq. 4.17).

Now, the question remains, *why does l_1 norm induce more sparsity than l_2 norm?* In Figure 4.8, H0 represents the task, $H0: y = Wx$, which is a straight line, where y is the true label, W is the weight matrix, and x is the activation vector. Then, the disc represents the norm ball of l_2 in 2D, and the diamond represents the norm ball of l_1 in 2D. They are the constraints to the H0 task.

$$\|x\|_1 = |x_1| + |x_2| \tag{4.16}$$

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2} \tag{4.17}$$

Then, the objective is to find the minimum point (x_1, x_2) between the line and constraint while maintaining feasibility. In most cases, l_2 tends to find a solution that is tangential to

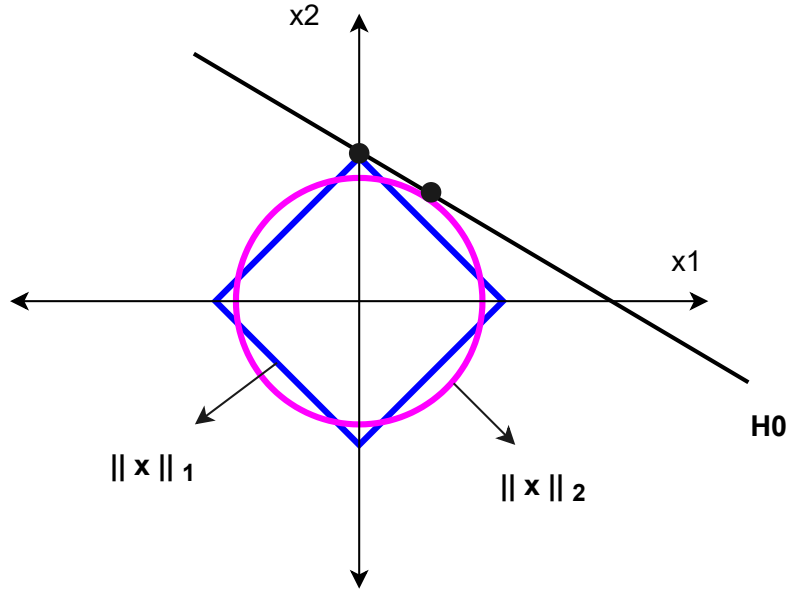


Figure 4.8: l_1 norm generating sparser solution better than l_2 . Line H0 represents the task, disc (pink) represents the l_2 ball, diamond (blue) represents the l_1 ball.

H0, i.e x_1 and x_2 are non-zero. However, l_1 , due to its nature, finds a solution where one of the x values is on the axis. In Figure 4.8, l_1 has removed the relevance of x_1 . Extending this explanation to higher dimensional cases, it can be justified how l_1 penalty induces sparsity.

4.3 Proposed Algorithms

Putting it all together, two algorithms are presented in this work. One uses delta calculations and sparsity penalty concepts with fixed point quantization, and the other uses modified learned step size quantization. The flowcharts of the methodology is given in Figure 4.9 and 4.10.

4.3.1 Temporal Delta Layer with Fixed Point Quantization

1. The output from the activation layer is fed into the *quant_layer* along with the bitwidth to which the activations are to be quantized.
2. Given the activations and bitwidth, required integer bits and fractional bits are calculated using Eq. 4.6 and 4.7.
3. Using the parameters bitwidth, fractional bits, and sign bit, activations are quantized using Eq. 4.8.
4. These quantized activations are then passed onto the *temporal_delta_layer* along with the penalty co-efficient, λ .
5. Then delta maps are calculated considering whether the activations are a part of convolutional layer (5 dimensional) or fully connected layer (3 dimensional).

6. The l_0 and l_1 norm of delta maps for each batch is calculated along with the total number of neurons. Then, l_1 norm of the delta map is normalized with the number of neurons, i.e. *delta_l1_normalized*.
7. Sparsity penalty is introduced to minimize *delta_l1_normalized*. λ is used to control the weight of this penalty or constraint.
8. The outputs of the *temporal_delta_layer* are quantized delta activations, the total number of neurons and the activation density of the delta maps, which are passed onto the next layer.

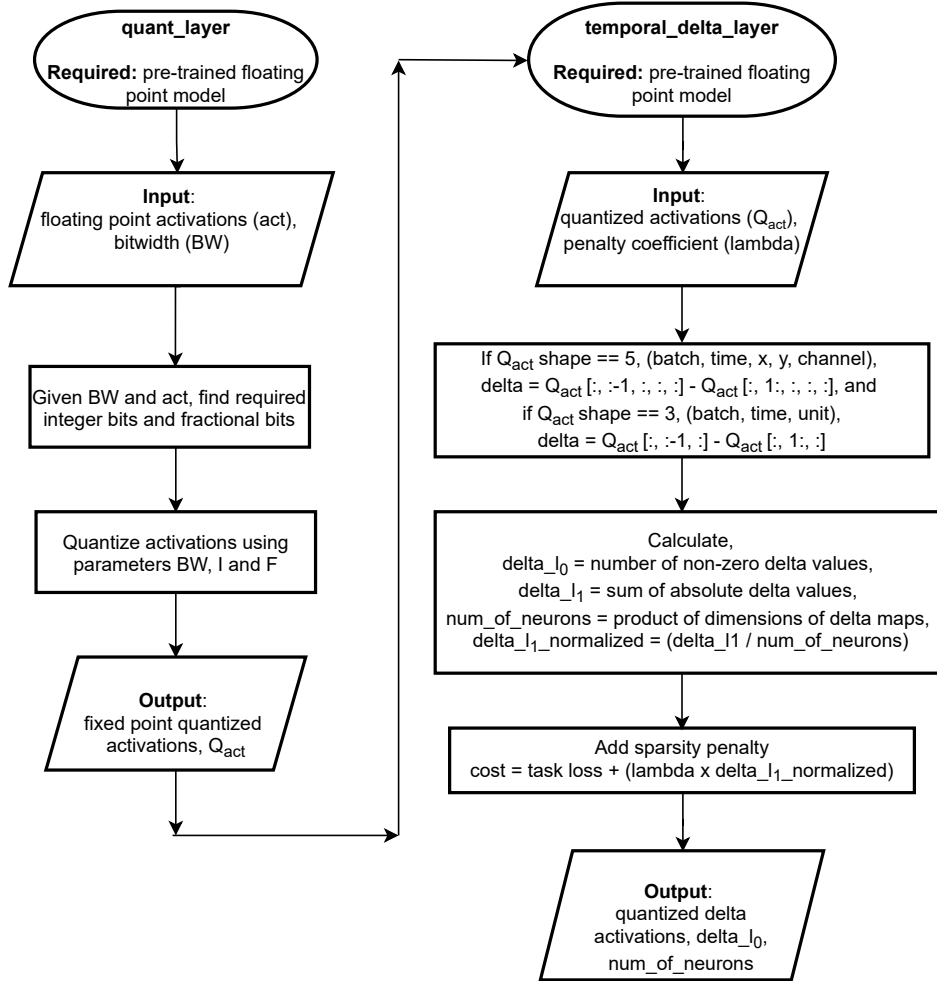


Figure 4.9: Methodology flow of temporal delta layer with fixed point quantization

4.3.2 Temporal Delta Layer with Learned Step Size Quantization

1. The output from the activation layer is fed into the *temporal_delta_layer* as an input along with penalty coefficient (λ), quantization granularity (*q_level*) and step size initialization (*step_init*).

2. The parameter step size is initialized with *step_init* according to the quantization granularity, *q_level*. It could be channel-wise or layer-wise.
3. Then, the activations and step size is passed to the *quantize_lsq_function*.
4. Within the function, activations are quantized using the method mentioned in Section 4.2.2.6.
5. Steps 5 to 7 from Section 4.3.1 are repeated.
6. The outputs of the *temporal_delta_layer* are quantized delta activations, the total number of neurons and the activation density of the delta maps, which are passed onto the next layer.

4.4 Conclusion

The chapter explains the proposed methodology of this thesis in detail. A new layer is introduced which exploits the concept of temporal redundancy with the network to reduce the computations and memory accesses. Two types of quantization methods are considered to facilitate this layer. Also, the layer consists of a new penalty function which exclusively tries to minimize the temporal activation density. The results of adding this layer to a network are detailed in chapter 5.

