ICNN: The Iterative Convolutional Neural Network

KATAYOUN NESHATPOUR, HOUMAN HOMAYOUN, and AVESTA SASAN,
Electrical and Computer Engineering Department, George Mason University

Modern and recent architectures of vision-based Convolutional Neural Networks (CNN) have improved detection and prediction accuracy significantly. However, these algorithms are extremely computationally intensive. To break the power and performance wall of CNN computation, we reformulate the CNN computation into an iterative process, where each iteration processes a sub-sample of input features with smaller network and ingests additional features to improve the prediction accuracy. Each smaller network could either classify based on its input set or feed computed and extracted features to the next network to enhance the accuracy. The proposed approach allows early-termination upon reaching acceptable confidence. Moreover, each iteration provides a contextual awareness that allows an intelligent resource allocation and optimization for the proceeding iterations. In this article, we propose various policies to reduce the computational complexity of CNN through the proposed iterative approach. We illustrate how the proposed policies construct a dynamic architecture suitable for a wide range of applications with varied accuracy requirements, resources, and time-budget, without further need for network re-training. Furthermore, we carry out a visualization of the detected features in each iteration through deconvolution network to gain more insight into the successive traversal of the ICNN.

CCS Concepts: • **Computing methodologies** → *Object detection*; *Object recognition*; *Reconstruction*;

Additional Key Words and Phrases: Energy-efficiency, Convolutional Neural Networks, wavelets

ACM Reference format:
Katayoun Neshatpour, Houman Homayoun, and Avesta Sasan. 2019. ICNN: The Iterative Convolutional Neural Network. *ACM Trans. Embed. Comput. Syst.* 18, 6, Article 119 (December 2019), 27 pages.
https://doi.org/10.1145/3355553

1 INTRODUCTION

The rapid advancement of computing technologies, innovation in parallel processing, development of learning-model oriented languages, and the availability of large data sets has given new momentum to research and innovation in the field of machine learning in the past decade. Machine learning solutions are becoming prevalent and have found their ways into many domains and applications from social media services [1], health and wellness solutions [2–4], security and privacy [5–7], predictive and scheduling solutions [8–10], to computer vision tasks such as voice recognition [11], text recognition [12], object detection and image classification [13–15], and object localization[16].

This research has been supported by National Science Foundation (NSF) Award No. 1718538.
Authors' address: K. Neshatpour, H. Homayoun, and A. Sasan, Electrical and Computer Engineering Department, George Mason University; emails: {kneshatp, hhomayou, asasan}@gmu.edu.

1539-9087/2019/12-ART119 $15.00
https://doi.org/10.1145/3355553

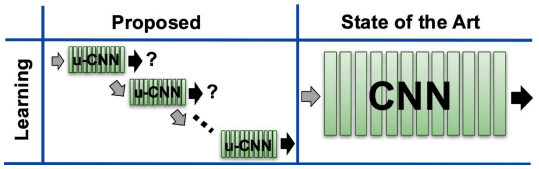


Fig. 1. Reformulating the CNN into an iterative solution.

The state of the art machine learning solutions for object detection has gained a significant boost in their prediction accuracy. This is credited to the recent developments in the design of deep and modern Convolutional Neural Networks (CNN), and processing power provided by Graphical Processing Units (GPU) for training them. However, many Neural Network algorithms and CNN as a part of this family, due to their deep networks and dense connectivity, are computationally intensive. For example, AlexNet [14], a CNN architecture that won the 2012 ImageNet visual recognition challenge, contains 650K neurons and 60M parameters, which demand computational performance in the order of 0.8G–1.0G Floating Point Operations (FLOP)s per classification. Next generations of vision-based CNN algorithms have further improved the prediction accuracy; however, this is achieved via even deeper networks. VGG [15], GoogleNet [17], and ResNet [18] have improved the prediction accuracy by increasing the FLOPs to 20G, 1.5G, and 11G, respectively, while keeping the CNN a computationally intensive and power-hungry solution.

Albeit higher performance requirements, there is a need to aggressively reduce the power consumption of these solutions as many desired platforms for vision-based applications are energy-constrained [19, 20]. Adopting complex vision algorithms in many of mobile and hand-held, embedded systems and IoT applications will not be feasible if energy consumption barrier is not addressed. At the same time, many of the desired applications require real-time and short-latency responses. Therefore, the optimization space involves Accuracy, Latency, Power and Area (ALPA).

With this in mind, in Reference [21], we propose a radically different approach from modern and deep CNN models; we reformulate the learning from a single feed-forward network to a series of smaller networks that are executed iteratively.

Figure 1 illustrates a high-level abstraction of the proposed iterative CNN (ICNN). With iterative learning, each iteration processes a small set of sub-sampled input features and enhances the accuracy of the classification. The proposed ICNN model removes the need for a large neural network and constructs a learning model based on iterative execution of substantially smaller networks. In each iteration, by combining the processing results of the previous iteration with new features extracted from the sub-sampled input image, ICNNs classification accuracy is refined.

2 BACKGROUND

CNNs are constructed from multiple computational layers formed as Directed Acyclic Graph (DAG) [22, 23]. Each layer extracts an abstraction of data from the previous layer, called a feature map (fmap). Most common layers are Pooling (*POOL*), Convolution (*CONV*), and Fully Connected (*FC*). In CONV layers, as illustrated in Figure 2, two-dimensional filters slide over the input images/feature-maps (*Ifmaps*) performing convolution operation to extract feature characteristics from local regions and generating output images/feature-maps (*Ofmaps*). Computation of CONV layer in popular CNNs accounts for more than 90% of the overall operations and requires a large amount of data movement and memory operations [24]. The large size of *Ifmaps*, *Ofmaps* and partial results generated during the CONV processing increases the memory requirements for these architectures. After CONV layers, a non-linear operation is applied to each *Ofmap* pixel to introduce non-linearity in the network. An example of such a non-linear operator is the Rectified Linear

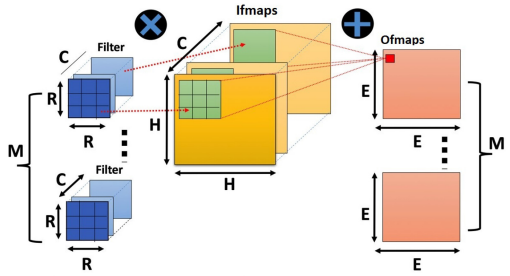


Fig. 2. Computing one CONV layer using input Ifmap/image and filters to produce the output (*Ofmaps*).

Unit (*ReLU*) that replaces all negative pixel values by zero. Other non-linear functions include *Tanh* and *Sigmoid* operators.

POOL layers perform down-sampling along the spatial dimensions of *Ifmaps* by partitioning them into a set of sub-regions and combining the values in each sub-region into a single value. Max-pooling and Average-pooling are examples of POOL operators that use the maximum and the average values for each sub-region, respectively. The outputs from the CONV and POOL layers represent high-level features of the input image. Each CNN includes multiple CONV, non-linear and POOL layers with their outputs fed to FC layers. FC layers combine all the neurons in the previous layer and connect them to every single neuron in the next layer. FC layers fuse the features extracted in the CONV layers to generate a relational representation of these features for each class in the classifier detection set. In the last layer, a Softmax classifier uses the outputs of the last FC layer to produce normalized class probabilities for various classes. Softmax classifier is a multi-class version of the binary logistic regression classifier, which produces un-normalized log probabilities for each class using cross-entropy loss.

3 RELATED WORK

In this section, the detection accuracy of CNN architectures that have competed in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in recent years is discussed. The ILSVRC [25] is an object detection competition for classification of images into 1,000 different classes by training on 1.2 million labeled images. AlexNet architecture, which consists of five convolutional layers and 3 fully connected layers, achieves a top-5 accuracy of 80.2% on ImageNet. VGG [15] advocates the idea that going deeper with CNNs increases accuracy [26]. It is proven that the effective receptive fields of 2 and 3 back-to-back 3×3 CONV layers are equivalent to the receptive field of 5×5 and 7×7 CONV layers, respectively. With this idea, VGG-16 achieves a top-5 accuracy of 92.5% by stacking 13 3×3 CONV layers, and 3 FC layers.

Considering the huge variation in the location and size of the networks, the selection of the size of kernels is tricky in large networks. For objects distributed more globally a larger kernel size is more desirable, whereas smaller kernels are better suited for objects that are locally distributed. GoogLeNet's solution to this problem is to make the network wider using the inception layer. In inception layer, pooling layers and multiple convolutional layers with different kernel sizes process the same input. All the outputs are then concatenated allowing the model to take advantage of multi-level feature extraction from each input. For instance, it extracts global (3×3 and 5×5) and local (1×1) features at the same time. With 22 weight layers, GoogLeNet (aka. Inception v1) achieves a top-5 accuracy of 93.3% and wins ILSVRC 2014. Inception v2 and Inception v3 were proposed in Reference [27], where 7×7 and 5×5 kernels were factorized to smaller kernels (3×3

kernels) and $n \times n$ kernels were replaced by $1 \times n$ and $n \times 1$ kernels to reduce the computational load.

Finally, Microsoft ResNet [18] uses residual blocks in which, each input goes through a series of CONV-ReLu-CONV layers before being added to itself. The formulation of residual blocks is realized through shortcut connections [28, 29], which allows features to skip one or more CONV layer and be combined with other features at a later stage in the network. The authors show that these residual networks are easier to optimize and they benefit from increased depth. With 152 layers, ResNet wins the ILSVRC 2015 with a top-5 accuracy of 96.4%.

The number of layers and the complexity of the CNNs in terms of FLOP count has dramatically increased over time to enhance their classification accuracy (see Table 2). In the literature, various optimization approaches have been proposed to reduce the computational complexity and to enhance the energy-efficiency of the CNN architectures [30–32], including Tensor decomposition and low-rank optimization [33, 34], parameter compression and exploitation of sparsity [35], binarized neural networks [36–41], and precision reduction of weights and/or neurons by fixed-point tuning and quantization [42–46]. In addition to changes in the model, researchers have also investigated the means of enhancing the energy-efficiency of these applications by designing new ASIC or FPGA solutions [47–53]. All these approaches are orthogonal to the proposed ICNN, which modifies the structure of the network and could be applied to ICNN as well.

Other works have focused on deploying a dynamically configured structure that allows the complexity of CNN to be adjusted in the run-time. In Reference [54] a dynamic approach is proposed that breaks the CNN into partial networks and the number of active channels per layer is adjusted based on classification confidence of partial networks. This allows CNN to be partially or fully deployed. While Reference [54] reduces the computational of classification, the memory footprint required to keep all the intermediate features in case of non-satisfactory confidence results from partial networks is quite large. In ICNN, this is addressed by keeping the feature-maps of only the last CONV layer.

In Reference [55], a Conditional Deep-learning Network (CDLN) is proposed in which, FC layers are added to the intermediate layers to produce early classification results. CDLN starts with the first layer and monitors the confidence to decide whether a sample can be classified early, skipping the computation in the proceeding layers. While CDLN only uses FC layers at each exit point, BranchyNet [56] uses additional CONV layers at each exit point (branch) to enhance the performance.

