An automated approach for improving the inference latency and energy efficiency of pretrained CNNs by removing irrelevant pixels with focused convolutions

Caleb Tung* tung3@purdue.edu
Nicholas Eliopoulos* neliopou@purdue.edu
Purvish Jajal* pjajal@purdue.edu
Gowri Ramshankar* gramshan@purdue.edu
Cheng-Yun Yang* yang2316@purdue.edu
Nicholas Synovic† nsynovic@luc.edu
Xuecen Zhang^ xxz1037@case.edu
Vipin Chaudhary^ vipin@case.edu
George K. Thiruvathukal† gkt@cs.luc.edu
Yung-Hsiang Lu* yunglu@purdue.edu

* Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, USA
†Department of Computer Science, Loyola University Chicago, Chicago, Illinois, USA
^Computer and Data Sciences Department, Case Western Reserve University, Cleveland, Ohio, USA

Abstract— Computer vision often uses highly accurate Convolutional Neural Networks (CNNs), but these deep learning models are associated with ever-increasing energy and computation requirements. Producing more energy-efficient CNNs often requires model training which can be cost-prohibitive. We propose a novel, automated method to make a pretrained CNN more energy-efficient without re-training. Given a pretrained CNN, we insert a threshold layer that filters activations from the preceding layers to identify regions of the image that are irrelevant, i.e. can be ignored by the following layers while maintaining accuracy. Our modified focused convolution operation saves inference latency (by up to 25%) and energy costs (by up to 22%) on various popular pretrained CNNs, with little to no loss in accuracy.

I. Introduction

Pretrained Convolutional Neural Networks (CNNs) are prevalent because they enable anyone – even those without the means to personally train CNNs – to leverage state-of-the-art computer vision models. CNNs are very computationally intensive and require power-hungry GPUs to execute, making them difficult to deploy in contexts like battery/solar-powered, mobile, embedded, and Internet-of-Things systems when GPUs or cloud offloading are unavailable. Existing energy-efficient CNN techniques often require either an end-to-end retrain or a completely new model design. Though successful, those techniques often require training and thus cannot be used with pretrained CNNs.

This paper proposes an easy-to-use method to *reduce the energy consumption of pretrained CNNs without retraining.* We insert a threshold layer into the pretrained CNN; this layer applies a brightness threshold to the activations from the preceding layers, determining which regions (background, uninteresting objects, etc.) of a given input image are deemed *irrelevant* for an accurate inference. The CNN's remaining layers are replaced with *focused convolutions* which completely ignore those irrelevant regions, saving computation. Illustrated in Figure 1, *this method modifies a pretrained CNN using the following steps*:

First, to choose the kth layer at which to insert the threshold layer, we model the energy consumption of the layers in the CNN as a function of k, allowing us to automatically select the insertion point.

Next, to choose the brightness threshold τ, we propose an automated latency-versus-accuracy curve search that yields a single threshold value to be used on the target dataset.

Finally, we replace the remaining convolutional layers with our improved *focused convolutions* [1] to ignore the irrelevant pixels. In our prior work, the original focused convolution was a drop-in replacement for the standard General-Matrix Multiply (GEMM) convolutional technique used by contemporary AI libraries. Made to only perform convolutions inside a predetermined Area of Interest (AoI) mask, our prior method required a different AoI mask per layer per inference along with a compute-heavy depth-mapping AoI generation technique. This rendered their design less suitable for inference in environments where objects moved about or the scene changed regularly. In contrast, our improvements use only one mask for the entire CNN and a hardware-aware block size, dramatically reducing computational overhead and boosting parallel processing utilization.

> The proposed technique requires no training and instead modifies pretrained CNNs to save computation in resource-constrained deployments without GPUs or cloud offloading.
> Our method automatically adjusts for different datasets and trades off accuracy, energy efficiency, and latency to meet a range of deployment requirements.
> In some cases, considerable energy efficiency (up to 22%) and latency improvements (up to 25%) can be achieved without degrading accuracy.
> A notable advantage of this training-free approach is that improvements can be achieved with little effort.

