SeFAct2: Selective Feature Activation for Energy-Efficient CNNs using Optimized Thresholds

Farhana Sharmin Snigdha[1], Susmita Dey Manasi[1], Jiang Hu[2], Sachin S. Sapatnekar[1]
[1]Department of ECE, University of Minnesota, [2]Department of ECE, Texas A&M University
e-mail: {sharm304,manas018,sachin}@umn.edu, jianghu@ece.tamu.edu

***Abstract*—This work presents a framework for dynamic energy reduction in hardware accelerators for convolutional neural networks (CNN). The key idea is based on the early prediction of the features that may be important, with the deactivation of computations related to unimportant features and static bitwidth reduction. The former is applied in late layers of the CNN, while the latter is more effective in the early layers. The procedure includes a methodology for automated threshold tuning to detect feature activation. For various state-of-the-art neural networks, the results show that energy savings of up to about 30% are achievable, after accounting for all implementation overheads, with a small loss in the accuracy.**

***Index Terms*—Low-power design; CNN; deep learning; neural network; energy optimization**

I. Introduction

Deep neural network (DNN) architectures [2], [3] have multiple layers consisting of artificial neurons. Each layer is trained to recognize various features of the input, with deeper layers uncovering more complex features. In recent years, convolutional neural networks (CNNs) have emerged as a prominent branch of DNNs, and have been shown to solve increasingly complex problems with remarkably high accuracy [4]. As DNNs have grown deeper, adding more layers and features [3], their computational requirements have rapidly increased. As a result, platforms that implement DNNs in hardware are power-hungry and require large memory footprints.

Prior works have proposed several methods for reducing computations and computation latency [5], including data parallelization using multiple cores, specialized hardware [6]–[8], and approximate computation based on simplified architecture and static pruning methods [9]–[15]. Typically, the deeper layers of the network have higher error sensitivity compared to the earlier layers [10]. Hence, the architecture approximation becomes unprofitable in terms of accuracy and energy savings. In this work, we identify specific DNN properties related to how features are mapped to neurons in various layers that can provide further improvements in computational efficiency.

We make two observations regarding the feature activation patterns in CNNs. First, during both training and testing, specific features tend to be activated by the recognition of specific classes. Second, deeper layers have higher neuron activation sparsity, i.e., fewer neurons are activated per layer [16]. Some paths through the network are unlikely to be activated because specific sets of neurons are seldom activated together. These properties can be leveraged to switch off unnecessary computations to save energy, and we propose a selective feature activation approach, SeFAct, to develop quantitative metrics for identifying such scenarios, which are used to drive runtime energy-reduction optimizations. Specifically, during training, we prepare for selective feature activation by grouping similarly activated classes into clusters across multiple classes. During testing, unlike well-known static pruning methods [13]–[15], [17]–[19], we perform dynamic pruning: we use the clusters to dynamically approximate a large section of the CNN that does not help interpret the input. This optimization is particularly beneficial in the later layers of a DNN. The energy requirement of a neural network comes from two parts, data access from various hierarchies of memory, and computation of neuron activations. The memory access is the dominating part of the energy consumption in a neural network. Our proposed method achieves energy savings by reducing both the number of memory accesses and the computations. We couple these gains with reduced bitwidth in earlier layers of the DNN, again without significantly affecting accuracy.

A preliminary version of this work appeared in [1]. This work was supported in part by the NSF under awards CCF-1525925, CCF-1763761, CCF-1525749.

II. CNN Architecture

A CNN consists of various combinations of layers such as convolutional (*Conv*), fully connected (*FC*), pooling (*Pool*) as well as normalization (*Norm*) layers [3].
Conv Layer Each of the *Conv* layers is composed of high-dimensional convolutions. There are three types of data associated with a *Conv* layer:

- **ifmap**, the input feature map, $F^{if} \in \mathbb{R}^{K^- \times H^- \times W^-}$, which comes from the computations in layer L_{i-1}.
- **filter**, the filter weights, $F^{filter} \in \mathbb{R}^{K \times K^- \times d \times d}$, i.e., K 3D filters with each feature dimension $d \times d$ are applied to the ifmap, and
- **ofmap**, the output feature map, $F^{of} \in \mathbb{R}^{K \times H \times W}$, which acts as the ifmap for layer L_{i+1}.

For a stride size U under bias $F^{bias}[u]$, the computation in layer L_i is given by [3]:

$$F^{of}[u][x][y] = \text{ReLU}\Bigg(F^{bias}[u] + \sum_{k=1}^{K^-}\sum_{i=1}^{d}\sum_{j=1}^{d} F^{filter}[u][k][i][j] \times F^{if}[k][Ux+i][Uy+j]\Bigg) \quad (1)$$

where, $1 \le u \le K, \;\; 1 \le x \le H, \;\; 1 \le y \le W$

The k^{th} feature of F^{if} is represented by the 2D feature plane, $F^{if}_k \in \mathbb{R}^{H^- \times W^-}, 1 \le k \le K^-$. The rectified linear

unit (ReLU) [20], a nonlinear activation function, introduces nonlinearity and sparsity into the CNN using $\text{ReLU}(x) = \max(0, x)$. It is typically applied after each *Conv* or *FC* layer.
FC Layer An *FC* layer also applies filters on the *ifmaps* as in the *Conv* layers, but the filters are of the same size as the *ifmaps*. Equation (1) still holds for the computation of *FC* layers with additional constraints on the shape parameters: $H^- = W^- = d, H = W = 1$, and $U = 1$.
Pool and Norm Layer The *Pool* and *Norm* layers are used to realign the data for faster training as well as dimensionality reduction process [3]. A *Pool* layer is typically applied to each 2D feature plane of a *Conv* layer to reduce the spatial size of the feature map representation. In a normalization (*Norm*) layer, the distribution of the input features are normalized such that the data has a zero mean and a unit standard deviation. Normalizing the input distribution across layers can help significantly speed up training and improve accuracy.
Network in Network Layer Various state-of-the-art networks such as GoogLeNet [21] and SqueezeNet [22] incorporated *network in network* (NIN) [23] modules named *inception* and *fire*, respectively, to improve accuracy. The NIN modules can have multiple parallel paths unlike a conventional CNN (i.e. LeNet [24], AlexNet [25]), where only a single convolution path exists. The use of multiple filter sizes has the effect of processing the input at multiple scales. In addition, intermediate layers with 1×1 filters are applied to reduce the number of computations as well as to increase the depth. The idea of processing the input at multiple scales and using a higher depth is seen to improve the CNN accuracy significantly.

III. Overview of the Selective Feature Activation

A. Motivating Example and Outline of SeFAct

We illustrate our key idea on a simplified neural network for letter recognition with three fully connected layers, L_1, L_2, and L_3, shown in Fig. 1. We later implement this idea on larger, standard CNN topologies. Each neuron represents one feature, and the outputs map to six classes, corresponding to the letters A through F. Two feature activation patterns are shown in Fig. 1, where the activated neurons for classes B and D are highlighted and the unactivated neurons are white.

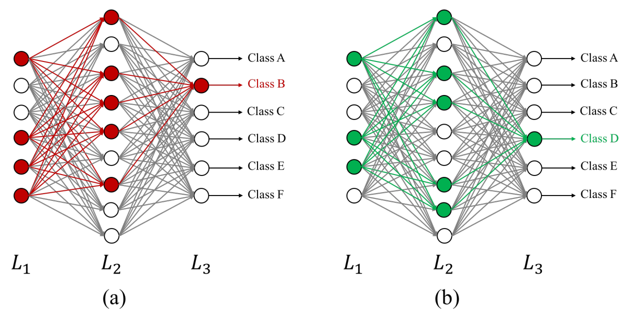

Fig. 1: Activation pattern of features for (a) Class B and (b) Class D in layers, L_1 to L_3 of an example neural net.

It can be seen that the activation patterns of classes B and D are very similar. Here, class B (D) has four (three) and five activated features in layer L_1 and L_2, respectively. Among these three/four (five) activated features in L_1 (L_2), three (four) i.e. 75% or more activated features are common. Based on this high feature activation similarities, we can cluster these two classes together in training phase. Additionally, we mark these three (four) commonly activated features in $L_1(L_2)$ as the "stamp" of the cluster for the corresponding layer. During the testing phase, if the feature activation pattern of an input matches with a particular stamp, then the input is said to belong to one of the classes from the corresponding cluster. This clustering helps predict classes in early layers, and can be used to reduce both computation and memory access energy in the subsequent layers. For example, the feature activation pattern of class B or D will match the cluster stamp prepared in layer L_2. Hence, we can early predict the input class and narrow down the classification options from six to two in layer L_3, and skip computations and data loads for other classes.

We build a framework, SeFAct, that systematically identify *important features* in each layer of the network based on four thresholds T_1, T_2, T_3 and T_4 to selectively activate neurons. The framework has two major steps:

1) Cluster learning phase: We augment the traditional training phase to identify clusters. The trained weights and training data are used in this step. The additional steps are:

(a) Class-specific important feature identification using parameters, T_1 and T_2. The usage of these thresholds are discussed in Section III-B1 and III-B2 respectively.
(b) Cluster preparation based on threshold T_3 that measures the feature activation similarities (Section III-B3).
(c) Stamps and cluster data preparation for the testing phase (Section III-B4).

2) Testing phase: The traditional testing phase is modified to use the cluster learning results. At each CNN layer, we

(a) compare feature activation patterns with the cluster stamps and identify the activated cluster(s) based on a threshold parameter, T_4.
(b) load and compute data only for the predicted class(es).
(c) propagate the predicted class sets to the next layer.

The cluster learning phase is a preprocessing step associated with training. Changes to the testing phase incur overheads, but also generate savings, and is implemented in real-time. The overall testing phase is discussed in Section III-C.

B. Cluster Learning Phase

1) The Concept of Important Features: The input feature map, i.e., *ifmap* (F^{if}), is used to detect important features based on threshold T_1. In this section, we explain how we identify the important features for a single input image, using the notation for the DNN defined in Section II. Threshold T_1 can be used to modulate the important feature count and hence tune classification accuracy and energy savings.

For the ifmap, the summation of all data in the k^{th} feature plane, $\tilde{F}_k^{if} = \sum_{h=1}^{H^-} \sum_{w=1}^{W^-} F_{khw}^{if}$, is a "signature" for the feature. We calibrate the importance of a feature in a layer based on the relative magnitude of $\tilde{F}_k^{if}$ with respect to the average of all feature plane data, $\tilde{F}_{avg}^{if} = \sum_{k=1}^{K^-} \tilde{F}_k^{if} / K^-$.

Definition 1. *The indicator function* $1\{\cdot\}$ *is defined as* $1\{true\} = 1$*, and* $1\{false\} = 0$*.*

Definition 2. *Feature* k *is considered important if it satisfies*

$$I_k = 1\left\{\tilde{F}_k^{if} \geq T_1 \times \tilde{F}_{avg}^{if}\right\} \quad (2)$$

where T_1 *is a tunable threshold parameter.*

Computational simplification: In simple terms, Eq. (2) states that an ifmap feature is important if it is within a multiplicative factor T_1 of the average of all feature plane data. While this equation can be applied relatively easily to an *FC* layer where $H^- = W^- = 1$ and the volume of data is moderate, *Conv* layers must handle a much larger volume of data. In a realistic hardware implementation, the 3D data processed by a *Conv* layer of the CNN is fetched through a memory hierarchy. It is computationally expensive to fetch the data and compute $\tilde{F}_{avg}^{if}$ for use in Eq. (2). Moreover, ifmap data is very sparse, i.e., numerous elements are zero.

