SeFAct: Selective Feature Activation and Early Classification for CNNs

Farhana Sharmin Snigdha[1], Ibrahim Ahmed[1], Susmita Dey Manasi[1],
Meghna G. Mankalale[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,ahmed589,manas018,manka018,sachin}@umn.edu, jianghu@ece.tamu.edu

***Abstract*—This work presents a dynamic energy reduction approach for hardware accelerators for convolutional neural networks (CNN). Two methods are used: (1) an adaptive data-dependent scheme to selectively activate a subset of all neurons, by narrowing down the possible activated classes (2) static bitwidth reduction. The former is applied in late layers of the CNN, while the latter is more effective in early layers. Even accounting for the implementation overheads, the results show 20%–25% energy savings with 5–10% accuracy loss.**

I. Introduction

Neural architectures [1], [2] have multiple layers consisting of artificial neurons, where each layer is trained to recognize various features of the input, with deeper layers uncovering more complex features. Such deep neural networks (DNNs), often implemented as CNNs, can solve increasingly complex problems. The computational requirements of DNNs have rapidly increased as networks with more layers and features are utilized [2]. These platforms are thus power-hungry and require large memory footprints. Energy reduction of DNNs is paramount in both datacenters and mobile platforms.

Prior works have proposed several methods for reducing computations, including data parallelization using multiple cores, specialized hardware [3]–[5], and approximate computation [6]–[9] based on simplified architecture and arithmetic.

However, there are also more specific properties that may be leveraged, related to how features are mapped to neurons in various layers. 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 [10]. Some paths through the network are unlikely to be activated because specific sets of neurons are seldom activated together. We propose a selective feature activation approach, SeFAct, to develop quantitative metrics for identifying such scenarios, and to save energy by reducing unnecessary computations. During training, we group similar activations across multiple classes into clusters. During testing, we use these clusters to dynamically prune or approximate, unlike well-known static pruning [6], a large section of the CNN that does not help interpret the input, allowing us to reduce both the number of memory accesses and the computation energy, particularly in the later layers of a DNN. For earlier layers of the DNN, we use reduced precision to improve energy savings without significantly affecting accuracy.

II. Overview of the Selective Feature Activation

A. Motivating Example

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. The unactivated neurons are white.

This work was supported in part by the NSF under awards CCF-1525925, CCF-1763761, CCF-1525749.

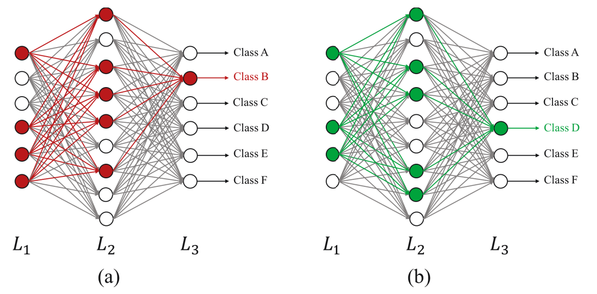


Fig. 1: Activation pattern of features for (a) Class B and (b) Class D in layers, L_1 to L_3 of an example neural net.

It can be seen that classes B and D have 75% or more common activated features in both layer L_1 and L_2. As the activation patterns of the classes B and D are very similar, we can cluster them together in training phase. More generally, we identify the common feature activation pattern for all classes in each cluster, which we refer to as its "stamp". 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 the neuron computation and memory access energy in the subsequent layers.

We build a framework for identifying and using important features to selectively activate neurons. We work in two steps:

1) *Cluster learning phase* We augment the traditional training phase with a step that identifies clusters. Based on the trained weights and training data, we
   a) identify the important features for each class
   b) cluster classes with similar feature activation patterns
   c) prepare stamps and cluster data for the testing phase
2) *Testing phase* The traditional testing phase is modified to use the cluster learning results. At each CNN level, we
   a) compare feature activation patterns with the cluster stamps and identify the activated cluster(s)
   b) load and compute data only for the predicted class(es)
   c) propagate the predicted class sets to the next layer

Thus, cluster learning is a preprocessing step associated with training. Changes to the testing phase incur overheads, but also generate savings, and are implemented in real-time.