While ICNN is most closely comparable to References [55] and [56], it significantly improves the computational complexity of the solution and provides a far more flexible and much richer set of features for trading off the accuracy vs computational complexity vs energy consumption vs classification time. While both References [56] and [54] allow dynamic adjustment of the CNN for various images through incremental networks, ICNN uses its contextual awareness from early iterations to provide hints to the next iterations for computational pruning. Moreover, unlike References [54, 56] ICNN, processes sub-sample of the images with reduced dimensions, which significantly reduces the computation load and required memory footprint. Regarding memory footprint, ICNN only saves the features extracted in the last CONV layer of the previous iterations, resulting in a much smaller memory footprint.

In addition, while the architectures in References [56], and [54] are proposed for up-to four iterations, ICNN shows promising results for up to seven iterations allowing a more flexible termination policy. The architectures in References [54] and [56] are trained on much for 10-class datasets (i.e., MNIST [57], CIFAR-10) and their scalability when the number of classes increases is questionable, while ICNN is trained on the entire ILSVRC dataset for 1,000 classes. ICNN, along with References [55] and [56] behaves perfectly and shows negligible degradation in the

classification accuracy for a small number of classes. However, achieving acceptable accuracy when the number of classes increases is quite challenging.

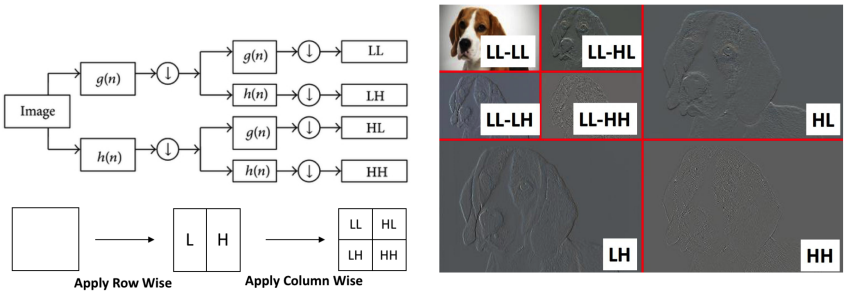
A major contribution of the ICNN is the training of multiple networks. GoogLeNet is another massively adopted CNN architecture that trains *auxiliary* classifiers. GoogLeNet adds auxiliary classifiers to the intermediate layers only in the training phase. During training, the weighted loss of the auxiliary networks is added to the total loss of the network. This allows the classifier to combat the vanishing gradient problem associated with the training of neural networks [58] while providing regularization. In the inference phase, the auxiliary networks are removed. Moreover, the effect of the auxiliary network is reported to be minor (0.5%). The basic contribution of the Inception architecture is the increase in the accuracy due to multi-level feature extraction. However, during the inference phase with inception architecture, each classification requires the whole images to go through all the layers of the network. ICNN, however, trains multiple smaller networks during the training phase and deploys these networks in the inference phase. During the inference phase the network processes only a DWT sub-sample of the image with each uCNN. As a result, the network could be tuned at the run-time to meet the timing and confidence requirements of the applications. A large number of images produce high confidence classification results with the first few uCNNs, which precludes the need to process the whole image. In fact, the proposed approach is also applicable to GoogLeNet, VGGNet, and other networks.

4 ITERATIVE LEARNING

State of the art DAG-based CNN networks are composed of a single feed-forward computational network, where the prediction is given and its confidence is determined after performing all necessary computations. Our proposed reformulation is driven by the needs of resource-constrained vision applications for reducing energy consumption and the classification latency when deploying CNN solutions. In the proposed solution, a large CNN block is decomposed into many smaller networks (uCNN in Figure 1), allowing iterative refinement and greater control over the execution of the algorithm. Thus, not all images pass through all the uCNNs; by monitoring the successive execution of uCNN networks, a thresholding mechanism decides when to terminate the forward uCNN traversal based on the current classification confidence of the images [59]. In addition, the ability to change the required confidence threshold from classification to classification, allows the system to trade off the accuracy of prediction versus energy consumed per iteration.

Using ICNN requires a methodology for sub-sampling the input image for the use of each iteration. Preferably, each sub-sample is obtained using a different sampling method to push each smaller network (uCNN) to learn new features that are not extracted in the previous iterations. For this reason, we propose the application of *Discrete Wavelet* sampling to decompose an input image into various input sets (sub-bands). The classification is then initiated by digesting the first sub-sampled as input in the first iteration. Upon completion of the first computational round (first uCNN), the classification confidence is tested. If the confidence is unsatisfactory, then it could be progressively increased by working on additional input samples (chosen from remaining sub-bands). Discrete Wavelet Transformation (DWT) provides the proposed learning algorithm with an attractive start point, because, in addition to frequency information, it also preserves temporal information of an image [60]. However, note that other sampling mechanisms could also be used for ICNN.

The DWT of an image is calculated by passing it through a series of high and low pass filters. Consider $h[n]$ as the impulse response of the high pass filter, and $g[n]$ as the impulse response of low pass filter, and each row of the image as a signal $x[n]$. To obtain the first order DWT transform of the image, for each row of the image, the low and high filters are first convolved with the image values in each row. Note that since half the frequencies of the signal (row of pixels) are



(a) (b)

Fig. 3. (a) DWT filter structure, (b) sub-bands generated from a two-level two-dimensional Haar Discrete Wavelet Transformation (DWT) of an input image.

removed by each filter, the low pass and high pass filter sub-sample the input signal by half the frequency:

$$y_{low}[n] = (x * g)[n] = \sum_{n=0}^{Width} x[k]g[2n-k], \tag{1}$$

$$y_{high}[n] = (x * h)[n] = \sum_{n=0}^{Height} x[k]h[2n-k]. \tag{2}$$

As illustrated in Figure 3(a), application of the DWT to each row results in two sub-sampled images. Each image has the same height as the original image, but half the width. By placing these two images next to one another, the resulting image is equal to the size of the original image. The same sampling mechanism could now be applied in a vertical direction, considering each column of pixel values as discrete signal x. This results in four sub-bands. Each sub-band is a sub-sample of the original input obtained using a different filtering mechanism. If more sub-samples are required, then we could apply the DWT to any of the sub-bands hierarchically. In this article, we have used seven iterations. Hence, we have applied a second transformation to the LL sub-band to generate four level-2 sub-bands. Figure 3(b) shows the two resulting sub-bands by applying two-level DWT to a dog image.

A high-level representation of envisioned ICNN fed by DWT is illustrated in Figure 4. Each iteration is a uCNN, which processes a new DWT sub-band and refines the confidence of learning network. DWT, being a convolutional filter, could be readily computed using processing elements (PE) in CNN processing engine of interest, or could be provided directly to each uCNN.

Figure 5 shows the decomposition of the ith uCNN, while Table 1 captures the configuration of each uCNN. The Concat layer in Figure 5 fuses the *Ofmaps* of the last CONV layer in the current iteration with the *Ofmaps* of the last CONV layers from previous iterations. Note that the number of *Ofmaps*, which are processed at any given CONV layer in each uCNN is considerably smaller than that of original AlexNet (see Table 1). Hence, the computational complexity of each uCNN is considerably smaller than that of AlexNet.

5 IMPLEMENTATION FRAMEWORK

An iterative version of AlexNet was designed to build a 1,000-class image classifier for the ImageNet dataset. The Iterative AlexNet was implemented in Caffe [61], a deep-learning framework developed by Berkeley AI Research (BAIR). Following the methodology in Reference [17] to solve the over-fitting problem, the training set was augmented by reshaping images to $3 \times 256 \times 256$ and

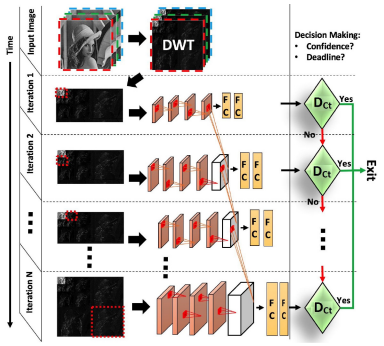


Fig. 4. Iterative CNN (ICNN) general architecture where each uCNN is fed by features extracted from its previous uCNN and a DW sub-band generated from DWT transformation of the input image.

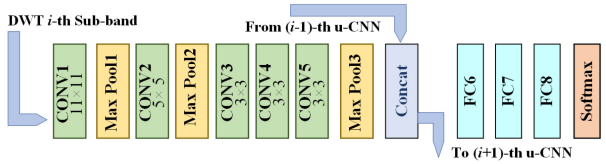


Fig. 5. The architecture of the *i*th uCNN for the iterative AlexNet.

Table 1. The Configuration (Number of Output Channels) of the Iterative AlexNet

iter	i1	i2	i3	i4	i5	i6	i7
CONV1	24	24	24	24	24	24	24
CONV2	64	64	64	64	64	64	64
CONV3	96	96	96	96	96	96	96
CONV4	48	48	48	48	48	48	48
CONV5	32	32	32	32	32	32	32
FC6	1,024	1,024	1,024	1,024	2,048	2,048	4,096
FC7	1,024	1,024	1,024	1,024	2,048	2,048	4,096
FC8	1,000	1,000	1,000	1,000	1,000	1,000	1,000

extracting $3 \times 228 \times 228$ crops from each image (resulting in 784 crops). Moreover, the horizontal mirror of each crop was added to the dataset.

To prepare the sub-band for the ICNN, a two-level two-dimensional DWT was captured through the Haar filter [62] for each channel (i.e., R, G, and B) of the images. The resulting sub-bands are four RGB images of size $3 \times 57 \times 57$ in smaller sub-bands (corresponding to LL-LL, LL-LH, LL-HL, LL-HH), and three RGB images of $3 \times 114 \times 114$ in larger sub-bands (corresponding to HL, LH and HH). The seven resulting sub-bands allows sampling of the input image by 7×, and as a result building the ICNN with seven iterations. To increase the number of iterations, the DWT depth can be increased or alternatively, DWT can be applied to the other sub-bands (LH, HL, HH).

For training the ICNN, we initialize the weights from a Gaussian distribution with zero mean and a standard deviation of 0.01. We start each training step with a learning rate of 0.01. The learning rate was reduced by 2× every 20-epoch until the learning rate was as low as 10^{-6}. By one-epoch,

we refer to one pass of all the 1.2 million images in ImageNet; however, for data augmentation purposes, every few epochs, the order of the images was modified, the image crops were altered and horizontal mirrors of the images were utilized. To train the network, Tesla K80 GPUs were deployed.

6 ICNN TRAINING SCHEMES

When training the ICNN, two different training policies could be adopted. In the first method, the iterations are trained in sequence and in the second method, all of the iterations are trained in parallel. Each training process, the implications of each training approach, and the trade-offs for adopting one training approach versus others is discussed next:

6.0.1 Sequential Training. The first proposed training method is a multi-step training process carried out sequentially. In this approach, the first uCNN network is separated and its weights are trained as described in Section 5. Starting from the second iteration and moving up, each uCNN is then trained without changing the filter weights in the previous iterations. When training the *i*th uCNN (uCNN[i]), this training constraint is enforced by keeping the learning rate of convolutional layers in uCNN[1]-uCNN[*i*-1] equal to zero. Hence, the weights in the previous iterations are only used during forward propagation of error during the training process, without affecting the back propagation's update process. To allow faster training, starting from the second iteration, instead of initiating the weights from Gaussian distribution, the weights in the CONV layer of each iteration were initiated to the values in the corresponding CONV layer of the previous iteration.