We simplify the computation of $\tilde{F}_{avg}^{if}$ based on mean and sparsity information of the *ifmap*, pre-calculated during the cluster learning phase using sampling. This helps to reduce computation without hurting the effectiveness of our method.

Theorem 1. *Under the above simplified assumption, feature* k *in a Conv layer is important, as defined in Definition 2, if*

$$I_k = 1\left\{(1 - S_k^{if})\mu_k^{if} \geq T_I\right\}, \quad (3)$$

where $T_I = T_1 \times (1 - S^{if}) \times \mu^{if}$*. We assume, the overall sparsity,* $S^{if} = \sum_{k=1}^{K^-} S_k^{if}/K^-$*, and the average of all nonzero data,* μ^{if}*, are constants for an* ifmap *layer.*

Proof. The number of nonzero elements associated with the k^{th} feature plane is $H^-W^-(1 - S_k^{if})$. If the average of all nonzero data in the feature plane is μ_k^{if}, we can write, $\tilde{F}_k^{if} = H^-W^-(1 - S_k^{if})\mu_k^{if}$. The Eq. (2) for a *Conv* layer then becomes:

$$I_k = 1\left\{H^-W^-(1 - S_k^{if})\mu_k^{if} \geq T_1 \times \frac{\sum_{k=1}^{K^-} H^-W^-(1 - S_k^{if})\mu_k^{if}}{K^-}\right\}$$

$$I_k \approx 1\left\{(1 - S_k^{if})\mu_k^{if} \geq T_1 \times (1 - S^{if}) \times \mu^{if}\right\}$$
$$= 1\left\{(1 - S_k^{if})\mu_k^{if} \geq T_I\right\} \quad (4)$$

The latter approximation operates under the assumption that the average of all feature plane data, $\tilde{F}_{avg}^{if} = (1 - S^{if}) \times \mu^{if} H^-W^-$, is *constant*, as H^-, W^-, S^{if} and μ^{if} are considered constants for a given *ifmap* layer. □

2) Important Feature Detection for Each Class: The relation in Eq. (2) shows how the k^{th} element of the class-based importance feature vector, $I \in \mathbb{R}^{K^-}$, is used to develop the importance criterion for the features of an *individual* input image in layer, L_i. However, during the cluster training phase, we work with *multiple* images in multiple classes to identify important features for the cluster. Threshold T_2 is used to determine whether a certain feature is important for all images of an input class.

Let us say that we apply Eq. (2) in layer, L_i, to determine the importance feature vector I' of the q^{th} image of one particular class c, i.e., $I'(c, q) = I$. During cluster learning, we repeat this operation over all N_{class} classes of the CNN, based on the learned data from $N_{image}(c)$ images for each class c. Due to the network training inaccuracies, or image pixel pattern variations, all the images with same class may not have the same activation pattern for a specific layer. Therefore, we deem a feature to be important for a particular class if it is important for a sufficiently large number of images in the class, i.e., if it is activated for a fraction T_2 of all training images. This provides a criterion for determining the important features through the vector $\tilde{I}_c \in \mathbb{R}^{K^-}$, for class c of the current layer:

$$\tilde{I}_c = 1\{\textstyle\sum_{q=1}^{N_{image}(c)} I'(c, q) \geq T_2 \times N_{image}(c)\} \quad (5)$$

3) Clustering: Classes with closely matched features are now grouped together as clusters based on a cluster affinity threshold, T_3. The clusters are also needed to satisfy a few additional properties. We first list the properties below and then illustrate the idea of clustering through an example.

Property: A cluster is a set of classes, with one or more clusters associated with each layer, satisfying these properties:

1) Clusters in the last layer are singleton sets, with each set containing one class.
2) No cluster is a subset of another cluster in the same layer.
3) If $\mathcal{C}$ and $\mathcal{C}^+$ are the sets of clusters in layer L_i and layer L_{i+1}, then each cluster $\mathcal{C}_x$ in layer L_i must satisfy

$$\mathcal{C}_y^+ \subseteq \mathcal{C}_x \quad \text{for some } y.$$

We refer to this as the **diverging clustering criterion**. We define a cluster graph in which each cluster forms a vertex, and an edge is drawn from x to y if the above criterion is satisfied. Moreover,

$$\mathcal{C}_x = \textstyle\bigcup_{x \to y} \mathcal{C}_y^+.$$

Fig. 2 shows a sample clustering of the network of Fig. 1. Clusters in the last layer are singletons, consistent with Property 1 above, and no cluster within the same layer is a subset of another, as stated in Property 2. Property 3 can be illustrated by cluster $\mathcal{C}_3^{L_1} = \{B, D, E, F\}$ of layer L_1, which diverges through its connections to clusters $\mathcal{C}_2^{L_2} = \{B, D\}$ and $\mathcal{C}_4^{L_2} = \{B, E, F\}$.

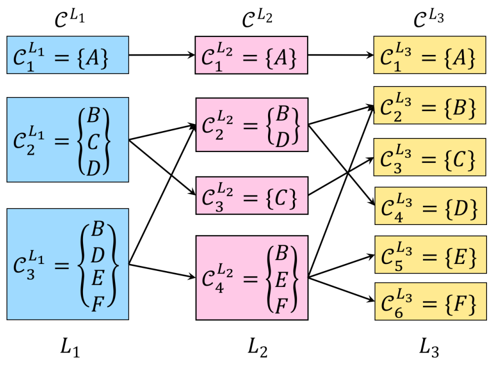
$\mathcal{C}^{L_1}$
$\mathcal{C}^{L_2}$
$\mathcal{C}^{L_3}$
$\mathcal{C}_1^{L_1} = \{A\}$
$\mathcal{C}_2^{L_1} = \{B, C, D\}$
$\mathcal{C}_3^{L_1} = \{B, D, E, F\}$
$\mathcal{C}_1^{L_2} = \{A\}$
$\mathcal{C}_2^{L_2} = \{B, D\}$
$\mathcal{C}_3^{L_2} = \{C\}$
$\mathcal{C}_4^{L_2} = \{B, E, F\}$
$\mathcal{C}_1^{L_3} = \{A\}$
$\mathcal{C}_2^{L_3} = \{B\}$
$\mathcal{C}_3^{L_3} = \{C\}$
$\mathcal{C}_4^{L_3} = \{D\}$
$\mathcal{C}_5^{L_3} = \{E\}$
$\mathcal{C}_6^{L_3} = \{F\}$
L_1
L_2
L_3

Fig. 2: Example to illustrate the properties of clusters.

The diverging clustering criterion allows the network to reduce the number of predicted classes monotonically. For example, if an image of the letter, E, is provided to the network, then the probable classes for layer L_1, L_2, and L_3 are $\{B, D, E, F\}$, $\{B, E, F\}$ and $\{E\}$, respectively.

The class-based importance vector, $\tilde{I}_c$, obtained in (5), lists the set of important features in a layer. We use this information to create diverging clusters from layer to layer. We begin by creating singleton clusters in the last layer; the number of clusters equals the number of classes. Then, we topologically move backwards maintaining the diverging clustering criterion.

For each cluster $\mathcal{C}_k^+$ of layer L_{i+1}, we prepare a cluster-based important feature vector for layer L_i, J_k, that lists the features important to all cluster members, using the AND ($\bigwedge$) operator:

$$J_k = \bigwedge\nolimits_{c \in \mathcal{C}_k^+} \tilde{I}_c \tag{6}$$

We then combine clusters in layer L_{i+1} into larger clusters in layer L_i. We create a graph $G(V, E)$ whose vertex set $V = \mathcal{C}^+$, and with edge set E connecting clusters that have high affinity, but keeping clusters with low affinity unconnected. The affinity metric for clusters $\mathcal{C}_i^+$ and $\mathcal{C}_j^+$ is given by $A_{ij} = \langle J_i, J_j \rangle$, the inner product between J_i and J_j. Since these vectors are Boolean, the inner product measures the number of important features that these clusters have in common. We add an unweighted edge to E if the affinity exceeds a threshold, $T_3 \times A_{max}$, where A_{max} is the maximum of all cluster affinities:

$$(i, j) \in E \text{ iff. } \{A_{ij} > T_3 \times A_{max}\} \qquad 1 \le i, j \le N_{\mathcal{C}^+} \tag{7}$$

The goal of clustering is to group the high-affinity clusters in layer L_{i+1} together. We map this problem to the *maximum independent set* (MIS) problem on graphs [26]. We defined any two vertices as independent if they have no edge between them, i.e., they have low affinity. The MIS problem finds a maximal set of independent vertices, which act as roots of individual clusters. By definition, elements of the MIS have low affinity for each other and should be in different clusters. Each cluster is thus defined by an MIS element, and includes its high affinity neighbors, i.e., classes, in Algorithm 1.

The algorithm finds the clusters in layer L_i by first identifying the vertices in G (i.e., cluster IDs) to be merged based on finding the MIS. Since the MIS problem is NP-hard, we employ a greedy heuristic [27]. After initialization (line 1), the algorithm sorts the vertices in non-ascending order of their degrees (line 2). In each iteration, the minimum degree node in V is added to the MIS (line 5). A new cluster is formed, combining the clusters associated with v and its neighbors with at least equal degree of v (line 7). The neighbors with lower degree than v are already included in previously formed clusters that should not be combined with the current cluster, since other elements of the other cluster may not have much affinity with the current cluster. Hence, this additional check helps creating more disjoint clusters as well as increasing affinity between classes of same cluster. The iteration ends with v and its neighbors eliminated from V (line 8). Once all clusters have been found, the cluster IDs in $\mathcal{D}^+$ are used to construct the set of classes in the cluster set $\mathcal{C}$ (line 11). The complexity of the algorithm is $\mathcal{O}\left(N_{\mathcal{C}^+} \log\left(N_{\mathcal{C}^+}\right)\right)$, which is dominated by the sorting operation of the vertices (line 2).