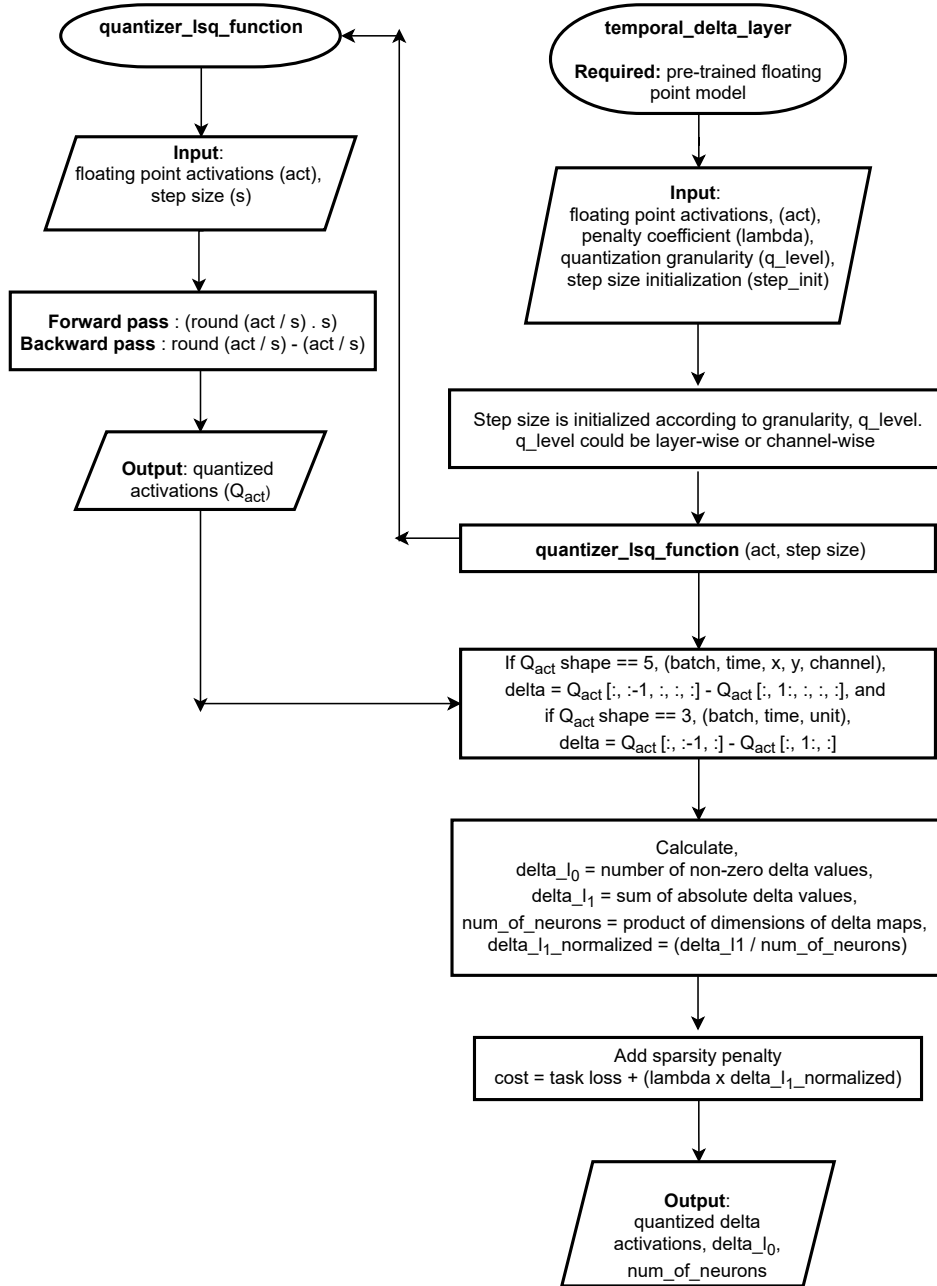


Figure 4.10: Methodology flow of temporal delta layer with learned step size quantization

Part III

Results

Experiments and Results

In this section, the proposed methodology explained in Chapter 4 is analyzed to study how it helps achieve the key objectives of this thesis.

5.1 Experiment Setup

5.1.1 Application: Human Action Recognition

The quantity of video data available is explosively increasing due to the easy access to digital recording devices and the popularity of video sharing websites. Among the video related applications, human action recognition (HAR) is one of the most relevant research topics in the computer vision community. Action recognition is the process of understanding high-level human behaviour by labelling video frames with corresponding action labels. Video-based action recognition has garnered immense interest in recent years due to its wide range of applications, such as video surveillance, human-computer interaction, autonomous driving, and health care [108].

Video-based action recognition involves various challenges [109]. First, the video data can have intra-class variations (for example, walking can differ in speed and stride length) and inter-class variations (for example, floating and swimming can appear similar). Second, environments and camera settings can vary significantly. For instance, human localization becomes much harder in a dynamic, cluttered environment. Finally, obtaining and labelling the training/testing video data is difficult.

Therefore, this work analyses the effectiveness of the proposed methodology against the backdrop of human action recognition.

5.1.2 Dataset - UCF101

UCF101 is a widely used human action recognition dataset of 'in-the-wild' action videos, obtained from YouTube, having 101 action categories. The dataset consists of 13320 videos with considerable variety in camera motions, illumination conditions, background, object scale, etc. (see Figure 5.1).

The videos in UCF101 are broadly grouped into 25 groups based on similar backgrounds, similar viewpoints, etc. Each group consists of 4-7 videos of an action. These actions can be divided into five macro-classes: 1) human-object interaction, 2) body-motion only, 3) human-human Interaction, and 4) group activity. All the videos have a fixed frame rate of 25 FPS and a resolution of 320×240 [110]. Thus, UCF101 is chosen as the benchmarking dataset for this work.

5.1.3 Architecture

The architectural approaches to perform human activity recognition can be divided into two major categories. (a) The classical image processing based approach, which is based on feature

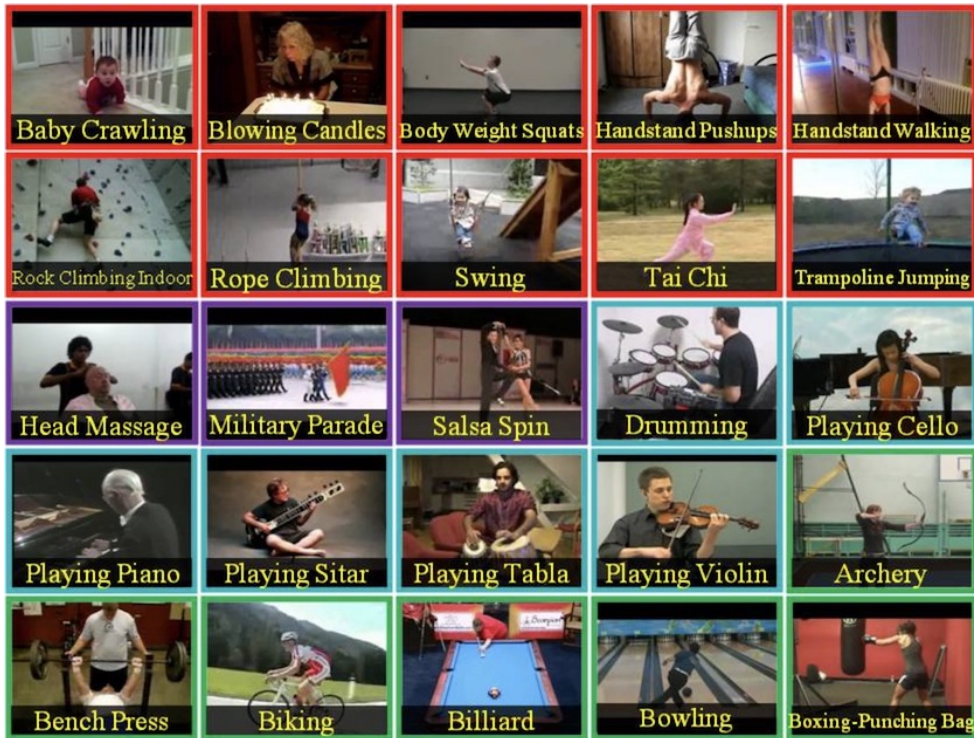


Figure 5.1: Samples of action categories from UCF101 dataset [110]

detectors and descriptors such as Hessian3D, Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), etc. (b) The end-to-end learning-based approach with the ability to learn features automatically from the dataset [111]. This rules out the need for handcrafted feature detectors and descriptors. A trainable classifier follows the feature extraction process in both approaches. This thesis uses the deep learning based approach.

5.1.3.1 Two Stream Network

Unlike still image-based recognition, video-based processing involves a temporal component that provides additional and important information for recognition. Theoretically, most of the actions can be reliably recognized based on temporal information. [112] was the first to explore the architecture based on two separate recognition streams (spatial and temporal), which are then combined by average fusion. The spatial stream performs action recognition from still video frames, whilst the temporal stream recognizes action from motion in the form of dense optical flow as shown in Figure 5.2. Both streams consist of CNNs. This two-stream architecture is also bio-inspired, considering the human visual cortex also contains two pathways: the ventral stream (which performs object recognition) and the dorsal stream (which recognizes motion).

In the original two-stream network, RGB images were used for the spatial stream and stacked optical flow fields were used for the temporal stream. However, inspired by [2], this work replaces stacked optical flow fields with stacked RGB temporal difference frames. RGB difference between two consecutive frames characterize the appearance change, which in turn correspond to the motion salient region (see Figure 5.3). Moreover, RGB difference is easier

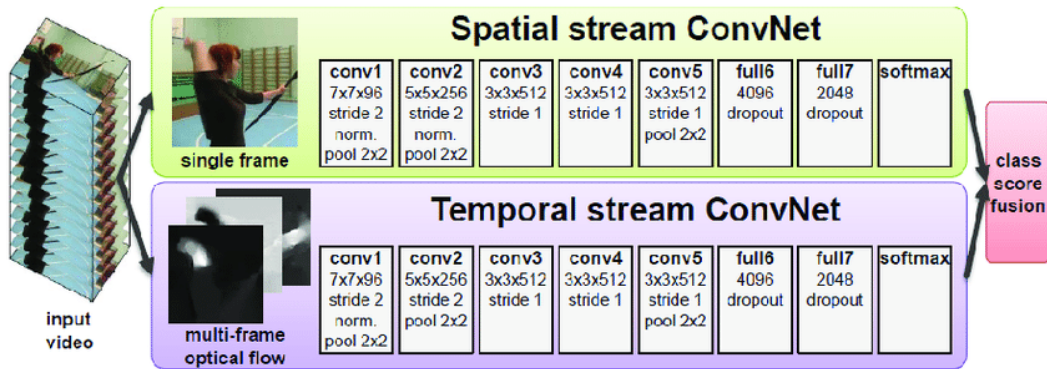


Figure 5.2: Two stream architecture for action recognition - original [112]

to compute. On the other hand, in an online scenario, computing the optical flow fields on the go require heavy computational resources and may lead to high latency.

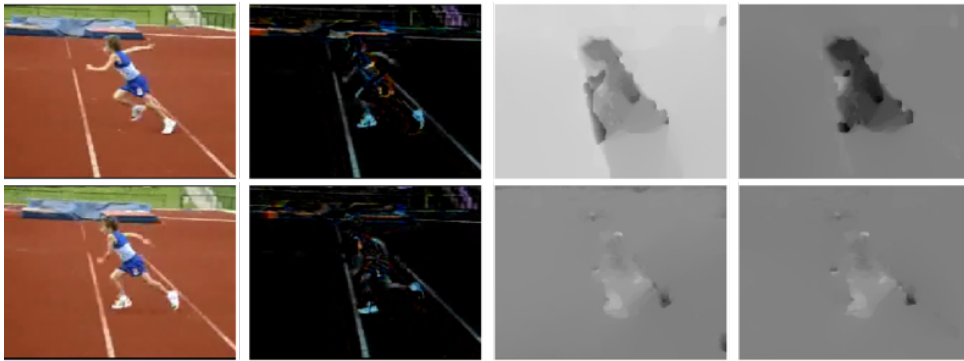


Figure 5.3: Possible input modalities for action recognition: (from left to right) RGB, RGB temporal difference, optical flow - U channel, optical flow - V channel [2]

5.1.3.2 ResNet50

Residual Network or ResNet [113] was initially developed by Microsoft Research and was implemented mainly to solve the vanishing gradient problem in deeper CNNs during back propagation. The residual block is the core of a ResNet. It introduces a skip connection or a "shortcut", which allows the gradient of a layer to be directly backpropagated to an earlier layer (skipping two or three layers in between). This architecture enables the construction of deeper networks without degrading the accuracy.