6.0.2 Parallel Training. The second proposed training method is a two-step training process. In the first step, the FC layers of the intermediate iterations are removed, and only the FC layers of the last iteration that combine the features from the last CONV layer of all iterations are kept. The resulting network, which is the last iteration of ICNN is trained first using the training process described earlier, yielding the weights for the FC layer of the last iterations, and CONV layers of *all* the iterations. In the second step, the FC layers in all other iterations are added to construct parallel networks for the rest of the iterations. Subsequently, the weights in all the CONV layers are kept constant by setting their learning rate to zero, and the weights in the FC layers of the intermediate iterations, are trained in parallel. Note that back-propagation at this step does not update the filter values in the CONV layers, and only the weights in FC layers are updated.

In terms of training time, parallel training is considerably faster. Using a single Tesla K-80 GPU, the parallel training was concluded in 7 days, while the sequential training took about 3 weeks to conclude. In terms of prediction accuracy, parallel and sequential training introduce an interesting trade-off. Figure 6 shows the top-5 and top-1 accuracy of both sequential and parallel training. Figures 6(a) and 6(c) capture the results for only one 1 crop. It should be noted, that to increase the prediction accuracy during the inference phase, rather than feeding only one image to the network, various work [14–18] extract 5 crops from images (4 corner and 1 center crop), along with their horizontal mirrors (10 crops in total), calculate the average of the output of SoftMax layer from all 10 crops and conclude the classification. To make a fair comparison with related work, in Figures 6(b) and 6(d) the results for the average of 10 crops are depicted.

As illustrated in Figure 6 sequential training provides higher accuracy in early iterations and has a front-loaded accuracy gain, which is advantageous for low-power and real-time systems that could benefit from improved accuracy with low delay. However, the parallel training suffers from lower classification accuracy in earlier iteration(s), while gaining a higher accuracy in the last iteration(s). This is due to the fact that the parallel scheme starts with training the last iteration. Another reason could be that in the parallel training approach all the weights are initialized from Gaussian distribution while in the sequential scheme, starting from the second

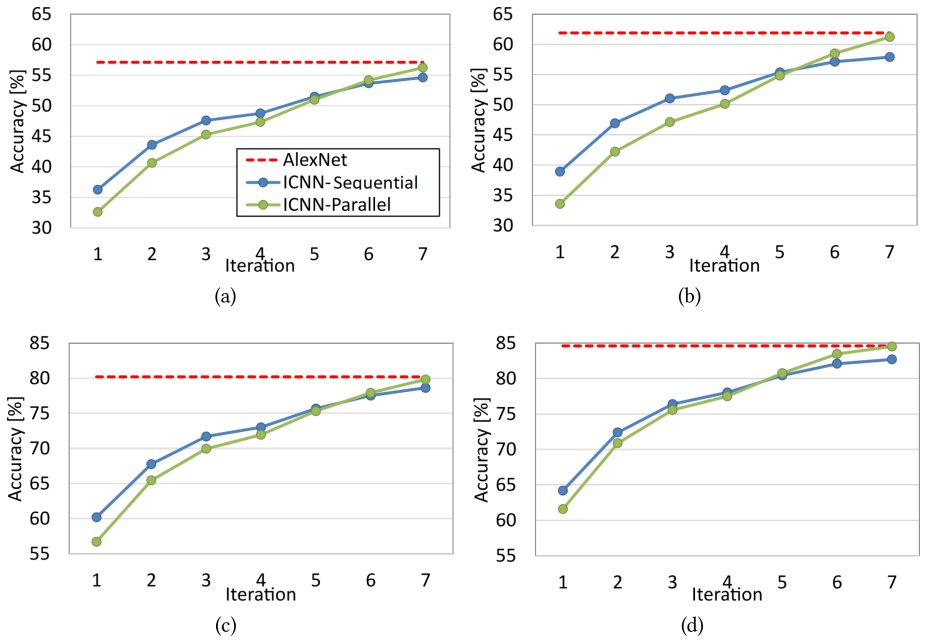


Fig. 6. AlexNet vs. Iterative AlexNet using (a) top-1 accuracy with 1 crop, (b) top-1 accuracy with an average of 10 crops, (c) top-5 accuracy with 1 crop, and (d) top-5 accuracy with an average of 10 crops.

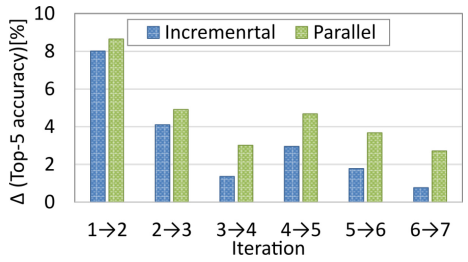


Fig. 7. Increase in the top-5 accuracy of ICNN in Sequential and Parallel training approaches.

iteration, the weights are initialized from the values in the previous iteration. The first sub-bands of the DWT are mainly dominated by low-frequency, while the other sub-bands are mainly dominated by higher-frequency. Due to this difference, the initialization strategy degrades the training procedure. As a result, the parallel training approach is used in the implementation results.

Based on Figure 6, the difference between the accuracy of the two approaches is more significant in both the early iterations and the latter ones. To get a better insight into the difference between the two training scheme, Figure 7 shows the growth of the top-5 accuracy in percentage. Since the parallel approach incurs lower and higher accuracy in the first and last iterations, respectively, parallel training covers a larger range of accuracy values, and each iteration contributes more significantly to the accuracy. Depending on the accuracy, latency, and power requirements of the hosting hardware (or application) when working on a batch or stream of images, one training process may become more appealing than the other.

7 DETECTION OF TINY IMAGES

The sub-sampling approach deployed in this article, the Wavelet Transform through Haar Filter, is a lossless transform. Note that DWT by itself is lossless, as it merely transforms pixels to a domain in which they can be more efficiently encoded; however, to generate compressed images, coefficients are modified, and compression becomes lossy [63, 64]. In this article, compression is carried out and any image detail lost in earlier sub-bands is eventually recovered in other sub-bands.

The images in ImageNet are of varied dimensions and they are all re-sized to match the input requirement of the networks. However, this predefined input size is not suitable when object scales vary. This problem seems exacerbated in ICNN, where images are scaled to $\frac{1}{4}$ and $\frac{1}{2}$ of the original size for the first four, and last three iterations, respectively. But it should be noted that while the size of the convolutions in the first layers are the same as the original network, with ICNN the size of the strides vary based on the image sizes to maintain the precision. In the original AlexNet, the stride in the first convolution is 4, while in ICNN the stride is 2 and 1 in the first four and last three iterations, respectively. As a result, the extracted feature maps in the CONV layers have the same dimensions, albeit a different number of channels with respect to the original Network.

In SPPNet [65], Spatial pyramid pooling has been proposed as a solution to remove the fixed-size constraint of the CNN networks. In this approach instead of resizing the input images, the original (larger) images are processed by the CNN and variable-size Poolings are applied to the features extracted in the last fully CONV layers to generate fixed-size outputs for the FC layers. As a result, smaller details in the network are better processed. This approach is well-suited for modifying any network for detection of tiny objects, where instead of scaling images to sizes that fit the network, we process larger images and then apply the spatial pyramid pooling to the extracted features. However, this would increase the processing time of the network.

8 VISUALIZATION OF CONVOLUTIONAL NEURAL NETWORKS

The major contribution of the ICNN is the training of multiple small networks and allowing the classification to improve sequentially in each iteration. The rate of improvement in the prediction accuracy in each iteration of ICNN depends on the network being able to learn distinct features in each iteration, and leverage all features learned in the previous iterations. To illustrate how ICNN learns distinct features in each iteration, visualization techniques are used to present the features learned in different iterations.

Deconvolutional networks have been widely used to project the feature activations back to the input pixel [66]. Deconvolutional networks construct layers from the image upwards in an unsupervised fashion [67]. In Reference [68] they were used to reveal the input stimuli that excite individual feature maps at various layers in the model. Since then, various works have used the Deconvolution networks for semantic segmentation [69], flow estimation [70], and generative modeling [71].

We use the Deconvolution networks to demonstrate how hidden convolution layers of each iteration extract features distinct from other iterations for CONV2 layers. Figure 8 depicts the top 9 activations for all 64 feature maps of CONV2 layers in all iterations. Each figure shows the reconstructed patterns from the training set that triggers the highest filter response in a given feature map on the left and the corresponding image patch that triggered the response on the right. The extracted features are drastically different in the same CONV layer in different iterations. The ability of ICNN to extract different features in different iterations and to combine these features after last CONV layer of each iteration is the reason why the accuracy of each iteration increases.

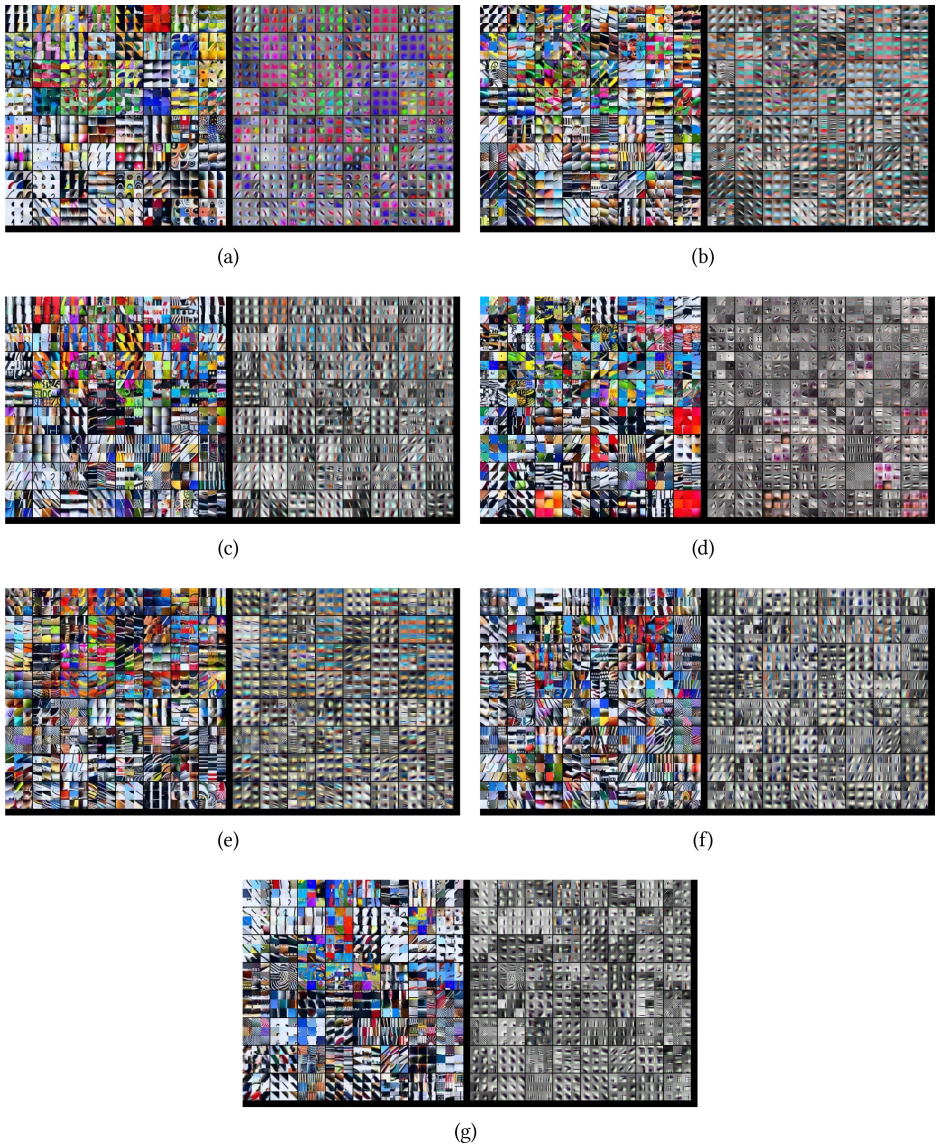


Fig. 8. Visualization of features in a fully trained model for layers CONV2 of iterations (a) 1, (b) 2, (c) 3, (d) 4, (e) 5, (g) 6, (h) 7. The figure shows the top 9 activations for all 64 feature maps of CONV2 layer across the validation data, projected down to pixel space using deconvolutional network approach. The reconstructed patterns from the validation set that cause high activations in a given feature map are depicted on the right, and the corresponding image patches are depicted on the left.