Algorithm 1 $\{\mathcal{D}^+, \mathcal{C}\} \leftarrow$ FindCluster$(G, \mathcal{C}^+)$

```
INPUT: Edge connectivity graph $G(V,E)$ based on $\mathcal{C}^+$
Cluster set of layer $L_{i+1}$: $\mathcal{C}^+$
OUTPUT: Diverging cluster set: $\mathcal{D}^+$          ▷ IDs of clusters to be merged
Cluster set of layer $L_i$: $\mathcal{C}$                 ▷ Set of classes in the cluster
METHOD:
 1: Initialize $I_s = \varnothing$, $\mathcal{D}^+ = \varnothing$
 2: Sort the vertices in $V$ in non-ascending order of degree
 3: while $V \neq \varnothing$ do
 4:     Set $v$ to be the minimum-degree vertex in $V$
 5:     $I_s = I_s \cup v$                         ▷ Add $v$ to the independent set
 6:     $n_v \leftarrow \{u\} \forall (u,v) \in E$ and degree$(u) \ge$ degree$(v)$   ▷ Set of neighbors of $v$
 7:     $\mathcal{D}^+ \leftarrow \mathcal{D}^+ \cup \{\{v\} \cup n_v\}$      ▷ Add $v$ and its neighbors to $\mathcal{D}^+$
 8:     $V \leftarrow V \setminus \{\{v\} \cup n_v\}$
 9: end while
10: $\forall \mathcal{D}_k^+ \in \mathcal{D}^+$, $\mathcal{C}_k \leftarrow \cup_{v \in \mathcal{D}_k^+} \mathcal{C}_v^+$     ▷ Build clusters using cluster IDs
11: $\mathcal{C} \leftarrow \bigcup \mathcal{C}_k$                      ▷ Set of all clusters at layer $L_i$
12: return $\{\mathcal{D}^+, \mathcal{C}\}$
```

4) Data Preparation for Testing Phase: We now encapsulate clustering information to enable its efficient use during the testing phase. At each layer L_i, the set $\mathcal{D}^+$ shows how the clusters in the current layer diverge to those in the next layer.

We prepare *stamps* for each cluster in layer L_i, corresponding to the features that can activate the cluster, i.e., the features that are important to all classes in the cluster. These stamps are used in the testing phase to determine the activated clusters, by comparing the list of important features in the input data with each cluster stamp. The stamp $\mathcal{S}_k$ for cluster k is:

$$\mathcal{S}_k = \bigwedge\nolimits_{c \in \mathcal{C}_k} \tilde{I}_c \tag{8}$$

For each cluster $\mathcal{C}_k$ in layer L_i, we now create a record of the features to be computed, $\mathcal{F}_k^+$, to identify potentially activated clusters in layer L_{i+1}. During the testing phase, for each activated cluster $\mathcal{C}_k$, only these features are inspected. For cluster $\mathcal{C}_k$, we compute $\mathcal{F}_k^+$ by combining, through a binary OR ($\bigvee$), the important features of layer L_{i+1} as:

$$\mathcal{F}_k^+ = \bigvee\nolimits_{c \in \mathcal{C}_k} \tilde{I}_c^+ \tag{9}$$

For example, in Fig. 2, the stamps for all clusters in layer L_1 are used to check which classes are activated. In the testing phase, for an input image B, depending on which features are activated, the stamps for $\mathcal{C}_3^{L_1}$ could trigger the identification of this cluster.[1] Next, from $\mathcal{D}_3^+$, we know that the activated clusters in layer L_2 may be $\mathcal{C}_2^{L_2}$ and $\mathcal{C}_4^{L_2}$. Accordingly, we use the list of important features given by Eq. (9) to compute the important features for classes in clusters $\mathcal{C}_2^{L_2}$ and $\mathcal{C}_3^{L_2}$. If any other cluster is activated, then a similar approach is used to add to the list of important features to be computed.

5) Overall Algorithm for Cluster Learning Phase: Algorithm 2 summarizes the cluster learning phase for layer L_i. Lines 1 and 2 prepare, respectively, the class-based and cluster-based importance feature vectors at level L_i. Based on the clusters and their affinities, the cluster graph G is formed. Next, Algorithm 1 is invoked to form the diverging clusters. Finally, in preparation for the testing phase, for each cluster, a cluster stamp and a record of important features for the next level are computed.

[1] Note that not all important features of a class are activated by each image: therefore, an image in class B may well activate only $\mathcal{C}_3^{L_1}$ and not $\mathcal{C}_2^{L_1}$.

Algorithm 2 The Cluster Learning Phase

INPUT: Layer L_i: ifmap data F^{if}; thresholds T_1, T_2, T_3
Layer L_{i+1}: importance vectors $\tilde{I}_c^+ \forall$ classes c; cluster set $\mathcal{C}^+$
OUTPUT: Layer L_i: importance vectors $\tilde{I}_c \forall$ classes c; cluster set $\mathcal{C}$; cluster stamps $\mathcal{S} \in \mathbb{R}^{N_{\mathcal{C}} \times K^-}$,
Layer L_{i+1}: divergent cluster set $\mathcal{D}^+$; features $\mathcal{F}^+ \in \mathbb{R}^{N_{\mathcal{C}} \times K}$
METHOD:

```
1: Create importance vectors for layer $L_i$, $\tilde{I}_c$        ▷ Use $T_1$, $T_2$, and (5)
2: Form cluster importance vectors in layer $L_i$, $J$        ▷ Use $\mathcal{C}^+$ and (6)
3: Create $G(V, E)$        ▷ Use (7) and $T_3$
4: $\{\mathcal{D}^+, \mathcal{C}\} \leftarrow$ FindCluster($G, \mathcal{C}^+$)        ▷ Algorithm 1
5: for $k = 1 : N_{\mathcal{C}}$ do
6:     Prepare stamp, $\mathcal{S}_k$; important features, $\mathcal{F}_k^+$        ▷ Use (8), (9) and $\tilde{I}_c^+$
7: end for
8: return
```

C. Testing Phase

In the *cluster testing phase*, the cluster activation pattern for each layer is identified. Based on this cluster activation, the class predictions are updated which helps to reduce computation for future layers. The identification of activated clusters is regulated by the tunable threshold, T_4.

The cluster testing phase for layer L_i is described in Algorithm 3. Note that this step is similar to the cluster learning phase, except here we work with only one test image at a time in real-time. Here, we will simultaneously discuss the overhead cost associated with each step. In Algorithm 3, first, we quantify the mismatch between important feature pattern of input class and cluster stamps (line 1). The L1 norm used in this computation is the sum of absolute differences of each element of the vector. The mismatch computation is performed inexpensively using summations of one-bit numbers. Here, we identify cluster activation for only $\tilde{D}$ clusters. This narrowed down cluster activation check significantly reduces energy overhead of SeFAct implementation. Next, the activated clusters are identified based on the following equation:

$$\mathcal{A}_k = 1\{\mathcal{M}_k \le \mathcal{M}_{min} + T_4 \|\mathcal{S}_k\|_1\}, \quad k \in \tilde{\mathcal{D}} \tag{10}$$

Here, minimum mismatch, $\mathcal{M}_{min}$, provides a minimum margin for cluster activation and $T_4 \times \|\mathcal{S}_k\|_1$ provides additional cluster specific tunability to modulate the classification accuracy. The implementation cost of line 2 is linear in the number of clusters and only require a few addition and shift operations.

Algorithm 3 The Cluster Testing Phase

INPUT: Layer L_i: ifmap F^{if}; filter F^{filter}; bias F^{bias}; importance features, I ; cluster set $\mathcal{C}$; cluster stamps $\mathcal{S}_k \forall$ cluster k; activated clusters $\tilde{\mathcal{D}}$; features $\mathcal{F}$; threshold T_4
Layer L_{i+1}: divergent cluster set $\mathcal{D}^+$; threshold T_1^+
OUTPUT: Layer L_i: ofmap data: F^{of}
Layer L_{i+1}: activated clusters $\tilde{\mathcal{D}}^+$; important features: I^+
METHOD:

```
1: Find cluster mismatch, $\mathcal{M}_k = \sum_{k \in \tilde{\mathcal{D}}} \|\mathcal{S}_k \wedge \neg I\|_1$
2: Obtain cluster activation vector $\mathcal{A}$ using (10)
3: Obtain features to compute for ofmap, $\tilde{\mathcal{F}} = \vee_{\mathcal{A}_k=1} \mathcal{F}_k$
4: Compute ofmap using (1) and $\tilde{\mathcal{F}}$
5: Obtain clusters to check for layer, $L_{i+1}$, $\tilde{\mathcal{D}}^+ = \bigcup_{\mathcal{A}_k=1} \mathcal{D}_k^+$
6: Detect important features for layer $L_{i+1}$, $I^+$        ▷ Use (2) and $T_1^+$
7: return
```

Next, the cluster activation information is used to update class predictions and identify the important features of ofmap for the predicted classes, $\tilde{\mathcal{F}}$ (line 3). It is used to compute reduced ofmap data (line 4). This is the prominent energy savings step. Here, we only load the ifmap and filter data based on the important features, I, which reduces the memory access cost as well as computational cost. In Section V, we will discuss about reduced memory access and computation related energy savings. Line 5 prepares cluster activation flags to limit computations at layer L_{i+1} which is similar as line 3. Line 6 detects the important features of ofmap in layer L_i, and only these features of the ofmap are written into the memory. This reduces the memory overhead and also effectively further increases the inherent sparsity (due to ReLU) for level L_{i+1}, which uses this ofmap as its ifmap. From (2), the energy overhead for detecting important features arises from (i) computation of the threshold and (ii) a comparison operation. The threshold computation and comparison can be simplified to a few fixed bit addition and multiplication operations. For the *Conv* layer, the check is simplified to (3), where the sparsity summation involves the addition of one-bit zero flags, an inexpensive operation. All the decision metrics are stored as single-bit register files and used in inexpensive decision circuitries to improve energy savings.

D. SeFAct Implemention in Various Layers

The *cluster learning* and *testing* phase of SeFAct implementation in a layer, as described in Section III-A, is a complex process which requires the two steps listed below:

- Step 1 Cluster preparation, class prediction updating and important feature identification
- Step 2 Computation reduction by detecting unimportant features based on the updated class prediction

The implementation of cluster preparation and prediction (*Step* 1) incurs energy overheads over the basic (non-SeFAct) implementation, whereas *Step* 2 is expected to substantially reduce energy, paying for the overhead of *Step* 1, by skipping the computation of unimportant features. Algorithms 2 and 3 are used together to implement these two major steps of SeFAct implementation. We implement SeFAct on various types of CNN layers (described in Sections II) as follows:

Conv and FC Layers Both steps of SeFAct are implemented in the *Conv* and *FC* layers, i.e., both layers prepare clusters during the cluster learning phase and update the class prediction to reduce computation in the cluster testing phase.

Pool and Norm Layer The *Pool* and *Norm* layers are used to realign the *Conv* layer data for faster training as well as dimensionality reduction process. Therefore, these layers are mere representation of its immediate predecessor *Conv* layer and important features of both layers are the same. Hence, both of these layers do not incur any computational overhead, as they receive SeFAct information from previous *Conv* layer: cluster prediction and importance of various feature planes. We skip the *pooling/normalization* operation for the unimportant feature planes.

NIN Module The *network-in-network* (NIN) concept incorporates additional nonlinearity by introducing intermediate nonlinear layers as described in Section II. There can be multiple intermediate layers in a NIN module [21]. To limit the energy overhead, we do not implement *Step* 1 in the intermediate layers of an NIN modules, and only propagate the cluster prediction from the previous layer to reduce computation (*Step* 2). However, there is a difference between *Pool/Norm* layers and intermediate layers of NIN module with regard to important feature detection. The *Pool* and *Norm* layers are mere modifiers of their previous *Conv* layer and they have the same important feature pattern. On the other hand, the intermediate layers of an NIN module are generally implemented as *Conv* layers. Therefore, the important feature patterns of intermediate layers may not be the same as those in their predecessor layer. We accommodate this difference in *Step* 2 by implementing certain additional measures in Algorithm 2 (cluster learning algorithm) as well as Algorithm 3 (cluster testing algorithm). Specifically:

- We compute class-based important features ($\tilde{I}_c$) as well as predictive data, cluster-activation-specific important features ($\mathcal{F}_k^+$) for a specific layer in Algorithm 2 (lines 1 and 6). We also determine these two metrics, $\tilde{I}_c$ and $\mathcal{F}_k^+$, for the additional intermediate layers.
- During the testing phase (Algorithm 3), important features for the predictive classes ($\tilde{\mathcal{F}}$) are computed to save *ofmap* computations (lines 3 and 4). We additionally compute $\tilde{\mathcal{F}}$ flags for the intermediate layers to identify their important features and reduce computations.