Notation 1. *We frequently refer to a variable x in multiple layers. We denote the variable x in the current layer, L_i, without a superscript; the variable in the previous layer, L_{i-1},* *and the next layer, L_{i+1}, as x^- and x^+, respectively.*

B. *Cluster Learning Phase*

1) The Concept of Important Features: A CNN has convolutional (*Conv*), fully connected (*fc*), or pooling (*Pool*) layers. 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}.

Fig. 2 illustrates various data types of *Conv* layer.

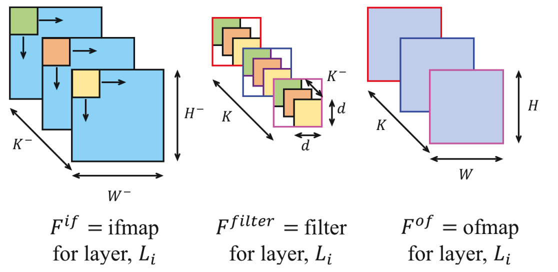

H^-
K^-
W^-
K
K^-
d
d
K
H
W
F^{if} = ifmap for layer, L_i
F^{filter} = filter for layer, L_i
F^{of} = ofmap for layer, L_i

Fig. 2: Illustration of ifmap, filter, and ofmap in a *Conv* layer.

For stride size U under bias $F^{bias}[u]$, layer L_i computes [2]:

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

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

The k^{th} feature of F^{if} is represented by the 2D feature plane, $F_k^{if} \in \mathbb{R}^{H^- \times W^-}, 1 \le k \le K^-$. We show how we mark the important features for a single input image.

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*

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

where T_1 is a tunable threshold parameter.

Computational simplification: While Eq. (2) 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 by assuming self-similarity between the data in various features in each layer: in particular, we assume that the average of all nonzero data for each feature of the ifmap of a *Conv* layer is identical, denoted as μ^{if}, and that various layers differ merely in the sparsity, S_k^{if} for feature k. This helps to reduce computation without hurting the effectiveness of our method, as seen in Section V.

Theorem II.1. *Under the self-similarity assumption, feature k in a Conv layer is important, as defined in Definition 2, if*

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

where $S^{if} = \sum_{k=1}^{K^-} S_k^{if} / K^-$ is the constant overall ifmap sparsity, and $T_I = 1 - T_1 \times (1 - S^{if})$.

Proof. The number of nonzero elements associated with each feature is $H^- W^- (1 - S_k^{if})$. Under the self-similarity assumption, we can write, $\tilde{F}_k^{if} = H^- W^- (1 - S_k^{if}) \mu^{if}$. Eq. (2) for a *Conv* layer then becomes:

$$I_k = 1\left\{ H^- W^- (1 - S_k^{if}) \mu^{if} \ge T_1 \times \frac{\sum_{k=1}^{K^-} H^- W^- (1 - S_k^{if}) \mu^{if}}{K^-} \right\}$$
$$= 1\left\{ S_k^{if} \le 1 - T_1 \times (1 - S^{if}) \right\} = 1\left\{ S_k^{if} \le T_I \right\} \quad (4)$$

The latter expression arises because H^-, W^-, and μ^{if} are constant for a given ifmap layer, F^{if}, and cancel out. □

2) Important Feature Detection for Each Class: The relation in (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.

