RELEVANT-FEATURES BASED AUXILIARY CELLS FOR ROBUST AND ENERGY EFFICIENT DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural networks are complex non-linear models used as predictive analytics tool and have demonstrated state-of-the-art performance on many classification tasks. However, they have no inherent capability to recognize when their predictions are wrong. There have been several efforts in the recent past to detect natural errors i.e. misclassified inputs but the proposed mechanisms pose additional energy requirements. To address this issue, we present a novel post-hoc framework to detect natural errors in an energy efficient way. We achieve this by appending relevant feature based linear classifiers per class referred as Relevant features based Auxiliary Cells (RACs). The proposed technique utilizes the consensus of RACs appended at few selected hidden layers to distinguish correctly classified inputs from misclassified inputs. The combined confidence of RACs is used to determine if classification should terminate at an early stage. We demonstrate the effectiveness of our technique on various image classification datasets such as CIFAR10, CIFAR100 and Tiny-ImageNet. Our results show that for CIFAR100 dataset trained on VGG16 network, RACs can detect 46% of the misclassified examples along with 12% reduction in energy compared to the baseline network while 69% of the examples are correctly classified.

1 INTRODUCTION

Machine learning classifiers have achieved high performance on various classification tasks, e.g., object detection, speech recognition and image classification. Decisions made by these classifiers can be critical when employed in real-world tasks such as medical diagnosis, self-driving cars, security etc. Hence, identifying incorrect predictions i.e. detecting abnormal inputs and having a well-calibrated predictive uncertainty is of great importance to AI safety. Note that abnormal samples include natural errors, adversarial inputs and out-of-distribution (OOD) examples. Natural errors are samples in the test data which are misclassified by the final classifier in a given network.

Various techniques have been proposed in literature to address the issue of distinguishing abnormal samples. A baseline method for detecting natural errors and out-of-distribution examples utilizing threshold based technique on maximal softmax response was suggested by Hendrycks & Gimpel (2017). A simple unified framework to detect adversarial and out-of-distribution samples was proposed by Lee et al. (2018). They use activations of hidden layers along with a generative classifier to compute Mahalanobis distance (Mahalanobis, 1936) based confidence score. However, they do not deal with detection of natural errors.Bahat et al. (2019); Hendrycks & Gimpel (2017); Mandelbaum & Weinshall (2017) focus on detecting natural errors. Mandelbaum & Weinshall (2017) use distance based confidence method to detect natural errors based on measuring the point density in the effective embedding space of the network. More recently, Bahat et al. (2019) showed that KL-divergence between the outputs of the classifier under image transformations can be used to distinguish correctly classified examples from adversarial and natural errors. To enhance natural error detection, they further incorporate Multi Layer Perceptron (MLP) at the final layer which is trained to detect misclassifications.

Most prior works on the line of error detection do not consider the latency and energy overheads that incur because of the detector or detection mechanism. It is known that deeper networks expend higher energy and latency during feed-forward inference. Adding a detector or detection mechanism on top of this will give rise to additional energy requirements. The increase in energy may make

these networks less feasible to employ on edge devices where reduced latency and energy with the ability to identify abnormal inputs is significant.

Many recent efforts toward energy efficient deep neural networks (DNNs) have explored *early exit* techniques. Here, the main idea is to bypass (or turn off) computations of latter layers if the network yields high *confidence* prediction at early layers. Some of these techniques include the adaptive neural networks (Stamoulis et al., 2018), the edge-host partitioned neural network Ko et al. (2018), the distributed neural network (Teerapittayanon et al., 2017), the cascading neural network (Leroux et al., 2017), the conditional deep learning classifier (Panda et al., 2016) and the scalable-effort classifier (Venkataramani et al., 2015). So far, there has been no unified technique that enables energy efficient inference in DNNs while improving their robustness towards detecting abnormal samples.

In this work, we target energy efficient detection of natural errors, which can be extended and applied to detecting OOD examples and adversarial data. We propose adding binary linear classifiers at two or more intermediate (or hidden) layers of an already trained DNN and utilize the consensus between the outputs of the classifiers to perform early classification and error detection. This idea is motivated from the following two observations:

- If an input instance can be classified at early layers Panda et al. (2016) then processing the input further by latter layers can lead to incorrect classification due to over-fitting. This can be avoided by making early exit which also has energy benefits.
- We have observed that on an average, the examples which are misclassified do not have consistent hidden representations compared to correctly classified examples. The additional linear classifiers and their consensus enables identifying this inconsistent behaviour to detect misclassified examples or natural errors.