There are two types of residual blocks used in a ResNet, depending on whether the input and output dimensions are equal [114].

Identity block: The identity block is one of the standard blocks used in ResNet and corresponds to the scenario where the dimensions of the input activation match that of the output activation. Figure 5.4.(a) depicts an identity block, where the lower one is the main path and the upper one is the shortcut path.

Convolutional block: The convolutional (or conv) block is used in scenarios where the input and output dimensions do not match. The difference between identity block and conv block is that in the latter, there is a convolutional layer in the shortcut path (see Figure 5.4.(b)). This layer resizes the input x such that the shortcut path's dimension match the dimensions in the main path.

The ResNet-50 model is a variety of ResNet where convolution and identity blocks are stacked, as shown in Figure 5.4.(c). This thesis uses ResNet50 in both spatial and temporal streams as feature extractors.

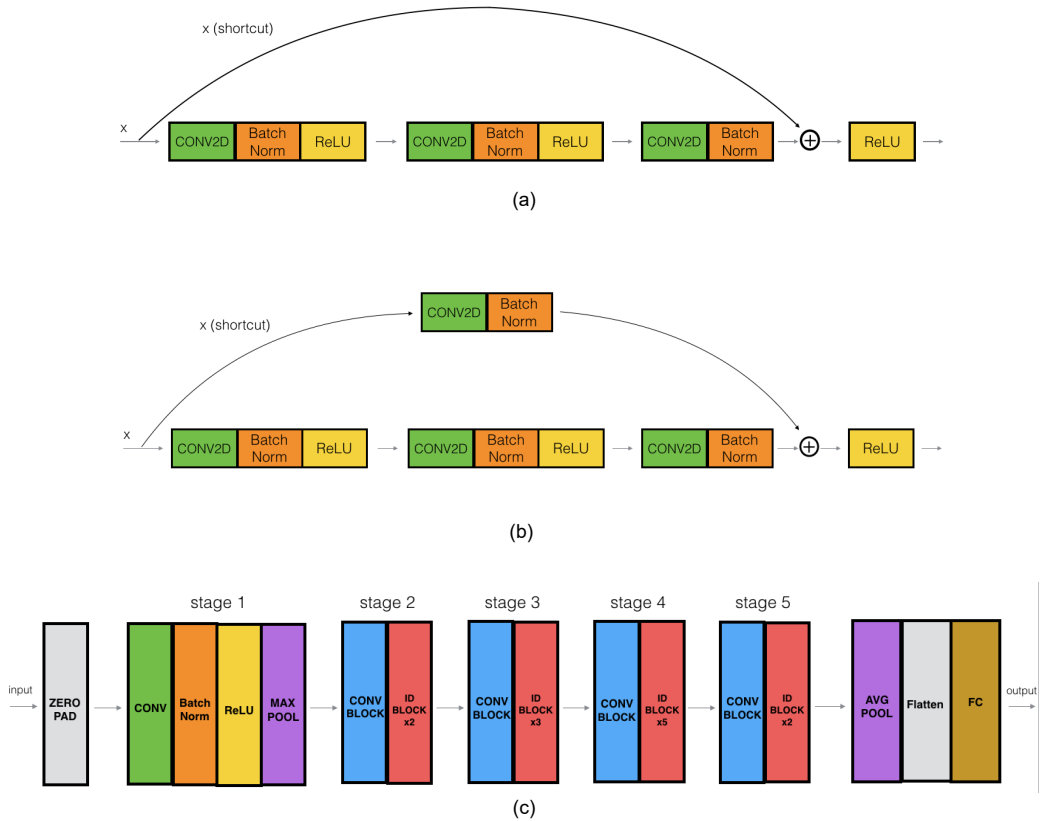


Figure 5.4: ResNet architecture: (a) Identity block, (b) convolution block, and (c) ResNet50 block diagram [114]

5.1.4 Pre-processing - Frame Generation

The video inputs from the UCF101 dataset are converted to their corresponding frames using *Keras video generators* [115]. The package contains three types of generators:

- *VideoFrameGenerator*, which provides a pre-determined number of consecutive frames from the entire video.
- *SlidingFrameGenerator*, which provide several sequences of the same "action" by sliding the video's cursor.

- *OpticalFlowGenerator*, which provides an optical flow sequence from frames using different methods including absolute difference between two frames and Farneback method.

An additional module was created during this work which enabled the vertical stacking of frames of form 3D video cubes.

5.1.5 Evaluation Metrics

- **Classification accuracy:** The ratio of the number of correct predictions made by the model over the total number of predictions. The predictions are averaged over all test frames.
- **Spatial activation sparsity:** The ratio of zero activations within a feature map to the total number of activation neurons in the feature map. This is often termed as activation sparsity.
- **Temporal activation sparsity:** The ratio of zero activations within a delta map (or the difference of two consecutive feature maps) to the total number of activation neurons in the delta map.

Note: The cost function of the model, which is the combination of a task loss term and a weighted penalty term, is a minimization problem. Therefore, it is to be noted that the task loss term is trying to minimize the classification error and, the penalty term is trying to minimize the activation density of the model. Activation density represents the ratio of non-zero activations over the total number of activations, i.e. $activation\ density = 1 - activation\ sparsity$. This work, within context, uses both density and sparsity to explain the results.

5.1.6 Baseline

For baseline, the two-stream architecture (with ResNet50 on both streams) was fine-tuned with UCF-101 as the input, without the sparsity penalty. The spatial stream used single-frame RGB images of size (224, 224, 3) as the input, while the temporal stream used stacks of 10 RGB difference frames of size (224, 224, 10×3) as the input. Also, both these inputs were time distributed to apply the same layer to multiple frames simultaneously and produce output that has time as the fourth dimension. Both the streams were initialized with pre-trained ImageNet weights and fine-tuned with an SGD optimizer.

As shown in Figure 5.5, under the above-mentioned setup, spatial and temporal streams achieved an accuracy of 75% and 70%, respectively. Then, both streams were *average fused* to achieve a final classification accuracy of 82%. Also, in this scenario, both streams were found to have an activation sparsity of $\sim 47\%$.

5.2 Experiments

Scenario 1: A l_1 regularization based penalty was placed on the activation layer of the baseline to induce spatial sparsity to the network. This setup essentially tries to reduce the number of non-zero activations within a feature map spatially by explicitly encoding the sparsity inducing l_1 prior to the cost function. In this setup, the model starts with the baseline

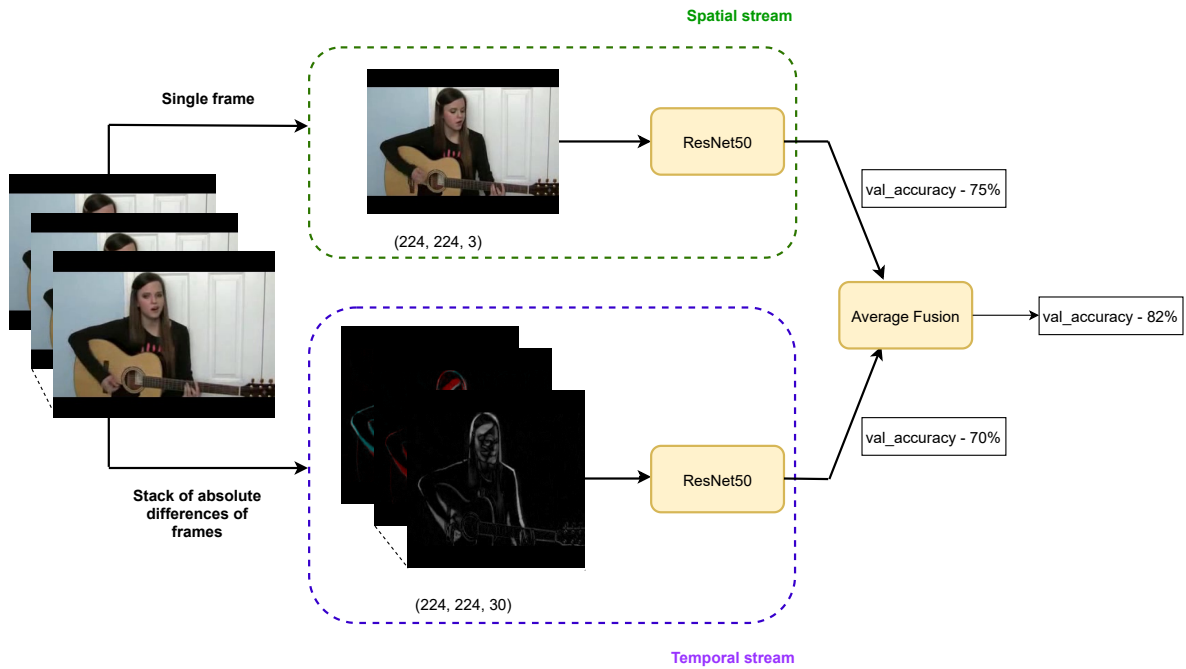


Figure 5.5: Baseline: two stream network with single frame RGB images for spatial stream and stack of RGB difference frames for temporal stream. The model achieved 82% validation accuracy.

weights from Section 5.1.6. All the layers are fine-tuned to induce spatial activation sparsity while maintaining the baseline accuracy.

This scenario is considered for a fair comparison of the proposed (temporal sparsity based) method with the conventional spatially regularized method. As spatial sparsity in feature maps is not a focus of this work, the results from the aforementioned scenario are not analyzed in detail.

Scenario 2: The setup consecutively places the fixed point based quantization layer and temporal delta layer after every activation layer in the network (as explained in Section 4.3.1). The temporal delta layer here also includes a l_1 norm based penalty. What differentiates this from the previous scenario is that the regularization is applied to sparsify the delta map rather than the feature map itself. Fixed point quantization, in this setup, is used as a means to decrease the precision of input activation maps. Both of these techniques promote temporal sparsification.

Again, the baseline weights were used as a starting point, and all the layers including the temporal delta layer is fine-tuned until acceptable convergence. The hyper-parameters specifically required for this setup were bitwidth (to which the activations were to be quantized) and penalty co-efficient to balance the tussle between task loss and sparsity penalty.

