Supplementary for DCT-SNN: Using DCT to Distribute Spatial Information over Time for Low-Latency Spiking Neural Networks

Anonymous ICCV submission

Paper ID 9512

1. Overall Training Methodology

1.1. Spiking Neuron Model

In this work, we employ the bio-plausible Leaky Integrate and Fire (LIF) model [6], which is described by-

$$\tau_m \frac{dU}{dt} = -(U - U_{rest}) + RI, \quad U \leq V_{th} \tag{1}$$

where U denotes the membrane potential, I is the input current representing the weighted summation of spike-inputs, τ_m indicates the time constant for membrane potential decay, R represents membrane leakage path resistance, V_{th} is the firing threshold and U_{rest} is resting potential. The discretized version of Eqn. 1 implemented in our work is given as-

$$u_i^t = \lambda u_i^{t-1} + \sum_j w_{ij} o_j^t - v_{th} o_i^{t-1}, \tag{2}$$

$$o_i^{t-1} = \begin{cases} 1, & if\ u_i^{t-1} > v_{th} \\ 0, & otherwise \end{cases} \tag{3}$$

where u is the membrane potential, subscripts i and j represent the post and pre-neuron, respectively, t denotes timestep, λ is the leak constant= $e^{\frac{-1}{\tau_m}}$, w_{ij} represents the weight of connection between the i-th and j-th neuron, o is the output spike, and v_{th} is the firing threshold. As evident from Eqn. 2, whenever u crosses this threshold, it is reduced by the amount v_{th}, implementing a soft-reset. We implement the proposed DCT-SNN using the model described above with our encoding scheme, the code for our work is submitted as part of the supplementary material.

1.2. Surrogate-Gradient Based Learning

To train deep SNNs, we use surrogate-gradient based backpropagation which performs both the temporal as well as the spatial credit assignment of errors. Spatial credit assignment is achieved by spatial error distribution across all layers, while for temporal credit assignment, we unroll the network in time and employ backpropagation through time (BPTT) [15]. The output layer neuronal dynamics is given as-

$$u_i^t = u_i^{t-1} + \sum_j w_{ij} o_j^t, \tag{4}$$

here the u_is correspond to the membrane potential of i-th neuron of final (L-th) layer. The final layer neurons do not spike, rather we just accumulate their potential over time for classification purpose. These accumulated outputs are passed through a softmax layer to obtain the class-wise probability distribution and then the cross-entropy between the true output and the network's predicted distribution is used as loss for backpropagation. The governing equations are-

$$L = -\sum_i y_i log(z_i), \tag{5}$$

$$z_i = \frac{e^{u_i^T}}{\sum_{k=1}^{N} e^{u_k^T}}, \tag{6}$$

where L is the loss function, y denotes true output, z is the prediction, T is the total number of timesteps and N is the number of classes. The derivative of the loss w.r.t. to the membrane potential of the neurons in the final layer is given as-

$$\frac{\partial L}{\partial u_i^T} = z_i - y_i, \tag{7}$$

and the weight updates at the output layer are done as-

$$w_{ij,L} = w_{ij,L} - \eta \Delta w_{ij,L}, \tag{8}$$

$$\begin{aligned} \Delta w_{ij,L} = \sum_t \frac{\partial L}{\partial w_{ij,L}^t} &= \sum_t \frac{\partial L}{\partial u_i^T} \frac{\partial u_i^T}{\partial w_{ij,L}^t} \\ &= \frac{\partial L}{\partial u_i^T} \sum_t \frac{\partial u_i^T}{\partial w_{ij,L}^t}, \end{aligned} \tag{9}$$

where η is the learning rate, and $w_{ij,L}^t$ is the weight between i-th neuron at layer L and j-th neuron at layer $L-1$ at timestep t. Since the output layer neurons are non-spiking, the non-differentiability is not an issue here. On