IV. Design Optimizations for Energy Reduction

We enable energy-efficient neural computation by combining selective feature activation, SeFAct, described in Section III, with optimized reduced-precision approximation. Reduced precision schemes have been explored in recent research [6], [9] as well as commercial platforms [2], [7]. Some approaches have used fixed bitwidths (for example, 8-bit [2]) for all the layers. Other approaches [28] have used layerwise bitwidth optimization for controlled error introduction and improved power savings. The reduced precision approximation and our SeFAct approach are two orthogonal processes that introduce controlled levels of error in network to achieve energy savings. We obtain optimized bitwidths for various layers with Monte-Carlo simulations.

A. Choice of Layers for Selective Activation Implementation

The SeFAct scheme is useful in network layers where a relatively few features are activated for each class. However, in early layers, individual neurons do not have enough information from the input to narrow down the set of possible classes, and many neurons may be activated, regardless of the class.

An alternative way to explain the usefulness of SeFAct implementation in later layers is through the concept of the *receptive field* [24]. The receptive field of a neuron is the region in the input image that affects the neuron. The dimensions of the square receptive field for different layers of LeNet, AlexNet, and GoogLeNet are given in Tables I, respectively. The size of input images for LeNet (AlexNet/GoogLeNet) is 28×28 (227×227). There are multiple parallel paths for GoogLeNet with three different filters sizes (1×1, 3×3, 5×5) [21]. We use the median filter size, 3×3, to compute the receptive field.

TABLE I: Layerwise receptive fields of various networks.

Layer	Dimension	Layer	Dimension
LeNet			
$c1$	5	$p2$	16
$p1$	6	$fc1$	28
$c2$	14	$fc2$	28
AlexNet			
$c1$	11	$c5$	163
$p1$	19	$p5$	195
$c2$	51	$fc6$	355
$p2$	67	$fc7$	355
$c3$	99	$fc8$	355
$c4$	131		
GoogLeNet			
$c1$	7	$inception4b$	137
$p1$	11	$inception4c$	171
$c2/reduce$	11	$inception4d$	203
$c2$	19	$inception4e$	235
$p2$	27	$p4$	267
$inception3a$	43	$inception5a$	331
$inception3b$	59	$inception5b$	395
$p3$	75	$p5$	459
$inception4a$	107	$fc1$	459

To improve energy savings, SeFAct should be implemented at the earliest possible layer. However, the neurons in a specific layer can characterize the classes only if they see enough of the image to identify specific objects. Neurons in layer $c2$ of LeNet, AlexNet and GoogLeNet process information about $(14/28)^2 = 25\%, (51/227)^2 = 5\%$ and $(19/227)^2 = 0.7\%$ of the input image. Empirically, we choose to implement SeFAct from the layers whose receptive field covers about a quarter of the image, namely, from $c2$, $c5$ and $inception4b$ onwards in LeNet, AlexNet and GoogLeNet, respectively.

B. Choice of Data Bitwidth for Various Layers

SeFact cannot be implemented in the early layers of a CNN as they are unable to process enough data to correlate to specific classes due to limited receptive field. However, these early layers are highly error resilient. This provides an opportunity to implement error sensitivity based circuit approximations for early layers. The reasons are as follows:

- The number of resilient neurons is significantly higher in initial layers of the network [10] in comparison to later layers. The reason is, neurons in the initial layers typically process features local to a certain region of the image, while the later layer neurons infer global features from the previously extracted local features.
- Errors in neurons near the inputs are more likely to be compensated/filtered out later in the network.

Therefore, we have achieved power savings through reduced precision in early layers along with our selective activation approach for deeper layers.

V. Hardware Implementation

We implement our SeFAct scheme in combination with *optimized reduced precision bitwidths* in the testing phase. The *baseline implementation* of the testing phase is performed in three steps in each layer, L_i. First, the ifmap and the filter are loaded from the memory. Next, multiply-accumulate (MAC) operations are performed to compute the ofmap of layer L_i based on (1), and data is written back to the memory. The ofmap computation leverages the ifmap data sparsity.

Memory hierarchies are used in neural network accelerators to reduce the cost of data movement [3], [7], [8]. Similar to [7], we assume that the hierarchy consists of a DRAM, then a 108 kB global SRAM buffer that services $12 \times 14 = 168$ neural processing elements (PE). Each PE has a total of 0.5 kB local register file (RF) storage. Each MAC computation requires four RF accesses: three read operations for the operands, and one write operation for the result.

This hardware architecture can work with large batch size based on the on-chip memory size. The sliding window operation during a convolution allows the same filter plane to be shared across multiple sub regions of an ifmap plane during the computation. The impact of a larger batch size is in enabling greater degrees of filter reuse: the same filter data can be convolved with the ifmap of other images in the batch without having to fetch the filter weights again. Since cluster activation is dynamic and image-specific, SeFAct can be used for each specific image by performing separate selective activation of a different set of clusters for each image. In other words, the decision algorithm for SeFAct framework remains the same for single image or multiple image processing, and different decisions can be made for different images in a batch using the simple control circuitry that chooses selective activation.

The total computation energy is calculated as the product of the number of operations at each of the DRAM, SRAM, and RF levels, multiplied by the energy per unit operation (e_{DRAM}, e_{SRAM}, and e_{RF}) at each of these levels, incorporating the reduction in operation count from (1) due to sparsity.

Using $E_x^y, x \in \{r, w\}, y \in \{I, F, O\}$ to represent the energy for operation x and computation y, the energy requirement, E, for layer L_i of the baseline testing phase is:

$$E = E_r^I + E_r^F + E_w^O + E_{RF} + E_{MAC} \tag{11}$$

where the first three memory access terms are a weighted sum of DRAM and SRAM energies. The weights correspond to the number of memory access to each level, which depends on the data movement and reuse pattern in the DRAM and SRAM. For both the baseline and our enhancement, all data communication with the DRAM (i.e., ifmap read or ofmap write) is performed in run-length compressed (RLC) format, incorporating data sparsity, which is decoded in the SRAM.

Our energy savings appear due to two factors, outlined in Section IV. The first contribution is due to the use of reduced bitwidths, which is a static optimization performed during the training phase, primarily in early layers of the network. The other contribution is obtained from the SeFAct implementation on later layers of the network and it supports dynamic adaptation of computations during the testing phase, as described in Algorithm 3.

The algorithm performs reduced ofmap computation based on updated class prediction which contributes towards the majority of the energy savings. We only load the ifmap and filter data based on the important features, I, and compute reduced number of ofmap data, $\tilde{\mathcal{F}}$. Compared to the baseline, energy savings are achieved from (i) bitwidth reduction, (ii) fewer memory fetches, and (iii) a reduction in the number of MAC operations as only $\tilde{\mathcal{F}}$ features are computed. The change in bitwidth affects memory access energy linearly and computation energy quadratically, since the dominant component of MAC operations is multiplication, whose complexity is quadratic in the number of bits. All the flags, such as I, $\tilde{\mathcal{F}}$, are stored in single-bit register files which are used in inexpensive decision circuitries to reduce memory access operations. The size of the single-bit register files are in the order of 1–2 kB which is proportional to the number of features in all the layers where we implemented SeFAct. This memory overhead is less than 1% of the total memory requirement.

Additional energy savings is obtained by writing only important ofmap features to the memory. Only the identified important features of ofmap are written into the memory (note that due to the large volume of data at each level, the data within a level cannot be completely stored within the SRAM, and DRAM writes are essential). This reduces the memory overhead and also effectively further increases the inherent sparsity (due to ReLU) for level L_{i+1}. The energy savings, ΔE for layer L_i can be formulated using (11) as:

$$\Delta E = \Delta E_r^I + \Delta E_r^F + \Delta E_w^O + \Delta E_{RF} + \Delta E_{MAC} - E_{ov} \tag{12}$$

where E_{ov} includes the energy associated with additional hardware required for real-time decision circuitry mentioned in lines 1 through 3 and lines 5 through 6 in Algorithm 3. The percentage energy savings, PES, is computed based on the summation of required energy, E, and associated saved energy, ΔE of all layers of the network. This metric is used to estimate the effectiveness of the SeFAct framework.

Fig. 3 shows the modified architecture with the overhead decision circuitries for SeFAct implantation which requires three additional components: important feature detection, cluster activation check and predictive data calculation. The energy overhead for detecting important features arises from: (i) computation of the threshold, and (ii) a comparison operation. The threshold computation can be modeled as several multiplication and addition operations. Empirically, there are less than five such arithmetic operations for standard CNN topologies. For the Conv layer, the check is simplified to inexpensive summation of sparsity identifying one-bit flags. Similarly, we have implemented the other decision circuitries based on simple logic gates such as AND, OR, single adder.

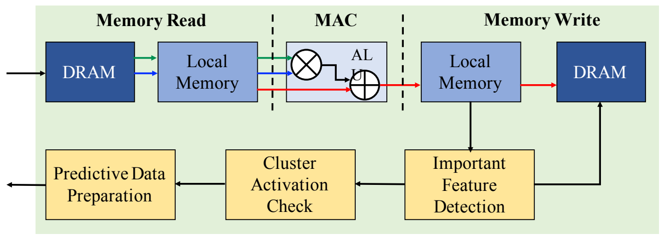

Fig. 3: Neural network hardware with overhead circuitry of SeFAct implementation.

VI. Threshold Formulation

The SeFAct implementation process in *each layer* of a neural network depends on four thresholds, T_1, T_2, T_3 and T_4 as described in equations (2), (5), (7) and (10), respectively. We update these threshold notations as $T_{1,D}, T_{2,D}, T_{3,D}$ and $T_{4,D}$ where *depth*, D, is a parameter that identifies the location of a layer from the input layer in a network. For example, $D = 0$ and $D = N_L - 1$ are the input and output layers for a network, respectively, where N_L is the total number of layers in the network. The SeFAct implementation prepares *predictive data* to save computation in the *subsequent layers*. Hence, we do not implement SeFAct in the last layer, $N_L - 1$, and start from the penultimate layer of the network, $D = N_L - 2$. We observe in Section IV-A, the earliest layer where SeFAct is beneficial, D_{min}, depends on the receptive field of the network. Hence, the total number of layers where SeFAct is implemented is $N_{SeFAct} = N_L - D_{min} - 1$.

The SeFAct implementation leverages a trade-off relation between accuracy and energy savings. We need many observations under various operating conditions of SeFAct to estimate the trend of this trade-off. However, to observe only one trade-off point, $4N_{SeFAct}$ parameters are required to be tuned. Additionally, there exists a complicated relationship between accuracy, energy savings, and thresholds of various layers of the network. This complexity makes the entire design space exploration very difficult.