It should be noted that the networks in 8 are trained with the sequential approach. As mentioned in Section 6.0.1 starting from the second iteration, the weights in the CONV layers of each iteration is initialized to the values in the corresponding CONV layer of the previous iteration. As a result, the patterns identified in each iteration for some feature maps are somewhat correlated albeit, different. See the first feature map in Figure 8 (top left 3×3 images). This feature map is triggered for image patches with vertical lines, diagonal lines and blurrier vertical lines

in iteration 1, 2 and 3, respectively. After the fourth iteration, the same feature map index starts identifying completely different patterns, which is indicative of the evolving nature of ICNN.

9 CONTEXTUAL AWARENESS IN ICNN AND ITS APPLICATION

Real-time application of deep-learning algorithms is often hindered by high computational complexity and frequent memory accesses. Network pruning is a promising technique to solve this problem [72]. The network can be pruned by learning only the important connections; i.e., if the weight of a connection is less than a threshold, then the connection is dropped [73, 74]. This approach is previously deployed in the training phase of CNN by discarding negligible weight values. However, ICNN enables us to apply the pruning at run-time. This is because early classification provides ICNN with a hint to avoid computation related to classes that are least probable.

Upon completion of the first uCNN, ICNN develops a form of contextual awareness as its first FC classifier outputs the probability of various classes with respect to an input image. We explore how this feature may be used to reduce the computational complexity of CONV or FC layers in the next iterations based on the class probabilities. To this end, We compare the class probabilities from various iterations to understand how the predictions and class probabilities derived in each iteration relate to the ones in the next iterations.

We introduce *Prediction Rank* (PR) as an indicator of the prediction accuracy of a model for each image. Consider an image with class label $C[i]$. After executing a uCNN, all the class probabilities are sorted in descending order. The location of the class $C[i]$ in the sorted array of class probabilities is called Prediction Rank. Needless to mention, PR can be any value from 1 to 1k, with 1 and 1k indicating highest and lowest image detection accuracy, respectively. If statistical analysis of the dataset for class $C[i]$ shows that PR is always smaller than L in all iterations, where $(1 \leq L \leq 1K)$, then limiting the number of computed class probabilities in the ICNN to L instead of 1k will have no impact on the probability of detection of class $C[i]$. However, if the number of computed classes (i.e., L) is chosen smaller than PR variation for class $C[i]$, then by pruning the computation for class $C[i]$, we will end up with miss-classification. Expanding this to all classes, the probability of miss-classification (MC) conditioned on pruning the classes to those with $PR \leq L$ is given by

$$P(MC|P_{th} = L) = \sum_{i=1}^{1000} P(C[i])P(PR(C[i]) > L). \tag{3}$$

In this equation the P_{th} is the Pruning threshold, $C[i]$ is the ith class, $PR(C[i])$ is the prediction rank for $C[i]$, and L is the chosen limit for pruning. For obtaining the probability $P(PR(C[i]) > L)$, we define the Decision function D as

$$D(\alpha^{C[i]}(j), L) = \begin{cases} 1, & \text{if } PR(\alpha^{C[i]}(j)) > L \\ 0, & \text{else.} \end{cases} \tag{4}$$

In this equation $\alpha^{C[i]}(j)$ is the jth image member of class $C[i]$. If we define $S[i]$ as the number/size of images in the dataset (or expected number of images in the batch) that belong to class $C[i]$, then $P(PR(C[i]) > L)$ is computed as follows:

$$P(PR(C(i)) > L) = \frac{\sum_{j=1}^{S[i]} D(\alpha^{C[i]}(j), L)}{S[i]}. \tag{5}$$

Based on Equation (3) the pruning threshold L is selected to set the Probability of miss classification due to pruning to the desired value. The higher the value of L, the higher the classification accuracy, but at the cost of higher computational complexity. The value L could reduce from iteration to the next iteration, making the pruning more aggressive as the accuracy of classification increases.

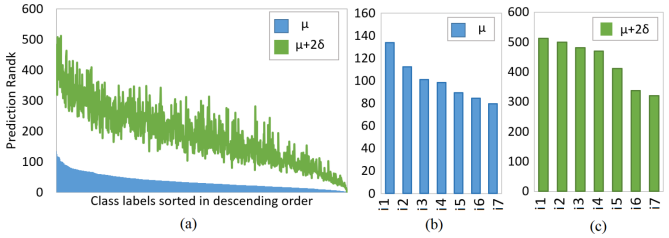


Fig. 9. Prediction Rank: (a) PR_μ & $PR_\mu + 2 \times PR_\sigma$ for various classes at iteration 1. (b) Maximum PR_μ among all classes in each iteration. (c) Maximum $PR_\mu + 2 \times PR_\sigma$ among all classes in each iteration.

To get a sense on the impact of choosing a value for L, for each class, we find the mean (μ) and variation (σ) of PR based on all the data in the validation set. Figure 9(a) shows the mean (PR_μ) and variation (PR_σ) of PR for all 1k classes sorted in descending order of PR_μ at the first iteration. Assuming a normal distribution, 95% of images of each class will have a PR below $PR_\mu + 2 \times PR_\sigma$. In Figure 9, $PR_\mu + 2 \times PR_\sigma$ of none of the classes exceeds 500, suggesting that by removing 50% of classes ($P_{th} = 500$) the prediction accuracy does not drop beyond 5% for any of the classes. Figures 9(b) and 9(c) show the maximum of PR_μ and $PR_\mu + 2 \times PR_\sigma$ among all classes for all the iterations. The maximum of $PR_\mu + 2 \times PR_\sigma$ is reduced as ICNN moves to the next iteration, indicating that in subsequent iterations, the pruning policy could be more aggressive. Note that based on the eliminated classes, the computational complexity of the next iterations is reduced by pruning the neurons in the FC layers and/or the filters in the CONV layers, which are highly correlated to classes with close to zero probabilities.

9.1 Pruning Neurons in FC Layers

For each image, all class probabilities are sorted in descending order. Assuming the rank of each class is the location of the class in the sorted array, by setting Pruning Thresholds (P_{th}) to L in a given iteration, all the classes with ranks greater than L are eliminated in the subsequent iteration. Hence, the computation of FC layer for these classes could be skipped. This results in a significant reduction in FC computation, and removes memory transfer of parameters related to the classification of eliminated classes.

9.2 Pruning Filters in CONV Layers

In addition to eliminating neurons in the FC layers, CONV filters in the final layer(s) can also be eliminated. To this end, for each class, we carry out a statistical analysis on the 1.2 million images in the training set. In this study, we identify the classes that are least affected by the elimination of each CONV filters in each iteration. When we set the Pruning threshold (P_{th}) to value L in a given iteration, all classes with ranks greater than L are eliminated in the subsequent iteration. Hence, the required CONV filters are only those needed by remaining classes, and the computation of other filters could be skipped.

It should be noted that while filter pruning has been a well-known approach for reducing the computational complexity of CNN [72–74], ICNN allows the pruning to be carried out based on the feedback from previous iterations. An ignorant pruning approach removes a number of filters that least affect the overall accuracy; however, context-aware pruning allows us to remove the filters that least affect the top-P_{th} classes of the previous iteration, promising an increased accuracy.

10 PROPOSED POLICIES FOR COMPLEXITY/ACCURACY TRADE-OFF

The iterative transformation of the learning algorithm can be exploited to yield the highest accuracy while meeting hard deadlines or create an efficient trade-off between computational complexity and accuracy. To this end, we propose the following policies to explore the benefits of ICNN and its trade-offs in terms of computational complexity (thus energy consumption) and accuracy.

10.1 Dynamic Deadline (DD) Policy for Real-Time Applications

Many real-time applications require fast and deadline-driven computations to operate safely and correctly. To speed up the CNN-based classifier, one could adapt more powerful resources and benefit from higher parallelism. However, such solutions are extremely expensive in terms of hardware cost and energy requirements. The proposed ICNN, however, provides a cost and energy effective solution for real-time applications. Each uCNN in ICNN produces a classification allowing early exit upon reaching a scheduled deadline. For the deadline-driven applications, the ICNN classifier processes the input image up until the uCNN iteration in which, the scheduled deadline is reached, and subsequently a classification is made. Consequently, the accuracy of the detection and computational complexity is dependent on the time-budget and the number of iterations that can be completed within the scheduled deadline. Note that in this mode of operation, only FC layer(s) in the last processed uCNN iteration is required to make a classification decision, and the FC layers in the previous iterations for the availability of computation-time-budget are skipped. This is due to the fact that for the deadline-driven applications, we rely solely on the classification results from the last processed iteration.

Algorithm 1 captures the implementation of DD policy. In this algorithm, N marks the number of ICNN iterations. $uCNN_{CONV}(i, img)$ invokes the convolution layers in the ith uCNN with img as input, returning $Ofmaps$ from the last CONV layer (Of_{μ}) and the elapsed time (t_{CONV}). The $Ofmaps$ are concatenated with the $Ofmaps$ from the previous iterations. Subsequently, t_{CONV} is compared to the deadline, t_D, and if lower with a t_{mrg} margin, the next uCNN is invoked. The $t_{mrg}[i+1]$ accounts for the projected time of the next iteration of uCNN. Upon reaching the deadline, $uCNN_{FC}(i, Of)$ invokes the FC layers of the ith uCNN producing a vector of probabilities P_r, one for each of 1k labels.

ALGORITHM 1: Dynamic Deadline (DD)

```
Of ← [];
t_mrg[i] ← iteration_time_budget;
for i = 1 to N − 1 do
    t_CONV, Of_μ ← uCNN_CONV(i, img);
    Of ← Of_μ + Of;
    if (t_CONV + t_mrg[i + 1]) < t_D then
        exit;
P_r ← uCNN_FC(i, Of);
```

10.2 Thresholding Policy (TP) for Dynamic Complexity Reduction

Using this policy, the ICNN terminates as soon as a uCNN produces the desired classification confidence. The classification confidence is calculated by summing the probabilities of top-C (e.g., $C = 5$) suggested classes. Note that, in this approach, the FC layers of the previous uCNNs are not skipped, as the calculations for classification confidence relies on the output of the FC layers. When processing a batch of images, while the first iteration in ICNN may yield high confidence for a test image, it might yield lower confidence for another test image. The iterative CNN, in this

case, terminates the classification after the 1st iteration for the first image but proceeds to the next iteration for the second image. This decision is made dynamically based on a predefined Detection confidence-threshold (D_{Ct}).