the other hand, the hidden layer parameter update is given by-

$$\Delta w_{ij,k} = \sum_t \frac{\partial L}{\partial w^t_{ij,k}} = \sum_t \frac{\partial L}{\partial o^t_{i,k}} \frac{\partial o^t_{i,k}}{\partial u^t_{i,k}} \frac{\partial u^t_{i,k}}{\partial w^t_{ij,k}}, \quad k = 2, 3, ...L-1 \tag{10}$$

where $o^t_{i,k}$ is the spike-generating function (Eqn. 7), k is layer index. We approximate the gradient of this function w.r.t. its input using the linear surrogate-gradient [1] as-

$$\frac{\partial o}{\partial u} = \gamma max\{0, 1 - |\frac{u - v_{th}}{v_{th}}|\}, \tag{11}$$

where γ is a hyperparameter chosen as 0.3 in this work.

1.3. Weight Initialization and Threshold Balancing

A key component in successful implementation of SNNs is proper initialization of weights and thresholds. As mentioned in section 2 of the main manuscript, we first pre-train an analogous ANN and copy the weights to the SNN for finetuning. It is critical to balance the layerwise neuronal thresholds to achieve satisfactory performance in SNNs. One approach is to choose the maximum input to the neurons computed over all timesteps at each layer as the threshold at that corresponding layer [12]. However, empirically, we have found this scheme to be unstable (training did not converge in some cases due to spike-vanishing in the deeper layers), hence we select the 99.9 percentile of the pre-activation distribution at each layer to be that layer's threshold. Again, such threshold balancing has been argued to be more robust [11]. Notably, the threshold computation has to be performed one layer at a time and sequentially from first layer to the end. Having initialized the SNN as discussed above, we perform the surrogate-gradient based learning, the details of which is depicted in Algorithm 1. In addition, next we also provide the experimental details in appendix section A.2.

1.4. 2D-DCT Equations

Conversion of pixels denoted by S_{xy} to DCT coefficients F_{uv} for an $M \times N$ block is given by:

$$C_x = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } x = 0 \\ 1 & \text{else} \end{cases}$$
$$F_{uv} = \frac{2}{\sqrt{MN}} C_u C_v \times \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} S_{xy} cos\left(u\pi \frac{2x+1}{2M}\right) cos\left(v\pi \frac{2y+1}{2N}\right) \tag{12}$$

Algorithm 1 Procedure of spike-based backpropagation learning for an iteration.

Input: pixel-based mini-batch of input (X) - target (Y) pairs, total number of timesteps (T), number of layers (L), pre-trained ANN weights (W), membrane potential (U), membrane leak constant (λ), array of layer-wise firing thresholds (V_{th}), dct block-size b, number of freq. component to train with f
Initialize: $U^t_l = 0,\ \forall l = 1, ..., L$
// M= generate dct encoded-inputs for the current mini-batch data for $b * b$ timesteps
// Forward Phase
for $t \leftarrow 1$ **to** T **do**
  $O^t_1 = M[t\%f]$; // here M[x] denotes the dct-encoded inputs sampled from frequency index x
  for $l \leftarrow 2$ **to** $L-1$ **do**
    // membrane potential integrates weighted sum of spike-inputs
    $U^t_l = U^{t-1}_l + W_l * O^t_{l-1}$
    if $U^t_l > V_{th}$ **then**
      // if membrane potential exceeds V_{th}, a neuron fires a spike
      $O^t_l = 1,\ U^t_l = U^t_l - V_{th}$
    else
      // else, membrane potential decays exponentially
      $O^t_l = 0,\ U^t_l = \lambda * U^t_l$
    end if
  end for
  // final layer neurons do not fire
  $U^t_L = U^{t-1}_L + W_L * O^t_{L-1}$
end for
//calculate loss, Loss=cross-entropy(U^T_L, Y)
// Backward Phase
for $t \leftarrow T$ **to** 1 **do**
  for $l \leftarrow L-1$ **to** 1 **do**
    // evaluate partial derivatives of loss with respect to weight by unrolling the network over time
    $\triangle W^t_l = \frac{\partial Loss}{\partial O^t_l} \frac{\partial O^t_l}{\partial U^t_l} \frac{\partial U^t_l}{\partial W^t_l}$
  end for
end for

2. Experimental details

2.1. Datasets and Models

We perform our experiments on VGG9 for CIFAR10 dataset, VGG11 for CIFAR100 and VGG13 for TinyImagenet. Some comparisons with other encoding schemes are done using VGG5.