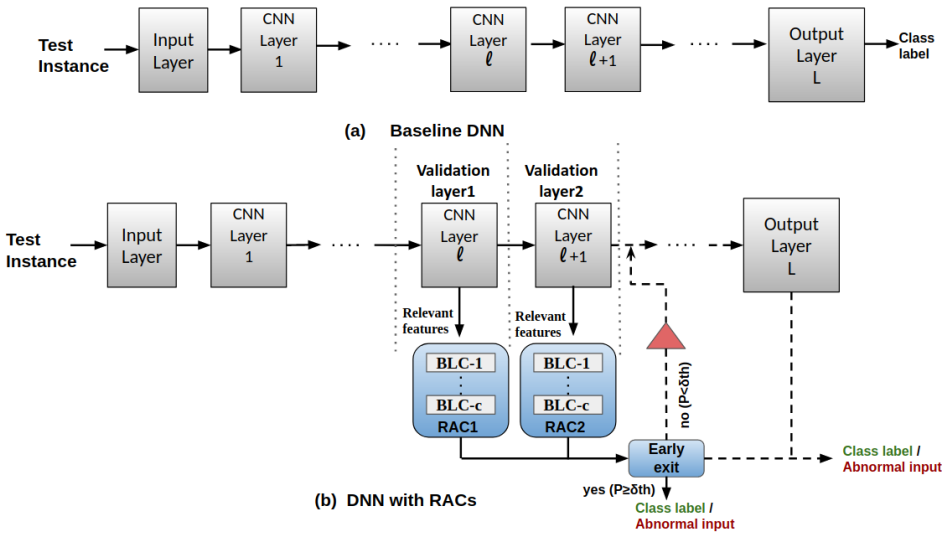


Figure 1: (a) Baseline Deep Neural Network (DNN). (b) DNN with Relevant features based Auxiliary Cells (RACs) added at validation layers (selected hidden layers) whose output is monitored to detect early classification

The training and construction of the linear classifiers is instrumental towards the accurate and efficient error detection with our approach. We find that at a given hidden layer, the error detection capability (detecting natural errors) is higher if we use class-specific binary classifiers trained on the corresponding relevant feature maps from the layer. In fact, using a fully connected classifier trained on all feature maps (conventionally used in early exit techniques of Panda et al. (2016)) does not improve error detection capability. Training these binary classifiers on relevant features can be considered as encoding prior knowledge on the learned hidden feature maps, thereby, yielding better detection capability. Besides improved error detection, a key advantage of using class wise binary

linear classifiers trained on only relevant features is that they incur less overhead in terms of total number of parameters, as compared to a fully connected classifier trained on all feature maps.

We use "relevant features" and class specific classifiers instead of using all the feature maps and one fully connected classifier at a hidden layer for the following reasons: (a) We have observed that at a given hidden layer, the detection capability (detecting natural errors) is higher if we use relevant feature maps with class-specific binary classifiers than using a fully connected classifier trained on all feature maps. Training these binary classifiers on relevant features can be considered as encoding prior knowledge on the learned hidden features maps and hence is expected to have better detection capability. (b) The class wise binary linear classifiers trained on relevant features have less number of parameters compared to a fully connected classifier trained on all the feature maps.

In the proposed framework, class-specific binary linear classifiers are appended at few selected hidden layers which have maximal information. These hidden layers are referred to as *validation layers*. The set of all binary linear classifiers at a validation layer constitute a *Relevant feature based Auxiliary Cell (RAC)*. We use the consensus of RACs to detect natural errors which improves the robustness of the network. The combined confidence of RACs is used to perform early classification that yields energy efficiency.

2 Feature Relevance Matrix

DNNs (or convolutional networks) trained for classification tasks compute a set of features at each convolutional layer. At each layer, there might be few feature maps which are highly responsible for activating an output node corresponding to class c compared to others. For example, a high-level feature can represent $whiskers$ (say) which are relevant to classes like cat and dog but not to classes like $truck$ and $airplane$. Hence, the feature map computed from this filter is considered as relevant feature to class cat and dog. Our approach of adding linear classifiers to trained DNNs follows two-steps: 1) First, we heuristically select few intermediate convolutional layers (or *validation layers*) with maximal information based on empirical observations of False Negative Rate (FNR) and True Negative Rate (TNR) (see sec. 4.1). 2) Then, we calculate the class-wise relevant features at the selected *validation layers* that are eventually used to train the binary linear classifiers.