We test the proposed technique on multiple popular pretrained models, including ResNet [2], VGG [3], ConvNeXt [4], Faster-RCNN [5], and SSDLite [6]. We test on ImageNet [7] for image classification and on Microsoft COCO [8] for object detection. Our method can reduce a pretrained CNN's inference energy consumption by up to 22% on different processor types (Intel, AMD, Arm), with little to no loss of accuracy (0-2% loss). Further, inference latency is shortened by up to 25%.

We also compare our technique's inference accuracy and latency with that of similarly inspired energy-efficient CNNs, *showing that our method is either competitive or better than those techniques*, all without needing the training that the other techniques require. Code is open-sourced on GitHub at https://github.com/purdueseris/focused-convolutions/.

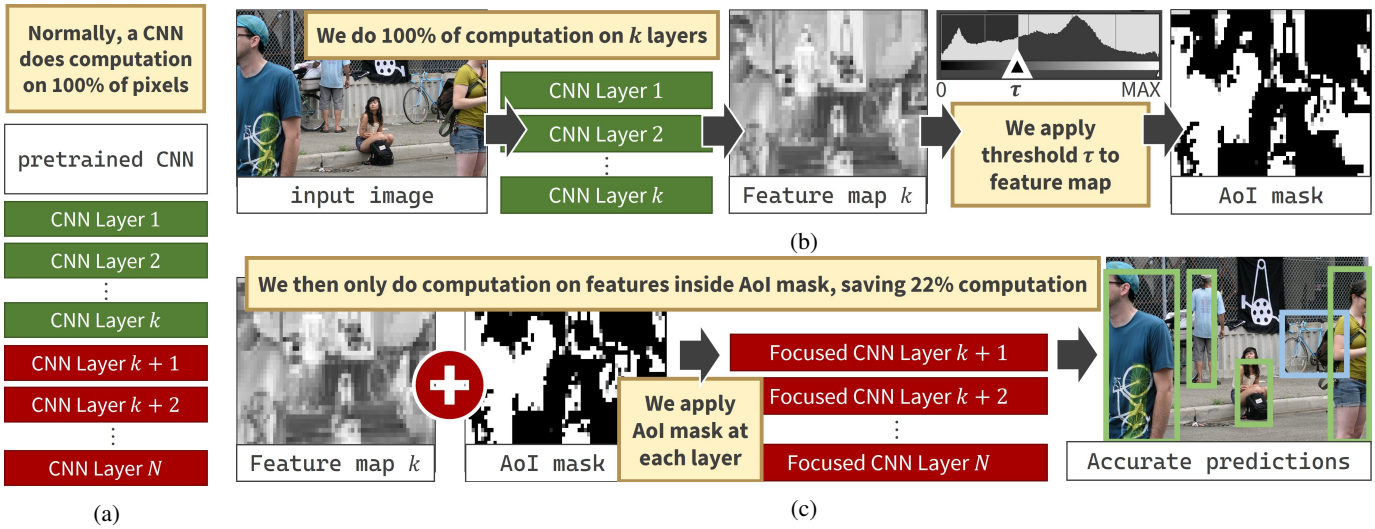

Fig. 1: **(a)** A pretrained CNN does computation on 100% of the input pixels for all N layers. The proposed method makes the CNN more efficient by: **(b)** First, only do 100% computation during the first k layers to collect contextual features. Second, apply a brightness threshold τ on the feature map from the kth layer to identify an Area of Interest (AoI) mask. As illustrated, white regions are relevant for an accurate prediction, black regions are irrelevant. Note: Select k, τ beforehand via a CNN energy consumption projection and an accuracy-vs-latency curve search, respectively. **(c)** Finally, in the last $N - k$ layers, completely ignore the irrelevant regions using focused convolutions. This saves computation with little to no loss in accuracy.

II. Related Work and Background

There are many efforts to improve CNN efficiency [9]. They typically focus on: (1) reducing the model size (allowing it to fit more effectively into processor caches), (2) reducing the number of operations (reducing the load on the processor) [10]. Specialized hardware, such as neural accelerators [11], can also run CNNs more efficiently. However, this paper focuses on software-side improvements.