Algorithm 2 implements the TP policy. The $uCNN(i, \bar{C_L}, img)$ function invokes the ith uCNN with img as input, requesting classification result for all class labels in the $\bar{C_L}$ vector and produces a vector of probabilities $\bar{P_r}$, one for each label in $\bar{C_L}$, which contains all 1,000 labels. After each uCNN, the sum of top-5 probabilities ($\sum_{k=1}^{5} \bar{P_r}[k]$) is compared with Detection Confidence Threshold ($\bar{D_{Ct}}[i]$), and if greater, the image is classified. Note that some of the images never reach a classification confidence above D_{Ct}. For these images, the results of the classification in the last iteration are used.

ALGORITHM 2: Thresholding Policy (TP)

```
for i = 1 to N − 1 do
    P̄r ← uCNN(i, C̄L, img);
    Sort_descending(P̄r);
    if ((∑_{k=1}^{5} P̄r[k]) > D̄Ct[i]) then
        exit;
P̄r ← uCNN(N, CL, img);
```

In this approach, classification for various images is terminated in different uCNN iterations. Thus, the total complexity of classification dynamically varies for each image. Moreover, the number of parameters required for images classified in early iterations significantly drops, which directly results in a reduction in the number of memory accesses, required memory footprint and energy consumption. Note that the D_{Ct} values could be different for each iteration of the ICNN allowing ICNN to exercise complicated thresholding policies. In this article, we explore two variants of TP: (1) Fixed Thresholding Policy (TP_F): In the fixed thresholding policy, a fixed value for D_{Ct} is used across all uCNNs. (2) Variable Thresholding Policy (TP_V): In TP_V the confidence threshold value for different iterations is varied.

10.3 Context-aware Pruning Policy (CAPP)

Based on the contextual awareness obtained upon completion of the initial uCNNs, CONV, and FC pruning policies are proposed.

(1) Context-aware Pruning Policy for FC layer ($CAPP_{FC}$): Based on the discussion in Section 9.1, Algorithm 3 proposes the implementation of CAPP policy for FC neurons. In the first uCNN, C_L contains all 1,000 labels. The $uCNN(i, \bar{C_L}, img)$ function invokes the ith uCNN for an input img returning a vector of probabilities $\bar{P_r}$, one for each label in $\bar{C_L}$. Subsequently, The less-probable classes are pruned based on pruning threshold stored in the pruning policy vector $\bar{P_{th}}$. For instance, $\bar{P_{th}}[i] = 100$ results in only choosing the 100 labels and disables all other neurons in the FC layer of the uCNN(i+1) associated with the eliminated labels.

ALGORITHM 3: Context Aware Pruning Policy for FC layer ($CAPP_{FC}$)

```
for i = 1 to N − 1 do
    P̄r ← uCNN(i, C̄L, img);
    Sort_descending(P̄r);
    C̄L ← C̄L[1 : P̄th[i]];
P̄r ← uCNN(N, C̄L, img);
```

Since the compute-intensive parts of the CNN architectures are the CONV layers, the pruning slightly reduces the computational complexity as it only affects the FC layers. However, it considerably reduces the number of weights needed to be moved to the memory. CAPP is yielding a dynamic trade-off between accuracy, the required memory footprint, and computational complexity. Pruning a larger number of classes results in higher complexity and memory footprint reduction, while negatively affecting the accuracy. It should be noted that, unlike the thresholding scheme, the pruning scheme yields the same computational complexity for all the images.

(2) Context-aware Pruning Policy for CONV layer ($CAPP_{CONV}$): Visualization of the ICNN filters by using deconvolution networks as described in Reference [68], makes it possible to identify and remove trained CONV filters that are closely associated with the classification of low-probability classes, extending the pruning feature into CONV layers. The pruning feature based on filter visualization is explored in Algorithm 4.

ALGORITHM 4: Context Aware Pruning Policy for CONV layer ($CAPP_{CONV}$)

```
Pre-Process:Obtain P̄re(i, cnv, c)
for i = 1 to N − 1 do
    P̄_r ← uCNN(i, C̄_L, img);
    Sort_descending(P̄_r);
    for cnv=1 in CONV_lst do
        Fltrem = {}
        for c = 1 to P̄_th[i] do
            Fltrem+ = P̄re(i, cnv, c)[1 : rm]
        Fltrem ← Maj(Fltrrem, rm))
        Update(uCNN, i , cnv, Fltrem)
P̄_r ← uCNN(N, C̄_L, img);
```

Algorithm 4, requires a statistical analysis by pre-processing a large dataset (i.e., 1.2 images in the training set). Based on this analysis, for each iteration, target CONV layer, and class, a vector is calculated in which, filters are arranged based on accuracy loss due to their removal from the network in ascending order. $\bar{Pre}(i, cnv, c)$ shows the pre-processing vector for the ith iteration, where cnv refers to the target CONV layer and c refers to class label. Thus, the removal of the filter associated with the first argument in $\bar{Pre}(i, cnv, c)$ has the least effect on the overall accuracy of class c. $CONV_{lst}$ is the list of CONV layers targeted for filter pruning. The variable rm is the number of filters to be removed from each CONV layer.

In the ith iteration, the filters least affecting the top-P_{th} classes are gathered in $Fltrem$ determining the candidate filters for removal. Subsequently, the majority function $Maj(Fltrrem, rm)$ returns rm most repeated arguments in $Fltrem$. This allows us to find the union of filters that least affect the top-$P_{th}[i]$ classes in the ith iteration. Subsequently, $Update(uCNNi, cnv, Fltrem)$ updates the ith uCNN by removing the rm filters in $Fltrem$ from cnvth CONV.

10.4 Pruning and Thresholding Hybrid Policy (PTHP)

The PTHP scheme takes advantage of both early termination in the thresholding scheme and the pruning of the FC layers in context-aware pruning policy. In this article, two variants of hybrid PTHP policies are studied:

(1) Fixed-Percentage Pruning & Thresholding ($PTHP_{FP}$): Algorithm 5 shows the first studied hybrid policy. In the first uCNN, $\bar{C_L}$ contains all 1000 labels. After each uCNN the top-5 confidence ($\sum_{k=1}^{5} \bar{P_r}[k]$) is compared with $D_{Ct}[i]$. If confidence is greater, then the image is

ALGORITHM 5: Fixed-Percentage Pruning & Thresholding ($PTHP_{FP}$) Policy

```
for i = 1 to N − 1 do
    P⃗_r ← uCNN(1, C⃗_L, img);
    Sort_descending(P⃗_r);
    if ((Σ_{k=1}^{5} P⃗_r[k]) > D_Ct[i]) then
        exit;
    else
        C⃗_L ← C⃗_L[1 : P_th[i]];
        P⃗_r ← uCNN(i, C⃗_L, img);
P⃗_r ← uCNN(N, C⃗_L, img);
```

classified. Otherwise, a number of classes are pruned based on Pruning policy $P_{th}[i]$ defined for each iteration.

(2) Confidence-Tracking Pruning & Thresholding ($PTHP_{CT}$): Algorithm 6 illustrates the $PTHP_{CT}$ policy. In the first uCNN the probability for all classes (all labels) is computed. In the while loop, if top-5 confidence is above the detection threshold $D_{Ct}[i]$, the classification is terminated. Otherwise, based on the value of a *Saturation Threshold* $S_{th}[i]$, a number of labels are selected for further processing in the next layer. The selected classes are the minimum number of labels with an accumulated probability of no less than $S_{th}[i]$. In this algorithm $C\bar{C}_L$ is the shrunk version of $\bar{C}_L$ that only contains the labels of interest.

ALGORITHM 6: Confidence-Tracking Pruning & Thresholding ($PTHP_{CT}$) Policy

```
P⃗_r ← uCNN(1, C⃗_L, img);
for i = 1 to N − 1 do
    Sort_descending(P⃗_r);
    if ((Σ_{k=1}^{5} P⃗_r[k]) > D_Ct[i]) then
        exit;
    else
        Sum = 0, CC̄_L = [], label=1;
        while Sum < S_th[i] do
            Sum+ = P⃗_r[label];
            CC̄_L[label] = C̄_L[label];
            label++;
        P⃗_r ← uCNN(i + 1, CC̄_L, img);
```

11 IMPLEMENTATION RESULTS

For reporting the accuracy values, we evaluated the 50K images in the validation set of ImageNet repository. We use the FLOP count as an indicator of computational complexity (MFLOP is equivalent to Mega FLOP). It should be noted that since the accuracy of the parallel training approach is higher, the parallel trained networks were used in the implementations. The results for the implementation of proposed policies are summarized next:

11.1 Dynamic Deadline Policy for Real-time Applications

Figure 10 shows the overall accuracy and FLOP count of each iteration when using DD policy. Each bar representing the flop count of an iteration accumulates the total flop count of its previous iteration with that of CONV and FC layer of the current iteration. The figure shows that assuming a large time-budget, continuing the ICNN to the last iteration still results in lower computational

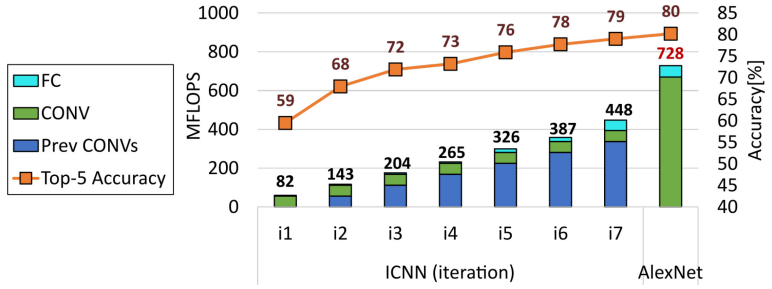


Fig. 10. The FLOP count and accuracy for dynamic deadline policy.