We have developed a threshold tuning knob, τ, to modulate thresholds $T_{1,D} - T_{4,D}$ of N_{SeFAct} layers through a set of analytical equations with the help of eight additional parameters, $\alpha_0 - \alpha_7$. We systematically explore N_τ trade-off observations by varying the parameter τ for a specific ensemble of $\alpha_0 - \alpha_7$.

For each value of τ, we sample N_α ensembles of α parameters to attune the relationship among τ and thresholds $T_{1,D}, T_{2,D}, T_{3,D}$ and $T_{4,D}$. The exploration requires minimum and maximum limits for τ and the α parameters. Hence, the number of tunable parameters becomes 18. However, based on the interdependencies of the analytical equations, we will show that the number of tunable parameters can be reduced from 18 to 5. Finally, we choose optimal operating conditions based on two additional parameters, β_{N_τ} and β_{APC_α}. Therefore, the designer specifies a total of 7 parameters that enable the exploration of the trade-off space. Note that the total number of tuning parameters does not change even if we seek a larger number of trade-off points: these points can be obtained by increasing the number of parameter samples.

TABLE II: Number of parameters for SeFAct implementation in N_{SeFAct} layers with and without analytical equations.

# Ensembles	# Data points	# Tunable parameters	
		Without analytical equations	With analytical equations
1	1	$4N_{SeFAct}$	7
1	N_τ	$4N_{SeFAct}N_\tau$	7
N_α	N_τ	$4N_{SeFAct}N_\tau N_\alpha$	7

A naïve alternative to our approach would be to explore the design space by tuning all $4N_{SeFAct}$ parameters. To find N_τ trade-off points with N_α sampled ensembles of parameters, exploring the same volume of the design space would require the empirical tuning of $4N_{SeFAct}N_\tau N_\alpha$ parameters. The prohibitively large number of tunable parameters can make it hard to explore the design space to find optimal solution. Table II summarizes the required number of tunable parameters for N_α ensembles of α parameters and N_τ tuning parameters with and without the analytical equations.

Next, we describe the formulation of analytical equations modeling the relationship between τ and the thresholds of various layers and the algorithm that systematically analyze the trade-off between accuracy and energy savings. During this discussion, we will use the following notation:

- $T_{x,D}^{max/min}$ denotes the max/min value of threshold $T_{x,D}$ over all layer depths D, $\forall\ \tau \in \{\tau_{min}, \tau_{max}\}$
- $T_{x,y}^{max/min}$ denotes the max/min value of threshold $T_{x,y}$ for a specific layer at depth y, $\forall\ \tau \in \{\tau_{min}, \tau_{max}\}$.

A. *Formulation of Thresholds* $T_{1,D} - T_{4,D}$

$\underline{T_{1,D}}$: The first step of the cluster learning phase of the SeFAct implementation is *important feature identification*, which is dependent on the threshold $T_{1,D}$ as described in Eq. (2). The earlier layers have smaller receptive fields, and they detect simpler and basic features. This threshold must be kept low to avoid discarding any essential features in the earlier layers. On the other hand, the later layers have larger receptive fields, and increasing $T_{1,D}$ can help reduce the number of predicted classes. Hence, for two layers with depth D_1 and D_2, where $D_2 > D_1$, we prefer $T_{1,D_2} > T_{1,D_1}$. We model the threshold $T_{1,D}$ in layers $D = \{D_{min}, \cdots, N_L - 2\}$ based on a geometric progression of the threshold tuning knob, τ. The energy savings can be increased by tuning τ to increase $T_{1,D}$. The empirical equation for $T_{1,D}$ is as follows:

$$T_{1,D} = \tau \times \left[\frac{D}{N_L-2}\right]^{\alpha_0}, D_{min} \leq D \leq N_L - 2, \alpha_0 > 0 \quad (13)$$

where, α_0 is a fitting parameter.

$\underline{T_{2,D}}$: According to Eq. (5), threshold $T_{2,D}$ is used to determine whether a certain feature is important for all images of an input class in layer D. We model $T_{2,D}$ using a network-depth-independent parameter α_1.

$$T_{2,D} = \alpha_1, \qquad 0 \leq \alpha_1 \leq 1 \quad (14)$$

$\underline{T_{3,D} \text{ and } T_{3,D}}$: There are two additional thresholds that we use for cluster preparation and identification. Threshold $T_{3,D}$ of layer D quantifies the affinity between two clusters and converts this affinity into a graph edge using Eq. (7). According to Eq. (6), the measure of cluster affinity is obtained based on the cluster-based important features, J_k, which evidently relies on the number of class-based important features, $\tilde{I}_c$. We compute $\tilde{I}_c$ using threshold $T_{1,D}$ as shown in equations (2)-(5).

The cluster activation operation for the *cluster testing phase* is regulated by the threshold $T_{4,D}$ as shown in (10). This threshold provides a cluster-specific margin for the activation of a cluster. For aggressive approximation to achieve higher energy savings, a smaller margin for the activation, i.e., lower $T_{4,D}$, is required. On the other hand, higher $T_{4,D}$ results in higher cluster activation, which leads to increased accuracy and reduced energy savings.

The values of both $T_{3,D}$ and $T_{4,D}$ depend on $T_{1,D}$. These dependencies are twofold:

1) Same layer Higher $T_{1,D}$ in a specific layer corresponds to reduced class-specific important features, $\tilde{I}_c$, which corresponds to more aggressive approximation. Hence, to achieve a more rigid clustering criteria, higher (lower) $T_{3,D}$ ($T_{4,D}$) is required. We empirically model the positive (negative) nonlinear correlation between $T_{1,D}$ and $T_{3,D}$ ($T_{4,D}$) using exponential (power-law) equation.
2) Across layers For any two layers at depth D_1 and D_2, where $D_2 > D_1$, the input classes will produce more similar feature patterns in the shallower layer at depth D_1 due to its smaller receptive field. Hence, for the same threshold $T_{1,D_1} = T_{1,D_2}$, the clusters in D_1 will have higher affinity than the clusters in layer D_2. Hence, we prefer to set the cluster threshold $T_{3,D_1} > T_{3,D_2}$ as a shallower layer would need a higher threshold to distinguish between classes. We model this observation using a product term of deeper layer of $T_{1,D}$ threshold. Moreover, the deeper layer thresholds affect the clustering preferences in shallow layers. Similar trend is observed for the threshold $T_{4,D}$.

The analytical equations for $T_{3,D}$ and $T_{4,D}$ that address both of these effects are as follows:

$$T_{3,D} = \alpha_2 \times \exp\{\alpha_3 \left(T_{1,D} - T_{1,N_L-2}^{max}\right)\} \prod_{d=D+1}^{N_L-2} T_{1,d}^{-\alpha_4} \quad (15)$$

$$T_{4,D} = \alpha_5 \times \left[\frac{T_{1,D}}{T_{1,N_L-2}^{min}}\right]^{-\alpha_6} \times \prod_{d=D+1}^{N_L-2} T_{1,d}^{-\alpha_7} \quad (16)$$

Here $\alpha_2(\alpha_5) \in [0,1]$ scales $T_{3,D}(T_{4,D})$, $\alpha_3(\alpha_6) > 0$ attunes the same-layer positive (negative) nonlinear correlation between $T_{1,D}$ and $T_{3,D}(T_{4,D})$, and the product term incorporates the across-layer effect of network depth. The exponential (power-law) term lies within the range $(0,1]$.

B. Choice of SeFAct Parameters

The proposed analytical equations (13)-(16) facilitate significant reduction of tunable parameters: from $4N_{SeFAct}N_\tau N_\alpha$ to nine, τ and $\alpha_0 - \alpha_7$. We sample these nine parameters within an acceptable ranges for energy savings vs. accuracy trade-off trend exploration. The minimum (maximum) of τ and α parameters are τ_{min} (τ_{max}), and $\alpha_{min} = \{\alpha_{0,min}, \cdots, \alpha_{7,min}\}$ ($\alpha_{max} = \{\alpha_{0,max}, \cdots, \alpha_{7,max}\}$), respectively. The initial constraints on the ranges of the parameters are discussed earlier. With the help of network-dependent criteria and heuristics, as explained in Appendix A, the tunable parameters are further reduced down from 18 to 5. Some of the remaining parameters are network-independent constants whereas others can be derived from the tunable parameters using analytical equations (17)-(21).

Once the ranges of the SeFAct parameter are determined, we observe the trade-off between accuracy and energy by sampling N_α combinations of α vectors from the updated range. We have developed two additional tunable thresholds, β_{N_τ} and β_{APC_α} to discard suboptimal α choices from the design space based on the following criteria:

Number of Acceptable Data Points We implement our SeFAct scheme for N_τ accuracy and energy savings trade-off points for each α vector. We discard the trade-off data points for which the network accuracies are smaller than minimum acceptable accuracy, Acc_{min}. Hence, the number of remaining observations are $N'_\tau \leq N_\tau$ and if N'_τ is very small, the particular α vector is suboptimal. We use tunable parameter, β_{N_τ}, to discard the α set if $N'_\tau \leq \beta_{N_\tau} \times N_\tau$.

Number of Average Probable Classes During the testing phase, an input image activates the clusters that have similar feature patterns to reduce computations. We define *average probable classes*, $APC_{\alpha\tau D}$, as the average number of predicted classes for all the input test images in layer D for a specific combination of parameters, $\{\alpha, \tau\}$. We compute average probable class, APC_α, over all $D \leftarrow \{D_{min}, \cdots, N_L - 2\}$ and τ for each α vector. The APC_α acts as a metric for computation reduction for the α vector. A smaller value of APC_α indicates narrower class prediction for the given α which contributes to higher energy savings. The α vectors with very large APC_α are suboptimal for SeFAct implementation. The maximum number of probable classes in a layer can be the total number of classes in a network, N_{class}. Hence, the maximum average probable class over all N_{SeFAct} layers is: $APC_\alpha^{max} = N_{class}N_{SeFAct}$. We discard the suboptimal α vectors that produce $APC_\alpha > \beta_{APC_\alpha} \times APC_\alpha^{max}$, where β_{APC_α} is a tunable parameter.

C. Algorithm for Obtaining the Optimal SeFAct Parameters

Algorithm 4 summarizes the process to obtain optimal operating parameters for the SeFAct implementation process. The inputs for the algorithm can be divided as follows:

User input: The user will choose a neural network and the minimum acceptable accuracy, Acc_{min}, for the network.

Network parameters: The basic network properties such as total number of layers in the network, N_L, the total number of classes, N_{class}, and the baseline accuracy, Acc_{orig}, are included in the network parameters. Additionally, $N_{SeFAct} = N_L - D_{min} - 1$, the total number of SeFAct layers, is obtained based on the receptive field criteria described in Section IV-A.

Empirical parameters: In Appendix A, we have determined the values for some network-independent parameters based on empirical observations. For example, we prefer relatively small τ_{min} to attain the baseline accuracy. Additionally, we set the range of α_0 to obtain both convex and concave trends of Eq. (13) to observe how the accuracy and energy savings trade-off is affected. On the other hand, the ranges for α_1 is chosen within a moderate interval to balance the energy savings and accuracy.

