EFFICIENT AND CONSISTENT CONVOLUTIONAL NEURAL NETWORKS FOR COMPUTER VISION

by

Caleb Tung

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Electrical and Computer Engineering

West Lafayette, Indiana

August 2023

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Yung-Hsiang Lu, Chair

Elmore Family School of Electrical and Computer Engineering

Dr. Mahsa Ghasemi

Elmore Family School of Electrical and Computer Engineering

Dr. Qiang Qiu

Elmore Family School of Electrical and Computer Engineering

Dr. George K. Thiruvathukal

Loyola University Chicago, Department of Computer Science

Approved by:

Dr. Milind Kulkarni

To the greatest country in the world. Thanks for the opportunities.

ACKNOWLEDGMENTS

Completing this work would be unthinkable without the help of my advisor, Prof. Yung-Hsiang Lu. He taught me how to do research, win grants, and give speeches. When my PhD career encountered speedbumps, he patiently guided me through the discouragement. Through it all, he consistently tried to place my needs first, and I am deeply grateful.

I greatly appreciate my committee members. Prof. Mahsa Ghasemi kindly helped me refine my research problem on her office whiteboard. Prof. Qiang Qiu's thoughtful feedback formed the outline of my first proposal. Prof. George K. Thiruvathukal always brought his positive attitude and love of classical music and philosophy to our paper-writing video calls.

I am indebted to my internship mentors for their professional advice and valuable research suggestions. To Dr. Jan Ernst and Dr. Sek Chai at Latent AI, and to Danny Loh and Alex Chalfin at Arm. To Chloe, Honson, and KK: thanks for looking after me.

No student (especially not this one) makes it across the finish line without the camaraderie and technical contributions of his teammates in the lab: Abhinav, Sara, Ryan, Nick, Xiao, Purvish, King, and Gowri. It was an honor to work with you.

Thanks to Zachary, Ivan, Tim, Dan, Jennifer, AJ, and Sara for all the laughs, tears, encouragement, and memes. To Jane and Grant for always calling to check on me.

Thanks to my landlady, Catherine, for accommodating my DIY projects around the house when I needed a break from research.

Thank you to my church community. To Pastor Chuck and his wonderful family. To David, Alyssa, Qianqian, and Chris for being the inner circle nuts enough to launch Abide Christian Fellowship with me. To Ben for being the first to love Texas more loudly than me.

Thanks to my beloved grandparents and relatives for always being proud of me.

To my dear sisters, Corrie and Carey, and to Dad and Mom, for their unconditional love, abundant grace, and much-needed wisdom, offered despite the constant challenges of life. Thanks for the home.

To my brother, Tyler.

To my beautiful, unfailingly supportive girlfriend, Annie.

To my Lord and my God, to Whom all praise is due.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

ABBREVIATIONS

AoI	Area of Interest
Conv	Convolutional
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
FC	Fully-Connected
FLOPs	Floating Point Operations
FPS	Frames Per Second
GPU	Graphics Processing Unit
IoT	Internet of Things
IoU	Intersection over Union
MAC	Multiply Accumulate Count
mAP	Mean Average Precision
NAS	Neural Architecture Search
QAT	Quantization Aware Training
RPN	Region Proposal Network

ABSTRACT

Convolutional Neural Networks (CNNs) are machine learning models that are commonly used for computer vision tasks like image classification and object detection. State-of-the-art CNNs achieve high accuracy by using many convolutional filters to extract features from the input images for correct predictions. This high accuracy is achieved at the cost of high computational intensity. Large, accurate CNNs typically require powerful Graphics Processing Units (GPUs) to train and deploy, while attempts at creating smaller, less computationally-intense CNNs lose accuracy. In fact, maintaining consistent accuracy is a challenge for even the state-of-the-art CNNs. This presents a problem: the vast energy expenditure demanded by CNN training raises concerns about environmental impact and sustainability, while the computational intensity of CNN inference makes it challenging for low-power devices (e.g. embedded, mobile, Internet-of-Things) to deploy the CNNs on their limited hardware. Further, when reliable network is limited or when extremely low latency is required, the cloud cannot be used to offload computing from the low-power device, forcing a need to research methods to deploy CNNs on the device itself: to improve energy efficiency and mitigate consistency and accuracy losses of CNNs.

This dissertation investigates causes of CNN accuracy inconsistency and energy consumption. We further propose methods to improve both, enabling CNN deployment on low-power devices. Our methods do not require training to avoid the high energy costs associated with training.

To address accuracy inconsistency, we first design a new metric to properly capture such behavior. We conduct a study of modern object detectors to find that they all exhibit inconsistent behavior. That is, when two images are similar, an object detector can sometimes produce completely different predictions. Malicious actors exploit this to cause CNNs to mispredict, while image distortions caused by camera equipment and natural phenomena can also cause mispredictions. Regardless the cause of the misprediction, we find that modern accuracy metrics do not capture this behavior, and we create a new consistency metric to measure the behavior. Finally, we demonstrate the use of image processing techniques to improve CNN consistency on modern object detection datasets.

To improve CNN energy efficiency and reduce inference latency, we design the focused convolution operation. We observe that in a given image, many pixels are often irrelevant to the computer vision task – if the pixels are deleted, the CNN can still give the correct prediction. We design a method to use a depth mapping neural network to identify which pixels are irrelevant in modern computer vision datasets. Next, we design the focused convolution to automatically ignore any pixels marked irrelevant outside the Area of Interest (AoI). By replacing the standard convolutional operations in CNNs with our focused convolutions, we find that ignoring those irrelevant pixels can save up to 45% energy and inference latency.

Finally, we improve the focused convolutions, allowing for (1) energy-efficient, automated AoI generation within the CNN itself and (2) improved memory alignment and better utilization of parallel processing hardware. The original focused convolution required AoI generation in advance, using a computationally-intense depth mapping method. Our AoI generation technique automatically filters the features from the early layers of a CNN using a threshold. The threshold is determined using an Accuracy vs Latency curve search method. The remaining layers will apply focused convolutions to the AoI to reduce energy use. This will allow focused convolutions to be deployed within any pretrained CNN for various observed use cases. No training is required.

1. INTRODUCTION

Convolutional Neural Networks (CNNs) are deep learning models that are commonly used for computer vision tasks. In recent years, CNNs have achieved high accuracy in vision tasks like image recognition, object detection, and semantic segmentation [1]–[3]. To achieve high accuracy, modern CNNs are generally very deep and large - state-of-the-art CNNs are comprised of billions of parameters, trained to perform hundreds of billions of mathematical operations. The training process itself consumes tremendous amounts of energy; multi-Graphics Processing Unit (GPU) clusters can draw thousands of Watts for days on end, experimenting with different training recipes to find the one that achieves the best CNN accuracy. Thus, achieving high accuracy with state-of-the-art CNNs often requires lots of computation resources and energy to run the hardware.

Besides posing environmental energy use concerns, these demanding computation and energy requirements make CNNs challenging to deploy on affordable, low-power devices, e.g., mobile, Internet-of-Things (IoT), and embedded [4]. These devices are constrained by batteries, low power draw, or limited computational and memory capacity. However, it has become increasingly desirable to deploy computer vision on such low-power devices without requiring heavy re-training.

In many computer vision use cases, it is not possible to simply offload the heavy-duty CNN computation to the cloud; the computation must be completed on the low-power device. For example, driver assistance/collision avoidance systems in cars cannot afford the unreliability or the latency of a cloud connection to determine whether to apply brakes; the computation is executed on the car's embedded processors. Even battery-powered mobile phones, with high-speed cellular Internet connections, are often expected to perform on-device computer vision due to the privacy concerns of sending personal or sensitive image data to remote servers. Thus, it is beneficial to research methods to mitigate the computational requirements of CNNs so that they can better accommodate low-power devices.

Improving the energy efficiency of CNNs presents a new hurdle for deployment on low-power devices: mitigating the degradation of prediction correctness. Deployed CNNs must behave *accurately* and do so *consistently* - it would be disastrous if a collision avoidance

system mis-identified a pedestrian or if an automated package sorter misread a label and routed an important, time-sensitive parcel to the wrong address. Since CNNs tend to lose accuracy with improved energy efficiency, it thus becomes important to find methods to accurately characterize and mitigate the losses of consistency in a CNN.

1.1 Problem Exploration

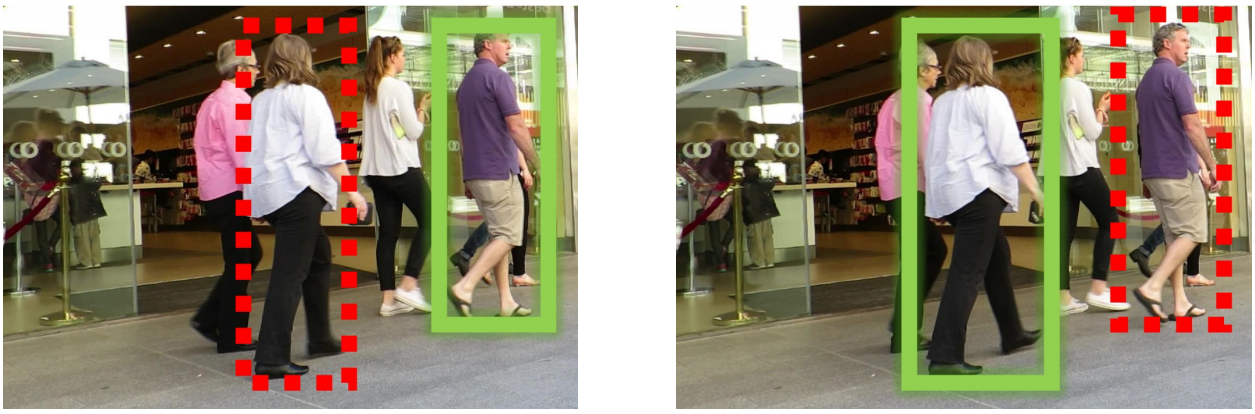
Maintaining consistent accuracy and improving energy efficiency, both without additional training, present challenges that we intend to address with the techniques in this dissertation. The challenges are detailed as follows:

1.1.1 Insufficiency of Accuracy As A Measure of CNN Prediction Correctness

At a high level, typical CNN accuracy metrics can be described as measures of how correct the neural network is, on average, over a given dataset. For example, if given a binary classifier, the CNN is either right or wrong for a given image. Accuracy denotes what percentage of the dataset the CNN predicted correctly. Although that is useful information, there is more to the story that is not captured by the accuracy metric: how *consistently* the CNN is right or wrong, assuming the input is similar.

If a CNN is presented with two images that appear similar or indistinguishable to human eyes, the desired behavior is for the CNN to give similar predictions for the images, even if the two images are slightly different from each other (e.g., offset by a couple pixels, slightly darker, etc.) However, accuracy metrics do not measure whether a CNN exhibits this desired behavior. Figure 1.1 shows an example of cases when slightly different images cause a CNN to behave very differently, but the accuracy metric does not capture that undesired behavior, remaining stable.

Finding a way to properly measure the behavior missed by accuracy metrics is important, because as CNNs are deployed, they are often faced with images that are slightly different from each other [5]. For example, traffic monitoring camera systems are placed outdoors and are constantly subject to slight changes in lighting and exhaust particles in the air, all of which introduce small differences in each picture recorded by the camera. Video cameras



(a) (b)

Figure 1.1. State-of-the-art object detector Mask-RCNN is inconsistent on two images taken 0.03s apart, even though both images look alike. In (a), the woman is missed (red dashed-line box) while the man is detected (green solid box). In (b), the reverse is true. (All other people are detected correctly in both images, giving an average 3/4 accuracy in both images.)

mounted on moving platforms like drones and cars will experience jostle and shake, such that successive video frames will be slightly different from each other. In these cases, it is important to capture how consistently the CNN behaves across such images with small differences. We introduce our consistency metric to do this in Section 1.2.1.

1.1.2 Resource Demands of a CNN and Limitations of a Low-Power Device

A CNN is computationally intensive, causing heavy energy expenditure during both training and inference. To achieve high accuracy, researchers use clusters of top-end NVIDIA GPUs to train CNNs with dozens of different training recipes, with each recipe requiring hundreds of epochs. During inference time, the trained CNNs can require billions of Multiply-Accumulate (MAC) operations, straining the limited computation resources of low-power devices. Table 1.1 shows examples of the demands imposed by several modern image classification and object detection CNNs.

Low-power devices have dramatically less computational capabilities than the GPUs typically used to run CNNs. The latest top-tier NVIDIA GPUs are priced well above $1,000

Table 1.1. Computation Demands of Popular CNNs

CNN	Measure of Computational Intensity	Value
VGG-16	Number of Parameters	138M
	Number of MACs	15.4G
	Model Size	553.4 MB
ResNet-18	Number of Parameters	11.7M
	Number of MACs	1.81G
	Model Size	46.7 MB
ConvNeXt-B	Number of Parameters	88.6M
	Number of MACs	15.4G
	Model Size	354.2 MB
Mask-RCNN	Number of Parameters	44.4M
	Number of MACs	134.4G
	Model Size	177.6 MB
RetinaNet	Number of Parameters	34.0M
	Number of MACs	151.6G
	Model Size	136.1 MB

Table 1.2. Comparison of Powerful GPUs and Low-Power Devices

Device	Power	Num Cores	Ops/Sec	Memory	Price ($USD)
NVIDIA RTX 4090	450 W	16384	100 TFLOPS	24 GB	$1500
NVIDIA Jetson Nano	10 W	128	472 GFLOPS	4 GB	$100
Raspberry Pi 4B	5 W	4	8 GFLOPS	4GB	$55
Raspberry Pi Zero	5 W	1	58 MFLOPS	512 MB	$5

USD, while low-power devices like an Arm-based Raspberry Pi costs just $40. The sheer price discrepancy means that GPUs can be equipped with tremendous amounts of computing cores to handle the many matrix operations demanded by CNN inference, while the low-power devices cannot. Table 1.2 shows a comparison between low-power devices and top-end GPUs to illustrate the discrepancy in computing capability.

To enable CNN deployment on low-power devices, we propose methods to reduce the computation done by CNNs. Our approach removes pixels from computation if the pixels are determined to have no impact on the output of the CNN. We emphasize the re-use of weights and biases in a pretrained CNN, allowing our techniques to be used without costly training. We introduce our method in Section 1.2.2.

1.2 Our Contributions

This dissertation develops techniques to address the challenges facing the deployment of computer vision on low-power devices: (1) improving CNN accuracy metrics and corresponding performance, and (2) improving energy efficiency while maintaining model accuracy. To address (1), we introduce a new metric called *consistency*, designed to cover the shortcomings of typical accuracy metrics. We investigate causes of consistency loss and propose image processing techniques to mitigate them. To address (2), we introduce the *focused convolution*, a replacement of the standard CNN convolution operation, designed to ignore pixels that do not significantly impact the outcome of the CNN. In so doing, the CNN performs less computation and achieves better energy efficiency. We demonstrate our techniques on popular computer vision datasets, using state-of-the-art CNNs, on different types of computer hardware. Our techniques do not require training. Instead, they are designed to work with a pretrained CNN, saving the additional energy cost of training. The methods are summarized below.

1.2.1 Consistency Metric

Consistent accuracy can be generally summarized as follows: for the same CNN, when inputs are similar, the output predictions should be similar. Since traditional accuracy metrics do not capture this behavior, we design a metric to measure this behavior concretely for object detectors: *Consistency*.

Our consistency metric relies on a rolling average across pairs of adjacent video frames. Modern videos are taken at relatively high framerates (e.g. 30, 60 FPS), so adjacent frames are similar. For a given pair of adjacent frames containing the same ground truth objects, we compare the predictions made by the CNN in the first frame with those made in the second frame. The more overlap between the two frames, the higher the pairwise consistency. We then take this averaged across the entire video. We measure consistency for several modern object detection CNNs using the MOT 2015 Challenge, a dataset of video frames taken with stationary cameras. We discover that all object detectors exhibit inconsistency.

Figure 1.2. Our focused convolution technique references a pregenerated Area of Interest (AoI) to ignore irrelevant pixels during computation. The irrelevant pixels are pixels deemed to not have an impact on the output of the CNN, and are chosen using a MiDaS [7] depth mapping neural network.

We also investigate natural image distortions (e.g., camera sensor noise, motion blur, etc.) within MOT 2015 as probable causes of inconsistent behavior in CNNs. We confirm that image distortion mitigation techniques (e.g., image compression, Gaussian denoise, etc.) can improve CNN consistency when applied to the images before they are sent for inference. These methods can all be performed on the input images themselves, without tampering with the architecture of the CNNs or requiring training.

1.2.2 Focused Convolutions

We observe that in many images, some pixels can be considered *irrelevant*, i.e., they do not significantly impact the predictions of the CNN. Existing work in CNN saliency indicates that, like the human eyes, some pixels are important, and others are not [6]. For example, in Figure 1.2, the buses and ground are important; the buildings are not.

We conduct a study on popular computer vision datasets, demonstrating that many pixels (an average of 48%) are irrelevant. We do this by using a neural network called MiDaS to generate depth maps on the images, and then counting pixels at the same depth level (ie same distance from the camera) as a ground truth label to be a relevant pixel. These additional pixels around the ground truth labels are used to provide additional information to the CNN to contextualize the pixels inside the ground truth. Other pixels are thus deemed irrelevant.

We then illustrate the energy consumption and inference latency savings possible by removing the irrelevant pixels from computation during a CNN's inference. We design the *fo-*

cused convolution, a replacement operation for the standard convolution operation in CNNs. The focused convolution uses the same weights and biases as the standard convolution, but it specifically skips over any pixel deemed as irrelevant during computation. We replace the convolutional layers in modern CNNs with focused convolutions, and then by deleting the pixels deemed irrelevant from our computer vision dataset study, we demonstrate that CNNs can save up to 45% energy in modern vision datasets, with little to no loss in accuracy.