2.2. Training Parameters

For all datasets, we follow some standard data augmentation techniques such as padding by 4 pixels on

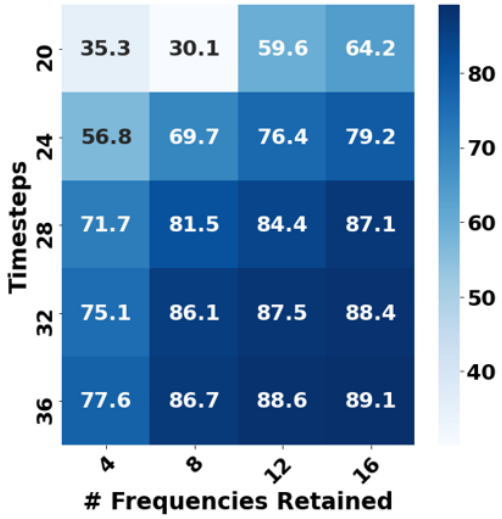

Figure 1: Training Accuracy with Limited Frequencies.

each side, and a 32×32 crop is randomly sampled from the padded image or its horizontally flipped version (with 0.5 probability of flipping). While testing, the original 32×32 images are used. Both training and testing data are normalized using 0.5 as mean and standard deviation for all channels. For training the ANNs, we use cross-entropy loss with stochastic gradient descent optimization (weight decay=0.0001, mometum=0.9). In the ANN domain, VGG5, VGG9 and VGG11 are trained for a total of 300 epochs, with an initial learning rate of 0.1, which is divided by 10 at each 100-th epoch. VGG13 with TinyImagenet is trained with similar learning rate schedule, but with initial learning rate of 0.01. The ANNs are trained with some architectural constraints to avoid significant loss during subsequent ANN-SNN conversion [4, 12]. The ANNs do not have bias terms as it increases the difficulty of threshold balancing. Again, batch-normalization is not used, rather dropout [13] is used as the regularizer and a constant dropout mask is used across all timesteps while training in SNN domain. Furthermore, average-pooling is used to reduce the feature map size since max-pooling causes significant information loss in SNNs [4]. During training the ANN, the weights are initialized using Xavier initialization [5]. After conversion, for training in the SNN domain, networks are trained for 20-30 epochs with cross-entropy loss and adam optimizer (weight decay=0.0005). Initial learning rate is kept at 0.0001, which is halved every 5 or 6 epochs. The leak constant λ is chosen as 0.9901 for all simulations.

3. Training with Curtailed Frequencies

We show the effect of training with limited frequencies in Figure 1. The frequencies are repeated cyclically until the specified number of timesteps. For instance, the accuracy for 8 frequencies given 3 times each (24 timesteps) is 69.7%. We note that during training, there is no benefit to dropping frequencies at iso-latency requirements.

4. Computational Cost

The equations for calculating the number of operations in a particular layer of an ANN are given by

$$\#\text{ANN}_{\text{ops}} = \begin{cases} k_w \times k_h \times c_{in} \times h_{out} \times w_{out} \times c_{out}, & \text{Conv layer} \\ n_{in} \times n_{out}, & \text{Linear layer} \end{cases} \quad (13)$$

where $k_w(k_h)$ denote filter width (height), $c_{in}(c_{out})$ is number of input (output) channels, $h_{out}(w_{out})$ is the height (width) of the output feature map, and $n_{in}(n_{out})$ is the number of input (output) nodes.