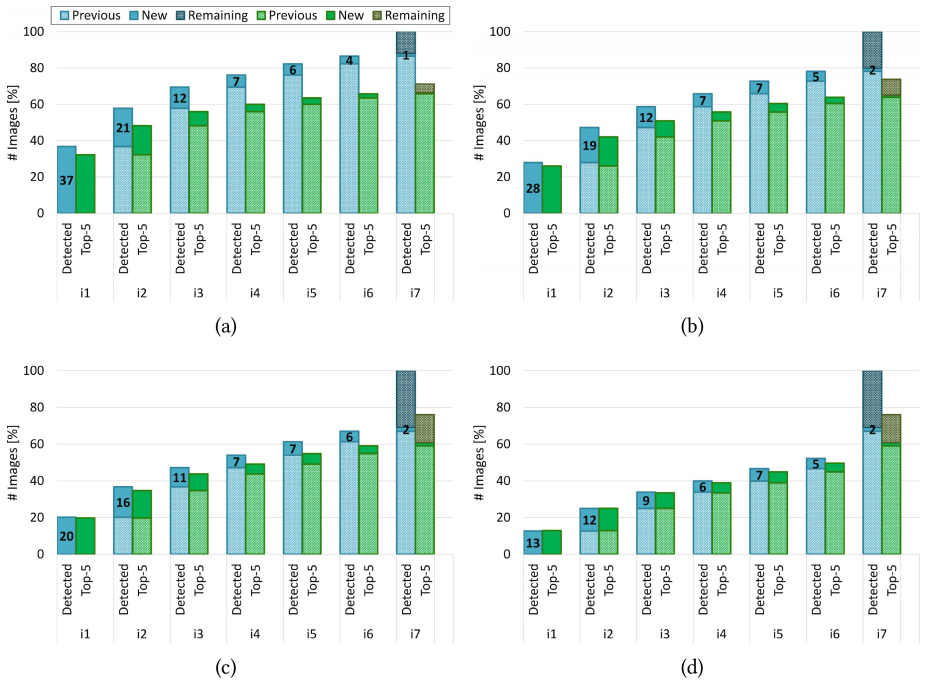
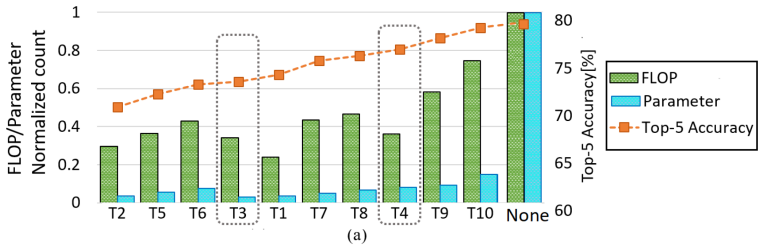


Fig. 11. Number of classified images and the number of classified images with a correct label in the top-5 probabilities for various confidence thresholds (Dc_t): (a) $Dc_t = 0.6$; (b) $Dc_t = 0.7$; (c) $Dc_t = 0.8$; (d) $Dc_t = 0.9$.

complexity (38% lower than original AlexNet) with only a 1% reduction in the top-5 accuracy. Note that the reduction in FLOPs in the last iterations of ICNN with *DD* policy is more significant than the thresholding policy. This is mainly a result of skipping the intermediate FC layers. However, assuming a limited timing budget, a reliable classification decision could still be made in the early iterations with a much lower computational cost. Thus, ICNN is no longer limited to a one-fits-all architecture and is allowed to make efficient use of the resources to make the best prediction, based on the requirements of each application.

11.2 Thresholding Policy for Dynamic Complexity Reduction

(1) TP_F: Figure 11 shows the total number of images classified up to each iteration and their top-5 score for various fixed D_{Ct} values. Each blue bar represents the number of classified images, and each green bar represents the number of correctly classified images. Hence, the difference



	Thresholding Policy									
it	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
i1	0.8	0.5	0.85	0.8	0.5	0.5	0.9	0.8	0.95	0.95
i2	0.7	0.5	0.7	0.8	0.55	0.6	0.8	0.8	0.9	0.95
i3	0.6	0.5	0.55	0.75	0.6	0.7	0.7	0.7	0.85	0.95
i4	0.5	0.5	0.4	0.75	0.65	0.8	0.6	0.7	0.8	0.95
i5	0.4	0.5	0.25	0.7	0.7	0.9	0.5	0.6	0.75	0.95
i6	0.3	0.5	0.1	0.7	0.75	0.9	0.4	0.6	0.7	0.95

e.g. T9 policy: for stage 3 D_{ct}=0.85

(b)

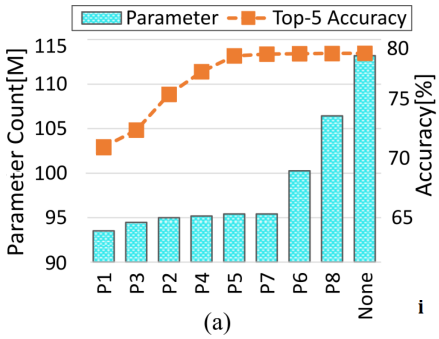
Fig. 12. Variable thresholding policies: (a) the accuracy and average FLOP count; (b) the table including the $\bar{D_{Ct}}$ values in each iteration for each thresholding policy. (For the last iteration, all the remaining images are classified.)

between the height of the blue and green bar is the number of incurred miss-classifications. Each bar is divided into two segments. The first segment is the accumulated number of classified images up to that iteration, and the second segment is the classifications being concluded in that iteration. The final bars on iteration 7 have an additional segment for *remaining* images that have not passed the classification threshold, however, are classified for lack of further iterations. For the *remaining* images, the top-5 classification accuracy is significantly lower than those with confidence values higher than the threshold. With increasing values of D_{Ct} the number of these *remaining* images increases, pushing the classifier toward the last iteration and thus increasing the total FLOP count.

Figure 11 shows that by increasing the value of D_{Ct}, the number of images classified in early iterations decreases; however the (top-5) classification accuracy increases. In Figure 11 this is illustrated by comparing the difference in the heights of *top-5* and *Detected* bars in each iteration, where a larger delta means larger miss-classification. More specifically, higher values of D_{Ct} enhances the accuracy at the expense of larger computation complexity.

(2) TP$_V$: Figure 12(a) shows the overall accuracy and average FLOP count for variable thresholding policy. It should be noted that since not all images go through all the iterations of ICNN, the FLOP count varies for each image. Thus, in Figure 12 the average parameter and FLOP counts are reported over 50K images in the validation set. Moreover, to better highlight how the thresholding policy contributes to reducing computational complexity, the FLOP and parameter counts are devised relative to the last iteration of ICNN. Figure 12(b) shows the $\bar{D_{Ct}}$ values for each iteration of the studied thresholding policies. T1 to T10 are sorted based on FLOP counts, with T1 corresponding to the policy with the lowest FLOP count and T10 the highest.

Figure 12 shows that even with a mild thresholding policy as in T10, the FLOP and parameter counts are reduced by up to 25% and 80%, respectively, with negligible accuracy loss. It should be noted that the number of parameters in the early iterations of ICNN is significantly lower than the later iterations. For instance, the first and the last iteration account for less than 3% and more than 50% of the total parameter count, respectively. This is due to the fact that FC layers require 99× more parameters compared to the CONV layers in the proposed ICNN structure. With the FC layers being smaller in the early iterations, a significant drop in the parameter count is observed due to skipping the parameters in FC layers of last iterations as a result of early termination.



	Pruning Policy							
it	P1	P2	P3	P4	P5	P6	P7	P8
i1	1k	1k	1k	1k	1k	1k	1k	1k
i2	10	20	150	100	100	200	400	400
i3	10	20	60	100	100	200	320	400
i4	10	20	30	100	100	200	256	400
i5	10	20	18	100	100	200	205	400
i6	10	20	13	50	100	200	164	400
i7	10	20	10	25	100	200	131	400

(b)

Fig. 13. Pruning policies: (a) the accuracy and average parameter count for FC layers; (b) the table including the number of labels not pruned in previous iterations. (The first iteration uses all 1,000 labels, as there is no feedback from the previous iterations.)

The complexity/accuracy trade-off suggests that by setting high values for confidence thresholds, and thus higher FLOP count, the accuracy increases from T1 to T10. However, extensive exploration of a large set of variable thresholding policies with ICNN yields the thresholding policy to be more of a Pareto optimization question, where increased accuracy can be achieved with lower FLOP count through intelligent tuning of the D_{Ct} values. For instance, while the FLOP counts of T1 and T4 are significantly lower than T3 and T8, respectively, they yield higher accuracy.

11.3 Context-aware Pruning Policy for Parameter Reduction

(1) CAPP$_{FC}$: Figure 13(a) shows the accuracy and average parameter count for various FC layer pruning policies, while the table in Figure 13(b) captures the setting of the pruning policy ($\bar{P_{th}}$) by listing the number of remaining classes after pruning the classes with low probability. The last bar shows the ICNN results when none of the classes are pruned. P1 to P8 are sorted based on their parameter count, with P1 having the lowest parameter count and P8 the highest.

Figure 13 shows that by increasing the number of pruned classes (i.e., lower P_{th}) the accuracy drops; however, intelligent selection of the pruning policy yields reductions in the parameter count, with negligible accuracy loss (see P4 yields with a 17% reduction in the parameter count). Note that in this scheme, all the images go through all the iterations. Thus, the only reduction in the FLOP count is due to the pruning of the last FC layer in each iteration. Newer and deeper CNNs (e.g., VGGNet and GoogleNet) deploy a single FC layer as opposed to three FC layers in AlexNet. Hence, Application of pruning policy to deeper CNNs increases the rate of FLOP count reduction as well as parameter count reduction in their iterative version. Note that moving the large set of parameters required for the FC from the memory accounts for a significant energy and execution delay.

(1) CAPP$_{CONV}$: Figure 14 captures the result of filter pruning for the CONV5 layer in iterations 2 to 7. Note that the CONV5 of each iteration consists of 32 filters, and in each iteration, 5, 10, and 15 filters least affecting the top-5 classes from the previous iteration are pruned (i.e., $rm = 5, 10, 15$ and $P_{th} = 5$). Our proposed smart approach takes advantage of the feedback from previous iterations and depending on the remaining classes removes the least contributing filters, while the ignorant approach prunes filters based on how strongly each filter contributes to the overall accuracy of the network across all classes and all images in the training dataset. As illustrated in Figure 14, the contextual knowledge generated after each classification, and selective pruning of filters based on remaining classes significantly reduces the loss in classification accuracy when pruning filters.

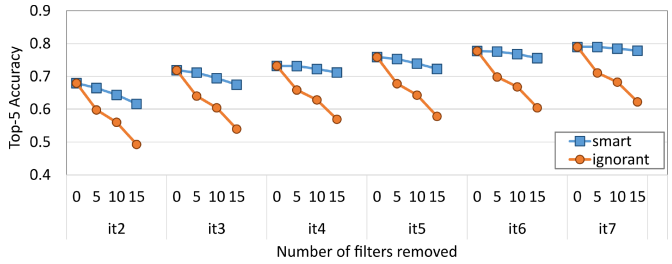
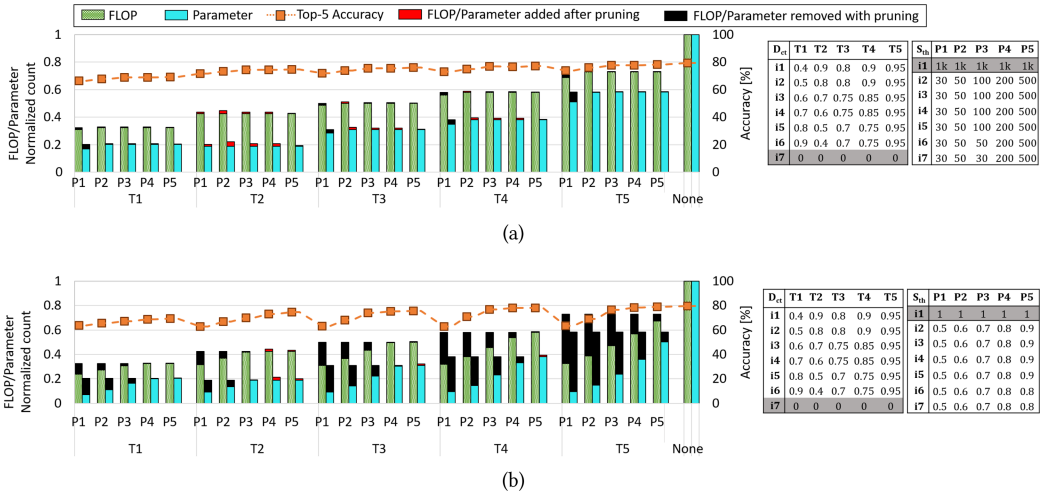


Fig. 14. The accuracy of $CAPP_{CONV}$ for the last CONV layer (i.e., CONV5) in iterations 2–7. The ignorant trend-line shows the results for pruning filters that least affect the accuracy for detection of all classes, while the smart trend-line shows the results for when feedback from previous iterations is used to prune the filters.