1.2.3 Training-Free, Automatically Generated Areas of Interest with Improved Focused Convolutions

Using MiDaS depth maps for AoI generation is computation-intensive and consumes a lot of energy. This completely offsets any energy savings gained from the use of the focused convolution, forcing MiDaS AoI generation to be done offline or only once in a while. This limits the use cases of focused convolutions.

We mitigate this and other shortcomings of the focused convolution using an end-to-end, training-free, automated technique for generating AoIs within the CNN itself. We apply a threshold to filter the activations from the early layers of the CNN to identify pixels that are irrelevant. The threshold is selected by briefly iterating over the training data (but not performing training) to conduct a search along the CNN's accuracy-versus-latency curve for a point that satisfies accuracy and latency targets. All irrelevant pixels are ignored by later focused convolutions in the CNN.

We also improve the focused convolution by using memory alignment to properly utilize the parallel processing cores on the hardware at inference time. The new focused convolution should use a single AoI mask (which can be comprised of multiple disjoint regions) across all layers, without needing the initial forward pass previously required.

1.3 Outline

This dissertation is organized as follows: The first three chapters present different aspects of our work, each with a contained background section that discusses relevant information and related work. Chapter 2 introduces our consistency metric. We present a study of a

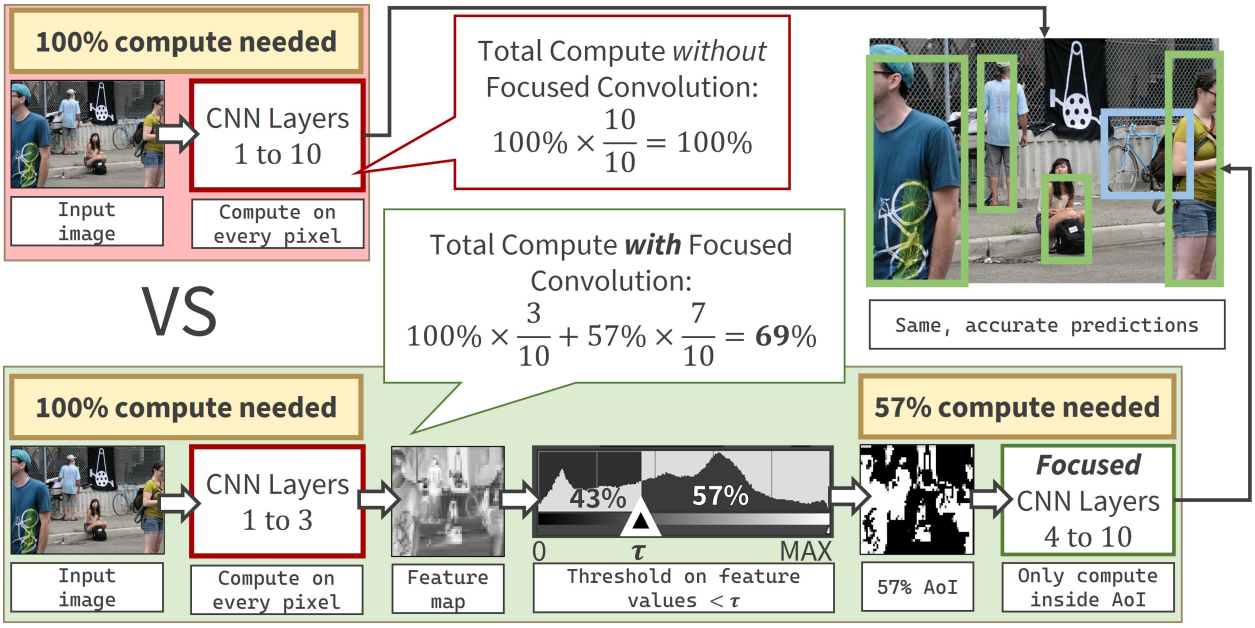


Figure 1.3. We propose a training-free application of a threshold that can filter the CNN feature maps via automated AoI generation. This allows the focused convolution to be widely adopted and used with any pretrained CNN.

major object detection dataset (MOT Challenge 2015 [8]) and demonstrate that modern CNNs are inconsistent on the dataset. We further investigate different image processing techniques to improve the consistency of the CNNs on the data. Chapter 3 presents the focused convolution. We conduct a study to show that many pixels are irrelevant in computer vision datasets. We then demonstrate that when those irrelevant pixels are ignored by the focused convolutions, pretrained CNNs see improved inference latency and energy efficiency. Chapter 4 describes our improvements to the focused convolution: designing a self-contained CNN that generates its own AoIs for focused convolution use. We also propose improving the parallel compute capability of the focused convolution. Chapter 5 summarizes the work and concludes the document. The software artifacts and source code for our techniques are available on GitHub at https://github.com/PurdueCAM2Project/focused-convolutions.

2. CONSISTENCY

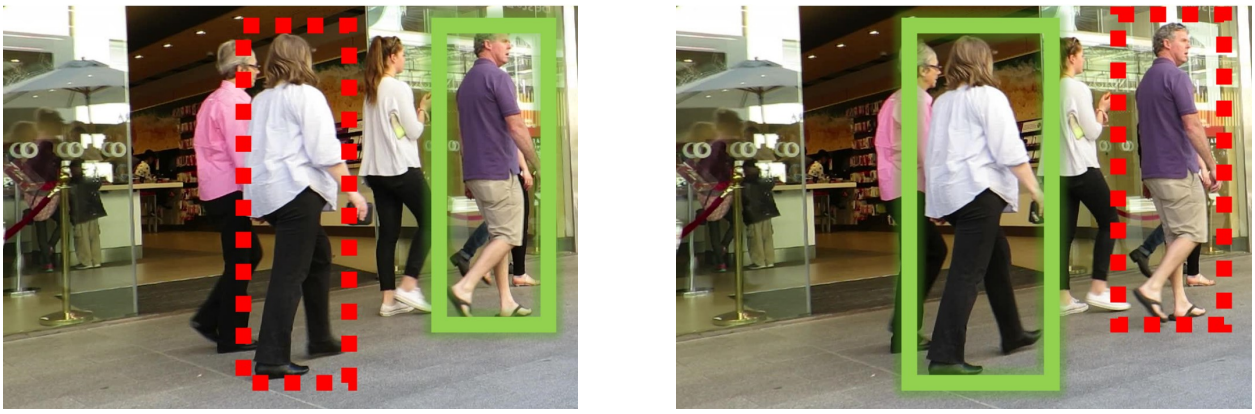
Much of modern computer vision relies on object detectors. An object detector is a Deep Neural Network (DNN) that takes an image as the input and then identifies the locations and types of objects found in that image. Across scientific disciplines, object detectors are increasingly ubiquitous. From electronic package sorting in e-commerce to collision detection in traffic monitoring, from remote sensing in low-orbit satellites to automated MRI screening in the fight against cancer: object detectors are driving an entire frontier of technology. With so many critical applications, object detectors need to be consistently accurate. Modern object detectors use different architectures (e.g., single-shot, R-CNN, etc.) and training methods (e.g., multitask loss, neural architecture search, etc.) to achieve state-of-the-art accuracy on popular image datasets like Microsoft COCO (Common Objects in Context) [9].

2.1 Introduction to Consistency

Even though object detectors are carefully tested for accuracy, this dissertation observes that *consistency* is also a valuable metric that receives less attention in literature. As we discuss later, common image datasets make it challenging to test for consistency.

2.1.1 The Consistency Problem

Accuracy typically measures how often an object detector is correct *on average*. This information is partially deficient because it does not capture the variation in an object detector's performance when input images are similar. Since accuracy is reported as an average, there could be multiple ways an object detector achieves a given accuracy, some of which may be less desirable. For example, in Fig. 2.1, a state-of-the-art object detector (Mask-RCNN [10]) detects three out of four people per image. The accuracy is the same *on average* (3/4), yet the detector behaves inconsistently: it misses a different person in each image.



(a) (b)

Figure 2.1. State-of-the-art object detector Mask-RCNN is inconsistent on two images taken 0.03s apart, even though both images look alike. In (a), the woman is missed (red dashed-line box) while the man is detected (green solid box). In (b), the reverse is true. (All other people are detected correctly in both images, giving an average 3/4 accuracy in both images.)

Inconsistent behavior becomes cause for concern in vision applications that demand strict performance guarantees. For example, a collision prevention algorithm with 95% accuracy that misses the same 5% of objects allows one to investigate the cause of the 5% error more easily, because the defective behavior is *consistent*. However, an algorithm that behaves inconsistently, missing different objects across each image in the test, is much harder to troubleshoot.

This dissertation investigates object detector *consistency* as a method to augment existing accuracy metrics. Consistency measures the difference in predictions from an object detector across *similar* images. We explore different methods to quantify consistency, ultimately choosing a metric that tracks a detectors behavior on time-series images from the MOT Challenge [8]—a dataset originally intended to benchmark object tracking. As shown in Figure 2.2, we find that state-of-the-art detectors (Mask-RCNN, Faster-RCNN [11], RetinaNet [12], SSD [13]) exhibit inconsistent behavior. In our experiments, we observe up to an average of 17% inconsistent detections. We evaluate methods to improve consistency

Table 2.1. Terminology used for calculating object detection accuracy.

	Object detected	**Object not detected**
Object exists	True Positive (TP)	False Negative (FN)
Object does not exist	False Positive (FP)	True Negative (TN)

and present a selection of methods (lossy image compression, gamma boosting, etc.) that successfully raise consistency by up to 5%.

2.1.2 What Accuracy Measures

We determine accuracy by comparing the predictions of an object detector against the ground truth in a dataset of images. Standard accuracy metrics like mean Average Precision (mAP) usually consider two factors simultaneously: (1) how much of the ground truth the detector successfully predicts, and (2) how many of the detectors predictions were incorrect. To get perfect accuracy, each of the predictions an object detector makes must be correct: it cannot blanket the image with guesses.

In computer vision, we define the mAP accuracy metric in terms of *precision* and *recall* using true/false positives and negatives (see Table 2.1). Precision and recall are defined in Equation 2.1 and Equation 2.2, respectively. mAP is calculated for a given image dataset by integrating the area under the precision-versus-recall curve with respect to recall (Equation 2.3).

$$\texttt{precision}\ p = \frac{TP}{TP + FP} \tag{2.1}$$

$$\texttt{recall}\ r = \frac{TP}{TP + FN} \tag{2.2}$$

$$\texttt{accuracy}\ = \int_0^1 p(r)dr \tag{2.3}$$

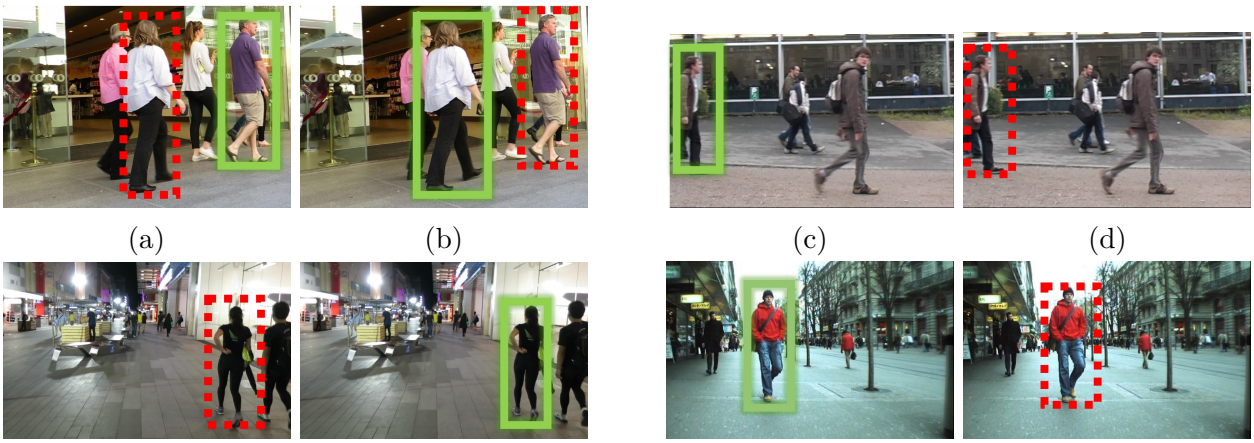


Figure 2.2. Different object detectors behave inconsistently on images that look visually similar. Left images are taken 0.03s before the right. Red, dashed-line boxes are missed detections, green solid boxes are correct detections. (a)-(b) Mask-RCNN. (c)-(d) Faster-RCNN. (e)-(f) RetinaNet. (g)-(h) SSD.

2.1.3 Accuracy does not measure consistency

When accuracy is not 100%, there may be more to the story (for context, the state-of-the-art Mask-RCNN reaches 63% mAP on the COCO dataset). Accuracy does not describe whether the detector is consistent. Consider an object detector that achieves 50% accuracy on a set of images, where each image is slightly different but still contains the same objects. One might hope that the detector would predict consistently as shown in Figure 2.3a-b. However, it is still possible for other detector predictions (Figure 2.3c-d) to achieve the same 50% accuracy, albeit inconsistently.

Additionally, simply reporting fine-grained accuracy statistics does not capture consistency. It is true that we can infer more information about the neural networks consistency by reporting additional statistics like the standard deviation or variance of per-image accuracy across a dataset of similar images. However, reporting the standard deviation still does not reveal the Figure 2.3c-d case discussed above, where the detector inconsistently detects and misses *different* objects in each frame even though the *number* of objects remains the same.

As we showed earlier in Figure 2.2, inconsistent behavior exists even in popular object detectors. Thus, accuracy does not always communicate the full picture of a detectors performance because it does not explicitly describe consistency. Inconsistency may have severe consequences in applications that involve safety.

2.2 Related work on consistency

There is a growing number of studies to improve computer vision, but they do not focus on consistency. These efforts can largely be grouped into two categories: (1) adversarial attacks and (2) synthetic image distortions.

2.2.1 Adversarial attacks

Adversarial attacks present a significant challenge to neural networks. Goodfellow, et al. [14] demonstrate that a well-trained image classifier can be tricked into misclassifying an image by slightly perturbing the values of the pixels. In a typical adversarial attack,

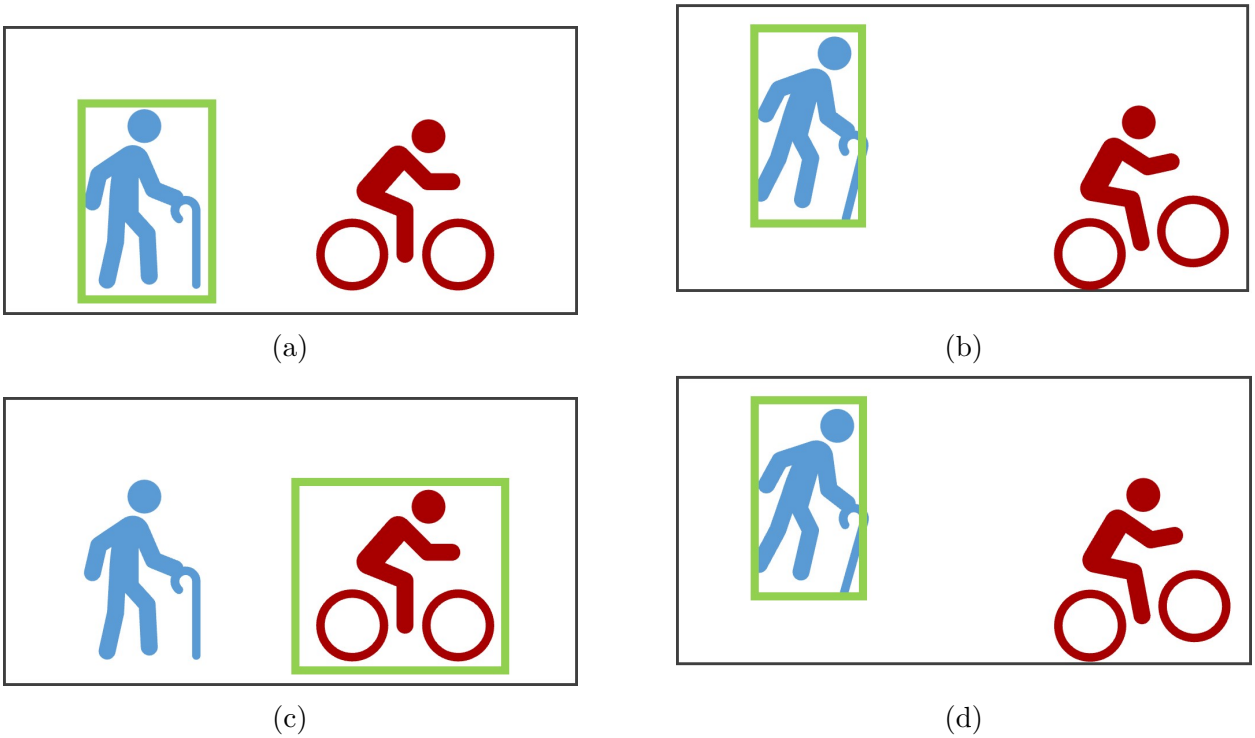


Figure 2.3. Accuracy does not capture consistency. For object detections shown in similar-looking images, (a)-(b) has the same average accuracy (i.e. 1/2 correct) as (c)-(d). (a)-(b) has consistent detections (same object detected in both images), but (c)-(d) is inconsistent (different objects detected in both images).

an algorithm makes minute, calculated adjustments to the pixels of a correctly predicted image until the neural network makes an incorrect prediction. The final image, known as the adversarial sample, appears to human eyes as very similar to the original image. Several different adversarial samples are shown in Figure 2.4.

Existing methods to defend against adversarial attacks often involve some combination of (1) including adversarial samples during network training [14], (2) transforming input images into a lower-dimensional space before feeding the neural networks [17], and (3) filtering adversarial samples through a custom neural network before they reach the main network [18].