5. Performance Comparison with Different Encoding Schemes

5.1. Performance Comparison with Temporal Encoding Schemes on MNIST

Table 1: Accuracy of various temporal encoding schemes on MNIST

Reference	Accuracy(%)	Timesteps
Kheradpisheh et al. [7]	97.4	256
Comsa et al. [3]	97.9	not reported
Stephan et al. [14]	85	10
Yu et al. [17]	78	100
Xu et al. [16]	87	not reported
Beyeler et al. [2]	91.6	500
This work	98.54	16
This work	86.7	2
This work	97.3	5

To further compare the performance of the proposed DCT-SNN encoding scheme with recent temporal methods on MNIST, we implement it on a shallow network with just 1 hidden layer consisting of 784-100-10 neurons (all fully-connected). The results are reported in Table 1. As can be seen, our methods outperforms these recent temporal methods in terms of accuracy and also converges at much lower timesteps.

5.2. Performance Comparison with Different Encoding Schemes on CIFAR

To further compare our results with other reported works in terms of timesteps required at iso-accuracy level, we

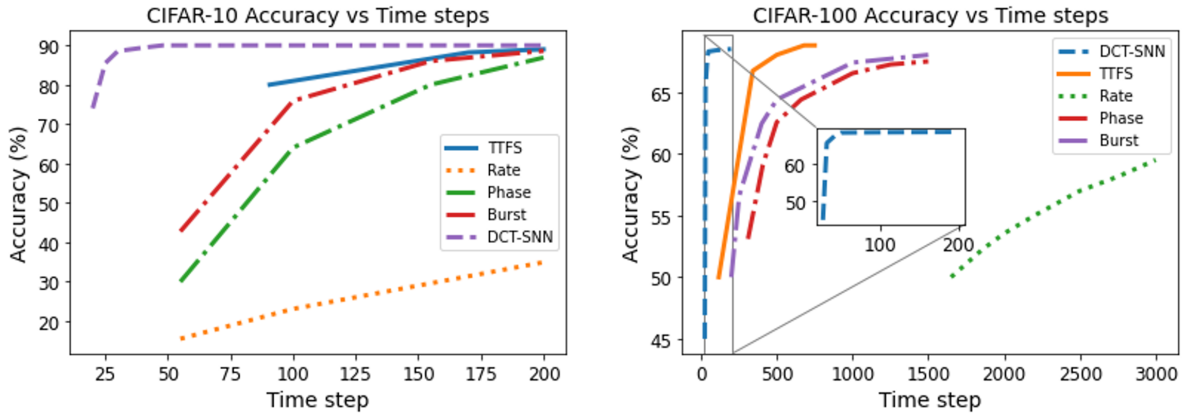

Figure 2: Accuracy versus Latency curve for various coding schemes, the values for TTFS [10], Phase [8], Burst [9] and rate [11] have been adopted from [10].

.

depict the inference curve across different timesteps in Fig. 2. The figure is adopted from Fig. 6 of [10] and demonstrates the results of "T2FSNN" encoding scheme, which is a temporal encoding scheme and other rate and temporal encoding schemes such as "Rate" [11], "Phase" [8], and "Burst" coding [9]. The left graph in Fig. 2 is recreated for CIFAR-10, and shows $\sim$ 200 timesteps for the fastest convergence among these encoding methods, in contrast, we achieve $\sim$ 90% accuracy in just 48 timesteps, saturating far earlier than any of these methods. From Fig. 6 of [10], we can tell that the best version of "T2FSNN" first reaches 90% roughly at 240 steps, "Burst" at 300, "Phase" at 425, and "Rate" at 1200 timesteps, showing that we reduce latency by orders of magnitude, resulting in convergence at much fewer timesteps. The network is slightly different here, ours is VGG9 and the network used in [10] is VGG16, but in our opinion, that affects final convergence accuracy more than it affects orders of magnitude of timesteps for inference. Similarly, we exceed 68% accuracy in 48 timesteps when training a VGG11 on CIFAR100. The graph on the right in Fig. 2 shows the convergence statistics for VGG16 on CIFAR100 using "T2FSNN", "Burst", "Phase" and "Rate". The best version of "T2FSNN" reaches 68% roughly at 500 steps, "Burst" at 1500, "Phase" at 2000, and "Rate" does not go above 60% in even 3000 timesteps.