A. Similarly Inspired Methods to Our Technique

Some techniques also use the concept of identifying unnecessary computation to skip at inference time. *Early Exit architectures* decide if the CNN is confident enough to make a prediction before it finishes executing all the layers. The *Spatially Adaptive Computation (SAC)* family of models can be trained end-to-end to identify regions of interest and then use special blocks to reduce computation outside those regions [12]. We compare our technique with these methods in Section IV.

B. Other Energy-Efficient, Low-Latency Methods

Traditional techniques make alterations to existing CNN architectures before training. *Quantization* shrinks the size of a CNN by reducing the CNN's precision (e.g. storing numbers as 8-bit integers instead of 32-bit floats) [13]. *Pruning* reduces both computation and model size by deleting channels/features or neurons that are not activating often; the smaller model is then retrained [9]. *Knowledge Distillation* uses a large, trained model to "teach" a smaller, more efficient model for deployment [9]. *Neural Architecture Search* automatically searches a space of CNN building blocks for energy-efficient architecture.

C. Irrelevant Pixels and Areas of Interest

Irrelevant pixels in an image do not contribute to the CNN's ability to make an accurate prediction. Pixels that are useful to the CNN comprise *"Areas of Interest" (AoI)* of Relevant Pixels; irrelevant pixels fall outside the AoI [1]. An image can have multiple, disjoint regions that comprise the image's AoI.

The existing approach of generating AoIs using depth mapping neural networks [14] is accurate but is too computationally expensive to deploy on resource-constrained systems.

D. How Focused Convolutions Work

In our prior work, the *focused convolution* [1] is based on the popular General-Matrix Multiply (GEMM) technique for doing convolutions [15]. In GEMM, an input image is segmented into convolution kernel-sized *patches*. Each patch is then vectorized into a matrix's column or row. That matrix is then multiplied with the weights matrix to produce the convolution output.

The focused convolution, a drop-in-replacement for a GEMM convolution, applies an AoI that was generated in advance. Any patches not found inside the AoI are deemed irrelevant and then excluded from the matrix. That results in a smaller matrix and thus a less computation-intense matrix multiplication. The AoI not being generated at inference time keeps the CNN from being able to truly replace existing models for any application.

E. Novelty of Our Contributions to Literature

The related techniques described above all require varying degrees of training or otherwise completely redesigning the CNN. This means that considerable human expertise and compute is required to apply those other techniques. *Our method, in contrast, requires no training and can be used on a pre-trained CNN.*

III. Automatically Identifying and Removing Irrelevant Pixels at Inference Time

The proposed technique takes a pretrained CNN and then produces the *fCNN*, a modified version of the CNN that applies a threshold to the early layers of the CNN to automatically generate AoIs and use focused convolutions for the remainder of the network with those AoIs. An fCNN behaves as follows:

1) Process input image using only the beginning k layers (k chosen in Section III-A) of the CNN (referred to as $NN.top$). Produces feature map X.
2) Sum X along the channels. Produces X_{sum}.
3) Filter X_{sum} with the activation brightness threshold τ (τ chosen in Section III-B) to produce the AoI. Activations bright enough to clear the τ threshold are allowed through as corresponding regions of relevant AoI pixels; the rest are discarded. Produces X_{thresh}.
4) All convolutional layers after $NN.top$ use our improved focused convolutions (Section III-C) on X_{thresh}, saving energy by discarding irrelevant pixels. The focused convolutions use the same weights and biases as the convolutions they replace.

To generate this modified fCNN, we need to choose k and τ, and replace the convolutional layers after the k-th layer with focused convolutions.

A. Choosing the k Layers in NN.top

The beginning k layers of an fCNN will process every pixel of the input image, gathering information that can then be filtered by an inserted layer which applies activation brightness threshold τ to generate an AoI part-way through inference. This AoI will then be used by the later layers' focused convolutions.

The smaller k is, the earlier the τ-threshold will get applied. This also implies that more layers remain in the CNN to take advantage of the energy improvements of the focused convolution. Therefore, a smaller k is beneficial. Meanwhile, k cannot be too small: too early on in the CNN, there is insufficient information encoded within the features to make a useful threshold AoI [9]. CNNs generally collect basic features about the input image in the first few layers. Deeper layers accumulate those into more complex features later in the network [3]. Therefore k must be large enough to capture useful information in the features produced at the kth layer.