Scenario 3: The setup is similar to the previous scenario except for the activation quantization method used. The previous experiment used fixed precision quantization where all the activation layers in the network were quantized to the same bitwidth. However, this experiment uses learnable step-size quantization (LSQ), which performs channel-wise quan-

tization depending on the activation distribution resulting in mixed-precision quantization of the activation maps (as explained in Section 4.3.2).

The layer also introduces a hyperparameter during training (apart from the penalty coefficient mentioned earlier) for the step size initialization. These step sizes are initialized stage-wise for the ResNet50 used. The concept of stages in ResNet50 is shown in Figure 5.4. According to the initialization, if stage 3 is initialized with a step-size of 4, every channel in every layer of stage 3 (layer 11 - 22) is initialized with 4. Then, during training, the step size increases or decreases depending on the activation distribution in each channel.

5.2.1 Accuracy v/s Activation Sparsity

Model setup (Spatial stream)	Accuracy	Activation sparsity
Baseline	75%	48%
Spatial L1 regularisation	60%	70%
Temporal delta layer with fixed point quantization	73%	74%
Temporal delta layer with learned step-size quantization	69%	86%

Table 5.1: Spatial stream - comparison of accuracy and activation sparsity obtained through the proposed scenarios against the benchmark. In the case of fixed point quantization, the reported results are for a bitwidth of 6 bits.

Model setup (Temporal stream)	Accuracy	Activation sparsity
Baseline	70%	47%
Spatial L1 regularisation	56%	64%
Temporal delta layer with fixed point quantization	68%	67%
Temporal delta layer with learned step-size quantization	65%	89%

Table 5.2: Temporal stream - comparison of accuracy and activation sparsity obtained through the proposed scenarios against the benchmark. In the case of fixed point quantization, the reported results are for a bitwidth of 7 bits.

Table 5.1 and 5.2 show the baseline accuracy and activation sparsity compared against the three scenarios mentioned¹. An aspect to take note is, although temporal sparsity between two feature maps is a metric of interest, it can also be interpreted as the spatial sparsity of the delta map itself. Moreover, in the proposed methodology, the delta maps are the ones propagated to the next layer and lead to the skipping of computations. That is why, colloquially, 'activation sparsity' is used as the metric to analyse the results.

¹The training and inference for all experiments were performed on NVIDIA Quadro RTX 5000 with CuDNN 10.1

Firstly, it can be seen that in the case of spatial l_1 regularization (in both streams), the activation sparsity increases in comparison to the baseline, but at the cost of losing accuracy. The reason for this could be that the complexity of the task is beyond the capacity of the sparse model, considering 67% of the activations are zero. Moreover, as explained in Section 3.2, in the case of irregular sparse tensors, very high sparsity is required for the model to see gains in terms of energy consumption.

Secondly, when the temporal delta layers with fixed point quantized activations are included in the baseline model, it can be observed that the activation sparsity increases considerably with a lower loss in accuracy. This method clearly do better both in terms of accuracy and sparsity than the spatial regularization method. One interesting observation is that, although the sparsity penalty in the temporal delta layer does a good job of decreasing the activation density, quantizing the activations from floating to fixed point representation pushes the activation sparsity of the model even higher. From the experiments, the temporal delta layer without any activation quantization managed to increase the activation sparsity only by 10% in comparison to the baseline, even after exhaustive training. However, the temporal delta layer with sparsity penalty and fixed point quantization managed to increase the activation sparsity by $\sim 25\%$. This is because lowering the precision from 32 bits to 8 bits (or less) leads to temporal differences of activations going to absolute zero as explained in Section 4.2.2.1.

Additionally, the reason for close-to baseline accuracy in the method involving fixed point quantization can be attributed to fractional bit allocation flexibility. That is, as the bitwidth is fixed, the number of integer bits required is decided depending on the activation distribution within the layer, and the rest of the bits are assigned as fractional bits. This makes sure that the precision of the activation is compromised for range. For instance, Figure 5.6 shows that when the model is quantized to 8 bits, in most layers of the network, the majority of the bits are used to represent the integer part. Also, another contributing factor for accuracy sustenance is that the first and the last layers of the model are not quantized, similar to works like [55] [54] [116]. This is because the first and last layer has a lot of information density. Those are the layers where input pixels turn into features and features turn into output probabilities, respectively, which makes them more sensitive to quantization.

Although the activation sparsity gain in the case of the temporal delta layer with fixed point quantization is better than the activation sparsity obtained due to l_1 regularization, it is still not sufficiently high as required. In this effort, the bitwidth of the activations are decreased in the expectation of increasing sparsity. However, as the bitwidth goes below a certain value (6 bits for spatial and 7 bits for temporal stream), sparsity increases, but accuracy starts to deteriorate beyond recovery, as shown in Table 5.3. This is because quantizing all layers of a network to the same bitwidth can mean that the inter-channel variations of the feature maps are not fully accounted for. Since the number of fractional bits is usually selected to cover the maximum activation value in a layer, the fixed bitwidth quantization tends to cause excessive information loss in channels with a smaller dynamic range. Therefore, it can be inferred that mixed-precision quantization of activations is a better approach to obtain good sparsity without compromising accuracy.

Finally, using the temporal delta layer where incoming activations are quantized using learnable step-size quantization (LSQ) gives the best results for both spatial and temporal streams. As the step size is a learnable parameter, it gives the model enough flexibility to result in a mixed precision model, where each channel in a layer has a bitwidth that suits its activation distribution. This kind of channel-wise quantization minimizes the impact of

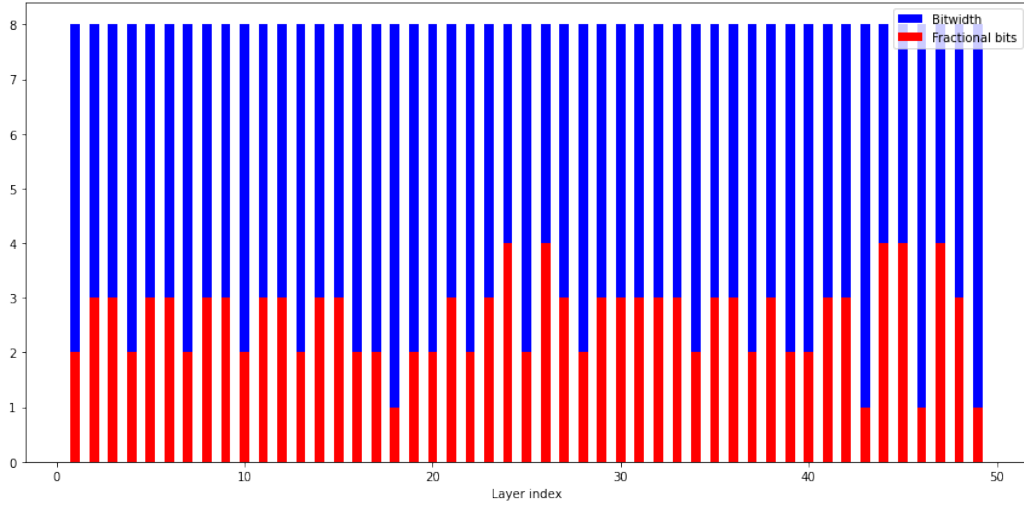


Figure 5.6: The distribution of fractional bits required over all the layers of ResNet50. The activations in all layers is quantized to 8 bits (blue). However, each layer selects its number of fractional bits (red) depending on the maximum value in the layer to maintain the range.

Activation bitwidth	Spatial stream		Temporal stream	
	Accuracy (%)	Activation sparsity (%)	Accuracy (%)	Activation sparsity (%)
32	75	50	70	47
8	75	68	70	65
7	75	71	68	70
6	73	75	61	73
5	65	80	-	-

Table 5.3: Result of decreasing activation bitwidth in an attempt to increase activation sparsity while maintaining accuracy. For spatial stream, decreasing below 6 bits caused the accuracy to drop considerably. For temporal stream, the same happened below 7 bits.

low-precision rounding. Figure 5.7 shows how the implementation of LSQ on floating point activations zeroes out the near zero activations and lead to more discrete feature maps.

The weights generated using this method was then average fused to find the final two-stream network accuracy and activation sparsity (Table 5.4). Finally, the proposed method can achieve 88% activation sparsity with a 5% accuracy loss.

5.3 Result Analysis

The last section concluded that fine-tuning the model with the temporal delta layer supported by learned step-size quantization gives the best results amongst the three scenarios considered. This section elaborates on the results from the said method by analyzing the hyperparameters required. Also, an estimate of the potential computation and memory access reduction is calculated.