Algorithm 1: Methodology to Compute Feature-Relevance Matrix at layer l

Input: Trained DNN, Training data $\{(x_i, y_i)\}_{i=1}^{N}$: $x_i \in$ input sample, $y_i \in$ true label
Parameters: number of classes $= c$, number of layers $= L$, feature maps at layer l: $\{f_1, f_2, \ldots, f_r\}$, relevance score of node p at layer l = R_p^l

Initialize feature-relevance matrix for given layer l: $F_l = zeros(c, r)$
for each sample (x_i, y_i) in training data **do**
  Forward propagate the input x_i to obtain the activations of all nodes in the DNN
  Compute relevance scores for output layer: $R_p^L = \delta(p - y_i) \;\; \forall p \in \{1, \ldots, c\}$
  $\delta(p - y_i) =$ Dirac delta function
  for k in $range(L-1, l, -1)$ **do**
    Back propagate relevance scores: $R_p^k = \sum_q (\alpha \frac{a_p w_{pq}^+}{\sum_p a_p w_{pq}^+} - \beta \frac{a_p w_{pq}^-}{\sum_p a_p w_{pq}^-}) R_q^{k+1}$
    $\forall\, p \in$ nodes of layer k, $\;\alpha - \beta = 1$
    $a_p = activations$, $w_{pq} = weights$
  end for
  Compute average relevance score per feature map at layer l
  Relevance score vector at layer l: $R^l = \left\{ R_{f_j}^l = \frac{1}{\sum\limits_{p \in f_j} 1} \Big(\sum\limits_{p \in f_j} R_p^l \Big) \right\}_{j=1}^{r}$
  Update feature-relevance matrix: $F_l(y_i, :) = R^l$
end for
Average rows of feature-relevance matrix: $F_l(p, :) = \frac{1}{\sum\limits_{\forall y_i \in p} 1} F_l(p, :) \quad \forall p \in \{1, \ldots, c\}$

return Feature-Relevance Matrix F_l

To obtain relevant features, we define a feature-relevance matrix at each validation layer which assigns class-wise relevance score to every feature map. The relevance score of a feature map for any given class (say cat) indicates its contribution in activating the output node (corresponding to cat). Algorithm 1 shows the pseudo code for computing the feature-relevance matrix. The process takes a pre-trained DNN and training data with corresponding labels as inputs and computes feature relevance matrix F_l for a particular layer l. Each row in F_l indicates the relevance scores of all the features maps at layer l corresponding to a unique class c from the dataset. In particular, $F_l(i, j)$ indicates the relevance score of feature map f_j at layer l corresponding to class i in the dataset.

We use *Layer-wise Relevance Propagation (LRP)* proposed by Sebastian et al. (2015) to compute class-wise relevance scores of feature maps. LRP computes the contribution of every node in the network to the prediction made for an input image. The relevance scores at output nodes are determined based on true label of an instance. For any input sample (x_i, y_i), the output node corresponding to true class, y_i, is given a relevance score of 1 and the remaining nodes get a score of 0. These relevance scores are then back propagated based on $\alpha\beta$*-decomposition rule* (Wojciech et al., 2016) with $\alpha = 2$ and $\beta = 1$. After determining the relevance scores at each node in the network, we compute relevance score of every feature map f_i at layer l by averaging the scores of all nodes corresponding to f_i. The relevance vector of a feature map f_i is obtained by taking class-wise average over relevance scores of all training samples and forms the i^{th} column of feature-relevance matrix F_l. The computed feature-relevance matrix is then used to determine relevant features for each class at validation layers.

3 RELEVANT FEATURES BASED AUXILIARY CELL (RAC)

In this section, we present our approach to designing DNNs with RACs. Fig. 1 shows the conceptual view of DNNs with RACs. Fig. 1(a) consists of the baseline DNN with L layers. We have not shown the pooling layers or the filters for the sake of convenience in representation. Fig. 1(b) illustrates our approach wherein the output relevant features from two hidden layer $l, l + 1$ which are referred as validation layers are fed to RACs. Note that the two validation layers need not be consequent.

An RAC consists of c *Binary Linear Classifiers (BLCs)*, where c represents the number of output nodes or the number of classes. Each BLC within an RAC corresponds to a unique class in the dataset and is trained on relevant features corresponding to that class. The output of BLC corresponding to class (say $c1$) in an RAC indicates the probability of a given instance x_i belonging to class $c1$, $P(y_i = c1|x_i)$. We can thus gather that output from an RAC (class label RAC_{class} and associated probability or confidence RAC_{prob}) will correspond to the BLC with maximum value as:

$$RAC_{class} = argmax_{i=1,2..c} BLC_i \tag{1}$$

$$RAC_{prob} = max_{i=1,2..c} BLC_i \tag{2}$$

The probability (RAC_{prob}) generated by the RAC is considered as its *confidence score*. Besides the RACs, an activation module is added to the network (traingle in Fig. 1(b)) similar to that in Panda et al. (2016). The activation module uses the consensus of RACs and their confidence scores to decide if an input instance classification can be terminated at the present layer or not.