To choose k, we use a heuristic: *choose the latest layer (i.e., largest k) that is still small enough such that the resulting fCNN can meet the deployment energy requirement.*

Focused convolution energy savings were determined to be approximately linear with respect to AoI size [1] and CNN computation is known to scale linearly with respect to input size [16]. Therefore, we linearly model the energy consumption of the CNN, and then use that information to project the energy savings of its fCNN equivalent, selecting k.

Let the energy consumption of the ith convolutional layer be $E_{c,i}$, the expected AoI size be a as a percentage of the original input size, and the measured energy from the computational overhead introduced by the focused convolution be c (this can be measured by manually setting the focused convolution AoI size to 100% and then subtracting $E_{c,i}$). Then, the energy use of the corresponding focused convolutional layer $E_{f,i}$ is modeled linearly as $E_{f,i} = aE_{c,i} + c$. For a CNN with k total conv layers, of which k belong to $NN.top$, then there will be $N-k$ focused convolutional layers. We model the total energy consumed by the convolutions in the fCNN in Equation 1.

$$E_{total} = (N-k)c + \sum_{i=1}^{k} E_{c,i} + \sum_{i=k+1}^{N} aE_{f,i} \qquad (1)$$

Thus, k can be selected such that E_{total} just meets the deployment constraint. Note: if the deployment constraint is lower than the overhead costs, then our technique determines that a suitable fCNN is unachievable.

B. Choosing Activation Brightness Threshold τ

The activation brightness threshold τ determines which pixels are considered relevant inside the AoI and which are deemed irrelevant. Our method tries the CNN with different τ values over a few iterations on the training dataset. After each full iteration over the dataset, it measures the average accuracy and inference latency to see if it meets the deployment requirements. If not, it iterates again with an adjusted τ-value. Although this bears similarity to training, our method does not backpropagate or modify any model weights at all, whereas training requires many epochs and backpropagation [2].

As shown in Figure 2, the proposed technique chooses the activation brightness threshold τ as follows: τ is used to filter the output of $NN.top$, the top k layers of the CNN. If a given region is brighter than the threshold, then it is allowed through as relevant pixels in the AoI. The higher τ is, the fewer pixels are allowed through and the smaller the AoI is (keep in mind that the AoI can be comprised of multiple distinct regions). τ is initialized to the minimum value of the sum of all $NN.top$ features, ensuring that any $NN.top$ output will pass through the threshold (i.e. 100% AoI). We wish to achieve a maximum target T for the CNN's inference latency $NN.t$, as well as a minimum target A for the CNN's accuracy $NN.a$.

The proposed technique increases τ by increments of some ϵ, shrinking the AoI and improving latency, until the latency target is met. Then, it checks to see if the accuracy target is also satisfied. If not, it begins reducing τ to attempt to find a smaller τ that can satisfy both targets. The size of the increment is adjusted based on the relative distance of $NN.a$ from A, getting smaller the closer the search gets to the target (i.e. as $|A - NN.a|$ shrinks in size, relative to A). The search succeeds if both T, A are both attainable, and times out if the search cannot succeed after a pre-set period of time. Thus, the search explores along the accuracy-latency tradeoff curve, succeeding when (T, A) is a point on or within the curve.

It is worth noting here that the k selection process can be rolled into the τ selection framework: an outer loop can try different k values, while the inner loop selects τ as shown. However, this paper recommends k be pre-estimated using a simple mathematical calculation based on energy predictions because it achieves energy usage and latency improvements with fewer iterations over the training dataset.