6. Training with Interleaved and Intermittent Frequencies

In this section, we analyze the effect of training with (a) interleaved and (b) intermittent frequencies, instead of all 16 frequency components given in a cyclic order.

For the interleaved case, instead of giving input as frequencies 0,1,2,...15,0,1,2,....15,0,1,2,...15 we input them as 0,0,0,1,1,1,2,2,2, ...15,15,15. So, one specific basis is repeated for 3 subsequent timesteps before giving the next frequency as input. Our original cyclic scheme gave the best reported accuracy of 89.94% and the interleaved scheme only achieved 79.7%. A similar cyclic vs interleaving experiment was done with the frequencies limited to top 8, repeated for 3 cycles (24 timesteps). The cyclic scheme achieved 69.7% and the interleaved achieved 53.73%. We believe this drop is due to the resetting of membrane potential as it fires between timesteps, causing temporal dependency to be incorporated between different timesteps and interleaving cannot leverage this dependency. Additionally, since the earlier DCT coefficients contain most of the energy, the spikes at the later timesteps start dying out with interleaved frequencies.

Next, we tried giving intermittent frequencies such as (0,2,5,7,9,10,12,15) given cyclically for 6 cycles (48 timesteps) and got 84.4%, an expected drop from 89.9% with all frequencies for 3 cycles (equivalent 48 timesteps) since we are only giving partial information for reconstruction.

As an additional experiment to re-emphasize that our ordering is beneficial, we give only the top 8 frequencies for the same number of cycles as the previous experiment (6 cycles, 48 timesteps) and get 87.2%, which is 3% better than the scheme with intermittent frequencies, validating the importance of ordering timesteps.

References

[1] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, pages 787–797, 2018.

[2] Michael Beyeler, Nikil D Dutt, and Jeffrey L Krichmar. Categorization and decision-making in a neurobiologically plausible spiking network using a stdp-like learning rule. *Neural Networks*, 48:109–124, 2013.

[3] Iulia M Comsa, Thomas Fischbacher, Krzysztof Potempa, Andrea Gesmundo, Luca Versari, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8529–8533. IEEE, 2020.

[4] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. ieee, 2015.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[6] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

[7] Saeed Reza Kheradpisheh and Timothée Masquelier. S4nn: temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, 30(6):2050027, 2020.

[8] Jaehyun Kim, Heesu Kim, Subin Huh, Jinho Lee, and Kiyoung Choi. Deep neural networks with weighted spikes. *Neurocomputing*, 311:373–386, 2018.

[9] Seongsik Park, Seijoon Kim, Hyeokjun Choe, and Sungroh Yoon. Fast and efficient information transmission with burst spikes in deep spiking neural networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[10] Seongsik Park, Seijoon Kim, Byunggook Na, and Sungroh Yoon. T2fsnn: Deep spiking neural networks with time-to-first-spike coding. *arXiv preprint arXiv:2003.11741*, 2020.

[11] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

[12] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[14] Andrew Stephan, Brian Gardner, Steven J Koester, and Andre Gruning. Supervised learning in temporally-coded spiking neural networks with approximate backpropagation. *arXiv preprint arXiv:2007.13296*, 2020.

[15] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[16] Qi Xu, Yu Qi, Hang Yu, Jiangrong Shen, Huajin Tang, and Gang Pan. Csnn: An augmented spiking based framework with perceptron-inception. In *IJCAI*, pages 1646–1652, 2018.

[17] Qiang Yu, Huajin Tang, Kay Chen Tan, and Haizhou Li. Rapid feedforward computation by temporal encoding and learning with spiking neurons. *IEEE transactions on neural networks and learning systems*, 24(10):1539–1552, 2013.