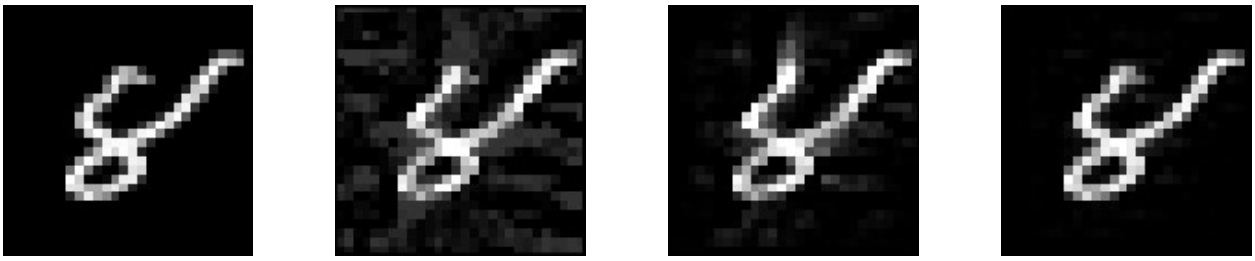
2.2.2 Synthetic image distortions and data augmentation

Computer vision models can make incorrect predictions on images from natural sources (i.e., not manipulated for adversarial attacks) [19]. Two images of the same scene can be captured by the same camera less than a second apart, yet the predictions of a neural network on those two visually similar images can differ dramatically. The small differences between the two images are called *image distortions*, and they can be caused by a range of factors, including ambient light level and camera sensor noise [5].

To make neural networks more robust against image distortions, data augmentation is widely used. The networks are trained on datasets that are modified, or synthetically distorted, to emulate the natural distortions. Common synthetic image distortions (shown in Figure 2.5) include perturbing the pixels with Gaussian noise, adding artificial motion blur, adjusting color saturation and brightness, and even adding computer-generated fog, snow, and rain. [19]

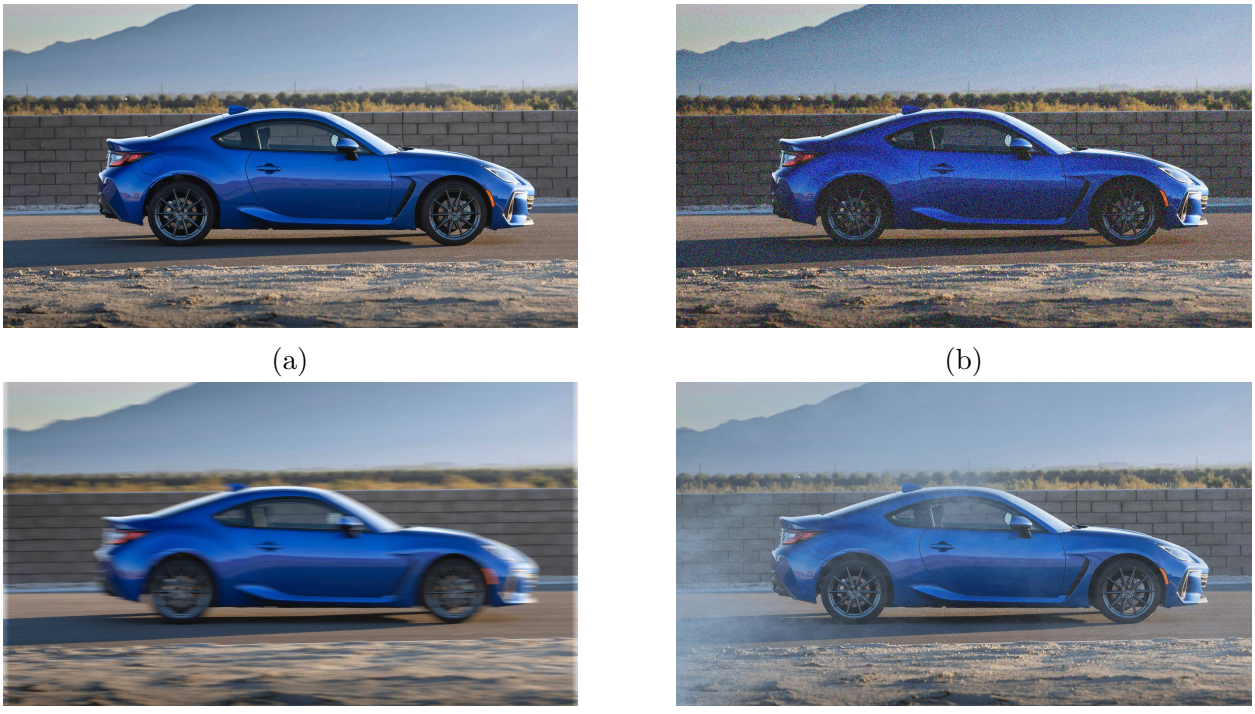
2.2.3 Quantifying image similarity

This dissertation emphasizes the importance of using similar-looking images to test for consistency. We use consecutive frames from videos in the MOT Challenge because such frames are taken up to 30 times per second, ensuring that adjacent frames look similar.



(a) (b) (c) (d)

Figure 2.4. Similar-looking adversarial samples generated from an image in the MNIST handwriting dataset. (a) Original image. Correctly classified as 8 with 0.9 confidence. (b) Fast-Gradient Sign Method adversarial sample. Wrongly classified as 4 with 0.9 confidence [14]. (c) Jacobian Saliency Map Attack adversarial sample. Wrongly classified as 9 with 0.6 confidence [15]. (d) DeepFool adversarial sample. Wrongly classified as 4 with 0.83 confidence [16].



(a) (b)

(c) (d)

Figure 2.5. Examples of synthetic image distortions. (a) Original image. (b) Increased brightness. (c) Motion blur. (d) Artificial fog.

Beyond the scope of this dissertation, other popular image similarity measurement methods attempt to relate two images beyond raw pixel value differences. The Structural Similarity Index (SSIM) [20] identifies structural details about the images and then compares the details. This means that SSIM identifies a noisy version of an image as similar to the original, even though raw pixel values are very different. Beyond SSIM, techniques such as the Feature Similarity Index (FSIM) [21] and deep-learning driven comparison extract low-level features to better approximate the way humans compare images.

2.2.4 IoU and non-maximum suppression

IoU (Intersection-over-Union) is a common measurement in object detection, used to determine if two bounding boxes overlap sufficiently to be counted as the same object. It is calculated by dividing the area of two bounding boxes overlap by the area of the union of the two boxes. If IoU = 1, then the boxes perfectly overlap. If IoU = 0, the boxes have no overlap. In literature, an IoU threshold of 0.5 is typically used [5] to decide if two bounding boxes sufficiently overlap.

Non-maximum suppression is a common application of IoU to filter an object detectors predictions so that only the best ones remain. It finds all overlapping predicted bounding boxes (determined by the IoU threshold) and then filters out the ones that have the same object class and lower confidence scores.

The discussed previous work uses only accuracy as the metrics, without explicitly measuring consistency. Although some publications [22], [23] explore ways to measure the robustness of a neural network beyond accuracy, they neither concretely define consistency nor propose solutions. This dissertation aims to do both.

2.3 Considerations when measuring consistency

In this dissertation, we define consistency as a metric of how differently an object detector behaves on similar images:

If images appear similar to the human eye, an object detector should consistently detect the same objects.

(a) (b) (c)

Figure 2.6. Consistency is only meaningful when measured across images that look similar (a), (b) (where (b) is taken a second later and camera autofocus is slightly blurred). Consistency is meaningless on images that look different (a), (c). Popular datasets use images that look different, making them appropriate for measuring accuracy, but less suited for measuring consistency.

Since accuracy does not always quantify consistency, we now discuss the considerations needed to properly capture consistency.

2.3.1 Use Video/Time-Series Data: consistency measurements require similar test images

A meaningful metric for consistency should use input images that are already consistent. Consider Figure 2.6a, b. Those pictures appear similar to the human eye, so we would expect consistent performance from an object detector. However, if the images are significantly different (Figure 2.6a, c), it is meaningless to use them to make claims about consistency.

As Tung, et al. [5] observe, popular image datasets (e.g., ImageNet, Microsoft COCO) are filled with visually dissimilar images. Thus, those datasets are largely unsuitable for consistency testing. Instead, those authors recommend consecutive frames from videos as a better source of visually similar images. Although adversarial attacks and artificial image transformations can also be used to generate consistency test data from such popular datasets, Gu, et al. [23] report that such techniques do not well represent image distortions that would occur naturally. Instead, this dissertation uses consecutive frames from video to test consistency - this way, any inconsistencies would be caused by natural image distortions.

2.3.2 Use MOT Ground Truth: Consistency measurements need additional labels

When benchmarking an object detector, bounding box and class ground truth labels are sufficient to report accuracy, but they cannot reveal all inconsistencies. In particular, that ground truth cannot show whether the same objects were detected between two similar images (the above Figure 2.3c-d problem): the ground truth is identifier-agnostic. Thus, we need a way to check whether objects from two images are the same, so that consistency can be measured.

We choose to use per-image object identifier (Object ID) ground truth labels to keep track of objects during measurement. Each unique object is assigned the same Object ID across the dataset.

2.3.3 Requirements for Consistency Test Data

In summary, object detection consistency test data has certain additional requirements beyond that of typical accuracy test data. Testing object detection accuracy merely requires ground truth labels. Testing consistency requires a series of images that (1) are visually similar to each other and (2) contain the same objects (and relevant Object ID labels). This is why we use the MOT Challenge datasets; the video datasets inside are high-framerate, ensuring that adjacent frames will be visually similar, while the object tracking labels allow us to identify the same object across frames.

Note: Measuring consistency can be done on adversarial attack datasets as well, since images are similar (slight noise injected) and contain the same objects. However, because we wish to test scenarios that occur naturally, we use MOT Challenge videos for our experiments.

2.4 Our method of measuring consistency

Based on our prior discussion of accuracy vs. consistency, we present a method that specifically tracks whether an object detector detects the same objects, given visually similar, time-series images. Our source of visually similar images is the Multi-Object Tracking (MOT) Challenge [8], consisting of high-quality videos from various datasets.

The *pairwise consistency* $C_{i,j}$ refers to the object detector's consistency on a pair of images I_i and I_j. It is calculated as shown in Equation 2.4. If the detector is perfectly consistent, $C_{i,j}$ is 1. If it is entirely inconsistent, then $C_{i,j}$ is 0. As shown in Figure 2.7, consistency looks to capture whether objects were detected in one image and missed in another.

$$C_{i,j} = \frac{|G_i \cap G_j| - |M_{i,j}| - |M_{j,i}|}{|G_i \cap G_j|} \tag{2.4}$$

Equation 2.4 is explained using the example in Figure 2.8. Figure 2.8a is image I_i, and Figure 2.8b is image I_j. G_i is the set of I_i's ground truth Object IDs {A, B, C}, and G_j is the set of I_j's ground truth Object IDs {D, A, B}. $M_{i,j}, M_{j,i}$ captures the objects that were inconsistently detected as follows: $M_{i,j}$ is the set of ground truth Object IDs that satisfy the following conditions: (1) the ground truth box is present in both images I_i, I_j (i.e. in $G_i \cap G_j$), (2) the object detector detected the object in frame I_i, and (3) the object detector missed the object in frame I_j. So $M_{i,j}$ is object B, while $M_{j,i}$ is empty.

If an object is present in both images, yet is detected in only one of the images, than consistency should decrease. Taken together, $M_{i,j}, M_{j,i}$ captures consistency decreases in I_i, I_j. This also implies that if both $|M_{i,j}|, |M_{j,i}|$=0, the detector can be said to be consistent.

Bounding boxes predicted by the object detector are eligible for consideration in $M_{i,j}, M_{j,i}$ calculations only after being filtered through non-maximum suppression (we use the common IoU threshold = 0.5) and a confidence threshold of 0.7 (see sidebar: "IoU and non-maximum suppression").

We measure consistency across a given video V with N frames by measuring pairwise consistency between each pair of adjacent frames in the video, and then averaging the results across all $N-1$ pairs. This is expressed in Equation 2.5.

$$C_V = \frac{1}{N-1} \sum_{i=1}^{N-1} C_{i,i+1} \tag{2.5}$$

As shown earlier in Figure 2.7, consistency and accuracy can work together to supply more information about object detection performance than either could on its own. Improving accuracy does not necessarily imply improved consistency and vice versa. Colloquially, one might say that consistency indicates how similarly an object detector behaves on two

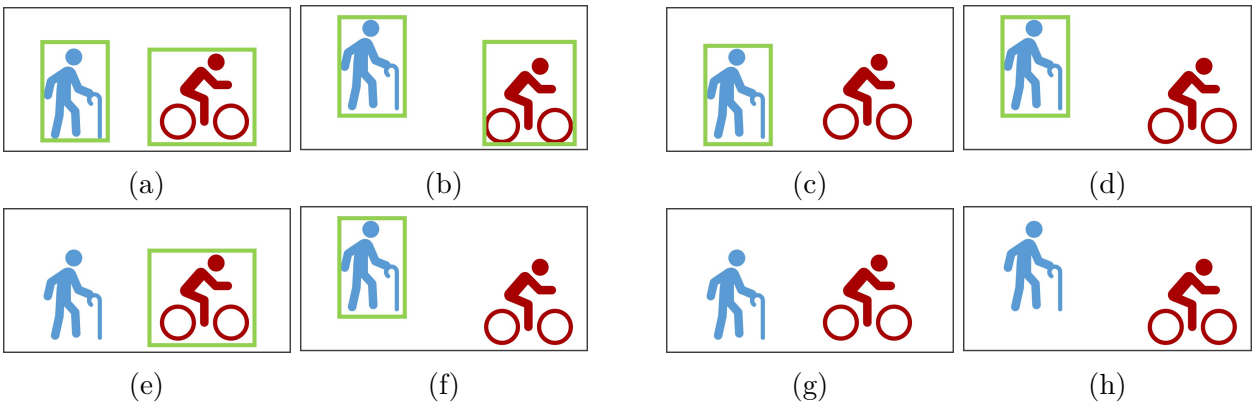
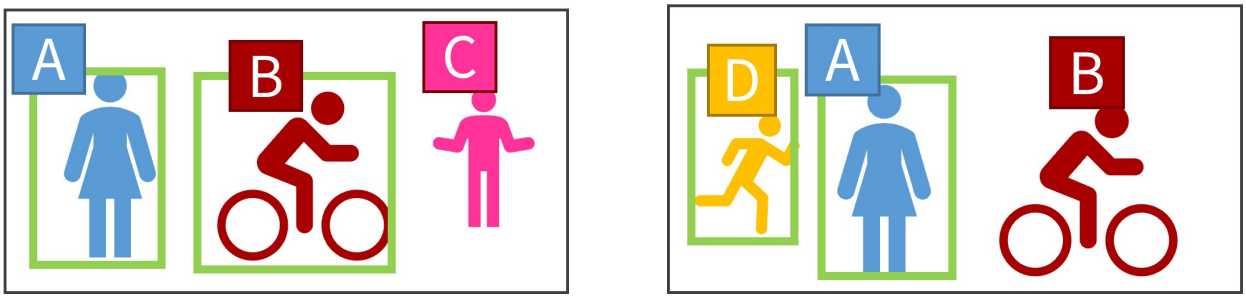


Figure 2.7. Consistency tracks whether an object is detected in one image and missed in another similar-looking image. This complements accuracy measurements. (a)-(b) 100% accurate (all objects correctly detected in both images), 100% consistent (nothing was missed in one image and detected in another). (c)-(d) 50% accurate (only one of two objects detected in each image), 100% consistent (nothing was missed in one image and detected in another). (e)-(f) 50% accurate (only one of two objects detected in each image), 0% consistent (any objects detected in one image was missed in the other). (g)-(h) 0% accurate (nothing detected), 100% consistent (nothing was missed in one image and detected in another). Improving accuracy does not necessarily imply improving consistency and vice versa.



(a) (b)

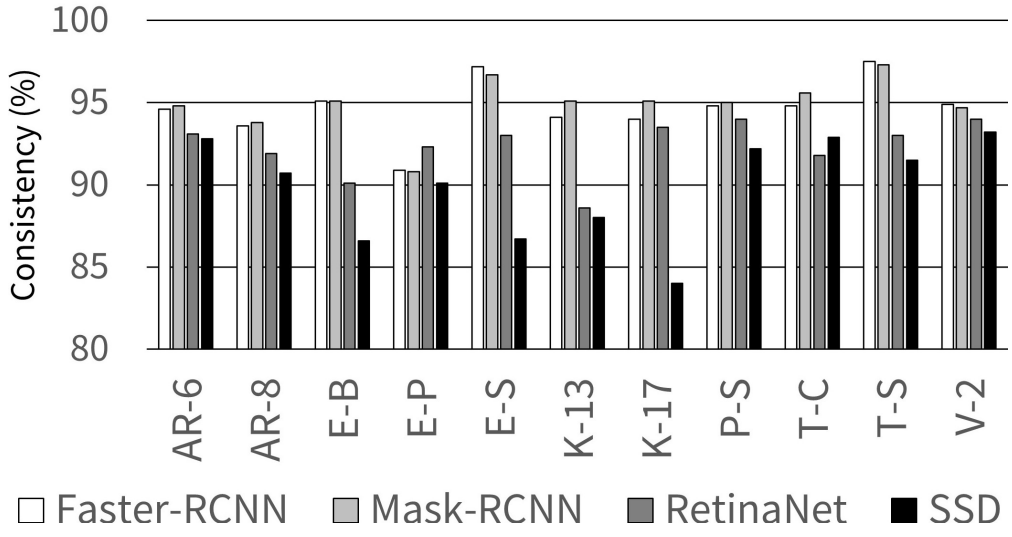
Figure 2.8. Visualization of Equation 2.4, where (a) is I_i, (b) is I_j and the green boxes indicate object detections. $G_i \cap G_j$ contains objects A and B since they appear in both I_i, I_j. Objects C and D are not included in calculations because they do not appear in both images. Because object B is detected in I_i but missed in I_j, so $M_{i,j}$ contains object B. No shared boxes are detected in I_j and missed in I_i, so $M_{j,i} = \emptyset$. Thus, consistency $C_{i,j} = (2 - 1 - 0)/2 = 0.5$.

similar images, while accuracy indicates whether that behavior is desirable (detects everything consistently) or undesirable (misses everything consistently).