In the current layer, let us say that we apply (2) 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, with $NI(c)$ 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 an empirically chosen 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:

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

3) Clustering: Classes with closely matched features are now grouped together as clusters that satisfy these properties:
Property: A cluster is a set of classes, with one or more clusters associated with each layer, satisfying these properties:

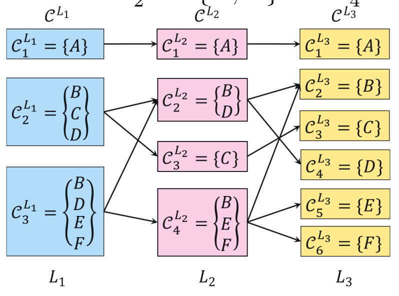
- Clusters in the last layer are singleton sets, with each set containing one class.
- No cluster is a subset of another cluster in the same layer.
- If $\mathcal{C}$ and $\mathcal{C}^+$ are the sets of clusters in layer L_i and layer L_{i+1}, then each cluster $\mathcal{C}_x$ in layer L_i must satisfy

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

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. 3 shows a sample clustering of the network of Fig. 1. Clusters in the last layer are singletons, consistent with the first property. The third property 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. 3: 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:

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

The goal of clustering is to combine the clusters in layer L_{i+1} so that high-affinity clusters are grouped together. We map this problem to the *maximum independent set* (MIS) problem on graphs. Two vertices are 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 also includes its neighbors, i.e., classes with high affinity.

Algorithm 1 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 [11]. 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 neighbor set (line 7). 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 $\mathcal{O}(N_{\mathcal{C}^+} \log(N_{\mathcal{C}^+}))$, 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:     $n_v \leftarrow \{u\} \forall (u, v) \in E$            ▷ 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. 3, 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. 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 (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: 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

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 $\mathcal{C}_2^{L_1}$.

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.

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

The cluster testing phase for layer L_i is described in Algorithm 3. Based on the cluster activations, the class predictions are updated and the information is used to reduce computation for future layers. First, we count the number of stamp elements that are not matched by the input data (line 1). By performing these computations only on $\tilde{D}$ clusters, we greatly reduce energy. Next, we find the minimum mismatch to detect the activated clusters that are within some margin of this mismatch (lines 2 and 3). The activated clusters contain the updated predicted classes. The L1 norm used in these computations is the sum of absolute differences of each element of the vector.

