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 might go wrong. There have been several efforts in the recent past to detect natural errors i.e. misclassified inputs but these 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 features based linear classifiers per class referred as Relevant features based Auxiliary Cells (RACs). The proposed technique makes use of the consensus between RACs appended at few selected hidden layers to distinguish the correctly classified inputs from misclassified inputs. The combined confidence of RACs is utilized 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 such as object detection, speech recognition, image classification among others. The decision made by these classifiers can be critical when employed in real-world tasks such as medical diagnosis, self-driving cars, security etc. Hence, identifying when a prediction is incorrect 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 examples. Natural errors are the samples in the training and test data which are misclassified by the final classifier in the network.

Various techniques have been proposed in literature to address the issue of distinguishing abnormal samples. Hendrycks & Gimpel (2017) proposed a baseline method for detecting natural errors and out-of-distribution examples utilizing threshold based technique on maximal softmax response. A simple unified framework to detect adversarial and out-of-distribution samples has been proposed by Lee et al. (2018). They use the activations from the hidden layers along with a generative classifier to compute Mahalanobis distance (Mahalanobis, 1936) based confidence score. However, this work does not deal with the detection of natural errors. Hendrycks & Gimpel (2017); Mandelbaum & Weinshall (2017); Bahat et al. (2019) 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 misclassification. But none of these works evaluate the robustness against all three types of abnormal inputs.

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. The advancement towards deeper net-

works has drastically increased the latency and energy required for feed-forward inference. Adding a detector or detection mechanism on top of this will give rise to additional energy requirements. This increase in energy may make these networks less feasible to employ in critical scenarios using portable devices where latency, energy and identifying abnormal inputs are important factors. There has been plethora of efforts to enable energy efficiency based on *early exit* conditions placed at individual layers of a convolutional neural network (CNN), aiming at bypassing later stages of a CNN if the classifier has a *confident* prediction in earlier stages. 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). However, there is no unified technique which enables energy efficiency and detects abnormal samples to improve robustness of deep neural network (DNN). We aim to improve the robustness of the network by detecting abnormal samples while being energy efficient.

In particular, we focus on detecting natural errors in an energy efficient manner. Our idea is to determine set of relevant features corresponding to each class at hidden layers which are utilized to detect natural errors and make early classifications. The relevant features are obtained from few selected hidden layers which have maximal information. The chosen hidden layers are referred as *validation layers*. We compute relevance scores of each feature map at validation layers with respect to all the classes in the dataset. Relevance score of a feature map corresponding to class c indicates its contribution in activating the final output node of c. Relevant features of c are the set of feature maps at validation layers that have high relevance scores corresponding to class c in the dataset. The relevant features thus obtained are fed to *Relevant feature based Auxiliary Cells (RACs)* which are a set of binary linear classifiers appended to the DNN. We use the consensus between RACs to detect natural errors and the combined confidence of RACs to decide on early classification. Thus we achieve robustness by detecting natural errors and energy efficiency through early classification.

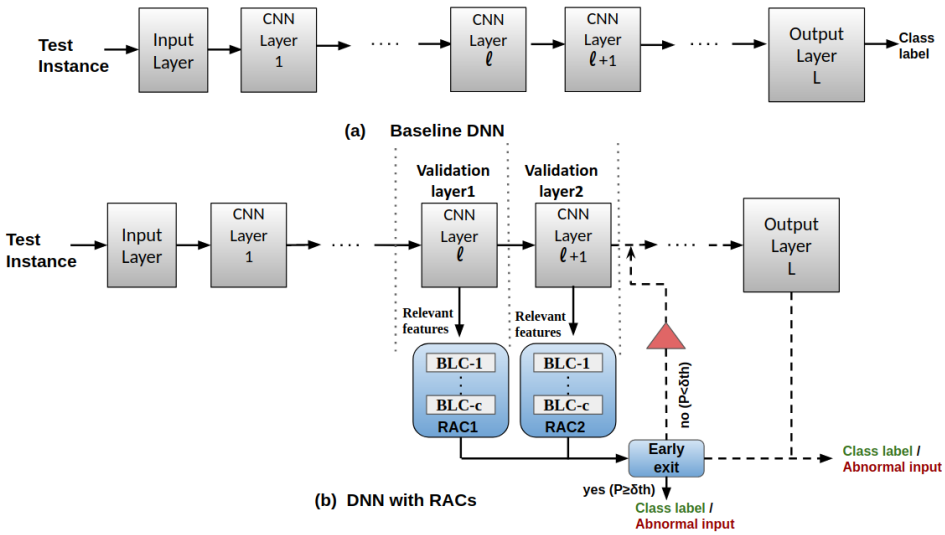


Figure 1: (a) Baseline Deep Neural Network (DNN). (b) DNN with Relevant features based Auxiliary Cells (RACs) added at validation layers (selected hidden layers) whose output is monitored to detect early classification