C. Improving Focused Convolution Parallelization

Modern computer architectures implement specialized hardware (e.g., "Neon" vector registers on Arm CPUs and "CUDA"

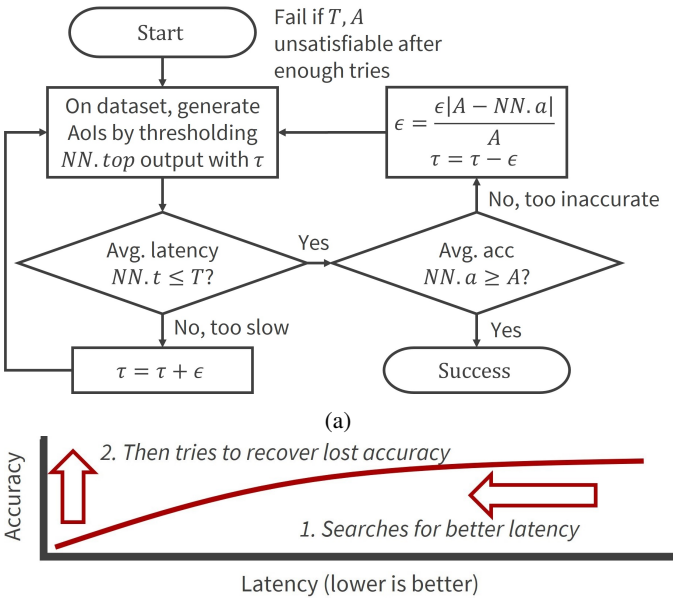

Fig. 2: **(a)** Training-free process to choose activation brightness threshold τ, given a maximum inference latency threshold of T and a minimum accuracy threshold of A. This proposed method will succeed if latency and accuracy targets T, A are simultaneously attainable, and fails otherwise. **(b)** This technique searches along the accuracy-latency curve, succeeding if (T, A) is on the curve.

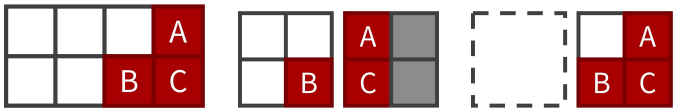

Fig. 3: Example of our technique's memory alignment. **(a)** 2×4 input data, where the three elements A, B, C belong to the AoI. **(b)** On hardware that can parallelize the data processing in blocks of 4, the original focused convolution's sliding-window patch selection sends two blocks of size-4 data for processing: B is contained in the first block, while A, C are contained in the second block. **(c)** The proposed technique downsamples to multiples of the hardware blocksize, resulting in only one size-4 block of data being sent with A, B, C, thus saving processing on one block.

cores on NVIDIA GPUs) used to parallelize operations on multiple, independent pieces of data. These parallel operations are supported by Single-Instruction-Multiple-Data (SIMD) instructions. The data is collated into blocks that perfectly fit the size of a vector register on the processor. The processor then uses specialized instructions to process the entire register's data in parallel. For example, a GEMM convolution operating on an entire input tensor could send multiple patches for simultaneous processing to improve inference latency.

Contemporary machine learning frameworks like PyTorch and TensorFlow already rely on C-native libraries to use parallel processing for built-in operations like "Conv2d". The focused convolution already uses these libraries [1]. However, we observe that the generic sliding-window indexing approach used by the original focused convolution results in inefficient utilization of the parallel processing cores. An example is illustrated in Figure 3: our technique will only select a single block of data instead of two, thus saving the processing.

To achieve this, we design the focused convolution to use a common computer architecture technique called *memory alignment* [17]. We pre-divide the input into a grid, where each cell is divided in multiples of the parallel processing block size. If a cell contains part of the AoI, the entire cell is sent for parallel processing. Memory alignment ensures a more efficient usage of the processor than the original focused convolution. Because we do not need to directly change the kernel parallelization primitives, and only change the way the data is laid out, the proposed technique is instantly compatible with any libraries using optimizations for SIMD, CUDA, etc.

IV. Results and Discussion

To demonstrate the utility of our method, we measure various performance aspects of different pretrained models when modified using our technique. We choose three pretrained image classifier models (VGG-16, ResNet-18, ConvNeXt-T) and two pretrained object detection models (Faster-RCNN and SSDLite) from Meta AI Research's Torchvision library. We then use the proposed technique, with ImageNet for image classification and Microsoft COCO for object detection, to determine k and τ. Then, we modify each model to use focused convolutions, and we measure energy consumption, inference latency, accuracy, and Multiply-Accumulate (MAC), comparing the focused convolution models with the unmodified ones.

Note: In this section, we often compare the perfomance of an unmodified, pretrained CNN with the focused-convolution fCNN version using "% improvement" or "% degradation". This is calculated using the formula $\frac{|\text{unmodified} - \text{focused conv}|}{\text{unmodified}}$.