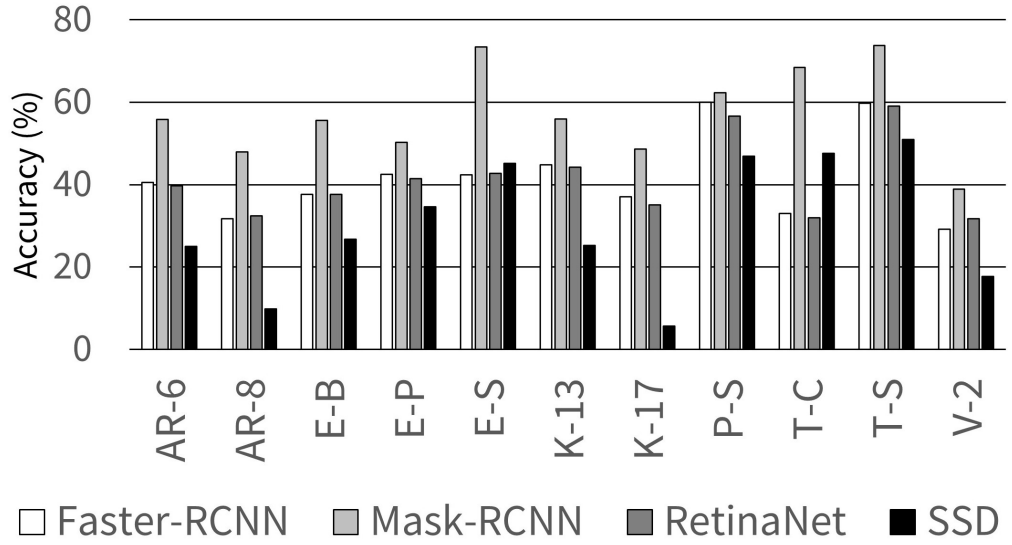
Finally, note that we choose to compare bounding box object IDs instead of output feature maps to calculate consistency. This is because comparing feature maps requires arbitrary similarity metrics - selecting an appropriate metric is itself an open problem. Additionally, bounding boxes are already used for accuracy measurements; re-using boxes for consistency will make it more convenient for researchers to take consistency measurements.

2.5 Measuring the consistency of modern object detectors

We present the consistency of several state-of-the-art object detectors, showing that they exhibit inconsistent behavior. We demonstrate using highly accurate two-stage object detectors (Faster-RCNN and Mask-RCNN) as well as the faster, but less accurate, single-shot detectors (RetinaNet and SSD). We use Facebooks official, pretrained models from their `torchvision` Python package. Measurements are taken on the videos found in the MOT Challenge.



(a)



(b)

Figure 2.9. Object detector consistency (our method, (a)) and accuracy (mAP, (b)) measured on the different videos in the MOT Challenge. Videos are: AR-6: ADL-Rundle-6, AR-8: ADL-Rundle-8, E-B: ETH-Bahnhof, E-P: ETH-Pedcross2, E-S: ETH-Sunnyday, K-13: KITTI-13, K-17: KITTI-17, P-S: PETS09-S2L1, T-C: TUD-Campus, T-S: TUD-Stadtmitte, V-2: Venice-2. State-of-the-art object detectors exhibit inconsistent behavior, ranging from 83.2% to 97.1% consistency. The two-stage models are both more consistent (a) and more accurate (b) than their single-shot counterparts.

As shown in Figure 2.9, all object detectors exhibit some inconsistent behavior, ranging from 83.2% to 97.1% consistency (C_V as calculated in Equation 2.5). We also see that the two-stage models are more accurate and more consistent.

2.6 Our work to improve consistency

Object detector inconsistency is caused by the detector missing an object. As described in Related Work, missed detections can be caused by adversarial attacks and image distortions. Because the MOT Challenge is not an adversarial dataset, we expect the inconsistencies to be caused by image distortions naturally present in the dataset. Therefore, we apply different image distortion corrections from literature to compare their efficacies at improving consistency.

Common training techniques such as *data augmentation* and *dropout* are known to improve a model's robustness to image transformations and distortions. Despite these techniques being employed to produce our pretrained models, we find that inconsistencies still persist. To improve consistency, we use post-training image distortion corrections because of their accessibility. However, emerging training methods appear promising for improving consistency. Zhang et al. propose weakly-supervised, context-based techniques [24], [25] that gather context for the scene (e.g., optical flow and prior knowledge) to provide additional information to an object detector - this information could stabilize the detector and improve consistency over time-series data from videos. Other techniques project neural network features into low-dimensional representations during training [17] - this could improve consistency on images of similar objects taken from different angles, lighting, etc.

1) Gaussian Denoise (GD). Random sensor noise and shot noise can decrease detection performance. As demonstrated by Kang et al. [26], we apply a normalized Gaussian filter to all images in the dataset in an attempt to reduce the noise.

2) Horizontal Flip (HF). Zhang, et al. [22] note that the slight translation of an object in an image could cause a previously misdetected object to become correctly detected. Further, Yin, et al. [27] observe that horizontally flipping an image can mitigate the detrimental effects of noise on a neural network. Thus, we horizontally flip all images.

3) WEBP Compression (WC). Compressing an image using the lossy JPEG format is already known to defeat adversarial attacks [19]. Yin, et al. [27] find that because WEBP compression introduced loop filtering, it is even better suited to breaking down the structures in an image that result from synthetic image distortions. Thus, we apply WEBP compression to the dataset images using a compression quality factor of 30 (Yin, et al. find that lower quality factors allow neural networks to perform better).

4) Unsharp Mask (UM). Tung, et al. [5] explain that if an object is moving, the motion blur can make it harder for a network to extract features along the objects edges. The Unsharp Mask is a linear image processing technique commonly used to remove blur. The technique first identifies a set of blurry details by subtracting a further-blurred image from the original. The details are then emphasized in the original image. We apply the Unsharp Mask to the dataset to reduce the blur on object edges.

5) Gamma Correction (GC). Yeu, et al. [28] show that artificially increasing an images contrast and perceived brightness can help object detectors like Faster-RCNN perform better, particularly when the detectors are trained on daytime images. Gamma Correction is a common brighten/contrast technique that is driven by the Power Law expression; we use it on the dataset as well.

Table 2.2. Avg. Consistency Improvements

	Faster-RCNN	Mask-RCNN	RetinaNet	SSD
GD	0%	-0.3%	0%	-0.6%
HF	-5.3%	-5.4%	-7.3%	-10.1%
WC	**0.6%**	**0.5%**	**0.7%**	**0.4%**
UM	**3.6%**	**2.6%**	**3.0%**	**1.1%**
WC+UM	**5.1%**	**3.0%**	**3.2%**	**1.3%**
GC	0.1%	0.1%	0.4%	0.1%

Table 2.3. Avg. Accuracy Improvements

	Faster-RCNN	Mask-RCNN	RetinaNet	SSD
GD	2.1%	2.4%	-0.6%	-1.1%
HF	-19.3%	-19.4%	-25.5%	-28.4%
WC	**1.5%**	**1.8%**	**0.5%**	**0.5%**
UM	**2.0%**	**3.2%**	**8.3%**	**3.6%**
WC+UM	**3.2%**	**4.1%**	**8.6%**	**3.9%**
GC	0.1%	-0.5%	-0.7%	-0.1%

Table 2.2 shows the average improvement across the MOT Challenge in terms of consistency percentage points (i.e. a table entry of Y% means that it raises consistency from X% to X+Y%), when the different distortion corrections are applied. Similarly, Table 2.3 shows the accuracy improvement.

We see that both WEBP Compression (WC) and Unsharp Mask (UM) improve both consistency and accuracy for all object detectors. Applying both effects at the same time (WC+UM) gives a further overall improvement. In fact, the example inconsistencies in Figure 2.1 and 2.2 are resolved using WC+UM. Gaussian Denoise (GD) and Horizontal Flip (HF) both degrade consistency and accuracy (likely because applying these effects on relatively un-noisy images degrades the feature structure of the images [21]).

Finally, we note that improvements in consistency do not always equate to improvements in accuracy. Gamma Correction (GC) improves consistency, but degrades accuracy: in other words, the detector is consistently worse on the GC data, as described earlier in Figure 2.7.

2.7 Dissertation Contributions on Consistency

Object detectors are vital to many modern computer vision applications. However, even state-of-the-art object detectors exhibit inconsistent behavior when the input undergoes small changes. This inconsistent behavior is not fully captured by existing measurement tools; accuracy metrics and popular image datasets cannot measure whether the same objects are detected consistently. We devise a consistency measurement method that uses images from videos and object ID labels. Our method compliments accuracy measurement. Using this method, we show that object detectors have consistency ranging from 83.2% up to 97.1%, depending on the input data. Additionally, applying image distortion corrections like WEBP Compression and Unsharp Masking can improve consistency by as much as 5.1%. There is still room for improvement by the community. We only explore post-training methods to raise consistency. Any future exploration should explore training-aware consistency improvements and further investigate the relationship between accuracy and consistency.

3. FOCUSED CONVOLUTIONS

Computer vision is often performed using Convolutional Neural Networks (CNNs). CNNs are compute-intensive and challenging to deploy on power-contrained systems such as mobile and Internet-of-Things (IoT) devices. CNNs are compute-intensive because they indiscriminately compute many features on all pixels of the input image. We observe that, given a computer vision task, images often contain pixels that are irrelevant to the task. For example, if the task is looking for cars, pixels in the sky are not very useful. Therefore, we propose that a CNN be modified to only operate on relevant pixels to save computation and energy. We propose a method to study three popular computer vision datasets, finding that 48% of pixels are irrelevant. We also propose the focused convolution to modify a CNN's convolutional layers to reject the pixels that are marked irrelevant. On an embedded device, we observe no loss in accuracy, while inference latency, energy consumption, and multiply-accumulate count are all reduced by about 45%.

3.1 Introduction to Focused Convolutions

Convolutional Neural Networks (CNNs) are known for their high accuracy at many computer vision tasks. However, this accuracy comes at a cost: CNNs are compute-intensive. ResNet, a popular computer vision CNN for performing the relatively simple task of image classification, needs to compute nearly 30 million parameters across all the pixels in the input image [29]. This high compute requirement is typically satisfied by running the CNN on powerful Graphics Processing Units (GPUs) or other hardware accelerators. However, *low-power systems* (i.e., Internet-of-Things (IoT), mobile, and embedded devices) often impose power and memory constraints that make it challenging to deploy CNNs on them [29].

To lessen the computation of a CNN on low-power systems, many methods opt to reduce the sheer magnitude of CNN parameters. These include *pruning* [30] and *quantization* [31] to cut away redundant parameters or reduce precision, and further include *knowledge distillation* [32] and *neural architecture search* [33] to train small CNNs with fewer parameters.

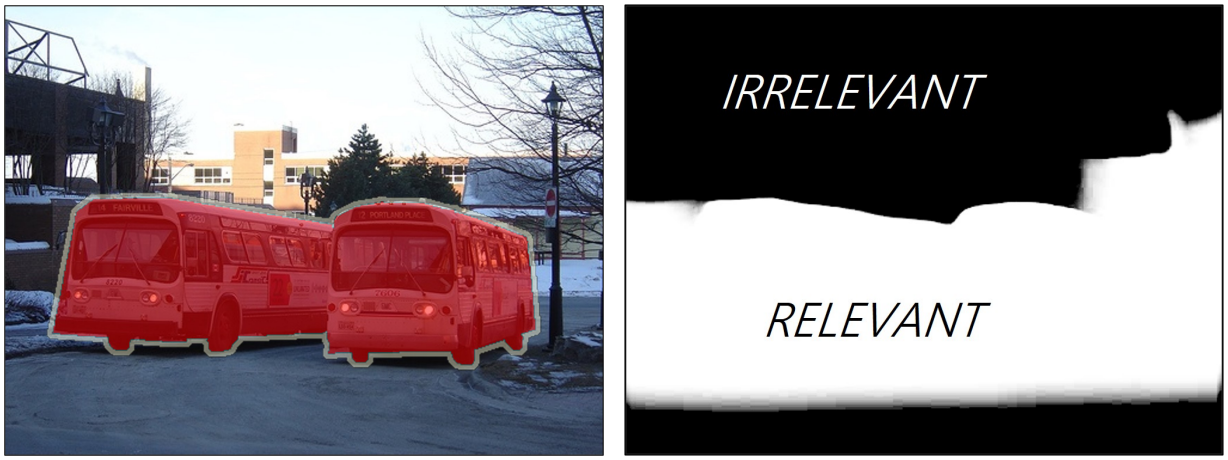
These methods all improve efficiency by changing the computer vision model itself; we instead propose changing the *input* to the model. CNNs operate indiscriminately on every single pixel in the input; therefore, if we reduce the input, we reduce the computation.

In this dissertation, we propose that input images have many pixels that can be deleted, and that by doing so, a CNN would save energy. We confirm this idea by creating methods to (1) identify that three different popular computer vision datasets all contain many such pixels and (2) demonstrate the energy and inference speed improvements possible by using our focused convolution to delete those pixels.

An *irrelevant pixel* (Figure 3.1) (formally defined in section 3.3) is one that is not useful for the computer vision task (e.g., a building's pixels are not useful when looking for cars). We use depth maps to convert ground truth labels into irrelevant pixel maps, showing that in the Microsoft Common Objects in Context (COCO, 164,000+ images) [9], Multi-Object Tracking Challenge (MOT Challenge, 5,500 images) [8], and PASCAL VOC [34] (17,900+ images) datasets, roughly 48% of pixels are irrelevant.

Given the irrelevant pixels, we further experiment to find that explicitly excluding those pixels during inference significantly reduces a CNN's compute expenses, saving energy and speeding inference. Our proposed *focused convolution* technique modifies a CNN such that the model itself can exclude pixels marked as irrelevant while still using the same parameters and General Matrix-Multiplication convolutional technique. Similarly inspired work includes Uber's SBNet and its variations; SBNet does convolution in ResNet-sized blocks and tries to fit the blocks into the relevant pixels [35], [36]. Those other methods still require the model to be changed and re-trained, while ours does not require training. Replacing normal convolution with the focused convolution on the three datasets reduces multiply-accumulate operations, energy consumption, and inference latency by about 45% in two popular CNNs.

This dissertation's contributions: (1) use depth maps to find how many pixels are irrelevant in three popular computer vision datasets, and (2) demonstrate that CNNs can save on inference latency and energy consumption by excluding irrelevant pixels from computation and only performing operations on relevant ones, via our focused convolution.



(a) (b)

Figure 3.1. Example of irrelevant pixels in a PASCAL VOC dataset image, determined using our depth mapping technique. **(a)** The shuttle buses are highlighted in red. **(b)** Our method shows that a significant number of pixels (black) are irrelevant, providing little utility, even though CNNs waste computation on them. Note that pixels deemed relevant (white) using this technique can include additional pixels in the vicinity of the shuttle buses as "contextually related", i.e. they degrade the final predictions of a neural network if they are removed.

3.2 Related Work to Focused Convolutions

This section describes the efforts creating energy-efficient CNNs and prior work about focused convolutions.

3.2.1 Novelty of Our Contributions

We present a novel design to modify any CNN to generate AoIs at *inference time, using its existing training*, so that the convolutional layers can use focused convolutions on the AoI. We design a new focused convolution to use a single AoI across all layers, without needing the initial forward pass previously required. As shown in Table 3.1, our approach exhibits notable improvements over existing work; mainly, we do not require additional training and do not lose accuracy. We test our work using a custom C kernel on different devices, on different datasets (including both image classification and object detection), for different CNNs. Detailed comparison with related work is presented below.

3.2.2 Techniques for Energy-Efficient CNNs

CNNs have high energy demands because of their computational intensity [3] to encode the necessary information for the model to make accurate predictions. Efforts to improve CNN efficiency is typically focused 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) with as little loss to accuracy as possible. We divide existing work into two categories: (1) require training once the method is applied, and (2) do not require training if the method is applied to a pretrained network.