2 Feature Relevance Matrix

Deep Neural networks (DNNs) 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 a particular class node (say c) at the output compared to other feature maps. We consider these features as relevant features of c. Relevant features can be computed at each layer for every class in the dataset. We focus on determining the relevant features at few selected hidden layers

which have maximal information. These layers are referred as validation layers and are selected heuristically (see sec. 4.1). For example, a filter at a higher layer i.e. a high-level feature can represent whiskers which are relevant to classes like cat and dog but not to classes like truck and airplane. So the feature map computed from this filter is considered as relevant feature to class cat and dog. (Note that we are considering special case of DNN models which are Convolutional Neural Networks (CNNs). However, our framework can be extended to any DNN model.)

Determining relevant features at validation layers plays a critical role in the proposed framework as RACs are trained on these features. To obtain relevant features, we define a feature-relevance matrix at each validation layer which assigns a class-wise relevance score to every feature map. The relevance score of a feature map for any given class (say cat) indicates its contribution to the softmax value of the final output node (corresponding to cat). Algorithm 1 shows the pseudo code for computing the feature-relevance matrix. The process takes pre-trained Deep Neural Network (DNN) and training data with corresponding labels as inputs and it outputs a 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 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. For example, consider a VGGNet with 16 layers trained on CIFAR10 dataset. The 7^{th} convolutional layer will output 256 feature maps and hence the feature-relevance matrix at this layer is of size 10×256. The first row in the matrix F_7 indicates the relevance scores of feature maps in layer 7 corresponding to class 0 i.e. airplane.

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

We use *Layer-wise Relevance Propagation (LRP)* proposed by Sebastian et al. (2015) to compute the class-wise relevance scores of hidden 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 i.e. 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 of each node in the network, we compute the relevance score of every feature map f_i at layer l by averaging the scores of all the nodes corresponding to f_i. The same procedure is repeated for all samples in the training data. For every feature map at layer l, all the relevance values computed from the training data are

averaged class-wise to obtain one relevance score per class i.e. class relevance scores. The class relevance scores of a feature map f_i at layer l forms the i^{th} column of feature-relevance matrix F_l. The computed feature-relevance matrix is then utilized to determine the relevant features for each class at the validation layers.

3 RELEVANT FEATURES BASED AUXILIARY CELL (RAC)

In this section, we present our approach to design the proposed Deep Neural Networks (DNNs) with Relevant features based Auxiliary Cells (RACs). The DNN models that are trained for classification behave as feature extractors by the removal of final output layer. We exploit the efficacy of the hidden layer features to develop an architecture in which natural errors can be detected and easy instances (Panda et al., 2016) can be classified earlier without activating the latter layers of the DNN. Fig. 1 shows the conceptual view of DNNs with RACs. Fig. 1(a) consists of the baseline deep neural network 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 number of output nodes i.e. number of classes. Each binary linear classifier with in an RAC corresponds to a unique class in the dataset and is trained on relevant features corresponding to that class. The output of binary linear classifier of a class (say c) in an RAC indicates $P(y_i = c|x_i)$ i.e the probability of a given instance x_i coming from class c. Every RAC outputs a class label and its probability which corresponds to the binary linear classifier with maximum output value. The probability outputted 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 of Panda et al. (2016) . The activation module utilizes the consensus of RACs and their confidence scores to decide if the next layers have to be activated for an input instance.

3.1 TRAINING RACS

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

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 (BIC-j) with no hidden layers
  Train BIC-j using $\{(x_i^{lj}, \tilde{y}_i)\}_{i=1}^{N}$ as training data
  return BIC-j
end for

The binary linear classifiers in an RAC can be trained in parallel as they are independent of each other. We use relevant features to train RAC instead of using all the feature maps for the following reasons:

(a) We found that at a given validation layer, the detection capability (detecting natural errors) is higher if we use relevant feature maps than using all available features maps.

(b) Reduced number of additional parameters added to the baseline DNN.

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 by Panda et al. (2016) and modify it to perform efficient classification with RACs. Given a test instance I_{test}, the methodology either produces a class label C_{test} or makes no decision (*ND*). If an input is classified at the RACs without activating the entire network i.e. without activating the layers beyond validation layers then it is considered as an early classification. The output from the RACs is monitored to decide if early classification can be made for an input. If the decision made by RACs do not agree with each-other then the network outputs *no decision* indicating the possibility of miss-classification by the final classifier. In this case, testing is terminated at layer $l+1$ without activating the entire network which results in an early classification. If both the RACs predict same class label c then we use pre-defined confidence threshold (δ_{th}) to decide on early classification. Along with predicting same class label c, if the confidence scores provided by both the RACs are greater than δ_{th} then we output c as final decision (early classification). The remaining layers from $l+2$ will be activated only if RACs output same class label but with confidence scores lower than confidence threshold δ_{th}. In this case, final classifier's decision is used to validate the decision made by RACs because of their lower confidence.