Designer choice: Based on the user-provided network information, the designer will tune the following network-dependent operating parameters:

1) τ_{max} is chosen based on the network complexity and classification task. According to the discussion of Appendix A, we have chosen larger τ_{max} for LeNet compared to AlexNet. As GoogLeNet is better trained for complex classification task of ImageNet data than that of AlexNet, τ_{max} is larger.

2) The minimum and maximum values of α_2 (α_5) are selected while maintaining positive (negative) correlation with τ_{max}, as mentioned in Appendix A.
3) The parameters, β_{N_τ} and β_{APC_α} are used to discard suboptimal α vectors. The designer may choose to impose less/more aggressive check for suboptimal points.

Algorithm 4 Algorithm to obtain optimal α parameters

```
INPUT: User Input: Minimum acceptable accuracy: $Acc_{min}$;
Network Parameters: Number of layers in the network: $N_L$; Total number of classes, $N_{class}$; Total number of SeFAct layer: $N_{SeFAct}$; Original accuracy: $Acc_{orig}$;
Network Indendent Empirical Parameters: Number of $\alpha$ samples: $N_\alpha$; Number of tunable threshold points: $N_\tau$; SeFAct parameters: $\tau_{min}, \{\alpha_{0,min}, \alpha_{0,max}\}$ and $\{\alpha_{1,min}, \alpha_{1,max}\}$;
Designer Choice: SeFAct parameters: $\tau_{max}, \{\alpha_{2,min}, \alpha_{2,max}\}$ and $\{\alpha_{5,min}, \alpha_{5,max}\}$; Data point threshold: $\beta_{N_\tau}$; Average probable class threshold: $\beta_{APC_\alpha}$
OUTPUT: Optimal parameters: $\alpha_{opt} \in \mathbb{R}^{N_\tau \times 8}$
METHOD:
 1: $S_D = \{D_{min}, \cdots, N_L - 2\}$
      ▷ Prepare $\alpha$ parameters using (17)-(21), $\tau_{min}$, $\tau_{max}$ and $D_{min}$
 2: Obtain $\alpha_{min} = \{\alpha_{0,min}, \cdots, \alpha_{7,min}\}$ and $\alpha_{max} = \{\alpha_{0,max}, \cdots, \alpha_{7,max}\}$
      ▷ Generate Latin hypercube samples
 3: $\alpha = \text{LHS}(\alpha_{min}, \alpha_{max}, N_\alpha) \in \mathbb{R}^{N_\alpha \times 8}$
 4: for each $\alpha[i] \in \mathbb{R}^8$   $i \leftarrow \{0, 1, \cdots, N_\alpha - 1\}$ do
 5:     Initialize $S_\alpha \leftarrow \varnothing$
 6:     Generate data points $\tau = \text{linspace}(\tau_{min}, \tau_{max}, N_\tau)$
 7:     for each $\tau[j]$   $j \leftarrow \{0, 1, \cdots, N_\tau - 1\}$ do
            ▷ Compute thresholds using (13)-(16) and $\alpha[i], \tau[j]$
 8:         Compute $T_{1,D}, T_{2,D}, T_{3,D}, T_{4,D}$,   $\forall D \in S_D$
 9:         Learn clusters for layer $D$ using Algorithm 2, $\forall D \in S_D$
10:         Test SeFAct for layer $D$ using Algorithm 3, $\forall D \in S_D$
11:         Compute network accuracy, $Acc_{\alpha\tau}$
12:         if $Acc_{\alpha\tau} > Acc_{min}$ then
13:             Compute percentage energy savings, $PES_{\alpha\tau}$ using (11)-(12)
14:             Obtain average probable classes, $APC_{\alpha\tau D}, \forall D \in S_D$
15:             $S_\alpha \leftarrow S_\alpha \cup \{\{Acc_{\alpha\tau}, PES_{\alpha\tau}\}\}$
16:             $APC_\alpha += \left(\sum_D APC_{\alpha\tau D}/N_{SeFAct} - APC_\alpha\right)/|S_\alpha|$
17:         end if
18:     end for
19:     if $N'_\tau = |S_\alpha| > \beta_{N_\tau} N_\tau$ and $APC_\alpha < \beta_{APC_\alpha} APC_\alpha^{max}$ then
20:         Compute Slope$_\alpha$, Intercept$_\alpha$ = curvefit($S_\alpha$)
21:         Slope = map ⟨key = $\alpha[i]$, value = Slope$_\alpha$⟩
22:         Intercept = map ⟨key = $\alpha[i]$, value = Intercept$_\alpha$⟩
23:     end if
24: end for
25: Generate $Acc = \text{linspace}(Acc_{min}, Acc_{orig}, N_\tau)$
26: for $x \leftarrow \{0, 1, \cdots, N_\tau - 1\}$ do
        ▷ Obtain optimal parameters, $\alpha_{opt}[x]$ for each $Acc[x]$ points
27:     Assign $\alpha_{opt}[x]$ = key for $\max_{\text{key}}$\{Slope[key] $\times$ $Acc[x]$ + Intercept[key]\}
28: end for
```

Algorithm 4 starts with creating a set, S_D, to store the depths of all the SeFAct layers (line 1). Line 2 prepares the acceptable minimum and maximum range of each α parameter based on the network-dependent input parameters. Next, N_α samples of the α parameters are generated in line 3 using latin hypercube sampling algorithm (LHS) [29]. For each α sample vector, an empty set, S_α, is initialized to store the accuracy and energy savings (line 5). Simultaneously, N_τ tunable thresholds are generated in the interval $\{\tau_{min}, \tau_{max}\}$ (line 6). Next, for each τ, all the necessary thresholds are obtained for the SeFAct implemented layers based on the analytical equations (line 8). The algorithm uses these thresholds to learn important features and clusters in N_{SeFAct} layers using Algorithm 2. Line 10 tests the SeFAct implementation in real time. It checkes the cluster activation and reduces the computations accordingly based on Algorithm 3. The network accuracy, $Acc_{\alpha\tau}$ is later computed in line 11. The $Acc_{\alpha\tau}$ and percentage energy savings, $PES_{\alpha\tau}$, are stored in the set S_α if $Acc_{\alpha\tau} \geq Acc_{min}$ (line 15). The algorithm also computes the average probable class of each layer, $APC_{\alpha\tau D}$. Simultaneously, a moving average of $APC_{\alpha R D}$ over all SeFAct layers of all the selected data points, APC_α is also computed. As described in Section VI-B, APC_α indicates the extent of probable class reduction for a given α.

After performing necessary computations for all N_τ points, the algorithm decides whether the current $\alpha[i]$ sample vector is suboptimal. The check of suboptimal α vector in line 19 depends on two parameters, β_{APC_α} and β_{N_τ}, based on the discussion of Section VI-B. The remaining α vectors are the candidates for optimal combination. Later, the algorithm models the trend of the accuracy and PES for the candidate combinations to predict PES for an acceptable accuracy interval. We use a linear fitting model for simplicity. The slope and intercept of the linear fit are stored in a map format with each candidate α vector as the key (lines 20 and 21). The final step of the algorithm is to choose optimal α combinations based on the maximum achievable energy savings for N_τ points in the accuracy interval $\{Acc_{min}, Acc_{orig}\}$.

VII. Results

In this section, we first report the simulation parameters and models. Next, we discuss the optimization of two orthogonal approximation processes implemented in this work: reduced bitwidth and selective feature activation. Finally, we present the effect of both approximation approaches on accuracy and energy savings for optimal operating points.

A. Simulation Parameters/Models

We demonstrate our energy-efficient CNN framework on three well-studied networks, LeNet [24] applied to the MNIST handwritten digit recognition dataset, AlexNet [25] and GoogLeNet [21] both applied to the ImageNet dataset using Caffe platform.

We assume the baseline network (defined at the beginning of Section V), uses 8-bit words for ifmaps, filters, and ofmaps [2], [6]. We use 5,000 (10,000) images for the cluster learning phase and 2,000 images for testing phase for LeNet (GoogLeNet/AlexNet). The top-1 (top-5) accuracy of LeNet (GoogLeNet/AlexNet) for the baseline is 99.06% (89.00%/77.95%). We use CNNergy, an open-sourced simulator [30], to determine the number of DRAM, SRAM, and RF memory accesses and computations as well as the batch size for GoogLeNet, AlexNet and LeNet. CNNergy estimates the energy consumption based on the analytical model provided by [31]. The batch size used for LeNet, AlexNet and GoogLeNet are 16 or higher [30]. Per unit energy for DRAM, SRAM, and RF memory accesses are obtained from [3].

B. Reduced Bitwidths

We implement reduced precision bitwidths from the $input$ layer to the $c1$, $c4$ and $inception4a$ layer in LeNet, AlexNet and GoogLeNet, respectively. For later layers, the bitwidths are the same as the baseline.

The reduced precision bitwidths in various layers of the CNN trade-off classification accuracy and energy savings. We used latin hypercube sampling [29] based Monte-Carlo simulations for 500 prospective bitwidth combinations in early layers and checked the classification accuracy on 5,000 test images. The optimal bitwidth is chosen based on the criteria of maximum energy savings for the minimum accuracy drop. The modified bitwidths in LeNet, AlexNet and GoogLeNet are listed in Tables III. The *Pool* and *Norm* layers use the same bitwidths as their immediately preceding *Conv* layers.

TABLE III: Modified bitwidths of early layers.

Network	Layer	ifmap/ofmap	filter
LeNet	$input$	4	3
	$c1$	3	4
AlexNet	$input$	6	8
	$c1$	7	7
	$c2$	5	7
	$c3$	6	6
	$c4$	5	7
GoogLeNet	$input$	7	7
	$c1$	7	7
	$c2/reduce$	6	8
	$c2$	7	7
	$inception3a$	6	8
	$inception3b$	7	7
	$inception4a$	7	7

C. Selective Feature Activation

We use our clustering based method for selective feature activation, as explained in Section III-B, for all layers from $c2$, $c5$ and $inception4b$ for LeNet, AlexNet and GoogLeNet, respectively. During the testing phase, the input image activates the clusters similar to its feature pattern to reduce class prediction. For example, we find that the images of various non-shedding dogs (e.g., shih-tzus, spaniels, and terriers) activate the same cluster in layer $fc7$ of AlexNet.

Early prediction of a reduced number of classes reduces the computations and the energy, as compared to the baseline. The amount of reduction in the number of classes based on the SeFAct prediction depends on thresholds $T_{1,D}, T_{2,D}, T_{3,D}$ and $T_{4,D}$, which rely on the α parameters and threshold tuning knob, τ, as detailed in Section VI. For each α parameter combination, the tuning knob τ can be varied to achieve various trade-off data points. Next, we will show the effect of parameters α and τ on accuracy and energy savings, as well as simulation setups to find optimal operating parameters.