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 $\mathcal{S}_k \forall$ clusters $k$; 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: $m = \min(\mathcal{M})$          ▷ Minimum cluster mismatch
3: $\mathcal{A}_k = 1\{\mathcal{M}_k \le m + T_4 \|\mathcal{S}_k\|_1\}, k \in \tilde{\mathcal{D}}$          ▷ Cluster activation
4: Obtain features to compute for ofmap, $\tilde{\mathcal{F}} = \vee_{\mathcal{A}_k=1} \mathcal{F}_k$
5: Compute ofmap using (1) and $\tilde{\mathcal{F}}$
6: Detect important features for layer $L_{i+1}$, $I^+$          ▷ Use (2) and $T_1^+$
7: Obtain clusters to check for layer, $L_{i+1}$, $\tilde{\mathcal{D}}^+ = \bigcup_{\mathcal{A}_k=1} \mathcal{D}_k^+$
8: return
```

Next, for the updated predicted classes, we identify the important features of ofmap, $\tilde{\mathcal{F}}$. The reduced ofmap data is determined in line 5. Finally, lines 6 and 7 determine, the important features and activated clusters in layer L_{i+1}, respectively. Note that these computations are identical to the cluster learning phase, except here we work with only one test image at a time.

III. Design Optimizations for Energy Reduction

We enable energy-efficient neural computation by combining selective feature activation, SeFAct, described in Section II, with optimized reduced-precision approximation. Reduced precision schemes have been explored in recent research [3], [7] as well as commercial platforms [1], [4]. Some approaches have used fixed bitwidths (for example, 8-bit [1]) for all the layers. Other approaches [12] have used layerwise bitwidth optimization for controlled error introduction and improved power savings. We obtain optimized bitwidths for various layers for accuracy and energy savings with Monte-Carlo simulations. The reduced precision approximation and our SeFAct approach are two orthogonal processes that introduce controlled levels of error in network to achieve energy savings.

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.

For image data, an alternative way to explain this is through the concept of the *receptive field* [13]. The receptive field of a neuron is the region in the input image that affects the neuron. The dimensions of square receptive field for different layers of LeNet (AlexNet) are given in Table I (Table II). The size of input images for LeNet (AlexNet) is 28×28 (227×227).

TABLE I: Receptive fields of various layers in LeNet

Layer	$c1$	$p1$	$c2$	$p2$	$fc1$	$fc2$
Dimension	5	6	14	16	28	28

TABLE II: Receptive fields of various layers in AlexNet

Layer	$c1$	$p1$	$c2$	$p2$	$c3$	$c4$
Dimension	11	19	51	67	99	131
Layer	$c5$	$p5$	$fc6$	$fc7$	$fc8$	
Dimension	163	195	355	355	355	

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. For example, neurons in layer $c2$ of LeNet (AlexNet) process information about $(14/28)^2 = 25\%$ ($(51/227)^2 = 5\%$) 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 $c2$ ($c5$) onwards in LeNet (AlexNet).

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 have high levels of resilience to inaccuracy. This provides an opportunity to implement error sensitivity based circuit approximations for early layers for these reasons:

- The number of resilient neurons is significantly higher in initial layers of the network [8] in comparison to later layers. The reason is that neurons in the initial layers typically process features local to a certain region of the image, while neurons in the final layers 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 optimized reduced precision to incorporate errors in early layers, along with our selective activation approach for deeper layers.

IV. 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 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 [2], [4], [5]. Similar to [4], 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.

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:

$$E = E_r^I + E_r^F + E_w^O + E_{RF} + E_{MAC} \qquad (10)$$

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 III. The reduction of bitwidths is performed statically during the training phase, primarily in early layers of the network. The other part, obtained in later layers of the network, involves the addition of hardware that supports dynamic adaptation of computations during the testing phase, as described in Algorithm 3. We now summarize these changes.

Line 1 performs inexpensive mismatch computations which involves summations of one-bit numbers. The min computation in line 2 is a sequence of compare (i.e., subtract) operations over all clusters. The implementation cost of both lines 2 and 3 are linear in the number of clusters. Line 4 associates flags in $\tilde{\mathcal{F}}$ for all activated clusters.