D_{ct}	T1	T2	T3	T4	T5
i1	0.4	0.9	0.8	0.9	0.95
i2	0.5	0.8	0.8	0.9	0.95
i3	0.6	0.7	0.75	0.85	0.95
i4	0.7	0.6	0.75	0.85	0.95
i5	0.8	0.5	0.7	0.75	0.95
i6	0.9	0.4	0.7	0.75	0.95
i7	0	0	0	0	0

S_{th}	P1	P2	P3	P4	P5
i1	1k	1k	1k	1k	1k
i2	30	50	100	200	500
i3	30	50	100	200	500
i4	30	50	100	200	500
i5	30	50	100	200	500
i6	30	50	50	200	500
i7	30	50	30	200	500

D_{ct}	T1	T2	T3	T4	T5
i1	0.4	0.9	0.8	0.9	0.95
i2	0.5	0.8	0.8	0.9	0.95
i3	0.6	0.7	0.75	0.85	0.95
i4	0.7	0.6	0.75	0.85	0.95
i5	0.8	0.5	0.7	0.75	0.95
i6	0.9	0.4	0.7	0.75	0.95
i7	0	0	0	0	0

S_{th}	P1	P2	P3	P4	P5
i1	1	1	1	1	1
i2	0.5	0.6	0.7	0.8	0.9
i3	0.5	0.6	0.7	0.8	0.9
i4	0.5	0.6	0.7	0.8	0.9
i5	0.5	0.6	0.7	0.8	0.9
i6	0.5	0.6	0.7	0.8	0.8
i7	0.5	0.6	0.7	0.8	0.8

Fig. 15. The accuracy and average FLOP and parameter counts (normalized to ICNN with no pruning or threshold policy) for (a) $PTHP_{FP}$ and (b) $PTHP_{CT}$. For each thresholding policy (i.e., $\bar{D_{Ct}}$ vector), multiple pruning policies (i.e., $\bar{S_{th}}$ vector) are investigated. The red and black bars show the increase and reduction, respectively, in the number of FLOP and/or parameter counts of a thresholding policy due to pruning.

11.4 Pruning and Thresholding Hybrid Policy

Figures 15(a-left) and 15(b-left) show the accuracy, and normalized FLOP and parameter counts for the $PTHP$ policy, while Figures 15(a-right) and 15(b-right) capture the setting used for experimented thresholding and pruning policies in the hybrid approach. The figure is divided into five sections, one for each thresholding policy. Each segment assesses the impact of five different pruning policies on total parameter and flop count when combined with thresholding in $PTHP$.

(1) PTHP$_{FP}$: Figure 15(a) shows the normalized accuracy, FLOP and parameter counts for $FTHP_{FP}$ policy. For each thresholding policy, we study how each $\bar{P_{th}}$ policy affects the FLOP and parameter counts by highlighting the increase and decrease in the average count by red, and black colors.

An inverse trend is observed when both thresholding and pruning policies are aggressive (images with lower confidence are classified in early iterations, and more classes are pruned in each iteration), where the parameter and FLOP counts are increased by pruning. This is due to the fact that an aggressive pruning policy that keeps only a small number of classes, might eliminate one or more classes in the top-5 classes of the next iteration, reducing the top-5 confidence of the next

iteration and forcing ICNN to proceed to its next iteration, which increases the parameter and FLOP counts.

(2) PTHP$_{CT}$: As illustrated in Figure 15, with aggressive thresholding (e.g., T1), the opportunities for pruning are limited due to the significant drop in accuracy. However, with moderate thresholding some (but not all) of the applied pruning policies allow the parameter and FLOP counts to significantly drop with small impact on the accuracy. This is mainly because these pruning policies allow only the high probability classes (with an accumulated probability of S_{th}) to proceed to the next iterations.

In the $PTHP_{CT}$ the classification would be terminated if (1) a top-5 confidence of D_{Ct} is reached, or (2) the number of classes selected for the next iteration is no bigger than 5. In the second case, proceeding to the next iteration does not increase the top-5 accuracy and ICNN is terminated while the confidence threshold is not reached. The effect of this early termination is twofold: On the one hand, it identifies images that would never reach high classification confidence, and for which processing of more iterations is a waste of resources, hence reducing computation with no accuracy loss. On the other hand, it prematurely terminates the processing of images that could have reached high confidences in the proceeding iteration, negatively affecting the accuracy. Hence, selection of the best combination of $\bar{D_{Ct}}$ and $\bar{S_{th}}$ is an optimization problem. Figure 15(b) shows that when the value of $\bar{S_{th}}$ in each iteration is selected to be slightly higher than $\bar{D_{Ct}}$ of the next iteration, reductions in FLOP and parameter counts has the least impact on the accuracy (see T4-P5 hybrid policy). It should be noted that the inverse impact of aggressive pruning and thresholding on parameter and flop count mentioned for Figure 15(b) is observed for Figure 15(a) too (see red bars in Figure 15).

11.5 Run-time and Overall Accuracy

11.5.1 Pruning and/or Thresholding. Adopting the pruning or thresholding policy creates a trade-off between the average delay of classification and the accuracy of classification. The pruning policy reduces the parameter count, and the thresholding policy reduces the average FLOP count. However, based on the results (see Figures 13 and 12), increased resources does not always translate into higher accuracy. The hybrid policies, which combine both pruning and thresholding, exhibit the same behavior, in which a higher number of parameters and FLOPs does not always contribute to higher classification accuracy. Consequently, finding the optimal strategy in which the target accuracy is reached with minimal resources (in this case execution time), requires thorough exploration and tuning of the thresholding and pruning parameters.

Figure 16 captures the design space of the ICNN when trading the accuracy to reduce run-time through 5k combination of thresholding and pruning policies. In this figure, each point denotes a unique combination of thresholding and pruning policies. For the real-time applications, waiting for a whole batch to be processed significantly adds to the latency [75], thus a batch size of 1 was selected to target real-time streaming applications.

As illustrated, the stand-alone thresholding policy yields better run-time/accuracy trade-off compared to the stand-alone pruning policy. However, their combination in hybrid policy could lead to optimal solutions. Several optimization points for the hybrid policies in Figure 16 exhibit lower run-time compared to the stand-alone thresholding policy with the same accuracy. For instance Figure 16 highlights a stand-alone thresholding policy and a hybrid policy derived by combining the same thresholding policy with pruning. Figure 16 shows that the hybrid policy reduces the classification time by 38% with only 1% accuracy loss.

11.5.2 Deadline-driven. Figure 17 shows the overall accuracy of deadline-driven (*DD*) policy for iterative AlexNet with a batch size of 1 for 50k validation images on K80 GPU for various

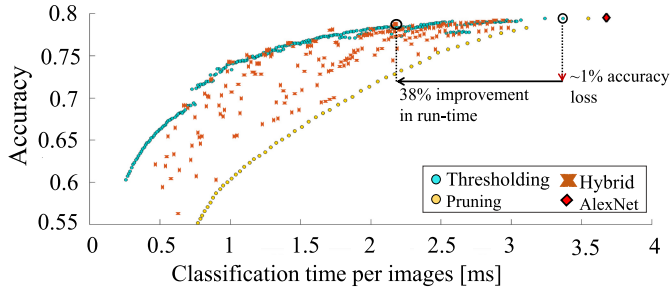


Fig. 16. The accuracy vs. average run-time per image of iterative AlexNet with a batch size of 1 for 50k validation images on K80 GPU. The diamond marks the original AlexNet. The policies on the top border of the diagram mark the most efficient ones in terms of accuracy and execution time.

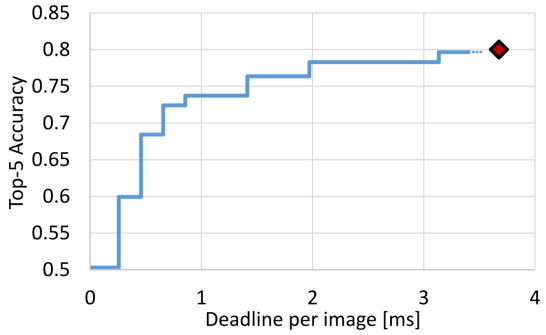


Fig. 17. The overall accuracy of *DD* for iterative AlexNet with a batch size of 1 for 50k validation images on K80 GPU. The diamond marks the original AlexNet. Higher time-budget only increases the accuracy when it is sufficient for execution of the next iteration.

time-budgets per image. As opposed to the pruning and thresholding policies, all the images go through the same iterations for any given deadline. Thus, the accuracy is captured as a function of the time-budget. Note that higher time-budgets only increase the accuracy when it is sufficient for execution of the next iteration, which results in the step-like behavior observed in the figure.

Unlike the thresholding and pruning policies, the *DD* policy only executes the FC layer of one iteration, more specifically the last iteration given the time-budget. However, the pruning and/or thresholding policies require the FC layers of all the iterations to be completed to draw a conclusion about the early termination and pruning of class labels. As a result, for higher accuracy values (i.e., high D_{Ct} and P_{th} and low S_{th} values in hybrid/thresholding/pruning policies, and large time-budget for the *DD* policy), where all images go through all the iterations, the *DD* policy yields lower execution time due to the fact that the computations in the FC layers of the preceding iterations are skipped in the *DD* policy. For instance, a top-5 accuracy of 79% is reached with a time-budget of 1.7ms with *DD* policy, while the same average run-time yields at least 1% drop in the top-5 accuracy with the *TPHP* policies. The *DD* policy allows adjusting the run-time per images in the range of 0.3–3.2 ms based on the available time-budget by trading off accuracy.

Table 2 captures the detection accuracy and FLOPS of ICNN with popular efficient models. The table shows that the complexity of the CNNs has dramatically increased over time to enhance their classification accuracy. However, ICNN proposes an architecture that allows a dynamic trade-off between FLOPs and accuracy. The Iterative version of AlexNet, for instance, achieves the same accuracy as the original network with 38% fewer FLOPs. Moreover, with ICNN, the FLOPs could

Table 2. Comparison with the Existing CNN Architectures

	ZFNet [68]	VGG [15]	GoogLeNet [17]	ResNet [18]	MobileNet V1 [76]	MobileNet V2 [77]	Inception V3 [27]	AlexNet [14]	ICNN (AlexNet version)
Top-5 Accuracy[%]	88.8	89.6	89.9	96.3	89.9	91.0	96.3	80.2	59.2–80.2
FLOPS	663M	19.6G	1.5G	11.3G	568M	300M	12G	729M	82M-448M

be further reduced by trading off accuracy. Although ICNN was constructed based on AlexNet in this article, by following the same approach, each of these networks could be converted to their iterative version, for trading off the accuracy versus computational complexity and energy efficiency.