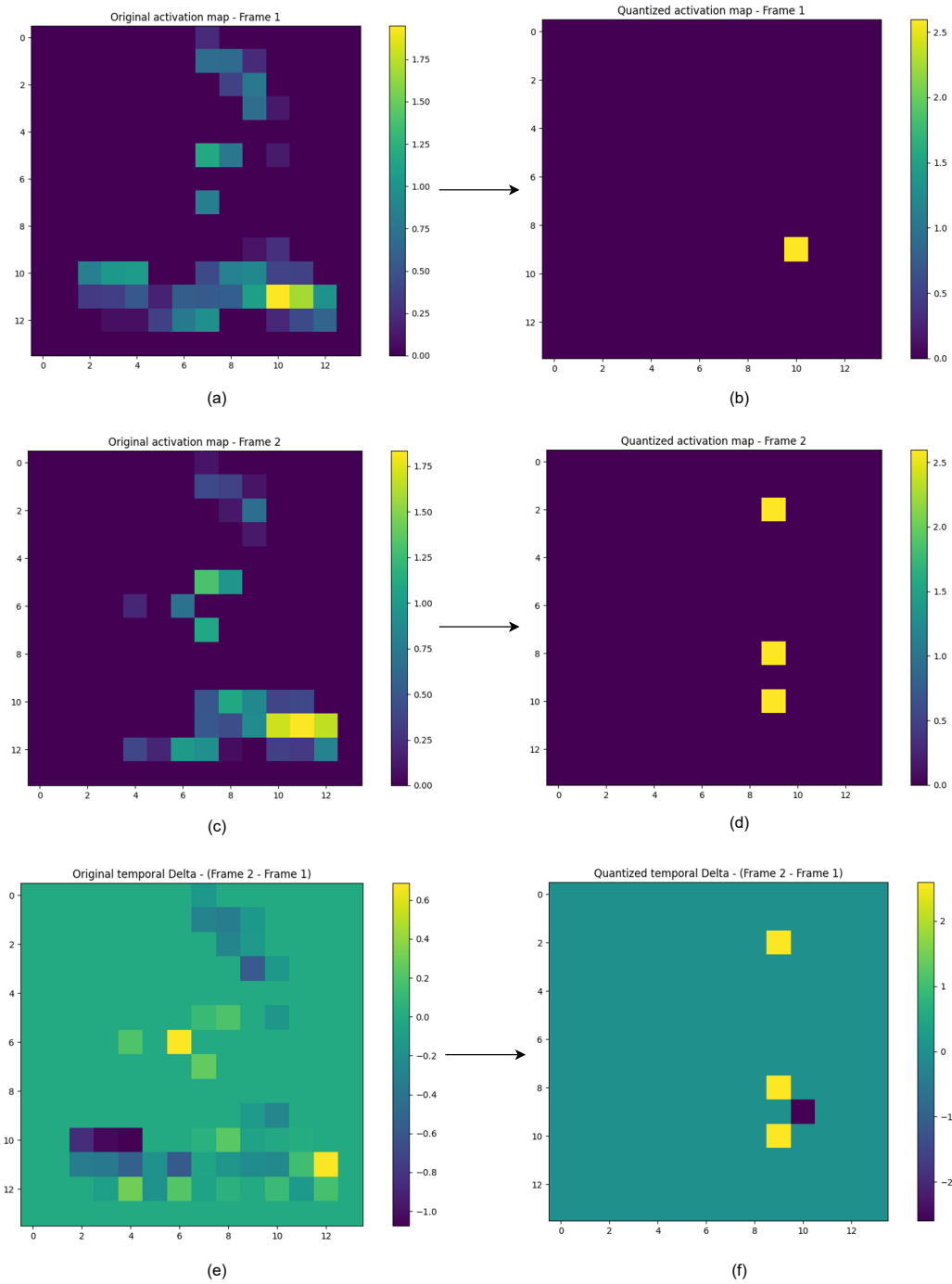


Figure 5.7: Effect of quantization on sparsity: in (a), and (c), the unquantized feature maps has a lot of near zero values, which in turn leads to near zero values in their delta map - (e), deterring the increase of sparsity. Whereas, quantizing the corresponding activations, (b) and (d), causes its delta map, (f), to have more sparsity.

Model type	Baseline		Proposed method	
	Accuracy (%)	Activation sparsity (%)	Accuracy (%)	Activation sparsity (%)
Spatial stream	75	50	69	86
Temporal stream	70	46	65	89
Two-stream (Average fused)	82	47	77	88

Table 5.4: Final results on 2 stream network after average fusing the spatial and temporal stream weights. With 5% accuracy loss, the proposed method almost doubles the activation sparsity available in comparison to the baseline.

5.3.1 Importance of Step Size Initialization

In the context of quantization, step size indicates the distance between two quantization transition points. Lesser values of step size lead to more quantization levels within the range, which implies more bitwidth. Similarly, larger values of step size result in lower bitwidth. As it is a learnable parameter like weight, it must be initialized properly.

Figure 5.8 depicts the effect of stage-wise step size initialization on accuracy and activation sparsity. For clarity, '2-2-3-3-3-2' indicates that stage 2 temporal delta layers are initialized with a step size of 2, stage 3 delta layers also with 2, and so on till the end of the network (refer to Figure 5.4.(c) to understand the concept of *stages* better). The last stage is the fully connected layer itself. It can be observed that if the step size is initialized with a lower value, it leads to more bitwidth, and therefore, the accuracy corresponding to that is the highest. However, as the bitwidth is high, the activation sparsity obtained is less than ideal. On the contrary, if the step sizes are initialized with larger values, all layers are quantized aggressively, resulting in high activation sparsity. But, the accuracy suffers due to the overall low bitwidth representation. Thus, a 'middle-ground' value was finalized, which gave high sparsity with comparatively high accuracy.

Additionally, Figure 5.9 explains the evolution of step sizes during training. The stages were initialized with '3-3-4-4-4-3', but when it nears convergence, the final values differ according to the activation distribution and bitwidth required to represent each layer. This flexibility is the very reason LSQ results in a mixed-precision model. Also, consistent with the literature [8], the first and last layers during training opts for smaller step sizes implying they need more bitwidth for their representation.

5.3.2 Importance of Penalty Co-efficient

Penalty co-efficient (or λ), in this work, represents the weight of the l_1 norm applied to the delta maps against the task loss. This hyper parameter can range between zero and one. If $\lambda = 0$, the sparsity penalty has no impact and if $\lambda = 1$, the sparsity term takes over the optimization, heavily compromising the accuracy.

As shown in Figure 5.10, a lower penalty co-efficient leads to near baseline accuracy, but the activation sparsity is not high enough. In comparison, larger co-efficient leads to

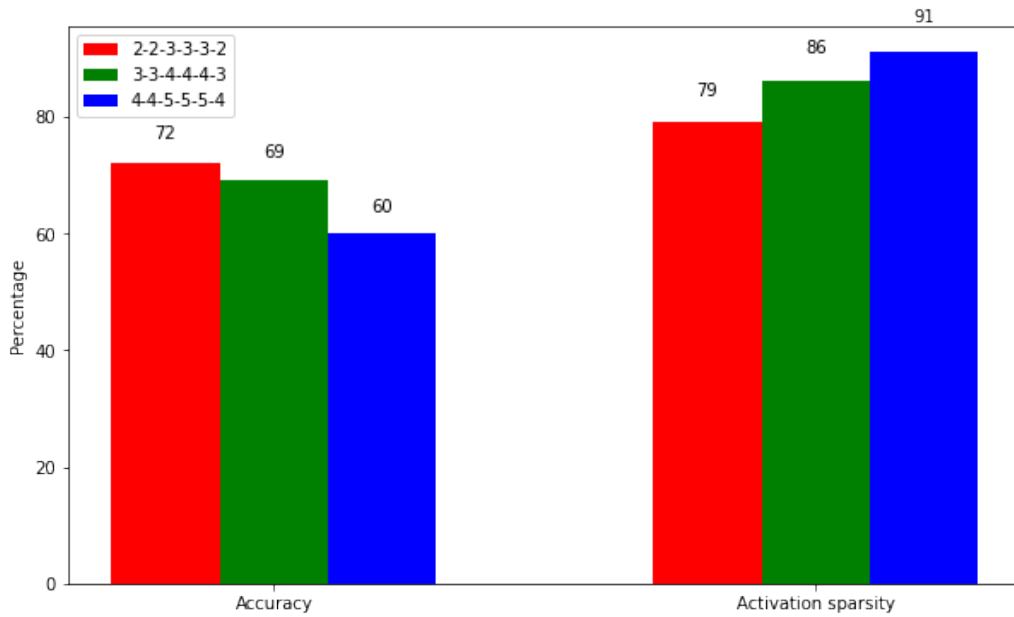


Figure 5.8: Effect of step size initialisation on accuracy and activation sparsity: '3-3-4-4-4-3' gives a balanced result of accuracy and sparsity.

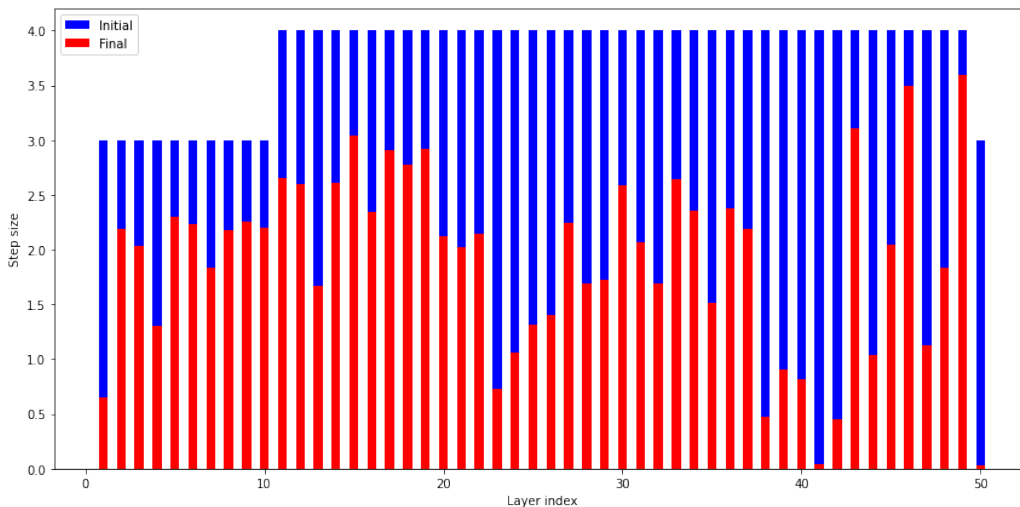


Figure 5.9: Evolution of step size from initialization to convergence. As step-size is a learnable parameter, it gets re-adjusted during training to cause minimum information loss in each layer.

> 90% sparsity but suffers unacceptable accuracy loss. Through experiment, it was found that $\lambda = 1e^{-4}$ resulted in an acceptable balance of both accuracy and sparsity, leading to the final results mentioned in Table 5.4. As an improvement, penalty co-efficient can also be found using grid search or Bayesian optimization rather than searching for it heuristically.

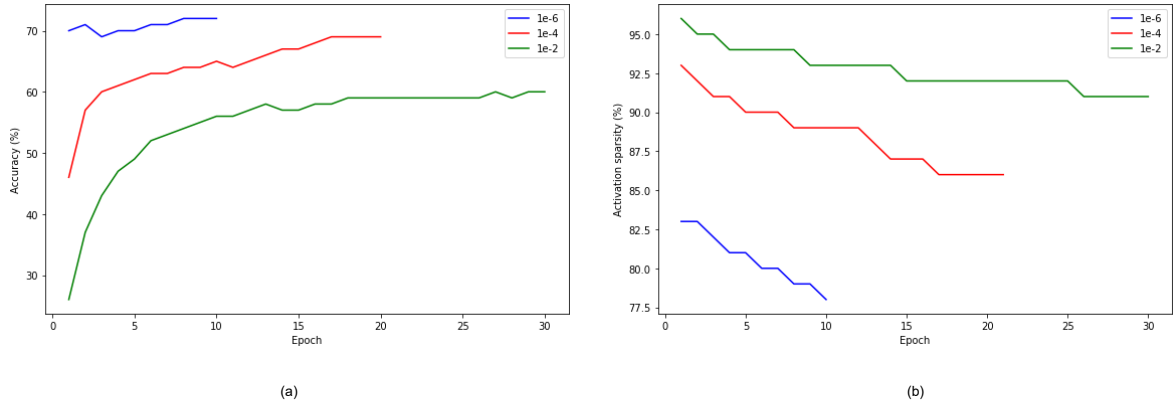


Figure 5.10: Effect of penalty co-efficient: (a) accuracy v/s epoch and (b) activation sparsity v/s epoch. Based on these, a penalty co-efficient of $1e^{-4}$ (red) was chosen for this work.