3.1 TRAINING RACS

We proceed to train RACs after determining feature-relevance matrices (see sec. 2) at validation layers. Algorithm 2 shows the pseudo code for training RACs. The initial step in this process is to determine the relevant features for each class at the validation layers using feature-relevance matrix. For every class j, we arrange feature maps in the descending order of their class relevance score and top 'k' feature maps are marked as relevant features for class j. Once the relevant features for each class are determined, they remain unchanged. The classifier of class j (BLC-j) is trained on the corresponding relevant features from the training data. Note, the relevant feature maps which are fed to RACs are obtained after the batch-norm and ReLU operation on selected convolutional layer (validation layer). The BLCs (BLC-1,...,BLC-c) in an RAC can be trained in parallel as they are independent of each other.

Algorithm 2: Methodology to Train an RAC at layer l

Input: Trained DNN, Training data $\{(x_i, y_i)\}_{i=1}^{N}$, feature-relevance matrix F_l
Parameters: number of class $= c$, feature-relevance matrix $= F_l$

for each class $j \in 1, \ldots, c$ **do**
    Determine top k relevant features of class j at layer l from $F_l(j, :)$
    Obtain relevant features i.e. $x_i^{lj} \ \ \forall i \in \{1, \ldots, N\}$ by forward propagating x_i through DNN
    Get the binary labels for training data: $\tilde{y}_i = \delta(j - y_i) \ \ \forall i \in \{1, \ldots, N\}$
    Initialize a binary linear classifier (BLC-j) with no hidden layers
    Train BLC-j using $\{(x_i^{lj}, \tilde{y}_i)\}_{i=1}^{N}$ as training data
    return BLC-j
end for

3.2 Early classification and Error detection

The overall testing methodology for DNNs with RACs is shown in Algorithm 3. We adopt the early exit strategy proposed in Panda et al. (2016) and modify it to perform efficient classification and abnormal input detection with RACs. Given a test instance I_{test}, the methodology either produces a class label C_{test} or makes *No Decision (ND)*. The output from RACs is monitored to decide if early classification can be made for an input. If the decision made by RACs across the selected validation layers do not agree with each other, then the network outputs *ND* indicating the possibility of misclassification at the final output layer of the DNN. If the RACs across all validation layers predict same class label c, then, we use a pre-defined confidence threshold (δ_{th}) to decide on early classification as follows:

- If $confidence$ score (RAC_{prob}) across all RACs is greater than δ_{th}, we output c as final decision and terminate the inference at the given validation layer without activating any latter layers.
- If $confidence$ score (RAC_{prob}) in any of the RACs is lesser than δ_{th}, the input is fed to the latter layers and the final output layer of the DNN is used to make the prediction.

In the second case above, all remaining layers from $l + 2$ onwards in Fig. 1(b) will be activated and the output of the final layer (L) is used to validate the decision made by RACs. If an input is classified at RACs either as ND or C_{test} (thus, not activating the layers beyond validation layers), then it is considered as an early classification. In Fig. 1, testing is terminated at layer $l + 1$ in case of early classification.

In summary, appending RACs into DNNs enables us to perform early classification with the ability to output a no decision ND that helps in detecting natural errors (and abnormal inputs). It is evident that early classification will translate to energy-efficiency improvements. The user defined threshold, δ_{th}, can be adjusted to achieve the best trade-off between efficiency and error detection accuracy. We believe that the proposed methodology is systematic and can be applied to all image recognition applications.

4 Experimental methodology

In this section, we describe the experimental setup used to evaluate the performance of Deep Neural Networks (DNNs) with relevant features based Auxiliary Cells (RACs). We demonstrate the effectiveness of our framework using deep convolutional networks, such as VGGNet (Szegedy et al., 2015) and ResNet (He et al., 2016) for image classification tasks on CIFAR (Alex & Geoffrey, 2009) and Tiny-ImageNet (Jia et al., 2009) datasets. For the problem of detecting out-of-distribution (OOD) samples, we have used LSUN (Fisher et al., 2015), SVHN (Yuval et al., 2011) and TinyImageNet datasets as OOD samples for networks trained on CIFAR10 and CIFAR100 datasets. For generating adversarial examples, Carlini and Wagner attack (Carlini & Wagner, 2017) has been used.