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

In summary, our approach tries to identify natural errors to improve robustness and modulates the number of layers used for classification for energy efficiency. This results in an energy efficient detection mechanism. The user defined threshold, δ_{th}, can be adjusted to achieve best trade-off between efficiency improvements and better detection of natural errors. Thus, the proposed methodology is systematic and hence 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 testing on adversarial samples, we used three attack methods such as FGSM (Ian et al., 2015), DeepFool (Seyed Mohsen et al., 2016) and CW (Nicholas & David, 2017) to generate adversarial examples.

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 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 baseline DNN i.e. by the final classifier. The inputs which are misclassified even by the DNN with RACs are considered as bad decisions. Therefore, 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.

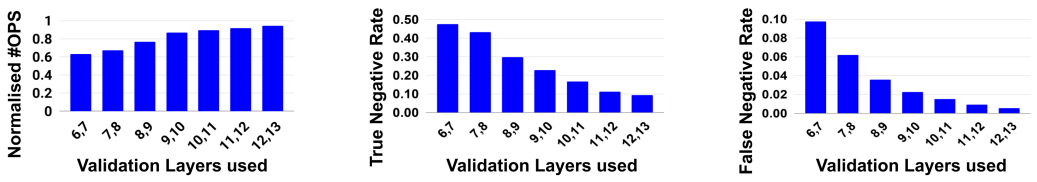
Our goal is to increase the true negative rate and improve energy efficiency (decrease # OPS) while maintaining the false negative rate as low as possible. We observed that these three metrics i.e. true negative rate, false negative rate and # OPS are sensitive to the hyper-parameters related to RACs 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):

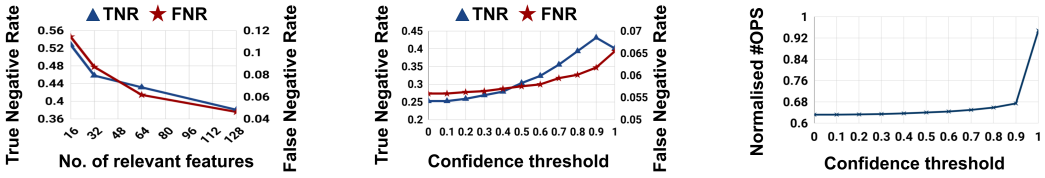
- The choice of validation layers (l, $l+1$)
- Number of relevant features (k) used at each validation layer
- Confidence threshold δ_{th}

We use heuristic based method to tune the above mentioned hyper-parameters. Firstly, lets understand how the validation layers are chosen and their effect on detection capability. The validation layers can not be the 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 the network (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) 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.



(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 the 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 value 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 the false negative and the 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 change in 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 power of 2, compute the corresponding FNR and TNR and select the optimal k value from these observations. Note that # OPS increase as 'k' increases. For example, consider the case of VGG16 network 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%. So, 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 this probability outputted by RACs to the confidence threshold δ_{th} to selectively classify 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 the 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 FNR of the detection mechanism. However, the change in FNR with confidence threshold is negligible (see Fig. 3b) which is around 0.1% for 0.1 change in δ_{th}. Fig. 3c shows the variation in the normalized OPS (with respect to baseline DNN which quantifies efficiency) with different δ_{th} for VGG16 network trained on CIFAR10 dataset with RACs appended at layer 7 and 8.

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

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 increase in confusion. This value of δ_{th} (0.9 in Fig. 3b) corresponds to maximum TNR that can be achieved for a DNN with RACs (validation layers and k are fixed). Beyond this point (say δ^*_{th}), TNR would decrease. 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 #OPS increase from 0.66 to 0.67. Further increase in δ_{th} degrades the TNR and increases #OPS by significant amount. Thus, δ_{th} serves as a knob to trade TNR for efficiency that can be easily adjusted during runtime to get the most optimum 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 baseline error, number of parameters in the baseline network, validation layers used and the additional number of parameters added due to inclusion of RACs. Table. 2 validates the performance of our suggested technique and fig. 4a indicates the reduction in classification error for various networks and datasets. We observe that DNN with RACs can detect $\sim (43-45)\%$ of the natural errors while maintaining the accuracy at $\sim (86-89)\%$ for CIFAR10 dataset. For CIFAR100 dataset, we observe slightly higher detection rate i.e. $\sim (46-49)\%$ with an accuracy range of $(67-69)\%$. The detection rate is much higher for Tiny-Imagenet dataset trained on ResNet18 i.e. 62%. 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 miss-classification but is because of false detection and these falsely detected examples fall into no-decision bucket. Therefore, even though the percentage of correctly classified examples decrease slightly, we avoid around 50% of miss-classifications compared to the baseline network which is advantageous in critical applications. Figure. 4b shows the normalized improvement in efficiency with respect to the baseline DNN for different datasets and networks. We observe that VGGNet has higher improvement in energy than compared to the network with residual connections.