12 CONCLUSIONS

In this article, an iterative architecture for processing convolutional neural network (ICNN) was proposed. As a case study, the proposed ICNN was used to break the large CNN network of AlexNet into a sequence of smaller networks (uCNN), each processing a sub-sample of the input image, providing the ability to terminate the classification early (when reaching a deadline in a real-time system or reaching a required classification accuracy in low-power systems), or carry the classification to the next iteration (if the deadline in a real-time system is not reached, or classification accuracy threshold is not satisfied in a low-power system). ICNN enable us to explore a wide range of complexity versus accuracy trade-offs. To explore these trade-offs, a dynamic deadline-driven exit policy for real-time applications, a confidence thresholding policy for dynamic complexity reduction, a context-aware pruning policy for parameter reduction and two hybrid pruning and thresholding policies for simultaneous parameter and complexity reduction were introduced. Our simulation results show that with an intelligent selection of the pruning and/or thresholding policies, ICNN reduces the average FLOP and parameter counts, and execution time across 50K validation images in ImageNet database by more than 25%, 80%, and 38%, respectively, with negligible accuracy loss. The reductions in the number of FLOP and parameter counts are directly translated to energy saving, motivating a power-aware CNN architecture. Moreover, the real-time systems could exploit the dynamic structure of the ICNN by reducing the execution time by up to 12× by trading off accuracy with execution time.

REFERENCES

[1] Kyumin Lee, James Caverlee, and Steve Webb. 2010. Uncovering social spammers: Social honeypots+ machine learning. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 435–442.

[2] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. 2017. Disease prediction by machine learning over big data from healthcare communities. *IEEE Access* 5 (2017), 8869–8879.

[3] Marjan Saadati, Jill Nelson, and Hasan Ayaz. 2020. Multimodal fNIRS-EEG classification using deep-learning algorithms for brain-computer interfaces purposes. In *Advances in Neuroergonomics and Cognitive Engineering*, Hasan Ayaz (Ed.). Springer International Publishing, Cham, 209–220.

[4] Marjan Saadati, Jill Nelson, and Hasan Ayaz. 2020. Convolutional neural network for hybrid fNIRS-EEG mental workload classification. In *Advances in Neuroergonomics and Cognitive Engineering*, Hasan Ayaz (Ed.). Springer International Publishing, Cham, 221–232.

[5] Ivan Firdausi, Alva Erwin, Anto Satriyo Nugroho et al. 2010. Analysis of machine learning techniques used in behavior-based malware detection. In *Proceedings of the 2nd International Conference on Advances in Computing, Control, and Telecommunication Technologies*. IEEE, 201–203.

[6] Igor Santos, Jaime Devesa, Felix Brezo, Javier Nieves, and Pablo Garcia Bringas. 2013. Opem: A static-dynamic approach for machine-learning-based malware detection. In *Proceedings of the International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions*. Springer, 271–280.

[7] Hossein Sayadi, Hosein Mohammadi Makrani, Sai Manoj Pudukotai Dinakarrao, Tinoosh Mohsenin, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2019. 2SMaRT: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. IEEE, 728–733.

[8] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal, and A. Cristal. 2017. A machine learning approach for performance prediction and scheduling on heterogeneous CPUs. In *Proceedings of the 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'17)*. 121–128. DOI: http://dx.doi.org/10.1109/SBAC-PAD.2017.23

[9] Hosein Mohammadi Makrani, Hossein Sayadi, Tinoosh Mohsenin, Avesta Sasan, Houman Homayoun et al. 2019. XPPE: Cross-platform performance estimation of hardware accelerators using machine learning. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 727–732.

[10] Hossein Sayadi, Nisarg Patel, Avesta Sasan, and Houman Homayoun. 2017. Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures. In *Proceedings of the IEEE International Conference on Computer Design (ICCD'17)*. IEEE, 129–136.

[11] Raymond Sepe Jr. 2005. Voice actuation with contextual learning for intelligent machine control. U.S. Patent 6,895,380.

[12] Charu C. Aggarwal and ChengXiang Zhai. 2012. *Mining Text Data*. Springer Science & Business Media.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. MIT Press, 1097–1105.

[15] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[16] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2921–2929.

[17] Christian Szegedy et al. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[18] Kaiming He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[19] Katayoun Neshatpour, Avesta Sasan, and Houman Homayoun. 2016. Big data analytics on heterogeneous accelerator architectures. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS'16)*. IEEE, 1–3.

[20] Katayoun Neshatpour, Arezou Koohi, Farnoud Farahmand, Rajiv Joshi, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. 2016. Big biomedical image processing hardware acceleration: A case study for K-means and image filtering. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'16)*. IEEE, 1134–1137.

[21] Katayoun Neshatpour, Farnaz Behnia, Houman Homayoun, and Avesta Sasan. 2018. ICNN: An iterative implementation of convolutional neural networks to enable energy and computational complexity aware dynamic approximation. In *Proceedings of the Design Automation and Test in Europe*.

[22] Mengchen Liu et al. 2017. Towards better analysis of deep convolutional neural networks. *IEEE Trans. Visual. Comput. Graph.* 23, 1 (2017), 91–100.

[23] Yann LeCun et al. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 4 (1989), 541–551.

[24] Yu-Hsin Chen et al. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circ.* 52, 1 (2017), 127–138.

[25] J. Deng et al. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'09)*.

[26] Kaiming He et al. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*. 1026–1034.

[27] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.

[28] Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.

[29] Brian D. Ripley. 2007. *Pattern Recognition and Neural Networks*. Cambridge University Press.

[30] A. Jafari, A. Ganesan, C. S. K. Thalisetty, V. Sivasubramanian, T. Oates, and T. Mohsenin. 2019. SensorNet: A scalable and low-power deep convolutional neural network for multimodal data classification. *IEEE Trans. Circ. Syst. I: Regular Papers* 66, 1 (Jan. 2019), 274–287. DOI: http://dx.doi.org/10.1109/TCSI.2018.2848647

[31] Morteza Hosseini, Mark Horton, Hiren Paneliya, Uttej Kallakuri, Houman Homayoun, and Tinoosh Mohsenin. 2019. On the complexity reduction of dense layers from $O(N^2)$ to $O(NlogN)$ with cyclic sparsely connected layers. In *Proceedings of the 56th Annual Design Automation Conference*. ACM.

[32] Bharat Prakash, Mark Horton, Nicholas Waytowich, William David Hairston, Tim Oates, and Tinoosh Mohsenin. 2019. On the use of deep autoencoders for efficient embedded reinforcement learning. In *Proceedings of the 29th Edition of the Great Lakes Symposium on VLSI (GLSVLSI'19)*. ACM.

[33] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866* (2014).

[34] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553* (2014).

[35] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 806–814.

[36] Morteza Hosseini, Hiren Paneliya, Mohit Kallakuri, Uttej Khatwani, and Tinoosh Mohsenin. 2019. Minimizing classification energy of binarized neural network inference for wearable devices. In *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED'19)*. IEEE.

[37] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV'16)*. Springer, 525–542.

[38] Shimeng Yu, Zhiwei Li, Pai-Yu Chen, Huaqiang Wu, Bin Gao, Deli Wang, Wei Wu, and He Qian. 2016. Binary neural network with 16 Mb RRAM macro chip for classification and online training. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'16)*. IEEE, 16–2.

[39] Jung H. Kim and Sung-Kwon Park. 1995. The geometrical learning of binary neural networks. *IEEE Trans Neural Netw.* 6, 1 (1995), 237–247.

[40] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems*. MIT Press, 4107–4115.

[41] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[42] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4820–4828.

[43] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bit-width gradients. *arXiv preprint arXiv:1606.06160* (2016).

[44] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Low precision arithmetic for deep learning. *CoRR abs/1412.7024* 4 (2014).

[45] Suyog Gupta, Ankur Agrawal, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the International Conference on Machine Learning (ICML'15)*. 1737–1746.

[46] Darryl Dexu Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Proceedings of the International Conference on Machine Learning (ICML'16)*. 2849–2858.

[47] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 367–379.

[48] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, Vol. 49. ACM, 269–284.

[49] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 92–104.

[50] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, and François Berry. 2018. Accelerating CNN inference on FPGAs: A survey. *arXiv preprint arXiv:1806.01683* (2018).

[51] Katayoun Neshatpour, Hosein Mohammadi Mokrani, Avesta Sasan, Hassan Ghasemzadeh, Setareh Rafatirad, and Houman Homayoun. 2018. Architectural considerations for FPGA acceleration of machine learning applications in MapReduce. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. ACM, 89–96.

[52] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. 2016. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In *Proceedings of International Symposium on Low-Power Electronics and Design*. 326–331.

[53] Katayoun Neshatpour, Hosein Mohammadi Makrani, Avesta Sasan, Hassan Ghasemzadeh, Setareh Rafatirad, and Houman Homayoun. 2018. Design space exploration for hardware acceleration of machine learning applications in

MapReduce. In *Proceedings of the IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'18)*. IEEE, 221–221.

[54] Hokchhay Tann, Soheil Hashemi, R. Iris Bahar, and Sherief Reda. 2016. Runtime configurable deep neural networks for energy-accuracy trade-off. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS'16)*. IEEE, 1–10.

[55] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. 2016. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *Proceedings of the Design, Automation & Test in Europe Conference*. IEEE, 475–480.

[56] Surat Teerapittayanon, Bradley McDanel, and HT Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *Proceedings of the International Conference on Pattern Recognition (ICPR'16)*. IEEE, 2464–2469.

[57] Li Deng. 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.* 29, 6 (2012), 141–142.

[58] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR abs/1211.5063* (2012).

[59] Katayoun Neshatpour, Farnaz Behnia, Houman Homayoun, and Avesta Sasan. 2019. Exploiting energy-accuracy trade-off through contextual awareness in multi-stage convolutional neural networks. In *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED'19)*. IEEE, 265–270.

[60] C. Sidney Burrus et al. 1997. Introduction to wavelets and wavelet transforms: A primer. (1997).

[61] Yangqing Jia et al. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*. ACM, 675–678.

[62] Stéphane Mallat. 1999. *A Wavelet Tour of Signal Processing*. Academic Press.

[63] Kamrul Hasan Talukder and Koichi Harada. 2010. Haar wavelet-based approach for image compression and quality assessment of compressed image. *arXiv preprint arXiv:1010.4084* (2010).

[64] V. Elamaran and A. Praveen. 2012. Comparison of DCT and wavelets in image coding. In *Proceedings of the International Conference on Computer Communication and Informatics*. 1–4.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2014. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Proceedings of the European Conference on Computer Vision*. Springer, 346–361.

[66] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. 2011. Adaptive deconvolutional networks for mid and high level feature learning. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'11)*. IEEE, 2018–2025.

[67] Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Christian Ledig, and Zehan Wang. 2016. Is the deconvolution layer the same as a convolutional layer? *arXiv preprint arXiv:1609.07009* (2016).

[68] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 818–833.

[69] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.

[70] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. 2015. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2758–2766.

[71] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[72] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.* 13, 3, Article 32 (Feb. 2017), 18 pages. DOI: http://dx.doi.org/10.1145/3005348

[73] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[74] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*. IEEE Press, 243–254.

[75] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[76] Andrew G. Howard, Menglong Zhu, Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[77] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

Received January 2019; revised May 2019; accepted August 2019