Hardware improvements generally also improve CNN performance. For example, *hardware accelerators* built into CPUs can increase floating-point operation parallelization [37]. *Eight-bit Floating Point (FP8)* is an upcoming hardware standard for representing floating-point numbers with eight bits, instead of the typical 16-, 32-, or 64-bit implementations found in the IEEE Standard [38]. Processors that support FP8 will allow new models to be trained from the ground up or quantized after training using 8-bit floating point numbers and

Table 3.1. Comparison of proposed method with existing work

Existing Work	**Benefits of our Approach**
Accelerators [37], FP8 [38]	Does not use special hardware
Quantization [39], pruning [3], knowledge distillation [3]	No re-training required, no loss of accuracy
Neural Architecture Search [40]	No expensive architecture search required
Hierarchical Neural Networks [41]	Compatible with all CV tasks without re-architecting the CNN
Throttlable Neural Networks [42]	No re-training required, no loss of accuracy
Restructurable Activation Networks [43]	No re-training required, no redesign of CNN architecture
NN-Mass [44]	Quicker search method, no redesign of CNN architecture
Post-Training Quantization [39]	No loss of accuracy

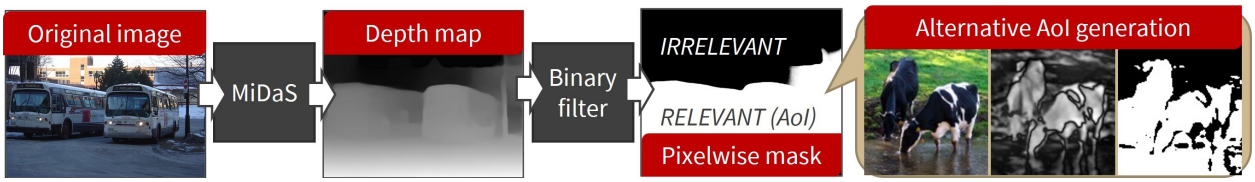


Figure 3.2. Area of Interest (AoI) contains the Relevant pixels for a CNN to make a correct detection on the original image; other pixels are deemed Irrelevant and should be excluded from computation. As illustrated, we [45] generate AoIs using a depth-mapping neural network. This is computationally intensive. Alternative AoI generation methods such as spectral residual saliency [46] and background subtraction [6] are also computationally intensive.

scaling factors. However, this dissertation focuses on software-side improvements, discussed below.

3.2.3 Methods that Require Training

Many methods attempt to improve existing CNN architectures before training so that they can be more energy-efficient without incurring steep accuracy degradation. *Quantization* reduces the size of an existing CNN by reducing a CNN's precision (e.g. storing numbers as 8-bit integers instead of 32-bit floats) [39]. "Quantization-Aware Training" is often used to mitigate accuracy losses that accompany the precision drop [39]. *Pruning* reduces both computation and model sizes by deleting channels/features or neurons that are not activating often (i.e. are not very useful during inference); the smaller model can then be retrained to tune the accuracy [3]. *Knowledge Distillation* uses a large, trained model to "teach" a smaller, more efficient model for deployment [3].

Other approaches build entirely new, energy-efficient CNN architectures. *Neural Architecture Search* treats a search space of CNN architecture hyperparameters (e.g. number of convolutional filters, number of layers, etc.) as differentiable, using a training process to optimize a CNN's design for energy efficiency (EfficientNet is an example) [40]. *Hierarchical Neural Networks* are trained to use small neural networks configured in a tree, instead of a large, monolithic DNN. The root nodes eliminate parts of the search space, and the later

neural network nodes in the tree refine the final prediction [41]. *Throttlable Neural Networks* are trained incrementally such that they can make predictions using only a subset of their convolutional filters, making increasingly accurate predictions the more filters are enabled at inference time. To run a lightweight, less accurate model, more features could be turned off [42]. *Restructurable Activation Networks* use a parameterized ReLU activation function to collapse entire sections of a CNN into a single lightweight, linear operation [43].

3.2.4 Training-Free Methods

Although the above methods have generated such designs as the lightweight, yet accurate EfficientNet – the methods all require training. Training consumes large amounts of energy and resources, sometimes at prohibitively high rates. For instance, EfficientNet uses a cluster of top-end GPUs to perform Neural Architecture Search, and the final architecture further required parameter tuning and 300 epochs of training time [40]. Therefore, some emerging techniques (including the focused convolution) are designed to improve upon an existing model, without additional training.

NN-Mass can predict the learning capacity and training behavior of a neural network without training, thus enabling the optimization of the model architecture for specific hardware constraints [44]. *Post-Training Quantization* can be beneficial in certain contexts as well. Typical static quantization only quantizes the activations at inference time, but dynamic post-training quantization pre-quantizes the weights in advance. This is useful when the model's latency is dominated by loading weights from memory. Static quantization fuses and quantizes the activations in advance, but this requires calibration with a dataset [39].

3.3 Irrelevant Pixels in Datasets

We propose a method to study datasets for irrelevant pixels. We find that irrelevant pixels are commonly found in the COCO, PASCAL VOC, and MOTChallenge datasets.

(a) (b)

Figure 3.3. Ground truth pixels alone **(a)** are not always a sufficient amount of relevant pixels for a CNN to make good predictions (red box is inaccurate, misses second bus); contextual pixels at similar depth levels are also needed **(b)** for CNN accuracy (two good detections).

3.3.1 Counting Irrelevant Pixels with Depth Maps

Our study defines a dataset's relevant pixels as: *all pixels that comprise the dataset ground truth objects and their associated depth levels.* In Figure 3.1a, the ground truth pixels are represented by the red area around the shuttle buses. The final pixelwise map of relevant pixels, shown in Figure 3.1b, is generated using depth level thresholding (explained below) and includes additional pixels.

Ground truth pixels alone do not represent all relevant pixels. A typical CNN uses not only the pixels comprising the object (Figure 3.3a), but also the *pixels from the surrounding area of an object* (Figure 3.3b) to extract features that contextualize and understand the object [47]. To properly include these contextual pixels, our method uses *depth maps*. Shown in Figure 3.4a, an image's depth map represents each pixel in the image with one value, referred to as the pixel's *depth level*, denoting how far away from the camera that pixel is.

Pixels that have similar depth levels to a given object on the ground are known to provide useful, contextual pixels that surround the object [48]. For example, a car will be at a similar depth level as its context: the road on which it drives. Therefore, by thresholding the depth level appropriately, we can capture all the relevant pixels: both the contextual pixels and their associated objects.

For each of our test datasets, we generate depth maps with "MiDaS," a depth estimation model by Ranftl, et al. [49] MiDaS is chosen from among other monocular depth estimation techniques because it is uniquely trained using a diverse mix of 11 datasets totaling nearly 3,000,000 images, instead of training on only one dataset at a time like other depth estimation models.

Our method thresholds the depth map to create a binary, pixelwise mask of relevant and irrelevant pixels, such that the relevant pixel region encapsulates the ground truth (see Figure 3.4). Camera focal lengths tend to cause pictures to be taken at mid-range depth levels [47], so we apply a threshold to an image's MiDaS depth map exactly at the mid-range of its distribution, filtering out pixels at short-range and long-range depth levels. The filtered pixels are considered irrelevant and the remaining pixels are considered relevant. We then record the number of irrelevant VS relevant pixels for each dataset, using this technique.

Simple subset testing on the ground truth indicates that this technique works well – it does not miss much ground truth. Specifically, we check if the relevant pixels resulting from the mask include all pixels inside the ground truth bounding boxes. Occasionally (roughly 5% of the time on our largest dataset, COCO), the threshold causes ground truth to be filtered out (i.e. rejects relevant pixels). When this occurs, we increase the threshold until the ground truth is fully captured (this would not be possible during test time, but is useful for our experiments). Therefore, we can guarantee that the masks used to test our focused convolutions contain 100% of the ground truth objects in all our datasets.

3.3.2 Exploring Datasets for Irrelevant Pixels

In our study, we use our depth map method to count irrelevant pixels in each image in the three datasets, totaling nearly 200,000 images. Results are shown in Figure 3.5. Irrelevant pixels are quite common in the three datasets. On average, 42% of all pixels are irrelevant in PASCAL VOC, 49% are irrelevant in MOT Challenge, and 45% are irrelevant in COCO. This suggests that CNNs are wasting significant amounts of compute resources on embedded devices by computing convolutions on all those irrelevant pixels.

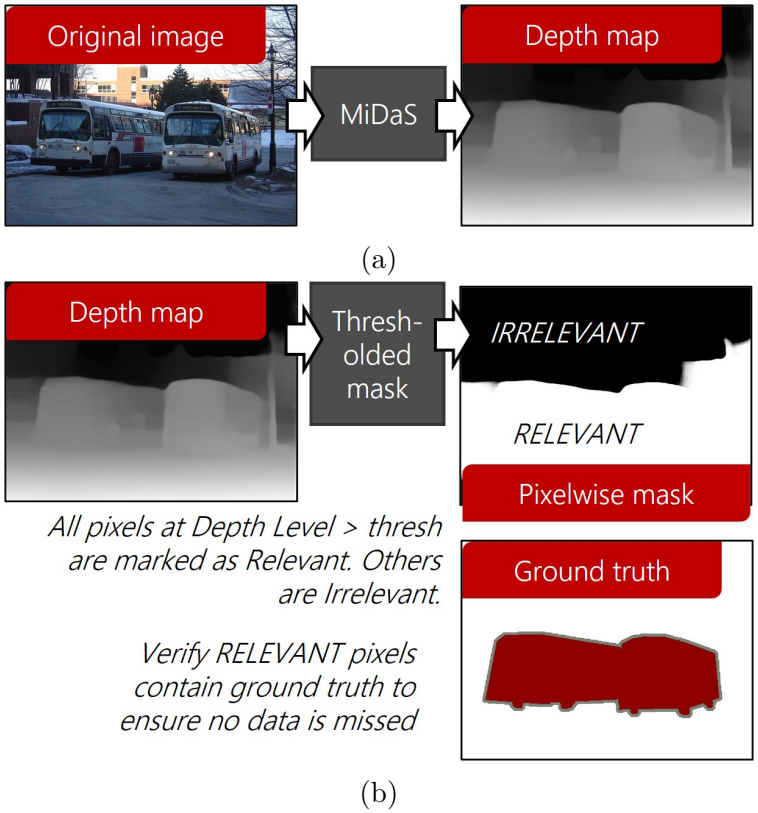


Figure 3.4. This is how to count the number of irrelevant pixels in datasets: First, use MiDaS to generate the image's depth map **(a)**. Next, threshold the depth map, producing a small Relevant Pixels region **(b)**. Verify against ground truth to ensure all important data is captured.

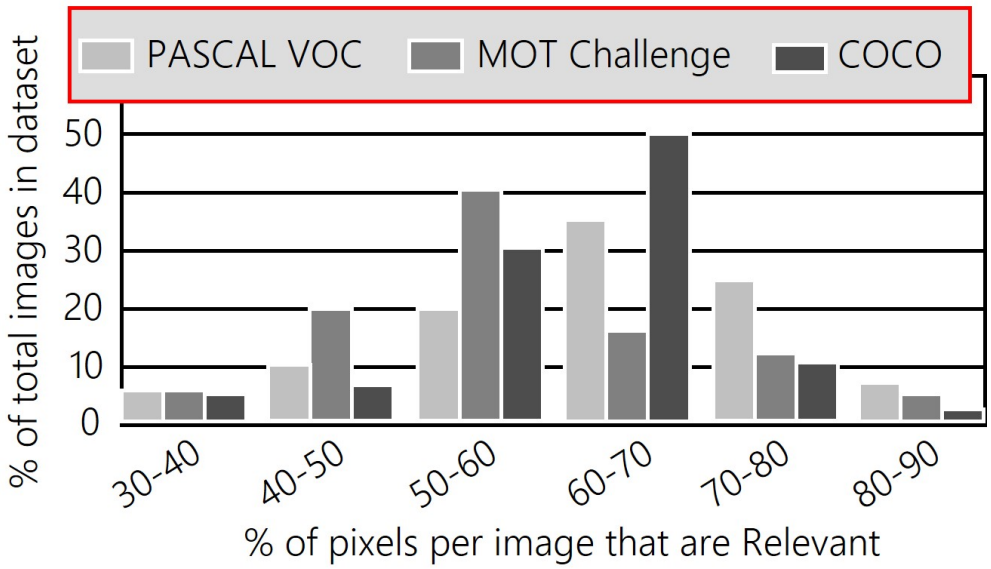


Figure 3.5. Average percentage of pixels that are relevant. Less than 1% of images contain 10-30% or 90-100% useful pixels and are not shown on the graph. As shown, most images contain 50%-60% relevant (40-50% irrelevant pixels). This means that in most images, 40-50% of pixels do not contribute to useful CNN computations and, therefore, can be excluded from processing.

3.4 Excluding Irrelevant Pixels in Datasets

To demonstrate that being able to exclude irrelevant pixels can reduce computer vision energy consumption and inference latency, we propose the *focused convolution* technique. In typical CNN computer vision models, the 2D-convolutional layer is responsible for up to 80% of the computations performed by the neural network [29]. Standard convolution layers are not designed to exclude irrelevant pixels. Instead, they operate on all the input pixels, wasting computation on irrelevant pixels. Because CNNs have multiple convolutional layers in succession, this waste is repeated across multiple layers, compounding the negative impact on the model's energy use and speed. The focused convolution reduces this waste by excluding any pixels marked irrelevant by some assumed oracle - in this case, our relevant VS irrelevant pixelwise masks generated in subsection 3.3.2.

3.4.1 Using Focused Convolutions to Exclude Irrelevant Pixels

Our focused convolution improves the popular General Matrix Multiplication (GEMM) technique [50] for convolutions on computer hardware. The GEMM technique reduces a sequential, sliding-window 2D-convolution to a parallelized, matrix-multiplication operation; matrix multiplications are heavily optimized on modern computers [50]. GEMM (Figure 3.6a) uses a procedure called *im2col* to convert patches of the 3D input tensor (e.g., an RGB image) into the columns of a 2D matrix, and then multiplies the matrix by the convolutional weights to retrieve the results of convolution.

The focused convolution enables the *im2col* procedure to accommodate a pixelwise mask (such as those generated in subsection 3.3.2) indicating whether a pixel is relevant or not. If a patch of pixels is labeled irrelevant, then the modified *im2col* will not convert the patch into a column of the matrix. Thus, those irrelevant pixels get excluded entirely by the matrix multiplication, as shown in Figure 3.6b.

Because the focused convolution is simply designed to accommodate the pixelwise masks as inputs, it does not need to change the model itself. It can be used in any pretrained CNN, requiring no additional training. It can replace existing convolution layers without changing

the weights and biases. If no pixelwise mask is supplied, the focused convolution behaves identically to a standard convolution.

For a given image and convolutional layer, we can count the number of operations needed to complete a GEMM convolution's matrix multiplication. Assume the input has dimensions $B_I \times C_I \times H_I \times W_I$ (batch, channels, height, width), and the convolutional weights are of dimensions $S_W \times S_W$ (side length, side length) with stride s and no padding. A normal *im2col* would convert each channel of the input to $\frac{H_I - S_W}{s} \frac{W_I - S_W}{s}$ columns of $S_W S_W$ pixels. That amounts to multiplying the weights with a $S_W S_W \times C_I \frac{(H_I - S_W)}{s} \frac{W_I - S_W}{s}$ matrix, B_I times. That is a total of $B_I \frac{C_I (H_I - S_W)}{s} \frac{W_I - S_W}{s} S_W S_W$ multiply/add operations.

If we convert the GEMM convolution to a focused convolution, we save matrix multiplication operations by excluding columns of irrelevant pixels. As we discovered in subsection 3.3.2, an average of 48% of pixels are irrelevant in popular datasets. If the region of relevant pixels is a rectangle that takes up $p\%$ of the pixels, then the focused convolution would produce only $p\% \frac{H_I - S_W}{s} \frac{W_I - S_W}{s}$ columns, resulting in a $100 - p\%$ reduction in operations. Therefore, the focused convolution's energy and latency improvements is a function of the linear relationship between number of irrelevant pixels and image size.

We simplify a notable portion of the focused convolution source code, shown below, illustrating how the focused convolution modifies the GEMM process to remove irrelevant patches from outside the AoI during convolution. This results in a matrix multiplication with smaller matrixes.

3.4.2 Evaluating the Compute Savings of Focused Convolutions

We implement and test the focused convolution using PyTorch on a Raspberry Pi 3 (average power 5 W). We compare the focused convolution with a normal PyTorch GEMM convolution. Excluding pixels is beneficial on more powerful hardware, too - we compare the focused convolution with an Intel MKL-optimized convolution [51] on a Intel Core i7 CPU (average power 28 W). For input images, we use the MOT Challenge, COCO, and PASCAL VOC. We exclude the pixels deemed irrelevant by our method from subsection 3.3.1.

Listing 3.1. Focused Convolution forward pass.

```
# Simplified code showing how a Focused Convolution layer works
# Detailed source on GitHub:
# https://github.com/PurdueCAM2Project/focused-convolutions

def FocusedConvolutionForwardPass(aoi_mask_tensor, x, k):
    """
        aoi_mask_tensor: AoI mask
        x: input tensor
        k: convolutional kernel, original weights
        k_unfolded: convolutional kernel, GEMM-unfolded using im2row
    """

    # GEMM unfold procedure using im2row
    # size: 1 X patch length X number of patches
    aoi_patches = gemm_unfold(aoi_mask_tensor, k.shape)

     # size: number of patches
    patch_idxes_to_keep = aoi_patches.sum(dim=1).squeeze(0) > 0

    # size: batchsize X patch length X number of patches
    in_patches = gemm_unfold(x, k.shape)

    # size: batchsize X patch length X number of patches selected
    selected_in_patches = in_patches[:, :, patch_idxes_to_keep]

    # This matrix multiplication only uses the patches inside
    # the AoI to multiply with the weight matrix
    selected_out_mat = matrixmultiply(k_unfolded, selected_in_patches)

    # Completes the GEMM process, reshaping the matrix
    # to a 3D output tensor
    return gemm_fold(selected_out_mat)
```

Table 3.2. By excluding irrelevant pixels from popular datasets (MOTChallenge, COCO, PASCAL VOC), CNNs equipped with our focused convolutions outperform those without (Normal). On an Intel CPU, MKL-optimized convolutions are comparable in performance to focused convolutions.

RPi: Raspberry Pi 3, Intel: Intel Core i7 CPU, ED: EfficientDet, SL: SSD-Lite, MKL: using Intel MKL optimized convolution.