We measure the following metric to quantify the performance: percentage of good decisions, percentage of bad decisions and percentage of early decisions. The inputs which are either correctly

Algorithm 3: Methodology to Test the DNN with RACs

Input: Test instance I_{test}, DNN with RAC-1 and RAC-2 at validation layers l and $l+1$ respectively
Output: Indicates class label (C_{test}) or detects abnormal input as No-Decision (*ND*)

Obtain the DNN layer features for I_{test} corresponding to layers l and $(l+1)$
Activate and obtain the output from RAC-1 and RAC-2
If class label outputted by RAC-1 $=$ RAC-2 **do**
    If confidence value of each RAC is beyond a certain threshold δ_{th} **do**
        Terminate testing at layer $(l+1)$
        Output C_{test} = class label given by RACs
    else do
        Activate remaining layers and obtain decision of final classifier (FC)
        If class label given by FC $=$ RACs then output C_{test} = class label given by FC
        If class label given by FC $\neq$ RACs then output *ND*
    end if
else do
    Terminate testing at layer $(l+1)$
    Output *ND*
end if

classified or classified as no-decision contribute towards good decisions. Note that DNN with RACs outputs 'no-decision', when the input can be potentially misclassified by the final classifier. The inputs which are misclassified even by the DNN with RACs are considered as bad decisions. Note the inputs fall into three different buckets in case of DNN with RACs: (a) Inputs which are correctly classified (b) Inputs which are classified as 'no-decisions' (c) Inputs which are incorrectly classified. We report False Negative Rate (FNR) and True Negative Rate (TNR) to evaluate the error detection capability and the average number of operations (or computations) per input (# OPS) to measure energy efficiency. The negatives are the inputs that are misclassified by the baseline DNN and positives are the inputs that are correctly classified by the baseline DNN. True negative rate is the percentage of misclassified (by baseline DNN) inputs which are classified as 'no-decisions' by DNN with RACs. False negative rate is the percentage of correctly classified examples (by baseline DNN) which are classified as 'no-decisions' by DNN with RACs.

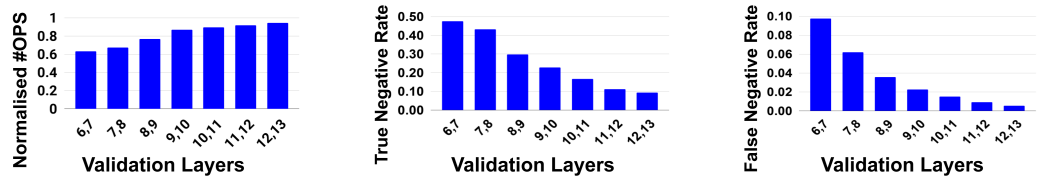
Our goal is to increase the true negative rate and improve energy efficiency (decrease # OPS) while maintaining the false negative rate as low as possible. We observed that these three metrics - true negative rate, false negative rate and # OPS are sensitive to hyper-parameters related to RACs and so we have carried out series of experiments to determine their effect. The details of these experiments are shown in the following section (Sec. 4.1).

4.1 Tuning Hyper-parameters

The following are the three hyper-parameters which affect true negative rate, false negative rate and energy efficiency (# OPS):

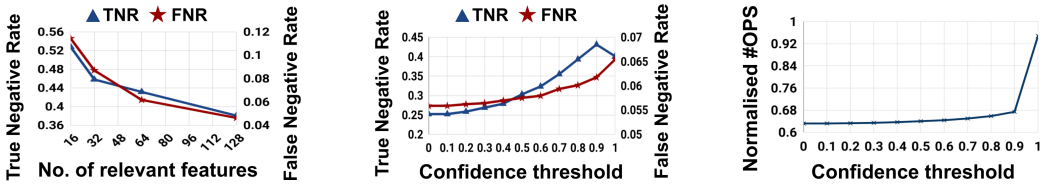
- The choice of validation layers (l, $l+1$)
- Number of relevant features (k) used at each validation layer
- Confidence threshold δ_{th}

We use heuristic based methods to tune the above mentioned hyper-parameters. First, lets understand how validation layers are chosen and their effect on detection capability. The validation layers can not be initial layers as they do not have the full knowledge of network hierarchy and the feature maps at these layers are not class specific. We observed that the hidden layers just before the final classifier (FC) make similar decisions as that of the final classifier and hence are not useful to detect natural errors. Thus, the hidden layers which are in between (yet, closer to FC) are suitable as validation layers. Fig. 2 shows the change in FNR, TNR and normalized #OPS with respect to change in the choice of the validation layers for CIFAR-10 dataset trained on VGG16 network.



(a) (b) (c)

Figure 2: (a) Normalised #OPS as the validation layers are shifted towards the final classifier (b) True negative rate as the validation layers are shifted towards the final classifier (c) False negative rate as the validation layers are shifted towards the final classifier



(a) (b) (c)

Figure 3: (a) TNR and FNR as the no. of relevant features k is increased at RACs (b) TNR and FNR as the confidence threshold δ_{th} is increased at RACs (c) Normalized #OPS as the confidence threshold δ_{th} is increased at RACs.

As validation layers move deeper into the network, both true negative rate and false negative rate tend to decrease (Fig. 2b, 2c). We heuristically select a pair of hidden layers as validation layers which result in the smallest false negative rate in the range of 5%-10%. For example, the smallest FNR in the range of 5%-10% is obtained when we choose layer 7 and layer 8 as validation layers for VGG16 network trained on CIFAR10 dataset (Fig. 2c).