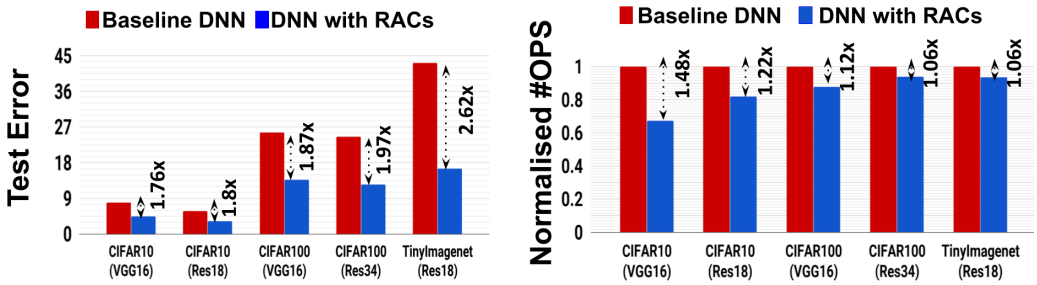
Table 2: Detecting incorrect classifications and making early decisions for image classification task. All the values are percentages.

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

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**	**adversarial TNR(%)**			**OOD TNR (%)**		
		FGSM	DeepFool	CW	Tiny-ImageNet	LSUN	SVHN
CIFAR10	VGG16	33.75	32.42	18.41	44.25	48.10	63.96
	Res18	45.97	53.48	20.31	60.55	68.46	70.13
CIFAR100	VGG16	44.16	35.51	23.70	43.03	38.28	38.02
	Res34	65.36	44.04	24.76	58.55	60.30	58.56

We also evaluate the robustness of our framework against adversarial and out-of-distribution (OOD) inputs for CIFAR10 and CIFAR100 datasets (Table. 3). The adversarial samples are generated using



(a) (b)

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

the following attacks: Fast Gradient Signed Method (FGSM), DeepFool and Carlini & Wagner (CW) attack (refer Appendix). We assume that the attacker has no knowledge about the detection mechanism but has complete knowledge about the baseline DNN (white-box attack with respect to baseline DNN). We observe that our technique make the architectures with residual connections more robust to adversarial and OOD samples as compared to VGG kind of networks. It can be seen that our results are comparable to the baseline detection technique proposed by Hendrycks & Gimpel (2017). The proposed framework not only helps in detecting natural errors but also provides some robustness towards adversarial and out-of-distribution examples while being energy efficient than the baseline network.

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 a set of binary linear classifiers per class at few selected hidden layers referred as Relevant features based Auxiliary Cells (RACs) which enable energy efficient error detection. 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 between 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 found that our proposed framework provides robustness towards adversarial inputs and out-of-distribution inputs to some extent. Beyond the immediate application to increase robustness 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.

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.

Goodfellow Ian, Shlens Jonathon, and Szegedy Christian. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*. 2015.

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.

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

Moosavi Dezfooli Seyed Mohsen, Fawzi Alhussein, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Computer Vision and Pattern Recognition (CVPR)*. 2016.

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 following adversarial attacks to evaluate the robustness of the proposed framework towards adversarial examples: fast gradient sign method (FGSM) (Ian et al., 2015), DeepFool (Seyed Mohsen et al., 2016) and Carlini-Wagner (CW) (Nicholas & David, 2017). The FGSM directly perturbs the input in the direction of the loss gradient. We have generated non-targeted adversarial examples using FGSM. These examples are constructed as

$$x_{fgsm} = x + \epsilon sign(\nabla_x L(y, P(\ \tilde{y}|x)))$$

where ϵ is the magnitude of the noise, y is the ground truth label, $P(\ \tilde{y}|x)$ is the predicted output probability and L is the loss function. We have chosen ϵ to be 0.05.

DeepFool method finds the closest adversarial examples with respect to a given distance measure. CW is an optimization based method and is considered to me most effective method. The non targeted CW attacks are constructed as

$$\arg\min_{x_{cw}} \ \{\lambda.d(x, x_{cw}) - L(y, P(\ \tilde{y}|x))\}$$

where λ is penalty parameter and d(.,.) is the distance measure and L is the loss function. λ is set as 1 for our experiments. We have used L_2 distance measure for both CW and DeepFool attack methods.