5.3.3 Layer-wise Sparsity Pattern

Figure 5.11 and 5.12 depicts the distribution of sparsity in each activation layer of ResNet50 for spatial and temporal stream respectively. One of the observations was that in the first ~ 25 layers, the activation sparsity has almost doubled under the proposed method. However, as the baseline sparsity itself is high in the later layers, drastic gain is not visible. One exception is the fully connected layer, which has less than 50% sparsity in both cases. This is because the number of neurons in the fully connected layer is very low compared to other layers, so sparsifying that layer beyond a threshold can adversely affect the performance. Also, as expected, the layers where the skip connections meet the layers in the main path of a block (i.e layer 4, 7, 10, ... 46), have less activation sparsity than their neighbouring layers.

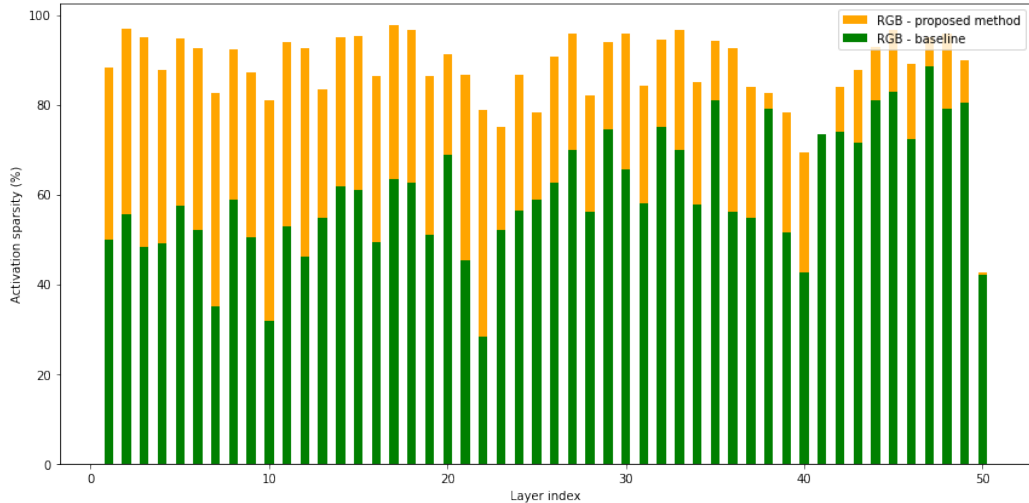


Figure 5.11: Comparison between layer-wise activation sparsity of baseline and proposed model - spatial stream. The proposed method results in $> 90\%$ sparsity in more than half of the layers.

One of the overheads of the proposed method is its memory cost. To calculate the temporal

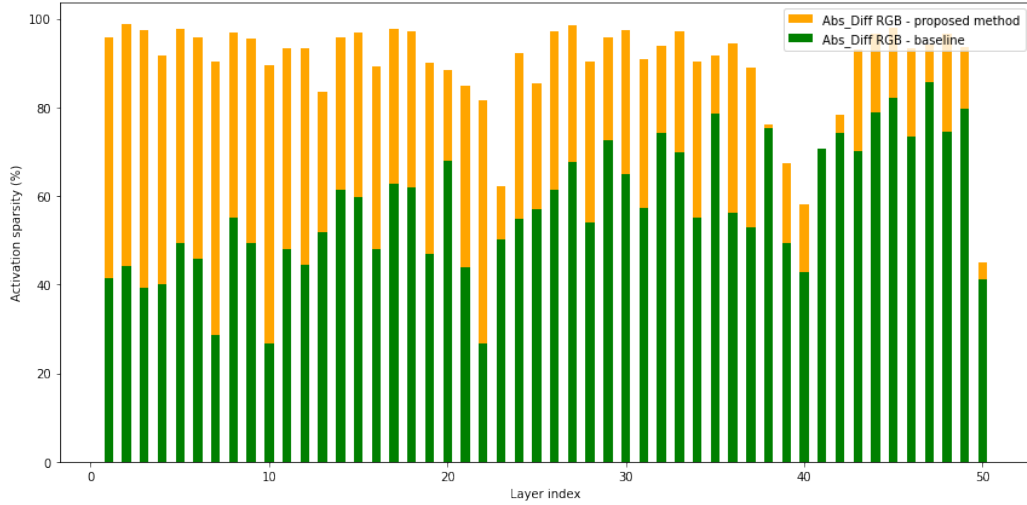


Figure 5.12: Comparison between layer-wise activation sparsity of baseline and proposed model - temporal stream.

difference between two feature maps in the delta layer, it needs to keep track of the previous neuron state. This increases the overall memory footprint of the algorithm. This memory overhead is not present in the traditional ResNet50. However, interestingly, as seen in Figure 5.11 and 5.12, more than half of the layers (spread throughout the network) gains $> 90\%$ activation sparsity. Therefore, the author proposes using temporal delta layers selectively, only at places where it can significantly improve the activation sparsity. This way, the gain in activation sparsity can potentially compensate for the memory cost.

5.3.4 Static v/s Moving Camera Input

UCF-101 has input videos shot with static as well as moving cameras. The hypothesis that static camera-based videos have more temporal sparsity than moving camera-based ones is tested in this experiment. Figure 5.13 shows the layer-wise sparsity pattern for two classes of the dataset, 'TaiChi' - shot with static camera and 'Rafting' - shot with moving camera, are compared against the average sparsity of all classes. In general, the effect of movement in the input videos has little effect on the later layers of the model where high-level features are extracted. However, it should also be noted that the average activation sparsity of 'TaiChi' class is 90%, whereas that of 'Rafting' is 80%. As the activation sparsity directly affect the reduction of computation and memory access, it can be posited that the proposed method works best to save the energy cost in applications where the camera is fixed.

5.3.5 Estimated Reduction in Computation and Memory Access

Estimating energy savings is challenging as it depends on the mapping and hardware memory hierarchy. In this work, the activation sparsity achieved is used to calculate the potential reduction in multiplication cost and memory access, which influences the energy consumption, as explained in Section 3.2.

As [20] argues that in delta based networks, the estimated computational speedup and memory access reduction is the inverse of activation density achieved, under the assumption

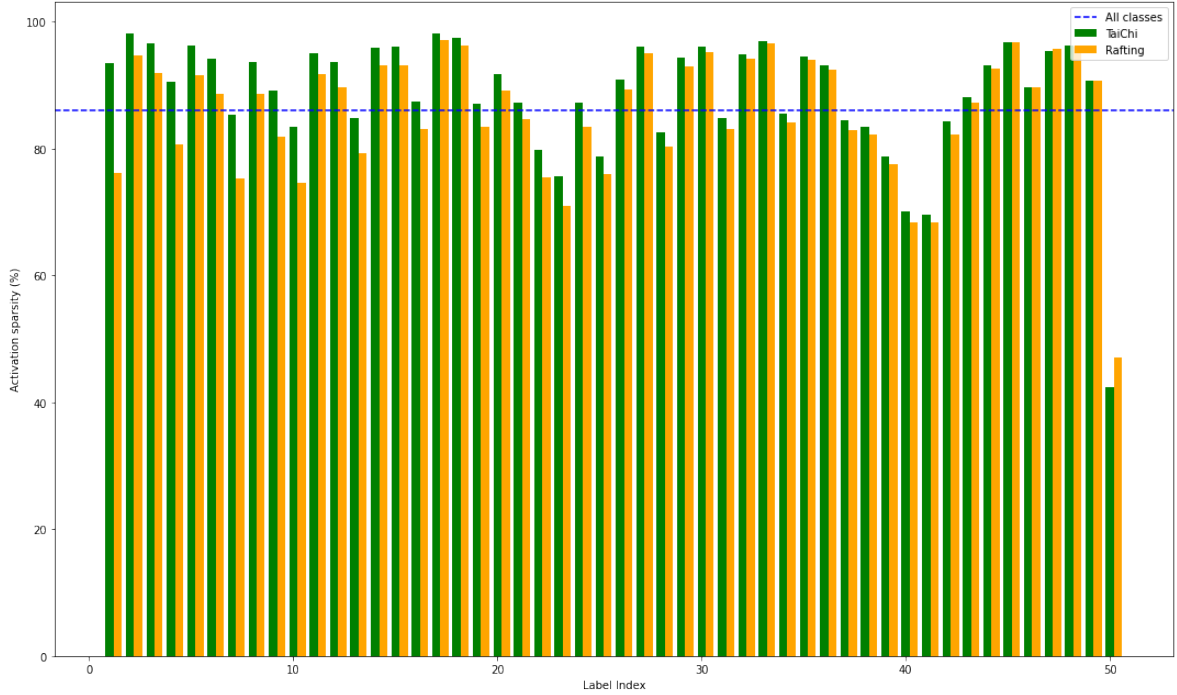


Figure 5.13: Activation sparsity of 'TaiChi' (green) and 'Rafting' (yellow) class in spatial stream. Blue dashed line is the average sparsity over all 101 classes.

that the total number of neurons is infinite.

$$\begin{aligned}
 \text{Computation reduction} &= \frac{Cost_{comp,dense}}{Cost_{comp,sparse}} \approx \frac{1}{\text{activation density}} \\
 &= \frac{1}{1 - \text{activation sparsity}} = \frac{1}{1 - 0.88} \approx 8.3
 \end{aligned} \tag{5.1}$$

$$\begin{aligned}
 \text{Memory access reduction} &= \frac{Cost_{mem,dense}}{Cost_{mem,sparse}} \approx \frac{1}{\text{activation density}} \\
 &= \frac{1}{1 - \text{activation sparsity}} = \frac{1}{1 - 0.88} \approx 8.3
 \end{aligned} \tag{5.2}$$

Therefore, the proposed method has the potential to reduce computations and memory access by a factor of 8. However, in practice, it will always be less than 8x because,

- the memory requirement is higher than the baseline as the algorithm needs to keep track of the previous state of the neurons to do delta calculations.
- on-chip memory is costly, causing the size of SRAM to be limited in the hardware. So, to accommodate the memory requirement, off-chip memory has to be used, which has two orders more energy cost than SRAM.