		MOT2015		*COCO*		*PASCAL VOC*	
		ED	*SL*	*ED*	*SL*	*ED*	*SL*
Number of Mult-Add Operations (M/inference)							
	Normal	384.5	483.6	384.5	483.6	384.5	483.6
	Focused	**196.1**	**246.8**	**211.4**	**266.0**	**223.0**	**280.4**
Inference Latency (s/inference)							
RPi (5W)	Normal	2.10	2.26	2.00	2.33	2.06	2.29
	Focused	**1.11**	**1.30**	**1.33**	**1.51**	**1.47**	**1.56**
Intel (28W)	Normal	0.25	0.28	0.25	0.29	0.25	0.28
	MKL	0.18	0.19	0.18	0.20	0.18	0.20
	Focused	**0.17**	**0.18**	**0.18**	**0.20**	**0.19**	**0.20**
Energy Consumption (J/inference)							
RPi (5W)	Normal	10.22	11.80	10.15	11.81	10.20	10.90
	Focused	**5.60**	**6.11**	**6.71**	**7.50**	**7.44**	**7.80**
Intel (28W)	Normal	6.61	7.39	6.45	7.42	6.69	7.81
	MKL	5.18	5.09	5.09	5.61	5.23	5.60
	Focused	**4.76**	**5.04**	**5.10**	**5.60**	**5.29**	**5.62**

Our tests comprise of two popular computer vision CNNs designed for use on low-power devices: EfficientDet [52] and SSD-Lite [53]. We use PyTorch's pretrained weights, and we replace each CNN's convolutional layer with a focused convolutional layer. The pixelwise, binary Relevant-VS-Irrelevant masks we generated in subsection 3.3.2 are propagated through the CNN by scaling the mask's size for each layer.

Table 3.2 summarizes the observed energy consumption/inference latency improvements from the focused convolutions. As shown, we see that equipping a CNN with focused convolution allows it to dramatically reduce its computation expense by excluding irrelevant pixels on images from popular datasets. On Intel CPU, the MKL optimizations allow a fullsize, "wasteful" convolution to operate comparably to the focused convolution, but such optimizations are unavailable on low-power devices, so the focused convolution is preferable.

Finally, the focused convolution-equipped CNN's detection accuracy remains identical to the original CNN. This is reasonable, because we use the same convolutional weights in both the focused convolutions and the normal convolutions, and all ground truth is contained in the relevant pixel masks.

3.5 Dissertation Contributions on Focused Convolutions

We observe that in computer vision, CNNs are compute-heavy because they waste time looking at pixels that are irrelevant. We propose a thresholded depth mapping technique to study modern computer vision datasets for irrelevant pixels. We find that an average of 42%, 49%, and 45% of pixels per image are irrelevant, for three popular datasets (PASCAL VOC, MOT Challenge, and COCO, respectively). We further propose that we can significantly reduce a convolutional layer's multiply-accumulate operations, energy consumption, and inference latency with our focused convolution: this modifies the layer's standard GEMM operations to ignore pixels marked as relevant. Ignoring the irrelevant pixels we computed on the three datasets, we find that multiply-accumulate operations are reduced by an average of 48.3%, energy consumption by 42.7%, and inference latency by 47.4% for two popular object detection CNNs on an embedded device. This dissertation assumes that including all pixels at the same depth level as ground truth gives a complete count of relevant pixels. This may

not be true when a camera's focus is extended (e.g. wide, scenic shots). Still, our technique is demonstrably sufficient for three major datasets. We also assume the focused convolution's pixelwise mask-generating oracle negligibly impacts computational expense. This is reasonable because even computation-intensive oracles can generate reusable masks for fixed cameras (e.g., surveillance) or be offloaded to another device to send masks periodically.

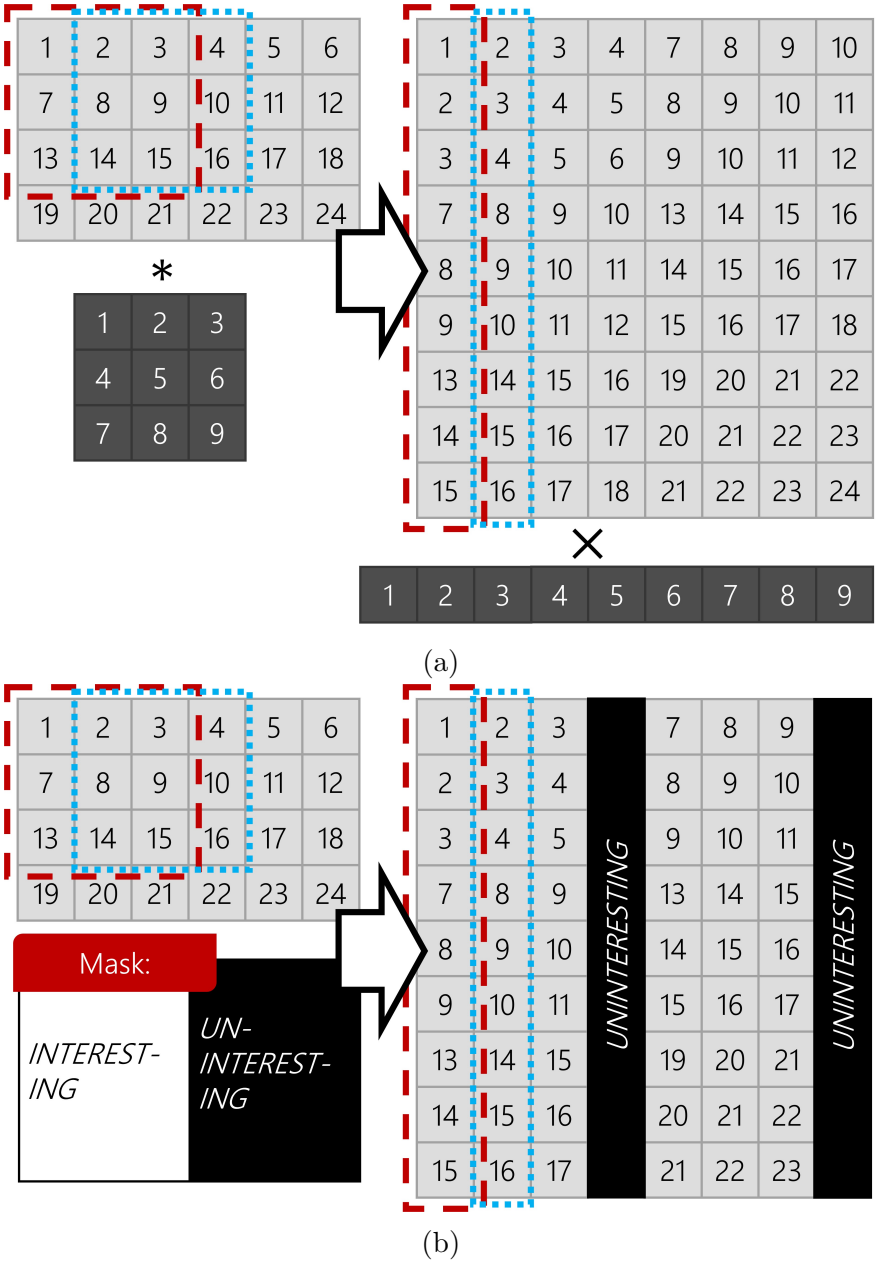


Figure 3.6. **(a)** Traditional GEMM convolution uses the *im2col* procedure to convert a $1 \times 4 \times 6 \times 1$ input tensor (light gray), assuming a stride-1, 3×3 weight kernel (dark gray), to a 9×8 matrix for matrix multiplication. Each 9×1 column of the matrix (e.g., red, blue dashed line) represents one 3×3 patch from the input tensor. **(b)** The proposed *focused convolution* modifies *im2col* exclude patches deemed irrelevant by an oracle-generated pixelwise mask. Thus, the final 9×6 matrix will have fewer columns than that of traditional GEMM, resulting in less computation on embedded devices.

4. AUTOMATED AOI GENERATION FOR FOCUSED CONVOLUTIONS

We note that there are areas for improvement on our previous focused convolution work. The existing focused convolutions have limited use cases; they can only be used in surveillance situations or when the AoI can be reused over time. The depth mapping AoI generation method is computationally intensive, and the focused convolutions require a complete forward pass of the CNN once with the AoI in order to use it for new inference. Therefore, we propose a method that generates an AoI automatically within the CNN itself. We modify an existing CNN to apply a threshold filter to the features from the CNN's early layers. Any features that make it past the filter are considered inside the AoI, and can be operated on by focused convolutions later in the CNN, making the entire modified CNN work with any computer vision use case.

4.1 Introduction to Training-Free, Automated AoI Generation

Modern computer vision models are often associated with ever-increasing energy and compute costs. In pursuit of higher accuracy, researchers have built deeper models, designed to leverage the computation of power-hungry—and sometimes multiple—Graphics Processing Units (GPUs) to train and deploy the Convolutional Neural Networks (CNNs). This approach raises concerns about energy waste [54]: apart from earthly sustainability concerns from the GPUs, we need energy-efficient methods for deployment in environments where a substantial electricity source may not be readily available, such as battery/solar power, mobile processing, and embedded Internet-of-Things systems. This is true for both training and inferencing [4].

This dissertation proposes a method to *reduce energy-consumption of pretrained CNNs without retraining or accuracy degradation.* Our proposed CNN modifications can be applied to any feedforward CNN (tested on popular object detection and image classification networks), by inserting a custom layer. This method improves both theoretical computation (e.g. number of Multiply-Accumulate operations, MACs) and practical hardware utilization

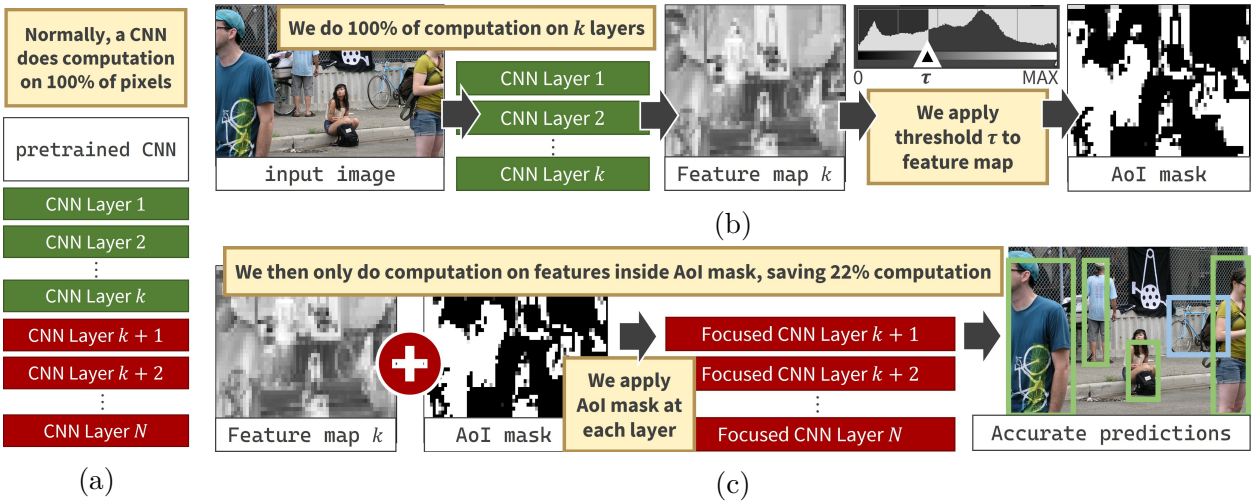


Figure 4.1. **(a)** A pretrained CNN does computation on 100% of the input pixels for all N layers. The proposed method makes the CNN more efficient by: **(b)** First, only do 100% computation during the first k layers to collect contextual features. Second, apply a brightness threshold τ on the feature 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 (subsection 4.3.2) and an accuracy-vs-latency curve search (subsection 4.3.3), respectively. **(c)** Finally, in the last $N - k$ layers, completely ignore the irrelevant regions using focused convolutions (subsection 4.3.4). This saves up to 22% computation for faster, less energy-hungry inference with little to no loss in accuracy (section 4.4).

(e.g., utilization of the processor's vector registers and instructions that can operate Single-Instruction-Multi-Data (SIMD) style), reducing inference latency and energy consumption.

As previously noted, in modern object detection datasets, many pixels were *irrelevant*, i.e. removing these irrelevant pixels did not degrade accuracy. The focused convolution, a modification of a standard General-Matrix-Multiply (GEMM) convolution operation, ignored irrelevant pixels outside an Area of Interest (AoI) as determined by a depth-mapping neural network to save on computation during inference for better inference latency and energy consumption. It is computationally intensive to deploy a depth-mapping neural network for generating an AoI; attempting to deploy that at runtime before a focused convolutional CNN would consume even more energy than the original CNN. Thus, generating an AoI must be done in advance, limiting their approach to such use cases as security camera object detection, in which the AoI will change less often.

Our new approach enables lightweight, automated generation of an AoI within the pre-trained CNN itself at inference time, as shown in Figure 4.1. We reuse features produced by the top layers of the CNN to generate an activation map which can then be manipulated by simple matrix operations and filtered by a threshold to produce an AoI. The threshold is decided using a training-free Accuracy vs. Latency curve search. The remaining layers of the CNN use improved focused convolutions to ignore pixels outside the AoI. Further, our improved focused convolutions employ a system-aware approach to improve compute efficiency. We set the convolutions to operate within a block-size that is a direct multiple of the parallel processing unit register sizes, improving utilization efficiencies. This enables users to make their existing, pre-trained CNNs more energy-efficient with little effort. Because no weights are changed, the modified CNNs retain their original accuracy.

This dissertation presents a method to automatically identify irrelevant pixels in pre-trained CNNs. The method excludes these pixels from computations via focused convolution blocks. We plan to test the proposed technique on multiple popular pretrained models, including ResNet [55], VGG [56], ConvNeXt [1], Faster-RCNN [2], and SSDLite [57]. We will test on ImageNet [58] for image classification and on Microsoft COCO [59] for object detection. We expect to find that our method can reduce a CNN's inference energy on dif-

ferent processor types (Intel, AMD, Arm), without loss of accuracy. This method requires no additional training or manual AoI generation.

4.2 Background on Automated AoI Generation

In this section, we highlight background from the focused convolution work that is relevant to automated AoI generation. It describes our previous work on focused convolutions and removing irrelevant pixels. We highlight a few areas of improvement of our method. This dissertation mitigates those issues by producing an entirely automated approach to focused convolution usage in CNNs.

4.2.1 Similarly Inspired Methods to Our Technique

Some techniques also use the concept of identifying unnecessary computation to skip at inference time. Specifically, they use *contextual information* to determine what computation is useful.

Contextual information is meta-information about the image that helps the CNN make a more informed decision. For instance, knowing that a car is usually found on roads and not underwater can help the system identify the car and ignore regions where it is unlikely to appear. This enables explainable attention mechanisms [60], and is also the inspiration for our approach to generating AoIs. We filter the learned information from the early layers of the CNN, enabling the later layers to make more informed decisions in an efficient manner.

Other techniques in this space include: *BranchyNet* is a custom CNN architecture that decides if the CNN is confident enough to make a prediction before it finishes executing all the layers; it employs this early-exit strategy to great effect on ImageNet [61]. The *Spatially Adaptive Computation (SACT)* model is another type of CNN architecture; it can be trained end-to-end to identify regions of interest and then use special blocks to reduce computation outside those regions [62]. We compare our technique with these methods in section 4.4.

4.2.2 Area of Interest Generation

Irrelevant pixels in an image do not contribute to the CNN's ability to make an accurate prediction. The pixels that are useful to the CNN comprise "Areas of Interest" (AoI) of Relevant Pixels; irrelevant pixels fall outside the AoI. An image can have multiple, disjoint regions that comprise the image's AoI (e.g. one region for cars on the road, another for boats on the water). Our previous method of generating AoIs (see Figure 3.2) uses a computationally intensive approach: we use a neural network called MiDAS [7] to generate a depth map of an input image, and then filter the pixels of the depth map to find an AoI. The method is suitable for stationary cameras (such as surveillance), it is impractical to generate AoIs on a per-image basis (e.g. for drone vision, detection datasets, etc.), because the overhead of AoI generation would outweigh the computational savings from ignoring the irrelevant pixels at inference time. Other AoI generation approaches include background subtraction [6], and spectral residual saliency [46].

We conduct a study to consider the techniques to generate the AoI before CNN inference. As shown in Figure 3.2, AoIs can be generated using techniques like depth mapping, background subtraction, and spectral residual saliency. We then test the generated AoIs with CNNs using focused convolutions to see if accuracy is negatively impacted. We find that depth mapping produces the best AoIs: CNNs achieve the highest accuracy using depth mapping. Unfortunately, the depth mapping approach is also compute-intensive since it requires the use of another neural network like MiDaS. Spectral residual saliency, while much faster, also produces the worst accuracy. Meanwhile, background subtraction techniques often produce AoIs that are too close to the image objects, obscuring contextual pixels needed to help the CNN make its predictions. Therefore, this dissertation uses a different approach: we apply a threshold to the feature maps generated by the CNN's early layers.

4.2.3 Focused Convolutions

The focused convolution is based on the popular General-Matrix Multiply (GEMM) technique for doing convolutions [63]. In GEMM, an input image is segmented into convolution kernel-sized windows called *patches*. Each window is then vectorized into a column of a

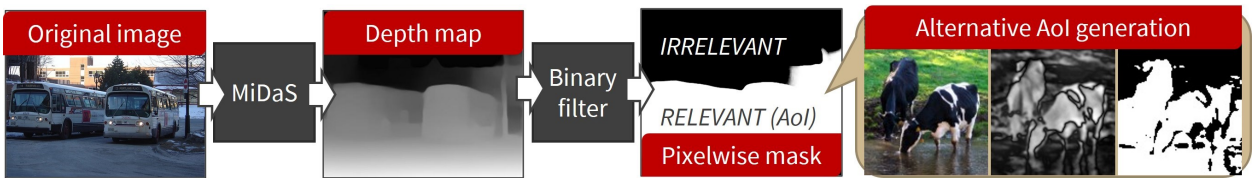


Figure 4.2. Existing methods (illustrated) to generate (Areas of Interest) AoIs are too computationally intensive. AoIs contain the Relevant pixels for a CNN to make a correct detection on the original image; other pixels are deemed Irrelevant and should be excluded from computation. AoIs can be accurately generated using a depth-mapping neural network [45]. This is computationally intensive. Other AoI generation methods like spectral residual saliency [46] and background subtraction [6] are also computationally intensive. *This dissertation inserts a layer into the CNN to filter out irrelevant pixels from computation.*

matrix via a process called *im2col*. That matrix is then multiplied with the kernel of weights to produce the convolution output.