Line 5 performs reduced ofmap computation using the flags in $\tilde{\mathcal{F}}$. Here, we only load the ifmap and filter data based on the important features, I. 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. 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.

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. Line 7 prepares cluster activation flags to limit computations at layer L_{i+1} which is similar as line 4.

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.

From (2), the energy overhead for detecting important features arises from (i) computation of the threshold and (ii) a comparison operation. We limit the summation of all ofmap data, F^{of} only to important features, $\tilde{\mathcal{F}}$, for the currently predicted classes. For these computations, we model energy for an n-bit adder as $n^{0.922} \times e_{add}$ [14]. The multiplication of this summation by T_1^+/K corresponds to a few add-and-shift operations: empirically, this number varies from 3–5 for standard CNN topologies, and the computed threshold can be represented by at most 6 bits. For the *Conv* layer, the check is simplified to (4), where the sparsity summation involves the addition of one-bit zero flags, an inexpensive operation.

The energy savings, ΔE for layer L_i can be formulated as:

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

where E_{ov} includes the energy associated with lines 1 through 4 and lines 6 through 7 in Algorithm 3. The detailed computation of this overhead is simple and is omitted due to space constraints.

V. Results

Simulation Parameters/Models We demonstrate our energy-efficient CNN framework on two well-studied networks, LeNet applied to the MNIST handwritten digit recognition dataset, and AlexNet applied to the Imagenet dataset using Caffe platform [15]. Based on [1], [3], we assume the baseline, as defined in the beginning of Section IV, uses 8-bit words for ifmaps, filters, and ofmaps. We use 10,000 (5,000) images for the cluster learning phase and 5,000 (2,000) images for testing phase for AlexNet (LeNet). The top-5 (top-1) accuracy of AlexNet (LeNet) for the baseline is 78.30% (99.06%).

We use an internal simulator (details omitted due to space limitations) to determine the number of DRAM, SRAM, and RF memory accesses and computations for AlexNet and LeNet by modeling [4]. The per unit energies for DRAM, SRAM, and RF memory access are obtained from [2].

TABLE III: Modified bitwidths of early layers in AlexNet

Bitwidth	$input$	$c1$	$c2$	$c3$	$c4$	$c5$	$fc6$	$fc7$	$fc8$
ifmap/ofmap	6	7	5	6	5	8	8	8	8
filter	8	7	7	6	7	8	8	8	8

TABLE IV: Modified bitwidths of early layers in LeNet

Bitwidth	$input$	$c1$	$c2$	$fc1$	$fc2$
ifmap/ofmap	4	3	8	8	8
filter	3	4	8	8	8

Reduced Bitwidths We implement reduced precision bitwidths from the *input* layer to the $c4$ ($c1$) layer in AlexNet (LeNet). For later layers, the bitwidths are the same as the baseline.

The reduced precision bitwidths in various layers of CNN have trade-off relation with classification accuracy and energy savings. We used Monte-Carlo simulations for 1,500 prospective bitwidth combinations in early layers and checked the classification accuracy on 5,000 test images. The optimal bitwidth is chosen based on the maximum classification accuracy and energy savings. The modified bitwidths in AlexNet and LeNet are listed in Tables III and IV, respectively. The *Pool* layers use the same bitwidths as their previous *Conv* layer.

Selective Feature Activation We use our clustering based method for selective feature activation, as explained in Section II-B, for all layers beyond $c5$ for AlexNet and $c2$ for

LeNet. During the testing phase, the input image activates the clusters similar to its feature pattern to activate a smaller set of computations. For example, we find that the images of various non-shedding dogs (e.g., shih-tzus, spaniels, and terriers) are all seen to activate the same cluster in layer $fc7$ of AlexNet.

We define *average probable classes* of a layer as the average number of predicted classes for all the input test images in the layer. The average probable classes depends on the thresholds, T_1, T_2, T_3 and T_4 in *each layer* as described in (2), (5), (7) and line 3 in Algorithm 3, respectively. These can be tuned during training in the same way as any neural net parameter. Based on the empirical observations, we keep $T_2 = 0.6$ constant for all layers and vary thresholds T_1, T_3 and T_4 in various layers to obtain an accuracy vs. energy savings trade-off. Table V shows that for a particular combination of thresholds in all SeFAct layers, the average probable classes are reduced monotonically, which results in a significant improvement in percentage energy savings (PES) with respect to the baseline, for a small accuracy drop.