5.4 Conclusion

With this analysis, it can be observed that the new temporal delta layer proposed in this work achieves the objective of increasing sparsity with acceptable loss in accuracy. It is also estimated that the sparsity gain obtained has the potential to reduce computations and memory access by a factor of 8.

Conclusions and Future Work

6.1 Conclusion

After the analysis of the results, it can be concluded that the proposed methodology satisfies the key objectives of the thesis mentioned in Section 1.3.

1. **To induce sufficiently high temporal sparsity without suffering too much accuracy loss:** Intuitively, the new temporal delta layer casts the temporal activation sparsity between two consecutive feature maps into spatial activation sparsity of their delta map. This spatial sparsity is then exploited to reduce computations and memory access when performing sparse tensor multiplications in hardware. As shown in Table 5.4, the proposed method resulted in 88% activation sparsity with an accuracy drop of 5% on UCF-101 dataset for human action recognition. However, according to the experiments, input videos shot with a fixed camera showed more sparsity than those shot with a moving camera. So, it can be extrapolated that the temporal delta layer performs better in applications, like video surveillance, patient monitoring, etc., where the camera is fixed.
2. **To make the method flexible enough to be integrated with existing architectures:** The proposed layer can be deployed after any activation layer, and its incorporation does not require any adjustment to the preceding or following layer. However, the layer does add two hyperparameters during training; one for the quantization step-size initialization and the other to determine the weight of the sparsity penalty. Also, the experiments were performed exclusively on ResNet50. Therefore, to confirm the effectiveness of the method, it will be experimented on different network architectures and datasets as a follow-up to this work. Moreover, the new layer can be introduced during training, fine-tuning or inference.
3. **To study whether existing spatial sparsification methods (like quantization) can facilitate temporal sparsity:** The proposed mechanism compares the ability of two different quantization methods - fixed point quantization and learned step size quantization, to increase the temporal sparsity. Although fixed point quantization maintains the accuracy, the temporal sparsity obtained is not sufficiently high. Whereas learned step-size maintains acceptable accuracy along with high temporal sparsity. However, even though the quantization step-size is learnable in LSQ, similar to weights, the initialization of step-size is important and is found heuristically in this work which can be a "nuisance".
4. **To investigate the consequence of the method:**
 - The collateral advantage of temporal sparsity is that the computations does not increase linearly with the increase in frame rate. In standard DNN, doubling the

frame rate naturally would require double the computations. However, in the case of temporal delta layer based model, increasing the frame rate would not only increase the temporal precision of the network but also increase the temporal sparsity limiting the computations required [117].

- The drawback of using temporal delta layer derives from its requirement to keep track of the previous activations to perform delta operations. This increases the overall memory footprint which in turn increases the reliance on off-chip memory. For instance, external DRAM memory consumes two orders of magnitude more energy than SRAM [40]. Due to this, although according to Section 5.3.5, the proposed method can reduce the compute and memory cost by 8x, but in practice the amount of energy savings is less than 8x. However, the increasing popularity of new memory technologies (like resistive RAM [118], embedded Flash memory [119], etc.) may improve the aforementioned cost calculations in the near future.

6.2 Future Work

A few of the interesting directions to explore as an extension to this work are:

- The proposed architecture can be implemented on hardware that can exploit the available activation sparsity to obtain more accurate energy/time measurements.
- Instead of l_1 regularization, which results in non-structured sparsity, methods like group LASSO [120] or group Hoyer [87] can be considered for the sparsity penalty term. The latter methods result in structured sparsity, which can help the distribution of computations in parallel cores.
- As the result of training with temporal delta layer is a quantized DNN, knowledge distillation can be used to improve the accuracy of that model [121]. In this setup, the baseline model can be the teacher, and the temporally sparse delta model developed from this work can be the student.
- To increase efficiency at the input side, compressed videos can be considered as inputs [100]. The spatial stream can use keyframes (I-frames), and the temporal stream can use motion vectors (P-frames).
- In this work, weight related optimizations are not considered. But efficient storing and accessing of weights is also important in reducing energy consumption in DNN. So, weight quantization can be combined with the method. As it will increase the static sparsity of the model, it will add to the computation and memory access reductions from the proposed method.
- Instead of ReLU, new activation functions like Swish [122] and Mish [123] can also be considered to confirm the effectiveness of the new layer.

Bibliography

- [1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.
- [2] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool, “Temporal segment networks: Towards good practices for deep action recognition,” in *European conference on computer vision*, pp. 20–36, Springer, 2016.
- [3] K. Chen and W. Tao, “Once for all: a two-flow convolutional neural network for visual tracking,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 12, pp. 3377–3386, 2017.
- [4] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, *et al.*, “T-cnn: Tubelets with convolutional neural networks for object detection from videos,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 10, pp. 2896–2907, 2017.
- [5] L. Cavigelli and L. Benini, “Origami: A 803-gop/s/w convolutional network accelerator,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, 2016.
- [6] 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.
- [7] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, “Neuflow: A runtime reconfigurable dataflow processor for vision,” in *Cvpr 2011 Workshops*, pp. 109–116, IEEE, 2011.
- [8] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European conference on computer vision*, pp. 525–542, Springer, 2016.
- [9] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, “Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018.
- [10] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.
- [11] T. Gale, E. Elsen, and S. Hooker, “The state of sparsity in deep neural networks,” *arXiv preprint arXiv:1902.09574*, 2019.
- [12] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

- [13] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, “Cnvlutin: Ineffectual-neuron-free deep neural network computing,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.
- [14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, “Eie: Efficient inference engine on compressed deep neural network,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.
- [15] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, *et al.*, “Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps,” *IEEE transactions on neural networks and learning systems*, vol. 30, no. 3, pp. 644–656, 2018.
- [16] O. Moreira, A. Yousefzadeh, F. Chersi, G. Cinserin, R.-J. Zwartenkot, A. Kapoor, P. Qiao, P. Kievits, M. Khoei, L. Rouillard, *et al.*, “Neuronflow: a neuromorphic processor architecture for live ai applications,” in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 840–845, IEEE, 2020.
- [17] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, “Nvidia a100 tensor core gpu: Performance and innovation,” *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.
- [18] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” *arXiv preprint arXiv:1608.03665*, 2016.
- [19] M. Mahowald, “The silicon retina,” in *An Analog VLSI System for Stereoscopic Vision*, pp. 4–65, Springer, 1994.
- [20] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, “Deltarmn: A power-efficient recurrent neural network accelerator,” in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 21–30, 2018.
- [21] “GrAI Matter Raises 14M dollar for Sparsity-Driven AI SoC.” <https://www.eetimes.com/grai-matter-raises-14m-for-sparsity-driven-ai-soc/>. Accessed: 2021-08-10.
- [22] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [23] A. Pinkus, “Approximation theory of the mlp model,” *Acta Numerica 1999: Volume 8*, vol. 8, pp. 143–195, 1999.
- [24] C. Munker, “Using convolutional neural networks to distinguish vehicle pose and vehicle class,” Master’s thesis, TU Darmstadt, Nov. 2016.
- [25] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, 2015.
- [26] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Unsupervised learning of hierarchical representations with convolutional deep belief networks,” *Communications of the ACM*, vol. 54, no. 10, pp. 95–103, 2011.

- [27] P. Baldi, “Gradient descent learning algorithm overview: A general dynamical systems perspective,” *IEEE Transactions on neural networks*, vol. 6, no. 1, pp. 182–195, 1995.
- [28] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.
- [29] “The Difference Between Deep Learning Training and Inference.” <https://www.intel.in/content/www/in/en/artificial-intelligence/blog.html>. Accessed: 2021-06-24.
- [30] “Why so sigmoid??.” <https://alexander-schiendorfer.github.io/2020/02/17/why-so.sigmoid.html>. Accessed: 2021-06-24.
- [31] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation functions: Comparison of trends in practice and research for deep learning,” *arXiv preprint arXiv:1811.03378*, 2018.
- [32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
- [33] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.
- [34] “What is the Softmax Function?.” <https://deeptai.org/machine-learning-glossary-and-terms/softmax-layer>. Accessed: 2021-06-25.
- [35] “Softmax Activation Function Explained.” <https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60>. Accessed: 2021-06-25.
- [36] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, “Handwritten digit recognition with a back-propagation network,” *Advances in neural information processing systems*, vol. 2, 1989.
- [37] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning hierarchical features for scene labeling,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2012.
- [38] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1653–1660, 2014.
- [39] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708, 2014.
- [40] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

- [41] “Understanding Deep Self-attention Mechanism in Convolution Neural Networks.” <https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb> Accessed: 2021-08-17.
- [42] “Predictive Neural Network Applications for Insurance Processes.” https://www.researchgate.net/publication/335609766_Predictive_Neural_Network_Applications_for_Insurance_Processes. Accessed: 2021-06-25.
- [43] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pp. 448–456, PMLR, 2015.
- [44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [45] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, “Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead,” *IEEE Access*, vol. 8, pp. 225134–225180, 2020.
- [46] Y.-H. Chen, J. Emer, and V. Sze, “Using dataflow to optimize energy efficiency of deep neural network accelerators,” *IEEE Micro*, vol. 37, no. 3, pp. 12–21, 2017.
- [47] M. Horowitz, “Energy table for 45nm process,” in *Stanford VLSI wiki*, 2014.
- [48] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, IEEE, 2014.
- [49] A. Goel, C. Tung, Y.-H. Lu, and G. K. Thiruvathukal, “A survey of methods for low-power deep learning and computer vision,” in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pp. 1–6, IEEE, 2020.
- [50] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, “Aciq: Analytical clipping for integer quantization of neural networks,” 2018.
- [51] S. Migacz, “8-bit inference with tensorrt,” in *GPU technology conference*, vol. 2, p. 5, 2017.
- [52] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+1 or-1,” *arXiv preprint arXiv:1602.02830*, 2016.
- [53] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, “Wrpn: Wide reduced-precision networks,” *arXiv preprint arXiv:1709.01134*, 2017.
- [54] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *arXiv preprint arXiv:1606.06160*, 2016.