Effect of Parameters α and τ The trade-off between the percentage energy savings, PES, and the accuracy (all normalized to the baseline) for various combinations of α (represented with various colors) and τ parameters are shown in Fig. 4(a) for LeNet, GoogLeNet, and AlexNet networks. The red vertical lines show the accuracy of the baseline. The accuracy gap between the red vertical line and the rightmost points in the scatter plots can be attributed to the loss in accuracy due to reduced bitwidth approximation in the early layers. The reduced bitwidth contributes towards a fixed energy savings for all operating conditions as shown in green horizontal lines. The figure shows that, depending on the choice of the α parameters, different trade-offs between the energy savings and accuracy can be obtained for the same τ.

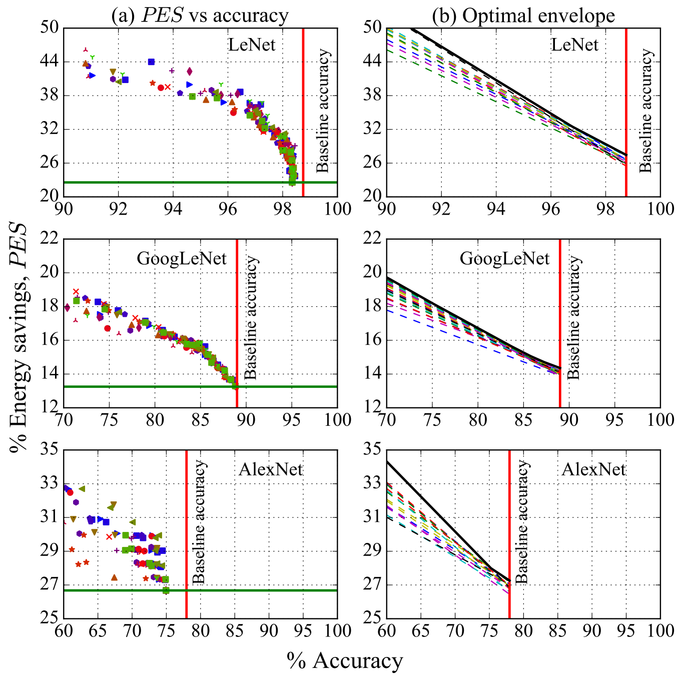

Fig. 4: (a) PES vs. accuracy for various combinations of the α parameters and (b) optimal PES vs. accuracy envelope using Algorithm 4 for LeNet, GoogLeNet and AlexNet.

Finding Optimal Operating Parameters Algorithm 4 automates the search for optimal α combination. It first discards the suboptimal α combinations and outputs the optimal α set based on the detection of envelope for the linearly modelled trade-off trends of various α parameter sets. Fig. 4(b) shows the linearly fitted trade-off curves of various α parameters using dashed lines. The envelope of all trade-off curves is shown using the black solid line. It can be seen that the envelope is piecewise linear i.e. for different accuracy interval, the α set is different.

From these plots, we observe that significant energy savings are achievable using our systematic analysis: for small (5–10%) degradations in accuracy, 15–25% energy savings are possible. The inexpensive overhead circuitries, described in Section V, consume less than 1% of the total energy. For scenarios where low accuracy is acceptable (e.g., in mobile embedded systems or edge devices, where battery limitations are the paramount consideration, and a best-effort accuracy is good enough), improvements of almost 30-40% are visible.

Simulation Setups for Optimal Operating Parameters The inputs for Algorithm 4 for various networks are provided in Table IV. We chose the number of α samples, $N_\alpha = 50$, and the number of threshold tuning parameters, $N_\tau = 16$.

D. Comparison of Static and Dynamic Pruning Approaches

Static pruning methods such as [17]–[19] avoid multiplica-

TABLE IV: Parameters used for SeFAct implementation.

Network	LeNet	GoogLeNet	AlexNet
Network parameters			
N_{class}	10	1000	1000
N_L	5	14	9
N_{SeFAct}	2	6	3
Acc_{orig}	98.35% *	89.00% **	74.95% **
User input			
Acc_{min}	90.00% *	70.00% **	60.00% **
Designer choice			
β_{N_τ}	0.50		
β_{APC_α}	0.95		
τ_{max}	1.50	1.20	0.80
$\{\alpha_{2,min}, \alpha_{2,max}\}$	{0.30,0.80}	{0.25,0.75}	{0.10,0.40}
$\{\alpha_{5,min}, \alpha_{5,max}\}$	{0.10,0.50}	{0.15,0.60}	{0.20,0.70}
Network independent empirical parameters			
τ_{min}	0.10		
$\{\alpha_{0,min}, \alpha_{0,max}\}$	{0.60,1.60}		
$\{\alpha_{1,min}, \alpha_{1,max}\}$	{0.55,0.65}		

* Top-1 accuracy, ** Top-5 accuracy

tions by zero. Another degree of freedom exploited by these works is in pruning filter values and retraining the network to reduce these computations. It should be noted that all such optimizations are static approximations, in that they must be valid for at least a vast majority of input images and classes.

SeFact, implemented in Eyeriss architecture, skips computations when the input activation is zero and stores the data in compressed format, thus naturally incorporating data sparsity in the evaluation of both the baseline method and SeFAct. Additionally, SeFAct is a dynamic, image specific approximation made at run-time. A specific image dynamically classifies some clusters as important and others as unimportant. As a result, a particular computation could be pruned for one image but not for another: something that static pruning cannot achieve. This implies that the gains of SeFAct are over and above any gains from static pruning, and these gains are largely orthogonal to static pruning.

A pruning approach removes features based on specific feature importance criteria. There are two metrics to consider for a feature to be deemed important:
1. Image sensitivity: A feature is considered important, if a large number of images activate that feature. However, if a feature is activated for a vast majority of the images, then it does not hold any special information and can be pruned out.
2. Feature sensitivity: When a small (large) number of features are activated in a layer for a specific image, then it suggests that the importance of each of the activated features is high (low): each of the activated features is highly (less) error-sensitive, i.e., discarding these features will (will not) prevent us from correctly identifying the image.

Fig. 5 illustrates the scope of static and dynamic pruning for AlexNet in three layers, conv5, fc6 and fc7. Each scatter data point suggests a feature in the layer, the x-axis indicates the image sensitivity and y-axis indicates the feature sensitivity. The scatter plot data are divided into four quadrants centered on the average of the image and feature sensitivities. The features activated for a small number of images with large number of simultaneously activated features (Quadrant II) are

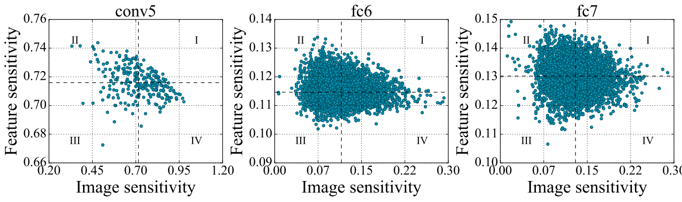

Fig. 5: Scope of static and dynamic pruning in AlexNet.

not *distinguishing features* and can be a good and safe choice for static pruning. On the other hand, the features that reside in Quadrants I and III have contrasting image and feature sensitivities. Aggressive static pruning techniques may remove many features from these two quadrants leading up to 70% feature reduction. However, not all networks can achieve this level of pruning as it depends on the dataset complexity. Finally, the features in the Quadrant IV have higher image sensitivity with lower feature sensitivity. These features cannot not be removed without careful consideration. The dynamic pruning approach such as our SeFAct increases energy savings by leveraging image specific information of all *four quadrants*, including Quadrant IV, where static pruning is less effective.

E. Accuracy and Energy Trade-off for Optimal Parameters

In this section, we show the combined effect of reduced bitwdith and SeFAct approximations for the optimal parameters. The average number of probable classes, $APC_{\alpha\tau D}$ of each layer, D, and the layer-wise energy for the baseline and our enhancement, are shown for GoogLeNet in Fig. 6. Here, the results are shown using the optimal α combinations obtained from Algorithm 4 for various tunable thresholds, τ.

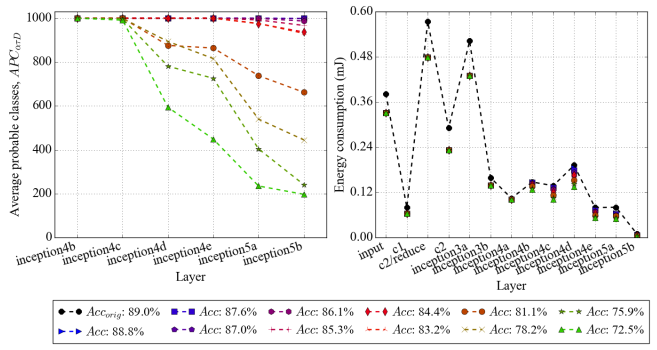

Fig. 6: (a) Average probable classes in layers with SeFAct. (b) layer-wise energy in GoogLeNet.

Fig. 6(a) shows the number of average probable classes, $APC_{\alpha\tau D}$ for each layer on which SeFAct is applied, whereas Fig. 6(b) reports the energy requirement for individual layers of the network. It can be seen that the number of average probably classes is reduced monotonically, resulting in a significant reduction in energy requirement with respect to the baseline, at the cost of a loss in accuracy. For example, the total number of classes in GoogLeNet is 1000 and the number of average probable classes is reduced to 445 in the $inception5b$ layer for a classification accuracy of 78.2%. Similar trends are seen for LeNet and AlexNet networks.

For these CNNs, the energy savings in the early layers are attributable to the optimized bitwidth, while those in later layers are attributed to the selective feature activation. For the same energy savings, the relative percentage contributions between reduced bitwidth approximation and SeFAct are about 60% and 40%, respectively.

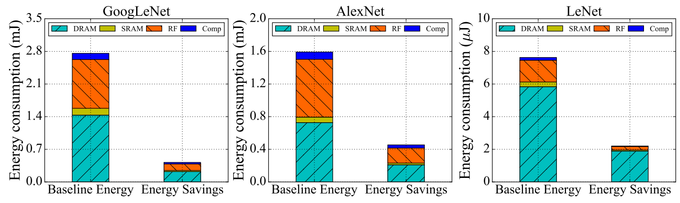

Fig. 7: Energy contribution breakdown for baseline energy and energy savings for (a) LeNet (b) GoogLeNet (c) AlexNet.

The total energy and the contributions from DRAM, SRAM, and RF memory accesses, and MAC computations for various networks are shown in Fig. 7. The results for both the baseline implementation and the energy savings for our approach indicate that the memory access operations are dominant.

VIII. Conclusion

This work has proposed an effective and automated way to dynamically reduce the energy of a CNN accelerator, using bitwidth reduction in early layers, and selective feature activation in later layers. Significant energy savings are seen, with small accuracy losses, for three well-known networks.

Appendix A
Choice of SeFAct Parameters: Details

The criteria to select the suitable ranges for the nine SeFAct parameters, τ and $\alpha_0 - \alpha_7$, are discussed here.

Choice of τ: The tuning parameter τ in Eq. (13) can be used to modulate $T_{1,D}$ to change the number of features deemed important, and hence alter energy savings. To get the baseline accuracy as the starting point, we choose $\tau_{min} \approx 0$ (constant for all networks). On the other hand, the value of τ_{max} is strongly dependent on the properties of the network and the input data. When relatively larger (smaller) networks are tasked with working with simpler (complex) input data sets, we can use a large (small) threshold margin, $T_{1,D}^{max} = T_{1,N_L-2}^{max} = \tau_{max} > 1$ $(0 < \tau_{max} < 1)$.