The focused convolution, a drop-in-replacement for a GEMM convolution, applies a provided AoI during this process. Any patches not found inside the AoI are deemed irrelevant and then excluded from the im2col matrix. That results in a smaller matrix and thus a less computation-intense matrix multiplication. An entire forward pass of the network is required to assign the AoI before deployment. However, the AoI not being generated at inference time keeps the CNN from being able to truly replace existing models for any application.

4.3 Proposed Method for Automated AoI Generation and Deployment

This section describes a new method that can identify and remove irrelevant pixels at runtime. The section presents the modified CNN. It reduces energy consumption without loss of accuracy and does not require training. We propose using features already generated within the CNN to determine the AoI. The method is discussed in the following section.

4.3.1 Generating AoIs by Feature Map Filtering

The proposed technique takes a pretrained CNN and then generates a modified version of that CNN – one that can automatically generate AoIs and apply them for focused convolution energy improvements. The new CNN behaves as follows during inference:

1. Process input image using the top n layers (n chosen in Section 4.3.2) 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 4.3.3) to produce the AoI. Activations bright enough to clear the τ threshold are allowed through as corresponding regions of relevant pixels; the rest are discarded. Produces X_{thresh}

4. Maxpool X_{thresh} to break the feature map into blocks that utilize the parallelization hardware on the processor (see Section 4.3.4).

5. All convolutional layers after $NN.top$ use our improved focused convolutions (Section 4.3.4) on the generated AoI, saving energy by discarding irrelevant pixels. The focused convolutions use the same weights and biases as the convolutions they replace.

To generate this modified CNN, we need to choose n and τ, and replace the convolutional layers after the n-th layer with focused convolutions. We refer to such modified CNNs as fCNNs. Our technique chooses n by estimating the amount of energy consumed by the CNN and selecting the n that allows the energy consumption estimate to satisfy a given target (e.g. 10% energy savings). Next, the CNN is iterated a few times over the training dataset with different τ threshold values applied to $NN.top$ and focused convolutions in the remaining layers, generating points on an accuracy-latency tradeoff curve for that CNN. The technique then chooses τ that can satisfy the user's accuracy and latency targets A and T, respectively. This newly configured fCNN can then be deployed for reduced energy consumption. While the concept bears similarities to attention mechanisms in some CNN and vision transformer models, the proposed method is more advantageous because it can modify any CNN without any training. The process is described in detail in the following subsections.

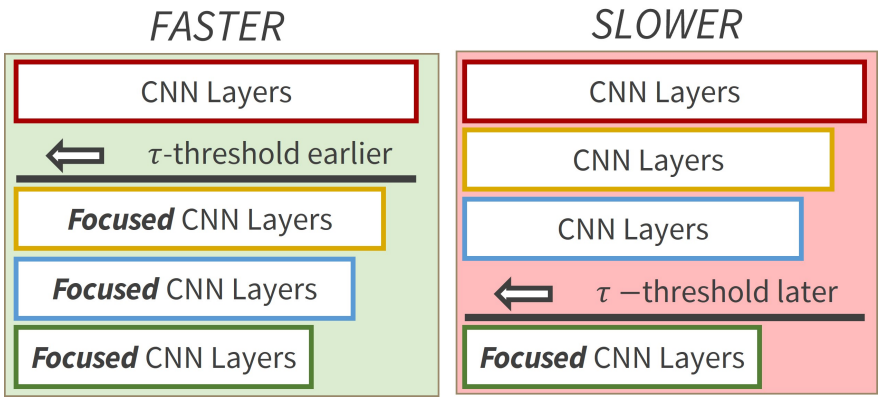


Figure 4.3. The later the τ-threshold is inserted (i.e., the larger the n, the fewer layers can take advantage of Focused convolutions, and the slower and less energy-efficient the CNN. Conversely, a smaller n allows for a faster, more energy-efficient CNN. Therefore, smaller n is generally beneficial.

4.3.2 Choosing the Layers

To convert a CNN into an fCNN, we first choose which layer's output to filter with threshold (i.e., choose n: after the nth layer, the τ-threshold is applied). The smaller n 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 n is beneficial, as illustrated in Figure 4.3.

Meanwhile, n cannot be too small: there is insufficient information encoded within the features to make a useful threshold AoI [3]. 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. The deeper layers are also responsible for spatially downsampling those feature maps [56]. Therefore n must be large enough to capture useful information in the features produced at the nth layer.

To choose n, we use a heuristic: *choose the latest layer (i.e., largest n) that is still small enough such that the resulting fCNN can satisfy the given energy consumption target.*

We now measure the energy consumption of the CNN, and then use that information to project the energy savings of its fCNN equivalent. Focused convolution energy savings were

determined to be approximately linear with respect to AoI size [45]. Therefore, if one has both an expectation of the average AoI size on the data and the known energy consumption of a CNN, they could easily infer whether the possible fCNN energy savings could meet their target.

Let the energy consumption of the ith convolutional layer be $E_{c,\mathrm{i}}$, 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 $E_{c,\mathrm{i}}$). Then, the energy use of the corresponding focused convolutional layer $E_{f,\mathrm{i}}$ is shown in Equation 4.1.

$$E_{f,\mathrm{i}} = aE_{c,\mathrm{i}} + c \tag{4.1}$$

We choose to use this linear estimation of energy because as noted by Yang, et. al [64], the energy use of the memory accesses and computation of a sliding-window convolution operation both scale linearly with respect to input size. We note that other factors such as the hardware platform, memory hierarchy, and optimization techniques employed may impact the runtime characteristics of the neural network, reducing the accuracy of our energy estimation. However, we empirically show in section 4.4 that the energy consumption improvements of our model are linear with respect to AoI size (i.e., effective input size of the layer), so we believe this energy estimation to be reasonable to use.

Thus, for a CNN with N total conv layers, of which n belong to $NN.top$, then there will be $N-n$ focused convolutional layers. Based on Equation 4.1, then, we can model the total energy consumed by the convolutions in the fCNN in Equation 4.2.

$$E_{total} = (N-n)c + \sum_{\mathrm{i}=1}^{n} E_{c,\mathrm{i}} + \sum_{\mathrm{i}=n+1}^{N} aE_{c,\mathrm{i}} \tag{4.2}$$

Thus, n can be selected such that E_{total} satisfies the energy target, selected according to the constraints of the target hardware. To keep n from becoming too small, we fix the lower-bound on n as the index of layer at which the first feature downsampling occurs in the CNN. Conversely, if the energy target cannot be satisfied due to the lower bound, then our technique determines that a suitable fCNN is unachievable.

Selecting an n does not guarantee that it is possible to achieve a model that can achieve a given accuracy target. Adjusting for accuracy uses a separate accuracy-latency curve search approach, described below in subsection 4.3.3.

4.3.3 Choosing the Activation Brightness Threshold

The activation brightness threshold τ determines which pixels are considered relevant inside the AoI and which are deemed irrelevant. Because each CNN is built differently, the threshold will be different for each CNN, even when trained on the same data. Thus, our method refines its τ selection by iterating the fCNN over the training dataset with different values of τ. After each full iteration over the dataset, it measures the average accuracy and inference latency to see if it satisfies a given target. 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 [55].

As shown in Figure 4.4, the proposed technique chooses the activation brightness threshold τ as follows: τ is used to filter the output of $NN.top$, the top 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. 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 $NN.a$. Note that this method is not restricted to only use latency VS accuracy to search. This method can be trivially modified to search energy use VS accuracy instead, for example. A decreasing latency is used in this case as a proxy for decreasing energy, as it is easier to measure latency than it is to instrument one's models for repeated energy measurements.

The proposed technique increases τ by increments of some ϵ, reducing 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

```
class AoiThresholder(nn.Module):
    ...
    def forward(self, x):
        """
        Assigns an AoI mask for remaining focused convolutions to use

        x: input, the features from the previous layers
        """

        # Assign the AoI based on the tau threshold
        # Remaining FocusedConv2d layers reference the `aoi_mask` field
        self.aoi_mask = (torch.sum(x, dim=1).squeeze(0) >= self.threshold)

        # Pass through the features to the remaining layers
        return x
```

Listing 4.1. AoI τ-threshold masking.

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.

Once configured, the τ-threshold is applied to the output of the previous layer to identify an AoI mask for all remaining focused convolution layers to reference, allowing the features to pass through to the remaining layers for focused inference. An example of the source code is as follows:

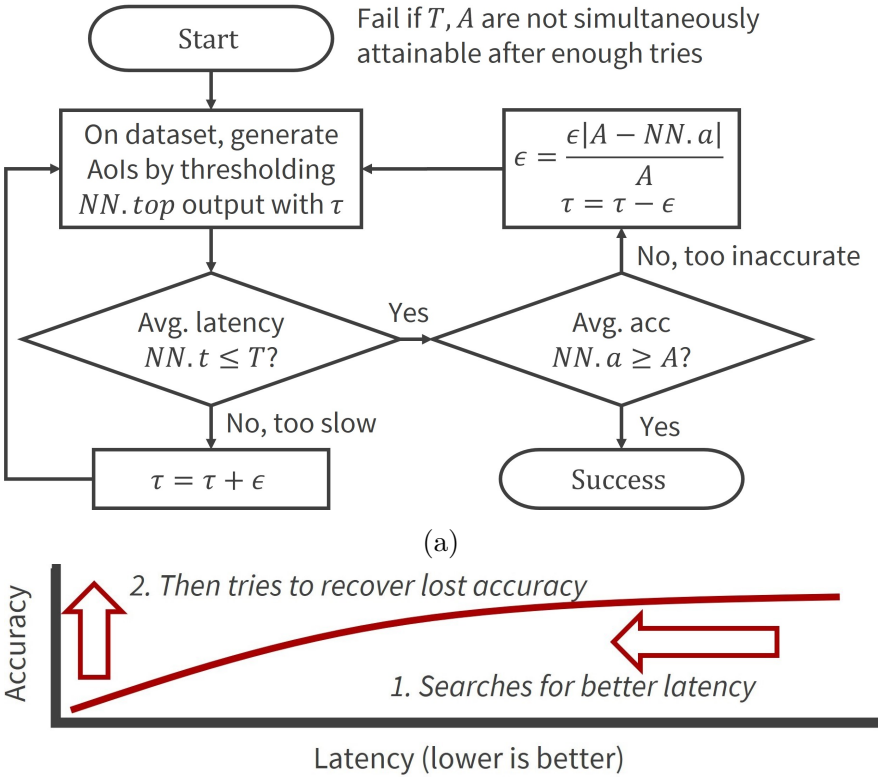
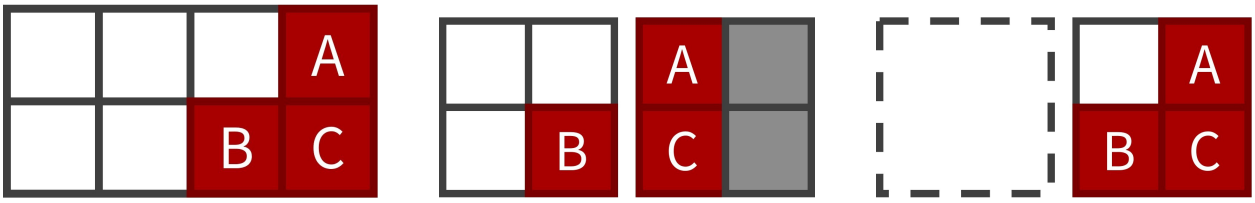


Figure 4.4. (4.4a) Training-free process to choose activation brightness threshold τ, given a maximum inference latency threshold of T and a minimum accuracy threshold of A. This proposed method will succeed if latency and accuracy targets T, A are simultaneously attainable. (4.4b) This technique searches along the accuracy-latency curve, succeeding if (T, A) is on the curve.



(a) (b) (c)

Figure 4.5. Our technique incorporates hardware-parallelization (e.g. Single-Instruction-Multiple-Data). The AoI is shown in red letters for some input data in (4.5a). On hardware that can parallelize the data processing in blocks of 4, the original focused convolution's sliding-window patch selection (4.5b) will result in two blocks of size-4 data to be sent for processing. The proposed technique (4.5c) instead indexes the input data downsampled to multiples of the hardware blocksize, resulting in only one block of size-4 data being sent, thus saving processing on one block. The algorithm is shown in Listing 4.2.

4.3.4 System utilization improvements

Modern computer architectures implement specialized hardware (e.g., "Neon" vector registers on Arm CPUs and "CUDA" cores on NVIDIA GPUs) used to parallelize code that needs to perform the same operation on multiple, independent pieces of data. These parallel operations are supported by Single-Information-Multiple-Data (SIMD) instructions. The data is collated into blocks that perfectly fit the size of a SIMD register on the processor. The processor then uses specialized instructions to process the entire registers data in parallel. For example, a GEMM convolution operating on an entire input tensor could use im2col to send multiple patches for simultaneous processing to improve inference latency. SIMD processing is so useful that many processor designers like Intel, AMD, and Arm all implement such techniques on their chips.

Contemporary machine learning frameworks like PyTorch and TensorFlow already rely on C-native libraries under the hood to use parallel processing for built-in operations like "Conv2d" and "MaxPool2d". The focused convolution already uses these libraries, but 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 4.5: imagine the processor can parallel-process blocks of size 4. The

```
# Here, Python is written sequentially so that it is easier to
# understand.
# In practice, accelerated numpy folding functions are used
def gemm_mask_to_patches(mask, k_side):
    """
    mask: AoI mask
    k_side: kernel side length
    """
    # Find the indices where the matrix is nonzero
    nonzero_indices = torch.nonzero(mask)

    # Block the nonzero elements into squares
    for index in nonzero_indices:
        row, col = index.tolist()
        row_patch_start = (row // k) * k
        col_patch_start = (col // k) * k
        patch_matrix[row_patch_start:row_patch_start + k,
            col_patch_start:col_patch_start + k] =
            mask[row_patch_start:row_patch_start + k,
                col_patch_start:col_patch_start + k]

    return patch_matrix
```

Listing 4.2. Algorithm for selecting GEMM patches based on the AoI mask in our improved focused convolution.

AoI (Figure 4.5a) contains three points of data (A, B, C). The original focused convolution design would use a parallelized sliding-window approach to select data in blocks. As shown, two size-4 blocks get selected (Figure 4.5b). Our technique will only select a single block (Figure 4.5c), thus saving the processing on one block.

To achieve this, we design the focused convolution to use memory alignment. We pre-divide the input into a grid, where each cell is sized 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 or frameworks using optimizations for SIMD, CUDA, etc.

This technique shares similarity with *sparse convolutions* [17]. Sparse convolutions are specifically designed to apply convolutions only on non-zero elements in an input. They do this by storing indexes of non-zero elements in a memory-efficient data structure, and then referencing those inputs during inference. Some form of aggregation then has to be performed at the end in order to produce the final output feature map. This approach tends to require specialized hardware or kernel support, while our approach can be drop-in swapped with standard convolution layers.

The previous focused convolution required a single forward pass of the CNN to scale the AoI correctly, pixel-by-pixel, for the field of perception at each layer before the focused convolutions could be deployed. This is unacceptable overhead for AoI generation at runtime. By using the parallel processing-aware blocks described above, our method can rapidly compute which blocks to keep and which to ignore from a single AoI. Thus, we do not need to pre-scale the AoI for each layer, removing the overhead cost entirely.

4.4 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 de-

Table 4.1. 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).