- [55] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, “Pact: Parameterized clipping activation for quantized neural networks,” *arXiv preprint arXiv:1805.06085*, 2018.
- [56] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, “Learning to quantize deep networks by optimizing quantization intervals with task loss,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4350–4359, 2019.
- [57] Y. Li, X. Dong, and W. Wang, “Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks,” *arXiv preprint arXiv:1909.13144*, 2019.
- [58] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, “Haq: Hardware-aware automated quantization with mixed precision,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8612–8620, 2019.
- [59] A. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, S. Gao, and H. Esmaeilzadeh, “Releq: An automatic reinforcement learning approach for deep quantization of neural networks,” in *NeurIPS ML for Systems workshop, 2018*, 2019.
- [60] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, “Hawq: Hessian aware quantization of neural networks with mixed-precision,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019.
- [61] J. Liu, S. Tripathi, U. Kurup, and M. Shah, “Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey,” *arXiv preprint arXiv:2005.04275*, 2020.
- [62] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.
- [63] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” *arXiv preprint arXiv:1506.02626*, 2015.
- [64] V. Lebedev and V. Lempitsky, “Fast convnets using group-wise brain damage,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2554–2564, 2016.
- [65] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, “Variational convolutional neural network pruning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2780–2789, 2019.
- [66] J.-H. Luo and J. Wu, “An entropy-based pruning method for cnn compression,” *arXiv preprint arXiv:1706.05791*, 2017.
- [67] M. Golub, G. Lemieux, and M. Lis, “Full deep neural network training on a pruned weight budget,” *arXiv preprint arXiv:1806.06949*, 2018.
- [68] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” *arXiv preprint arXiv:1506.02626*, 2015.

- [69] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.
- [70] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, pp. 4780–4789, 2019.
- [71] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.
- [72] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, “Single-path nas: Designing hardware-efficient convnets in less than 4 hours,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497, Springer, 2019.
- [73] “Neural Architecture Search — The Foundations.” <https://medium.com/digital-catapult/neural-architecture-search-the-foundations-a6cc85f7562>. Accessed: 2021-06-25.
- [74] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.
- [75] M. Gao, Y. Shen, Q. Li, C. C. Loy, and X. Tang, “Feature matters: A stage-by-stage approach for knowledge transfer,” *CoRR*, *abs/1812.01819*, vol. 3, 2018.
- [76] A. Mishra and D. Marr, “Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy,” *arXiv preprint arXiv:1711.05852*, 2017.
- [77] A. Polino, R. Pascanu, and D. Alistarh, “Model compression via distillation and quantization,” *arXiv preprint arXiv:1802.05668*, 2018.
- [78] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, “Block-wisely supervised neural architecture search with knowledge distillation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1989–1998, 2020.
- [79] J. W. Mink, R. J. Blumenschine, and D. B. Adams, “Ratio of central nervous system to body metabolism in vertebrates: its constancy and functional basis,” *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, vol. 241, no. 3, pp. R203–R212, 1981.
- [80] A. Yousefzadeh, M. A. Khoei, S. Hosseini, P. Holanda, S. Leroux, O. Moreira, J. Tapson, B. Dhoedt, P. Simoens, T. Serrano-Gotarredona, *et al.*, “Asynchronous spiking neurons, the natural key to exploit temporal sparsity,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 668–678, 2019.
- [81] R. Quian Quiroga and G. Kreiman, “Measuring sparseness in the brain: comment on bowers (2009).,” 2010.

- [82] S. Shoham, D. H. O’Connor, and R. Segev, “How silent is the brain: is there a “dark matter” problem in neuroscience?,” *Journal of Comparative Physiology A*, vol. 192, no. 8, pp. 777–784, 2006.
- [83] T. Delbruck and S.-C. Liu, “Data-driven neuromorphic dram-based cnn and rnn accelerators,” in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 500–506, IEEE, 2019.
- [84] C. Holmgren, T. Harkany, B. Svennenfors, and Y. Zilberter, “Pyramidal cell communication within local networks in layer 2/3 of rat neocortex,” *The Journal of physiology*, vol. 551, no. 1, pp. 139–153, 2003.
- [85] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” *arXiv preprint arXiv:1803.03635*, 2018.
- [86] P. O. Hoyer, “Non-negative matrix factorization with sparseness constraints.,” *Journal of machine learning research*, vol. 5, no. 9, 2004.
- [87] H. Yang, W. Wen, and H. Li, “Deepfayer: Learning sparser neural network with differentiable scale-invariant sparsity measures,” *arXiv preprint arXiv:1908.09979*, 2019.
- [88] P.-H. Yu, S.-S. Wu, J. P. Klopp, L.-G. Chen, and S.-Y. Chien, “Joint pruning & quantization for extremely sparse neural networks,” *arXiv preprint arXiv:2010.01892*, 2020.
- [89] S. Seto, M. T. Wells, and W. Zhang, “Halo: Learning to prune neural networks with shrinkage,” in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 558–566, SIAM, 2021.
- [90] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi, “Soft threshold weight reparameterization for learnable sparsity,” in *International Conference on Machine Learning*, pp. 5544–5555, PMLR, 2020.
- [91] T.-J. Yang, Y.-H. Chen, and V. Sze, “Designing energy-efficient convolutional neural networks using energy-aware pruning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5687–5695, 2017.
- [92] M. Rhu, M. O’Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, “Compressing dma engine: Leveraging activation sparsity for training deep neural networks,” in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 78–91, IEEE, 2018.
- [93] G. Georgiadis, “Accelerating convolutional neural networks via activation map compression,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7085–7095, 2019.
- [94] T. Foldy-Porto, Y. Venkatesha, and P. Panda, “Activation density driven energy-efficient pruning in training,” *arXiv preprint arXiv:2002.02949*, 2020.
- [95] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, W. Leiserson, S. Moore, B. Nell, N. Shavit, *et al.*, “Inducing and exploiting activation sparsity for fast neural network inference,” in *37th International Conference on Machine Learning, ICML 2020*, vol. 119, 2020.

- [96] M. Mahmoud, K. Siu, and A. Moshovos, “Diffy: A déjà vu-free differential deep neural network accelerator,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 134–147, IEEE, 2018.
- [97] Q. Yang, J. Mao, Z. Wang, and H. Li, “Dasnet: Dynamic activation sparsity for neural network efficiency improvement,” in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1401–1405, IEEE, 2019.
- [98] “Activation maps for deep learning models in a few lines of code.” <https://towardsdatascience.com/activation-maps-for-deep-learning-models-in-a-few-lines-of-code-ed9ced1e8d21>. Accessed: 2021-07-14.
- [99] L. Cavigelli, P. Degen, and L. Benini, “Cbinfer: Change-based inference for convolutional neural networks on video data,” in *Proceedings of the 11th International Conference on Distributed Smart Cameras*, pp. 1–8, 2017.
- [100] C.-Y. Wu, M. Zaheer, H. Hu, R. Manmatha, A. J. Smola, and P. Krähenbühl, “Compressed video action recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6026–6035, 2018.
- [101] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, “Eva²: Exploiting temporal redundancy in live computer vision,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 533–546, IEEE, 2018.
- [102] P. O’Connor and M. Welling, “Sigma delta quantized networks,” *arXiv preprint arXiv:1611.02024*, 2016.
- [103] P.-E. Novac, G. B. Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, “Quantization and deployment of deep neural networks on microcontrollers,” *Sensors*, vol. 21, no. 9, p. 2984, 2021.
- [104] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, “Learned step size quantization,” *arXiv preprint arXiv:1902.08153*, 2019.
- [105] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *arXiv preprint arXiv:1806.08342*, 2018.
- [106] Y. Yu, R. Hira, J. N. Stirman, W. Yu, I. T. Smith, and S. L. Smith, “Mice use robust and common strategies to discriminate natural scenes,” *Scientific reports*, vol. 8, no. 1, pp. 1–13, 2018.
- [107] “sparsity-in-the-neocortex-and-its-implications-for-machine-learning.” <https://www.slideshare.net/numenta/sparsity-in-the-neocortex-and-its-implications-for-machine-le>. Accessed: 2021-08-13.
- [108] M. Vrigkas, C. Nikou, and I. A. Kakadiaris, “A review of human activity recognition methods,” *Frontiers in Robotics and AI*, vol. 2, p. 28, 2015.
- [109] I. Jegham, A. B. Khalifa, I. Alouani, and M. A. Mahjoub, “Vision-based human action recognition: An overview and real world challenges,” *Forensic Science International: Digital Investigation*, vol. 32, p. 200901, 2020.

- [110] K. Soomro, A. R. Zamir, and M. Shah, “Ucf101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.
- [111] P. T. Biliński, *Human action recognition in videos*. PhD thesis, Université Nice Sophia Antipolis, 2014.
- [112] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” *arXiv preprint arXiv:1406.2199*, 2014.
- [113] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [114] “Residual Networks.” https://datascience-enthusiast.com/DL/Residual_Networks_v2.html. Accessed: 2021-08-06.
- [115] “Keras Sequence Video Generators.” <https://github.com/metal3d/keras-video-generators>. Accessed: 2021-08-06.
- [116] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.
- [117] M. A. Khoei, A. Yousefzadeh, A. Pourtaherian, O. Moreira, and J. Tapson, “Spar-net: Sparse asynchronous neural network execution for energy efficient inference,” in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 256–260, IEEE, 2020.
- [118] S. Huang, A. Ankit, P. Silveira, R. Antunes, S. R. Chalamalasetti, I. El Hajj, D. E. Kim, G. Aguiar, P. Bruel, S. Serebryakov, *et al.*, “Mixed precision quantization for reram-based dnn inference accelerators,” in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 372–377, IEEE, 2021.
- [119] M. Kang, H. Kim, H. Shin, J. Sim, K. Kim, and L.-S. Kim, “S-flash: A nand flash-based deep neural network accelerator exploiting bit-level sparsity,” *IEEE Transactions on Computers*, 2021.
- [120] M. Yuan and Y. Lin, “Model selection and estimation in regression with grouped variables,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.
- [121] S. Shin, Y. Boo, and W. Sung, “Knowledge distillation for optimization of quantized deep neural networks,” in *2020 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, IEEE, 2020.
- [122] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *arXiv preprint arXiv:1710.05941*, 2017.
- [123] D. Misra, “Mish: A self regularized non-monotonic neural activation function,” *arXiv preprint arXiv:1908.08681*, vol. 4, p. 2, 2019.