TABLE V: The effect of sample threshold on average probable classes, accuracy and PES

Network	Layer	T_1	T_3	T_4	Average probable classes	Accuracy (Baseline accuracy)	PES
AlexNet	$c5$	0.08	0.20	0.55	1000	72.5% (78.3%)	27.4%
	$fc6$	0.12	0.18	0.49	926		
	$fc7$	0.23	0.14	0.46	730		
LeNet	$c2$	0.60	0.60	0.22	10	96.1% (99.1%)	20.8%
	$fc1$	2.40	0.70	0.14	3		

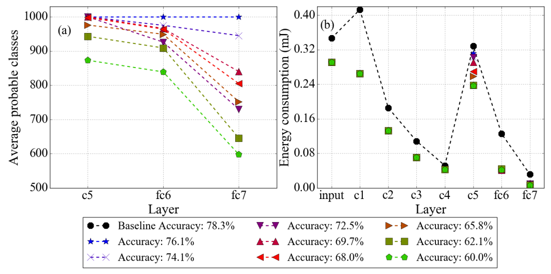


Fig. 4: (a) Average probable classes in layers with SeFAct implementation and (b) layer-wise energy consumption in AlexNet

The average number of probable classes of each layer using SeFAct in AlexNet for various threshold values, and the layer-wise energy for the baseline and our enhancement are shown in Fig. 4. Energy savings in the early layers come from the optimized bitwidth, while those in later layers change with the threshold values. Similar trends are seen for LeNet.

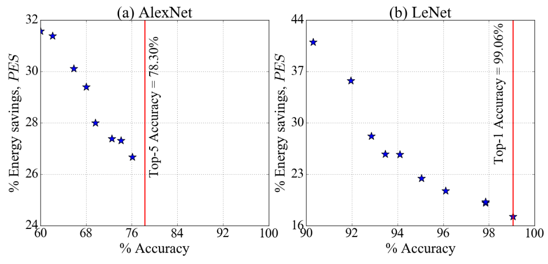


Fig. 5: PES vs. accuracy for (a) AlexNet (b) LeNet

Trade-off Between Energy Savings and Accuracy We compute the classification accuracy and energy savings for various combinations of threshold values. Fig. 5(a) (Fig. 5(b)) shows the trade-off between the PES and the accuracy (both normalized to the baseline) for AlexNet (LeNet). The red vertical line shows the accuracy of the baseline. The inexpensive overhead circuitries, described in Section IV, consume less than 1% of the total energy. Hence, significant energy savings are achievable: for small (5–10%) degradations in accuracy, 15–25% savings are possible. For the same energy savings, the relative percentage contributions between reduced bitwidth approximation and SeFAct are about 60% and 40%, respectively. For scenarios where low accuracy is acceptable (e.g., in mobile embedded systems, where battery limitations are the paramount consideration, and a best-effort accuracy is good enough), improvements of almost 40% are visible.

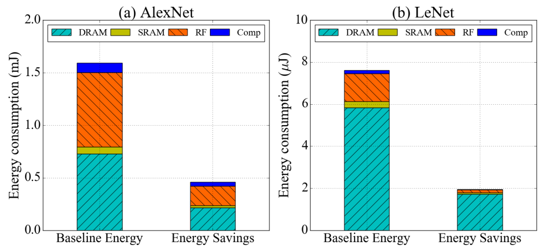


Fig. 6: Contribution of energy consumption of various operations for baseline energy and energy savings for (a) AlexNet (b) LeNet

The total energy and the contributions from DRAM, SRAM and RF memory accesses, and MAC computations are shown in Fig. 6. Results for both the baseline implementation and the energy savings for our approach in AlexNet and LeNet are shown, and it is easily seen that the memory access operations are the dominant components of both cases.

VI. Conclusion

This paper has proposed an effective way to dynamically reduce the energy of a CNN accelerator, using bitwidth reduction in early layers, and selective feature activiation in later layers. Large energy savings are seen, even for small accuracy losses.

References

[1] N. P. Jouppi, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ISCA*, pp. 1–12, 2017.
[2] V. Sze, *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of IEEE*, vol. 105, pp. 2295–2329, Dec 2017.
[3] P. Gysel, *et al.*, "Hardware-oriented approximation of convolutional neural networks," 2016. arXiv preprint arXiv:1604.03168.
[4] Y.-H. Chen, *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-St. Circ.*, vol. 52, no. 1, pp. 127–138, 2017.
[5] Y. Chen, *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. MICRO*, pp. 609–622, 2014.
[6] S. Han, *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," 2015. arXiv preprint arXiv:1510.00149.
[7] S. Gupta, *et al.*, "Deep learning with limited numerical precision," in *Proc. ICML*, pp. 1737–1746, 2015.
[8] S. Venkataramani, *et al.*, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proc. ISLPED*, pp. 27–32, 2014.
[9] V. Mrazek, *et al.*, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. ICCAD*, pp. 1–7, 2016.
[10] W. Yu, *et al.*, "Visualizing and comparing AlexNet and VGG using deconvolutional layers," in *Proc. ICML*, 2016.
[11] M. M. Halldórsson and J. Radhakrishnan, "Greed is good: Approximating independent sets in sparse and bounded-degree graphs," *Algorithmica*, vol. 18, pp. 145–163, May 1997.
[12] D. D. Lin, *et al.*, "Fixed point quantization of deep convolutional networks," in *Proc. ICML*, pp. 2849–2858, 2016.
[13] Y. Lecun, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of IEEE*, vol. 86, pp. 2278–2324, Nov 1998.
[14] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proc. ISSCC*, pp. 10–14, Feb 2014.
[15] Y. Jia, *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM Multimedia*, pp. 675–678, 2014.