A. Experimental Setup

We test using three devices with different levels of power consumption and different operating systems. We demonstrate improvements in energy consumption and inference latency without using any GPUs, hardware accelerators, or cloud offloading:

- Embedded (5 W): Broadcom Arm Cortex-A5, Debian
- Laptop PC (28 W): Intel Core i7, Ubuntu
- Desktop PC (142 W): AMD Ryzen 9, Windows

On the Arm embedded device, energy consumption is physically measured using a Monsoon Solutions HV Power Monitor. On the Intel laptop PC, measurements are taken using Intel's "Power Gadget" software, and on the AMD desktop, measurements are recorded with ASUS' "Armoury Crate" software. Baseline steady-state power consumption is recorded and subtracted from the numbers measured during inference.

The focused convolutions are compiled to use with Pytorch. On each device, the SIMD blocksize Section III is set according to the specifications from the processor's documentation.

We measure inference accuracy and MAC on ImageNet and Microsoft COCO using averaged numbers from the "torchbench" and "ptflops" libraries.

CNN	Dataset	Accuracy	MAC/inf	Energy/inference (J)			Latency/inference (ms)		
				Intel	AMD	Arm	Intel	AMD	Arm
VGG-16	ImageNet-1K	0.716	15.50G	6.9	11.2	10.1	242.1	83.2	2020.9
fVGG-16	**ImageNet-1K**	**0.716**	**14.19G**	**6.4**	**10.9**	**8.9**	**222.8**	**77.1**	**1799.0**
ResNet-18	ImageNet-1K	0.698	1.82G	2.0	3.1	2.3	54.2	18.2	457.7
fResNet-18	**ImageNet-1K**	**0.697**	**1.60G**	**1.5**	**2.6**	**2.1**	**50.19**	**16.4**	**410.8**
ConvNeXt-T	ImageNet-1K	0.821	4.47G	3.4	6.5	5.1	112.4	41.6	960.4
fConvNeXt-T	**ImageNet-1K**	**0.818**	**4.05G**	**2.9**	**5.2**	**4.3**	**99.9**	**37.0**	**854.3**
Faster-RCNN	COCO	0.370	120.87G	68.1	106.8	-	2390.1	751.9	-
fFaster-RCNN	**COCO**	**0.370**	**101.30G**	**57.7**	**88.9**	**-**	**2011.5**	**616.6**	**-**
SSDLite	COCO	0.210	716.42M	3.0	7.2	5.7	100.5	48.6	1083.7
fSSDLite	**COCO**	**0.192**	**599.06M**	**2.3**	**5.8**	**4.5**	**79.4**	**39.7**	**876.4**

TABLE I: Pretrained CNNs are compared with their corresponding "fCNNs" **(in bold)** using our method on an Intel laptop, an AMD desktop, and an Arm embedded device. Latency improvements can be achieved with little to no accuracy loss. Cells with "-" indicate that the model could not run on the device (exceeded memory capacity).

We report accuracy using the Top-1 ImageNet accuracy metric [7] and the Box mAP COCO accuracy metric [8].

B. Activation Brightness Threshold Selection

We follow the method described in Section III to create fCNNs from the pretrained CNNs and select k and τ.

For each model, several convolutional layers come before the first downsampling point, at which the input size shrinks due to either striding or pooling. k is selected at the first downsample point of the model. This allows for enough flexibility to choose a τ while retaining sufficient image-wide information from the early layers.

For image classification, we start the automated τ search for a latency target T that is 10% better than the pretrained CNN, with an accuracy target A matching the accuracy of the original CNN. As the τ value increases, fewer pixels are allowed past the threshold, shrinking the size of the AoI. This also causes the model's accuracy to begin dropping linearly. However, there are cases (e.g. VGG-16) where the accuracy holds steady while latency drops, indicating that **it is possible to achieve the accuracy of the original models while improving latency.**

The technique selects the best point, where the most latency is saved while dropping the least accuracy. Those models using our technique are denoted as the "fCNN" models. The same process is repeated to determine the "fCNN" models for the object detectors on Microsoft COCO.