Choice of α_0: From Eq. (13), $\frac{\partial^2 T_{1,D}}{\partial \alpha_0{}^2}$ = Constant $\times\ \alpha_0 \times (\alpha_0 - 1)$, where Constant > 0, hence $T_{1,D}$ follows a convex (concave) trend for variable τ when $\alpha_0 > 1$ $(0 < \alpha_0 < 1)$. A convex (concave) trend will assign lower (higher) thresholds to the shallower layers and more (less) aggressive thresholds in the later layers and save less (more) computations i.e. energy. Therefore, depending on the target accuracy, the value of α_0 that reflects the convex or concave trend can be chosen. We sample various α_0 and choose the optimal one based on the accuracy and energy savings trade-offs.

Choice of α_1: According to Section VI-A, α_1 is a network-independent parameter. We prefer α_1 to be assigned some moderate mid-range value within the interval $[0, 1]$ to balance the energy savings and accuracy.

Choice of α_2: The threshold $T_{3,D}$ is used to prepare clusters by quantifying the affinity between classes. According to Section VI-A, a network-dependent parameter, α_2, is used to scale the threshold $T_{3,D}$. Additionally, $T_{1,D}$ and $T_{3,D}$ have a positive correlation. Hence, if τ_{max}, the parameter controlling $T_{1,D}$, is set high value then α_2 must also be increased.

Choice of α_3: The exponential relation between $T_{3,D}$ and $T_{1,D}$ in Eq. (15) is regulated via the parameter, α_3. For efficient design space exploration, we want to keep T_{3,N_L-2}^{min} is within some factor, k_1, of T_{3,N_L-2}^{max}. The equation that ensures the condition is as follows:

$$T_{3,N_L-2}^{min} = k_1 \times T_{3,N_L-2}^{max}, \qquad 0.2 \le k_1 \le 0.6$$
$$\alpha_3 = -\frac{\log_e k_1}{T_{1,N_L-2}^{max} - T_{1,N_L-2}^{min}} \tag{17}$$

where the range of k_1 is empirically chosen.

Choice of α_4: The parameter α_4 in Eq. (15) is used to modulate the inter-layer relationship of cluster preparation threshold, $T_{3,D}$. It is observed that the product term of $T_{3,D}$ in Eq. (15) increases faster as we move to more shallower layers and the growth rate of the term depends on the parameter As described to Section VI-A, the exponential term $\alpha_2 \times \exp\left\{\alpha_3\left(T_{1,D_{min}} - T_{1,N_L-2}^{max}\right)\right\}$ is normalized to lie in the interval $(0, 1]$. Therefore, we can assume:

$$\prod_{d=D_{min}+1}^{N_L-2} \tau^{-\alpha_4} \times \left[\frac{d}{N_L-2}\right]^{-\alpha_0\alpha_4} = C, \qquad \text{from (13)} \tag{18}$$
$$-\alpha_4\left[(N_{SeFAct} - 1)\log\tau + \sum_{d=D_{min}+1}^{N_L-2}\log\left[\frac{d}{N_L-2}\right]^{\alpha_0}\right] = \log C$$

where $C = \left[\alpha_2 \times \exp\left\{\alpha_3\left(T_{1,D_{min}} - T_{1,N_L-2}^{max}\right)\right\}\right]^{-1} \ge 1$ and N_{SeFAct} is the total SeFAct layers. Based on some heuristics, the simplified equation becomes:

$$\alpha_4 = \frac{k_2}{(N_{SeFAct} - 1)}, \qquad k_2 = -\frac{\log C}{\log \tau_{min}} \tag{19}$$

Empirically, we set $1.25 \le C \le 2.5$ which results in $0.05 \le k_2 \le 0.40$ for $\tau_{min} = 0.10$.

Choice of α_5: The threshold $T_{4,D}$ in Eq. (10) is used as a cluster activation margin for layer D in the *cluster testing phase*. Networks with complex identification tasks should have higher $T_{4,D}$ for higher cluster activation to avoid significant accuracy degradation. According to (16), $T_{1,D}$ and $T_{4,D}$ have a negative correlation. The thresholds $T_{1,D}$ and $T_{4,D}$ are controlled by τ and α_5, respectively. Hence, the choice of α_5 should follow an opposite trend of τ, i.e., τ_{max}.

Choice of α_6: The criteria to choose α_6 is similar to α_3. The parameter α_6 is used to modulate the nonlinear relation between $T_{4,D}$ and $T_{1,D}$ as described in (16). We ensure that the minimum of T_{4,N_L-2} stays within a factor, k_3, from the maximum value, T_{4,N_L-2}^{max}. For simplicity, we choose $k_3 = k_1$ as they have similar ranges. The equation that ensures the condition is as follows:

$$T_{4,N_L-2}^{min} = k_1 \times T_{4,N_L-2}^{max}, \qquad 0.2 \le k_1 \le 0.6$$
$$\alpha_6 = -\frac{\log k_1}{\log T_{1,N_L-2}^{max} - \log T_{1,N_L-2}^{min}} \tag{20}$$

Choice of α_7: The parameter α_7 is used to modulate the inter-layer relationship between thresholds $T_{1,D}$ and $T_{4,D}$ as described in (16). The conditions to choose α_7 is similar to that of α_4. Hence, for simplicity we use same equation for both of these parameters. The equation for α_7 is as follows:

$$\alpha_7 = \alpha_4 = \frac{k_2}{N_{SeFAct} - 1} \tag{21}$$

In summary,

- There are only five network-dependent parameters: $\tau_{max}, \alpha_{2,min}, \alpha_{2,max}, \alpha_{5,min}$, and $\alpha_{5,max}$.
- The parameters τ_{min}, $\alpha_{0,min}, \alpha_{0,max}, \alpha_{1,min}$ and $\alpha_{1,max}$ are network-independent constants.
- Min/max values of remaining parameters, $\alpha_3, \alpha_4, \alpha_6$ and α_7 are obtained using the analytical equations (17)-(21).

References

[1] F. S. Snigdha, *et al.*, "SeFAct: Selective feature activation and early classification for CNNs," in *Proceedings of the Asia-South Pacific Design Automation Conference*, pp. 487–492, 2019.

[2] N. P. Jouppi, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the ACM International Symposium on Computer Architecture*, pp. 1–12, 2017.

[3] V. Sze, *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of IEEE*, vol. 105, pp. 2295–2329, Dec 2017.

[4] Y. LeCun, *et al.*, "Deep learning," *Nature*, vol. 521, pp. 436 EP –, 2015.

[5] P. Panda, *et al.*, "Cross-layer approximations for neuromorphic computing: From devices to circuits and systems," in *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pp. 1–6, 2016.

[6] P. Gysel, *et al.*, "Hardware-oriented approximation of convolutional neural networks," 2016. arXiv preprint arXiv:1604.03168.

[7] Y.-H. Chen, *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[8] Y. Chen, *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, 2014.

[9] S. Gupta, *et al.*, "Deep learning with limited numerical precision," in *Proceedings of the International Conference on Machine Learning*, pp. 1737–1746, 2015.

[10] S. Venkataramani, *et al.*, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proceedings of the ACM International Symposium on Low Power Electronics and Design*, pp. 27–32, 2014.

[11] V. Mrazek, *et al.*, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–7, 2016.

[12] M. Jaderberg, *et al.*, "Speeding up convolutional neural networks with low rank expansions," in *Proc. BMVC*, 2014.

[13] S. Han, *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," 2015. arXiv preprint arXiv:1510.00149.

[14] Y. He, *et al.*, "Channel pruning for accelerating very deep neural networks," in *The IEEE International Conference on Computer Vision*, Oct 2017.

[15] B. Reagen, *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proceedings of the ACM International Symposium on Computer Architecture*, pp. 267–278, June 2016.

[16] W. Yu, *et al.*, "Visualizing and comparing AlexNet and VGG using deconvolutional layers," in *Proceedings of the International Conference on Machine Learning*, 2016.

[17] J. Albericio, *et al.*, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proceedings of the International Symposium on Computer Architecture*, pp. 1–13, 2016.

[18] A. Gondimalla, *et al.*, "SparTen: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pp. 151–165, 2019.

[19] A. Parashar, *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the Annual International Symposium on Computer Architecture*, pp. 27–40, 2017.

[20] V. Nair *et al.*, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the International Conference on Machine Learning*, pp. 807–814, 2010.

[21] C. Szegedy, *et al.*, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2015.

[22] F. N. Iandola, *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," 2016. arXiv preprint arXiv:1602.07360.

[23] M. Lin, *et al.*, "Network in network," 2013. arXiv preprint arXiv:1312.4400.

[24] Y. Lecun, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[25] A. Krizhevsky, *et al.*, "ImageNet classification with deep convolutional neural networks," in *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 1097–1105, 2012.

[26] T. H. Corman, *et al.*, *Introduction to algorithms*. Boston, MA: MIT Press, 3 ed., 2009.

[27] M. M. Halldórsson *et al.*, "Greed is good: Approximating independent sets in sparse and bounded-degree graphs," *Algorithmica*, vol. 18, pp. 145–163, May 1997.

[28] D. D. Lin, *et al.*, "Fixed point quantization of deep convolutional networks," in *Proceedings of the International Conference on Machine Learning*, pp. 2849–2858, 2016.

[29] M. Stein, "Large sample properties of simulations using latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.

[30] "CNNergy: An analytical CNN energy model." https://github.com/manasiumn37/CNNergy/. Accessed June 27, 2020.

[31] S. D. Manasi, *et al.*, "NeuPart: Using analytical models to drive energy-efficient partitioning of cnn computations on cloud-connected mobile clients," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2020.

Farhana S. Snigdha received the B.Sc. degree from Bangladesh University of Engineering and Technology, Dhaka and the M.S. and the Ph.D. degrees from the University of Minnesota. She is currently working as a Lead Software Engineer at Cadence Design Systems, Austin, TX, USA. Her research interests include design and optimization of low power system and architectures.

Susmita Dey Manasi is currently pursuing her Ph.D. degree at the Department of Electrical and Computer Engineering, University of Minnesota. She received the B.Sc. degree in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. Her current research interests include low-power embedded deep learning processors, area-power-performance optimization techniques in VLSI design automation, and approximate computing.

Jiang Hu received the B.S. degree from Zhejiang University (China), the M.S. and the Ph.D. degrees from the University of Minnesota. He has worked with IBM Microelectronics from 2001 to 2002, and has been a faculty at the Texas A&M University, thereafter. He has received two conference Best Paper awards, the IBM Invention Achievement Award, and the Humboldt Research Fellowship. He has served on the editorial boards of IEEE TCAD and ACM TODAES.

Sachin S. Sapatnekar (S'86, M'93, F'03) received the Ph.D. degree from the University of Illinois, and is a Distinguished McKnight University Professor and the Robert and Marjorie Henle Chair Professor in Electrical and Computer Engineering at the University of Minnesota. He has received eight conference Best Paper awards, a Best Poster Award, two ICCAD 10-year Retrospective Most Influential Paper Awards, the SRC Technical Excellence award and the SIA University Research Award. He is a Fellow of the ACM and the IEEE.