Number of relevant features 'k' is another hyper-parameter which affects false negative and true negative rates. As we increase the number of relevant features k, both FNR and TNR decreases. Fig. 3a shows the change in FNR and TNR with respect to the change in the number of relevant features k for CIFAR-10 dataset trained on VGG16 network with validation layers at layer 7 and 8. The optimal value of k depends on the dataset and the network used. We increment k by powers of 2, compute the corresponding FNR and TNR and select the optimal k from these observations. Note that # OPS increase as 'k' increases. For example, consider the case of VGG16 trained on CIFAR10 with validation layers at layer 7 and 8. When we increment k from 64 to 128 at these validation layers, the FNR drops by 1.6% changing from 6.2% to 4.6% while TNR drops by 5%. Hence, we choose k as 64 for CIFAR10 with VGG16 network.

The confidence threshold δ_{th} is a user defined hyper-parameter which also influences energy efficiency and detection capability. The activation module discussed in Section. 3 compares the confidence score produced by RACs to the confidence threshold δ_{th} and selectively classifies the input at RAC stage or at the final classifier. Thus, we can regulate δ_{th} to modulate the number of inputs being passed to latter layers. Note that δ_{th} has no contribution to the decision made when the RACs do not output same class. The confidence threshold also affects the TNR and the FNR. However, the change in FNR with confidence threshold is negligible (see Fig. 3b), around 0.1% for 0.1 change in δ_{th}. Fig. 3c shows the variation in normalized OPS (with respect to baseline DNN) with different δ_{th} for VGG16 network trained on CIFAR10 dataset with RACs appended at layer 7 and 8.

As we increase δ_{th}, TNR increases because higher δ_{th} would qualify more inputs to be verified by the final classifier. However, beyond a particular δ_{th}, a fraction of inputs which are correctly classified at early stages can be detected as natural errors because of the increase in confusion. The maximum TNR can be achieved at δ_{th} = 0.9 (in Fig. 3b) and this point is referred as δ^*_{th}. The number of OPS increases as we increase δ_{th} but the rate of increase is significant beyond δ^*_{th}. In Fig. 3b, we observe that the TNR increases from 39.34% (δ_{th}=0.8) to 43.15% (δ_{th}=0.9) while the normalized

Table 1: Baseline network details and the complexity of hidden linear classifiers used for our technique.

Dataset	Network	Baseline Error	# of Params	validation layers	additional # of params
CIFAR10	VGG16	7.88	33.6 M	7,8	0.08 M
	Res18	5.76	11.2 M	12, 13	0.33 M
CIFAR100	VGG16	25.62	34.0 M	9, 10	0.41 M
	Res34	24.56	21.3 M	31, 32	0.82 M
TinyImageNet	Res18	43.15	11.3 M	15, 16	0.41 M

#OPS increase from 0.66 to 0.67. Further increase in δ_{th} degrades the TNR and increases the #OPS by significant amount. Thus, δ_{th} serves as a knob to trade TNR for efficiency and can be easily adjusted during runtime to get the optimal results.

4.2 Experimental Results

This section summarizes results on detection capability and energy efficiency obtained from DNN with RACs. We train VGGNet with 16 layers and ResNet with 18 layers for classifying CIFAR10. For training CIFAR100 dataset, we use VGGNet with 16 layers and ResNet with 34 layers. In addition, we have trained ResNet 18 architecture with TinyImageNet dataset. Table. 1 indicates the baseline error, the number of parameters in the baseline network, the validation layers used and the additional number of parameters added due to inclusion of RACs. Table. 2 validates the performance of our proposed technique. Fig. 4a indicates the reduction in classification error for various networks and datasets. We observe that DNN with RACs can detect $(43 - 45)\%$ of the natural errors while maintaining the accuracy at $(86 - 89)\%$ for CIFAR10 dataset. For CIFAR100 dataset, we observe slightly higher detection rate i.e. $(46 - 49)\%$ with an accuracy range of $(67 - 69)\%$. The detection rate is much higher (62%) for Tiny-Imagenet dataset trained on ResNet18. However, the accuracy drops from 56.85% to 41.28%. This can be potentially improved by using deeper networks such as DenseNet. Note that the decrease in accuracy is not because of misclassification but is because of false detection and the falsely detected examples fall into the no-decision bucket. Therefore, even though the percentage of correctly classified examples decrease slightly, we avoid around 50% of natural errors compared to the baseline network. Figure. 4b shows the normalized improvement in efficiency with respect to the baseline DNN for different datasets and networks.

Table 2: Detecting incorrect classifications and making early decisions for image classification task. All the values are percentages.