To choose the τ for our "fCNN" models, we do not need to retrain. Although our curve search will iterate over the training dataset, it is much faster than retraining a model, since we do not do backpropagation and only use 7 iterations to select a τ for each model.

C. Improvements On Pretrained CNNs

We compare our "fCNN" models with their corresponding unmodified pretrained CNNs in Table I. As shown, across desktop, laptop, and embedded processors, the technique successfully converts pretrained CNNs into faster, more energy-efficient models that still achieve the same or mildly degraded accuracy. Object detection models achieve more improvements because the COCO images often have smaller AoIs than the ImageNet images.

We also demonstrate qualitative results. In Figure 4, we show examples of the AoIs selected by our τ-thresholds in the different CNNs on images from COCO and ImageNet. Often, the selected AoIs draw the CNN's focus to the same areas that human eyes would focus on, although sometimes, the pretrained CNN seems to focus on areas of the image that seem less relevant. As shown, the technique can identify multiple AoIs in the pictures.

A notable regression is fSSDLite. The Torchvision pretrained SSDLite model is noted as more sensitive to perturbations in pixel values, so we suspect that the deletion of pixels marked irrelevant still negatively impacts the model. Additionally, SSDLite does not use the multi-scale Feature Pyramid Network from the Torchvision Faster-RCNN [5], struggling more when objects are smaller.

We also note that as more aggressive τ thresholds are selected, the energy consumption of the models improves more quickly than the accuracy degrades; for a more extreme example, it is possible to achieve a 28% energy consumption improvement on ConvNeXt-T with only a 15% loss in accuracy (accuracy dropped by 0.003, from 0.821 to 0.818).

In summary, our technique allows some latency and energy consumption improvements to be gained without suffering accuracy loss, and then allows accuracy and latency to be further traded off as the deployment scenario requires. We maintain this versatility without requiring any training.

D. Comparison with Similarly Inspired Techniques

While not an apples-to-apples comparison since our technique does not require the training that the other methods do, we provide a comparison with similarly inspired techniques (Section II) and an INT8-quantized baseline for ResNet-18.

Our focused fResNet-18 is both faster and more accurate than the Early Exit neural network [18], is faster than the standard ResNet-18 [2], is more accurate than quantized ResNet-18 [13], and is faster than the Spatially Adaptive Computation model (SAC) [12].

It is worth noting that although quantized ResNet-18 is shown to be roughly 15ms faster than our focused ResNet-18, the speedup from quantization may be challenging to achieve in practice. Quantization requires a time-consuming, often difficult-to-understand calibration process on the dataset. Meanwhile, the focused ResNet-18 achieves its speedup without any such training.

In short, our technique either outperforms or stays competitive with similarly inspired techniques (Figure 5), all while being easy to implement because it requires no training.

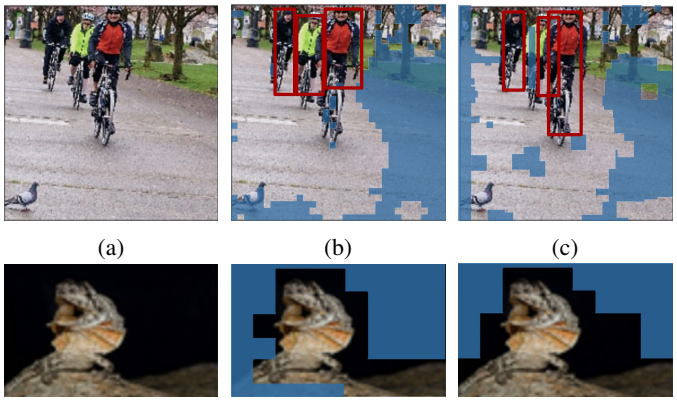

Fig. 4: The proposed technique can ignore regions of irrelevant pixels (marked in blue in the thresholded saliency maps) from the original images. The resulting Area of Interest (AoI) focuses on parts of the image that the human eye would. **(a)** Original COCO image. **(b)** fFaster-RCNN. **(c)** fSSDLite. **(d)** Original ImageNet image. **(e)** fResNet-18. **(f)** fVGG-16. By ignoring computation on the blue regions, fCNNs save up to **12% energy on ImageNet and up to 22% energy on COCO.**