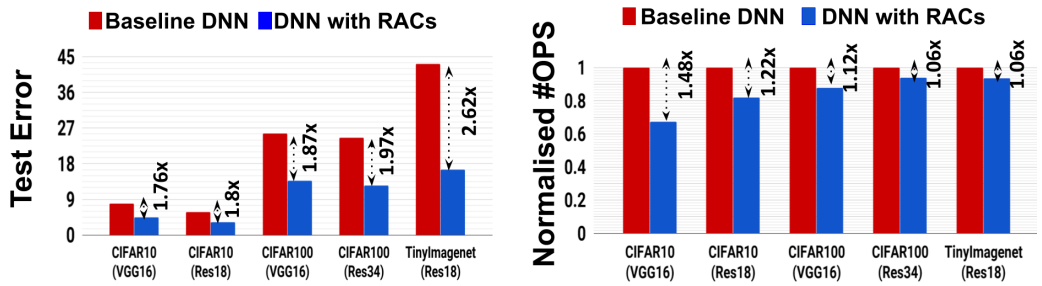
Dataset	Network	Good decisions (%)		Bad decisions (%) (Error)	FNR (%)	TNR (%)	Early decisions (%)
		Correct decisions	No decisions				
CIFAR10	VGG16	86.43	9.09	4.48	6.00	43.00	88.55
	Res18	88.81	7.99	3.20	5.76	44.44	74.58
CIFAR100	VGG16	68.6	17.67	13.73	7.7	46.4	87.03
	Res34	66.78	20.74	12.48	11.48	49.19	90.39
Tiny-ImageNet	Res18	41.28	42.26	16.46	27.39	61.85	95.24

We also evaluate robustness of our framework against adversarial and out-of-distribution (OOD) inputs for CIFAR10 and CIFAR100 datasets. The adversarial samples are generated using targeted Carlini & Wagner (CW) attack with L2 norm (Carlini & Wagner, 2017). We have considered both zero knowledge adversary and full knowledge adversary to evaluate robustness of DNN with RACs. The zero knowledge adversaries are created such that the attack has (95-100)% success rate in fooling the final classifier of the DNN. The mean adversarial distortion (average imposed L2 norm) and adversarial TNR is shown in Table. 3. For the zero knowledge evaluation, adversarial TNR indicates

Table 3: Performance of our technique on detecting adversarial and out-of-distribution data for image classification task. The reported TNR for adversarial and OOD detection is computed at FNR mentioned in Table. 2. All the values are percentages.

Dataset	**Network**	**Zero knowledge**		**Full knowledge**		**OOD TNR (%)**		
		Adv TNR	mean $\|.\|_2$	Adv TNR	mean $\|.\|_2$	Tiny Imagenet	LSUN	SVHN
CIFAR10	VGG16	38.42	1.74	57.35	1.78	44.25	48.10	63.96
	Res18	44.76	1.38	9.10	1.32	60.55	68.46	70.13
CIFAR100	VGG16	37.39	1.01	28.39	1.01	43.03	38.28	38.02
	Res34	43.49	0.79	13.95	0.73	58.55	60.30	58.56

the percentage of successful adversaries detected as no decisions. Note that the adversarial examples which can fool the final classifier are considered as successful adversaries in case of zero knowledge attack. Full knowledge adversaries are created by including the RACs' loss to the objective function that is optimized by CW attack (refer Appendix A.1). We have reported the adversarial detection rate of DNN with RACs for full knowledge adversaries at mean adversarial distortion similar to zero knowledge adversaries. Table. 3 also shows the detection capability of RACs in case of out-of-distribution examples. DNN with RACs are not very effective in detecting the adversarial examples when the threat model has complete knowledge of the detection mechanism. The proposed framework not only helps in detecting natural errors but also detects out-of-distribution examples while being energy efficient than the baseline network.



(a) (b)

Figure 4: (a) Test error comparison between baseline DNN and DNN with RACs (b) Normalized OPS benefits with respect to baseline

5 Conclusion

Deep neural networks are crucial for many classification tasks and require robust and energy efficient implementations for critical applications. In this work, we device a novel post-hoc technique for energy efficient detection of natural errors. In essence, our main idea is to append class-specific binary linear classifiers at few selected hidden layers referred as Relevant features based Auxiliary Cells (RACs) which enables energy efficient detection of natural errors. With explainable techniques such as Layerwise Relevance Propagation (LRP), we determine relevant hidden features corresponding to a particular class which are fed to the RACs. The consensus of RACs (and final classifier if there is no early termination) is used to detect natural errors and the confidence of RACs is utilized to decide on early classification. We also evaluate robustness of DNN with RACs towards adversarial inputs and out-of-distribution samples. Beyond the immediate application to increase robustness towards natural errors and reduce energy requirement, the success of our framework suggests further study of energy efficient error detection mechanisms using hidden representations.