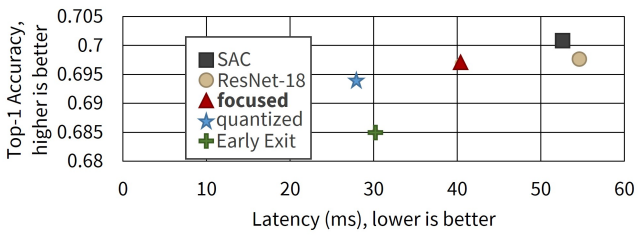

Fig. 5: Our technique "(focused)" compared with similarly inspired techniques on our Intel CPU. SAC [12] and Early Exit [18] CNNs require training and a complete redesign of the CNN; **our technique beats them or stays competitive, and also requires zero training, keeping the pretrained CNN intact.** Static-quantized [13] and unmodified ResNet-18 are shown to provide a baseline reference.

V. Conclusion

This paper presents a novel technique for converting a pretrained computer vision model into a more energy-efficient model, with no additional training. We apply a threshold (determined using an accuracy-latency curve search method) to the features produced by the few early layers of the CNN to automatically generate an Area of Interest (AoI) for the given input image. Pixels inside the AoI are relevant, the rest are irrelevant. Irrelevant pixels are ignored, reducing computational cost and energy expenditure while improving inference latency. The proposed technique uses a memory alignment method to ensure full utilization of parallel processing. By keeping the weights and biases of the original pretrained model, a CNN pretrained on one dataset can still use our method for computation savings on a different dataset. We achieve an average of 8%-12% energy savings with popular image classifiers (VGG, ConvNeXt) on ImageNet, and 15%-22% energy savings with popular object detectors (Faster-RCNN, RetinaNet) on COCO, with little to no loss in accuracy. Code is open-sourced on GitHub at https://github.com/purdueseris/focused-convolutions/.

Acknowledgements

This research project is supported by funding from Cisco and National Science Foundation 2107230, 2107020, 2104709, 2104319, 2104377. Any opinions, findings, and conclusions expressed in this paper are those of the authors and do not necessarily reflect the sponsors' views.

References

[1] C. Tung, A. Goel, X. Hu, N. Eliopoulos, E. S. Amobi, G. K. Thiruvathukal, V. Chaudhary, and Y.-H. Lu, "Irrelevant pixels are everywhere: Find and exclude them for more efficient computer vision," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 340–343, IEEE, 2022.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[4] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11976–11986, 2022.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, pp. 21–37, Springer, 2016.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, Ieee, 2009.

[8] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision (ECCV)*, pp. 740–755, Springer, 2014.

[9] A. Goel, C. Tung, Y.-H. Lu, and G. K. Thiruvathukal, "A survey of methods for low-power deep learning and computer vision," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pp. 1–6, IEEE, 2020.

[10] G. K. Thiruvathukal, Y.-H. Lu, J. Kim, Y. Chen, and B. Chen, *Low-power computer vision: Improve the efficiency of artificial intelligence*. CRC Press, 2022.

[11] A. Skillman and T. Edso, "A technical overview of cortex-m55 and ethos-u55: Arm's most capable processors for endpoint ai," in *2020 IEEE Hot Chips 32 Symposium (HCS)*, pp. 1–20, IEEE Computer Society, 2020.

[12] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1039–1048, 2017.

[13] T. Dubhir, M. Mishra, and R. Singhal, "Benchmarking of quantization libraries in popular frameworks," in *2021 IEEE International Conference on Big Data (Big Data)*, pp. 3050–3055, IEEE, 2021.

[14] J. Kopf, X. Rong, and J.-B. Huang, "Robust consistent video depth estimation," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1611–1621, 2021.

[15] B. Kågström, P. Ling, and C. Van Loan, "Gemm-based level 3 blas: high-performance model implementations and performance evaluation benchmark," *ACM Transactions on Mathematical Software (TOMS)*, vol. 24, no. 3, pp. 268–302, 1998.

[16] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5687–5695, 2017.

[17] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1145–1155, 1975.

[18] S. Teerapittayanon and B. McDanel, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, IEEE, 2016.