REFERENCES

Krizhevsky Alex and Hinton Geoffrey. Learning multiple layers of features from tiny images. 2009.

Yuval Bahat, Michal Iranu, and Gregory Shakhnarovich. Natural and adversarial error detection using invariance to image transformations. In *arXiv preprint arXiv:1902.00236v1*. 2019.

Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM workshop on AISec*. 2017.

Yu Fisher, Seff Ari, Zhang Yinda, Song Shuran, Funkhouser Thomas, and Xiao Jianxiong. Construction of a large-scale image dataset using deep learning with humans in the loop. In *arXiv preprint arXiv:1506.03365*. 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*. 2016.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*. 2017.

Deng Jia, Dong Wei, Socher Richard, Li Li-Jia, Li Kai, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*. 2009.

Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In *15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2018.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems 31*, pp. 7167–7177. 2018.

Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. The cascading neural network: building the internet of smart things. In *Knowledge and Information Systems 52, issue 3*. 2017.

Chandra Prasanta Mahalanobis. On the generalised distance in statistics. In *Proceedings of the National Institute of Sciences of India*, pp. 49–55. 1936.

Amit Mandelbaum and Daphna Weinshall. Distance-based confidence score for neural network classifiers. In *arXiv preprint arXiv:1709.09844*. 2017.

Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2016.

Bach Sebastian, Binder Alexander, Montavon Gregorie, Klauschen Frederick, Muller Klaus-Robert, and Samek Wojciech. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. In *Plos One*. 2015.

Dimitrios Stamoulis, Ting-Wu Chin, Anand Krishnan Prakash, Haocheng Fang, Sribhuvan Sajja, Mitchell Bognar, and Diana Marculescu. Designing adaptive neural networks for energy-constrained image classification. In *ICCAD '18 Proceedings of the International Conference on Computer-Aided Design, Article No. 23*. 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*. 2015.

Surat Teerapittayanon, Bradley McDanel, and H.T Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *ICDCS, IEEE*, pp. 328339. 2017.

Swagath Venkataramani, Anand Raghunathan, Jie Liu, and Mohammed Shoaib. Scalable-effort classifiers for energy-efficient machine learning. In *DAC*. 2015.

Samek Wojciech, Montavon Gregorie, Binder Alexander, Lapuschkin Sebastian, and Muller Klaus-Robert. Interpreting the predictions of complex ml models by layer-wise relevance propagation. In *arXiv preprint arXiv:1611.08191v1*. 2016.

Netzer Yuval, Wand Tao, Coates Adam, Bissacco Alessandro, Wu Bo, and Ng Andrew Y. Reading digits in natural images with unsupervised feature learning. In *Neural Information Processing Systems (NIPS) workshop*. 2011.

A APPENDIX

A.1 ADVERSARIAL ATTACKS

We consider the zero knowledge and full knowledge adversaries to evaluate the robustness of the proposed framework towards adversarial examples targeted Carlini-Wagner (CW) with L2 norm. The zero knowledge targeted CW-L2 attacks are constructed as

$$\underset{x_{cw}}{\arg\min} \ \{\|x - x_{cw})\|_2^2 + c.[f_y(x_{cw}) - f_t(x_{cw})]\} \tag{3}$$

where x is an input by the final classifier, x_{cw} is a required adversary, c is the penalty term, $f_y(x_{cw})$ is final classifier's logit value corresponding to true class and $f_t(x_{cw})$ is the logit value corresponding to target class when x_{cw} is given as the input to the network. The target class for correctly classified is chosen as the class with second highest output softmax value. The c value is adaptively chosen for each network such that the attack success rate CW attack is 95-100%.

For generating full knowledge attacks, we modify the objective function indicated by expression 3 to include RACs' loss. The full knowledge targeted CW-L2 attacks are constructed as

$$\underset{x_{cw}}{\arg\min} \ \{\|x - x_{cw})\|_2^2 + c_1.[f_y(x_{cw}) - f_t(x_{cw})] + c_2\big[\sum_{i \neq t}(f_i^1 - f_t^1)\big] + c_3\big[\sum_{i \neq t}(f_i^2 - f_t^2)\big]\}$$

where c_1, c_2, c_3 are penalty factors, t is the target class, f_j^1 indicates the output of the BLC-j in RAC1 before applying sigmoid non-linearity, f_j^2 indicates the output of the BLC-j in RAC2 before applying sigmoid non-linearity. We have adaptively chosen the constants c_1, c_2, c_3 such that the mean L2 distortion between x and x_{cw} is similar to the zero knowledge attack. For both zero knowledge and full knowledge attack, the chosen values of hyper-parameters are as follows: learning rate = 0.01, maximum number of iterations = 400.