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**

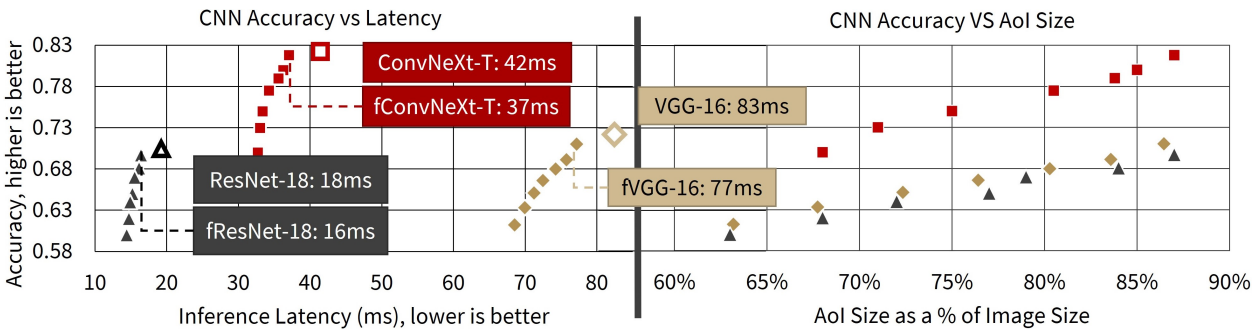


Figure 4.6. For a pretrained CNN equipped with focused convolutions, as we vary the activation brightness threshold τ, different amounts of activations are filtered, causing the CNN's accuracy, latency, and AoI size to change. We designate the CNNs that achieve the best accuracy as "fCNNs". Left: ImageNet accuracy trades off with inference latency. Fortunately, the tradeoff can be little to nonexistent – our fVGG-16 achieves faster inference than VGG-16 without losing accuracy. Right: Larger AoI sizes yield better accuracy. Note that latency is reported from an AMD desktop CPU.

tection 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 below formula:

$$\%\text{difference} = \frac{|\text{unmodified} - \text{focusedconv}|}{\text{unmodified}} \times 100\% \tag{4.3}$$

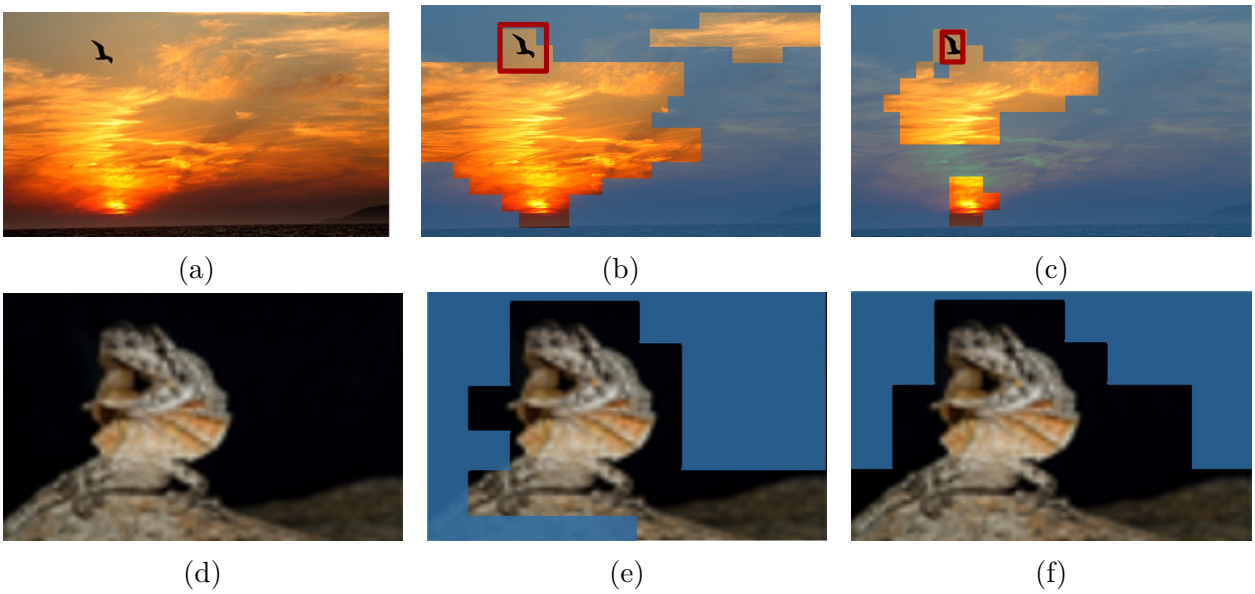


Figure 4.7. The proposed technique can ignore regions of irrelevant pixels (marked in blue in the thresholded saliency maps) from the original images. The resulting Area of Interest (AoI) focuses on parts of the image that the human eye would. **(a)** Original COCO image. **(b)** fFaster-RCNN. **(c)** 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 28% energy on COCO.**

```python
# Source code excerpt from
# https://github.com/PurdueCAM2Project/focused-convolutions/
def focusify_all_conv2d(m: nn.Module, aoi: AoI):
    """
    Recursively locate all Conv2d layers in a model m,
    then assign to use specified AoI as FocusedConv2d
    """
    for child_name in m._modules:
        child_m = m._modules[child_name]
        if type(child_m) == nn.Conv2d:
            in_channels = child_m.in_channels
            out_channels = child_m.out_channels
            kernel_size = child_m.kernel_size
            stride = child_m.stride
            padding = child_m.padding
            dilation = child_m.dilation
            groups = child_m.groups

            # All weights are kept
            new_conv = FocusedConv2d(in_channels, out_channels, kernel_size,
                stride, padding, dilation, groups, aoi=aoi)
            new_conv.weight = copy.deepcopy(child_m.weight)
            new_conv.bias = copy.deepcopy(child_m.bias)

            m._modules[child_name] = new_conv
        else:
            focusify_all_conv2d(child_m, aoi)
```

Listing 4.3. Function to replace any Conv2D layer in the model using Focused Convolution layers.

An unmodified VGG-16 operates on the entire image. We use our technique on this model to identify layer-index 3 as the layer at which to insert the AoI τ-threshold.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), )
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), )
    (3): ReLU(inplace=True)

    # AoI threshold layer will be inserted here
    # Layers 4-30 currently operating on
    # irrelevant pixels
    (4): MaxPool2d(kernel_size=2, stride=2, )
    (5): Conv2d(64, 128, kernel_size=(3, 3), )
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), )
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, )
    (10): Conv2d(128, 256, kernel_size=(3, 3), )
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), )
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), )
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, )
    (17): Conv2d(256, 512, kernel_size=(3, 3), )
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), )
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), )
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, )
    (24): Conv2d(512, 512, kernel_size=(3, 3), )
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), )
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), )
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, )
  )

...)
```

Our modified Focused fVGG-16. Conv2d layers extract features up to layer index 3. The AoI τ-threshold is applied, and remaining layers are FocusedConv2d layers.

```
VGG(
  (features): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), )
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), )
      (3): ReLU(inplace=True)
    )
    (1): AoIThresholder()
    (2): Sequential(
      (4): MaxPool2d(kernel_size=2, stride=2, )
      (5): FocusedConv2d(64, 128, kernel_size=(3, 3), )
      (6): ReLU(inplace=True)
      (7): FocusedConv2d(128, 128, kernel_size=(3, 3), )
      (8): ReLU(inplace=True)
      (9): MaxPool2d(kernel_size=2, stride=2, )
      (10): FocusedConv2d(128, 256, kernel_size=(3, 3), )
      (11): ReLU(inplace=True)
      (12): FocusedConv2d(256, 256, kernel_size=(3, 3), )
      (13): ReLU(inplace=True)
      (14): FocusedConv2d(256, 256, kernel_size=(3, 3), )
      (15): ReLU(inplace=True)
      (16): MaxPool2d(kernel_size=2, stride=2, )
      (17): FocusedConv2d(256, 512, kernel_size=(3, 3), )
      (18): ReLU(inplace=True)
      (19): FocusedConv2d(512, 512, kernel_size=(3, 3), )
      (20): ReLU(inplace=True)
      (21): FocusedConv2d(512, 512, kernel_size=(3, 3), )
      (22): ReLU(inplace=True)
      (23): MaxPool2d(kernel_size=2, stride=2, )
      (24): FocusedConv2d(512, 512, kernel_size=(3, 3), )
      (25): ReLU(inplace=True)
      (26): FocusedConv2d(512, 512, kernel_size=(3, 3), )
      (27): ReLU(inplace=True)
      (28): FocusedConv2d(512, 512, kernel_size=(3, 3), )
      (29): ReLU(inplace=True)
      (30): MaxPool2d(kernel_size=2, stride=2, )
    )
  )
...)
```

4.4.1 Experimental Setup

We test using three devices with different levels of power consumption and different operating systems:

- 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 4.3.4) is set according to the specifications from the processor's documentation.

We measure inference accuracy on ImageNet and Microsoft COCO using the "Torchbench" software. MAC counts are measured using the "ptflops" library. Inference latency is determined using Python's built-in timers. All timing/energy experiments are averaged across the dataset.

4.4.2 Activation Brightness Threshold Selection

We follow the method described in section 4.3 to create fCNNs from the pretrained CNNs. For image classification, we start the automated 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, less pixels are allowed past the threshold, shrinking the size of the AoI. This also causes the model's accuracy to begin dropping linearly; the accuracy-latency curves explored by the search are shown on the left in Figure 4.6. As shown by our method, there are points on the curve at which **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.

4.4.3 Improvements On Pretrained CNNs

We compare our "fCNN" models with their corresponding unmodified pretrained CNNs in Table 4.1. 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.

The qualitative results are positive as well. In Figure 4.7, 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.

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.The accuracy degradation is low relative to the energy improvements.

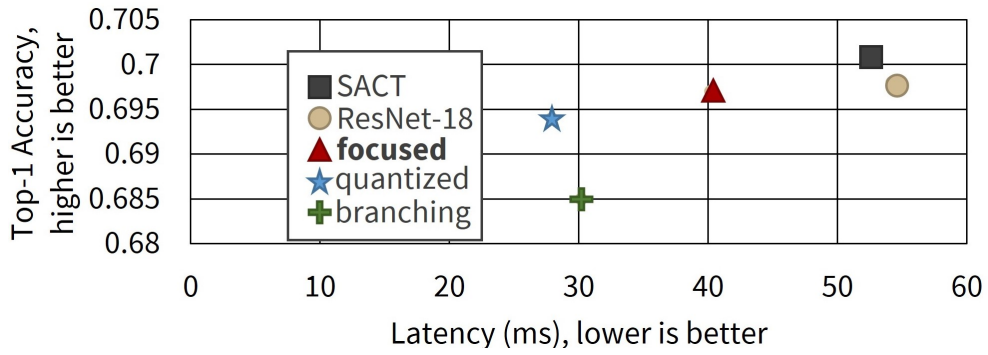


Figure 4.8. Our technique "(focused)" compared with ImageNet state-of-the-art on our Intel CPU. SACT [62] and branching [61] CNNs require training and a complete redesign of the CNN; **our technique not only either beats them or stays competitive, but it also requires zero training, keeping the pretrained CNN intact.** Static-quantized [39] and unmodified ResNet-18 are shown to provide a baseline reference.

4.4.4 Comparison with State-of-the-Art (SOTA)

While not an apples-to-apples comparison since our technique does not require the training that the SOTA methods do, we provide a comparison with similarly inspired techniques (subsection 4.2.1) and an INT8-quantized baseline for ResNet-18. Our focused fResNet-18 is both faster and more accurate than BranchyNet [61], is faster than the standard ResNet-18 [55], is more accurate than quantized ResNet-18 [39], and is faster than SACT [62]. In short, our technique either outperforms or stays competitive with the SOTA techniques (Figure 4.8), all while being easy to implement because it requires no training.

4.4.5 Saliency Mapping and Explainability

A Pretreained CNN's training naturally causes it to focus more heavily on certain regions of a given image. For example, a CNN trained to detect people might have its predictions more heavily weighed based on regions of the image that looked like a head. Those regions, then, contain the most relevant information for producing an accurate prediction, and it would behoove our technique to keep those regions during the threshold process.

For a given image, those relevant regions can be identified via saliency mapping techniques like Grad-CAM (Gradient-weighted Class Activation Mapping) [65]. After inference on the image, Grad-CAM generates a heatmap of the pretrained CNN's strongest activations, weighted by gradients from the last convolutional layer. This heatmap identifies the areas containing the most relevant information for the pretrained CNN's prediction.

Using the τ-threshold fFaster-RCNN, we compare our AoI for the input image with the heatmap to see what percentage the heatmap's relevant regions are contained in our AoI.

We do this comparison for the entirety of COCO, finding that on average, *97% of the Grad-CAM relevant regions are contained inside our AoI*. An example is shown in Figure 4.9. This confirms that our technique is explainable: it can cover the saliency maps of the neural networks.

4.5 Dissertation Contributions on Improved Focused Convolutions and AoI Generation

This dissertation presents a novel technique for converting a pretrained computer vision model into a more energy-efficient model, without losing accuracy or requiring additional training. We observe that highly accurate CNNs are typically accompanied by high energy consumption. We propose a technique to convert a CNN into a more energy-efficient model without requiring additional training. We apply a threshold (determined using an accuracy-latency curve search method) to the features produced by the top few 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. The AoI is used by the remaining convolutional layers in the CNN as focused convolutions. 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 instructions. By keeping the weights and biases of the original pretrained model, we expect to achieve both energy expenditure and inference latency improvements without losing accuracy or requiring additional training.

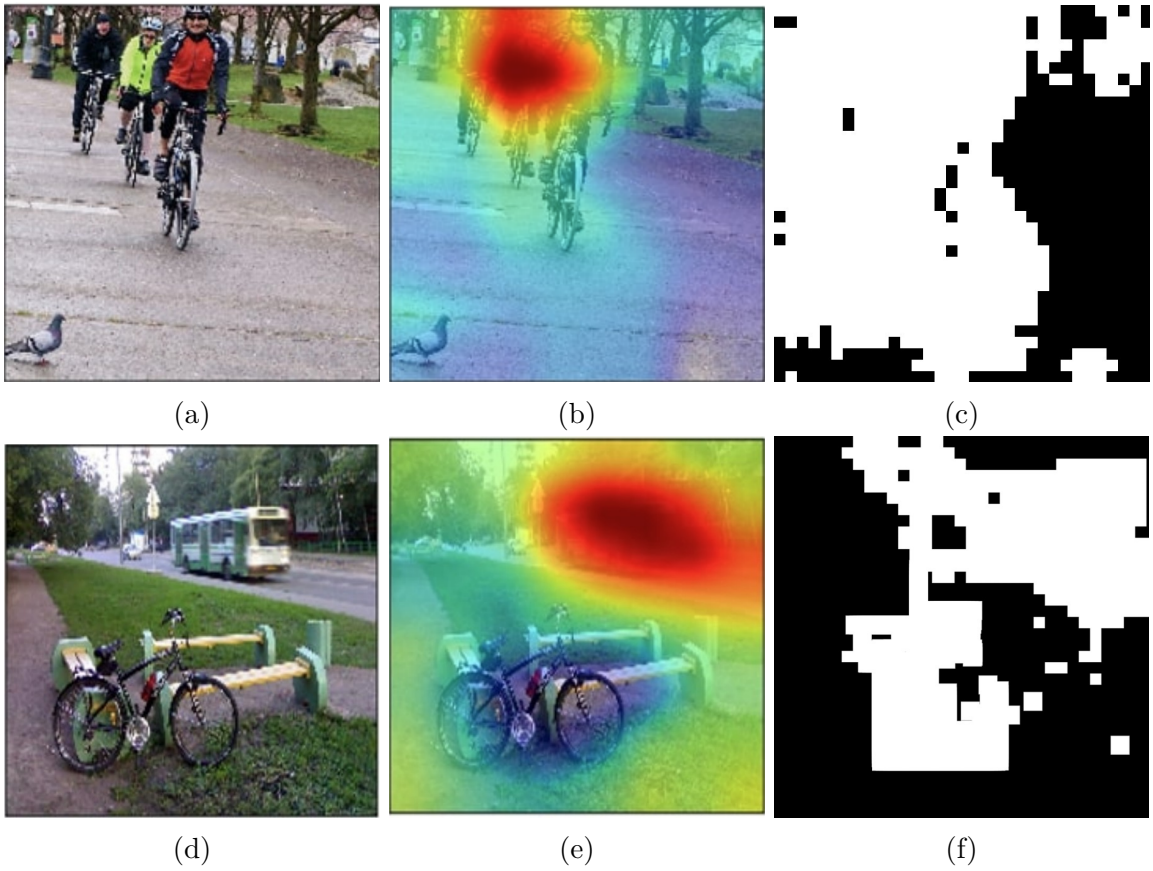


Figure 4.9. (a), (d) Original COCO images (cropped to square). **(b), (e)** Grad-CAM generated saliency map of Faster-RCNN. Warmer colors are more important, and colder colors are less important to the neural network. **(c), (f)** AoI generated by our τ-threshold AoI generation technique. Relevant pixels are white, irrelevant pixels are black. We see that our technique is explainable – it captures the most relevant regions (red, orange areas in (b), (e) are included in the white regions of (c), (f)) of the Faster-RCNN saliency map, and leaves out the irrelevant regions.

5. SUMMARY and CONCLUSION

In this dissertation, we present our work to measure and address the problem of inconsistent computer vision and to make CNNs more energy-efficient using focused convolutions to ignore pixels outside an Area of Interest (AoI). We also propose another technique to improve upon the focused convolutions: automatically generating AoIs within the CNN itself to filter out pixels at inference time.

5.1 Consistency Metric

In chapter 2, we discuss our work on measuring and improving CNN consistency. A CNN is expected to give similar predictions if the input images are similar. However, accuracy metrics do not measure whether CNNs meet that expectation. We create a consistency metric to measure that, and find that modern object detectors exhibit inconsistent behavior. We demonstrate consistency improvements by up to 5% using training-free, image processing techniques on the images.

5.2 Focused Convolutions

In chapter 3, we discuss our work on the focused convolution. We observe that in many images, there are pixels that are irrelevant to the computer vision task. The pixe ls that impact the CNN's prediction are relevant and comprise the AoI, and the rest are irrelevant. Our depth mapping dataset study finds that up to 48% of pixels in popular datasets are irrelevant. We design the focused convolution to ignore those irrelevant pixels during CNN inference, allowing CNNs to improve their energy consumption by up to 45%.

5.3 Automated AoI Generation for Improved Focused Convolutions

In chapter 4, we present our notable improvements to the focused convolution, enabling an end-to-end, training free solution to generate AoIs automatically. The existing focused convolutions require a full forward-pass of an existing AoI through the CNN before the CNN can then be used to do inference on the corresponding image. Additionally, the depth

mapping method used to generate AoIs is computationally intensive. Therefore, we filter the features from the early layers in a CNN to automatically determine the AoI at runtime, with little computational overhead. This method requires no training and could be customized for each CNN, mitigating any accuracy lost during the process. Implemented end-to-end, we demonstrate that pretrained CNNs can be modified to fully automate AoI generation and deploy focused convolutions for up to 28% inference latency improvement on large datasets like Microsoft COCO, with at most 2% accuracy degradation.

5.4 Concluding Remarks

To conclude, focused convolutions show promise as a method for improving energy efficiency and CNN latency by deleting irrelevant pixels outside an AoI, without requiring training. The consistency metric augments existing accuracy metrics as a way to ensure that energy-efficient CNNs maintain consistent accuracy. Any future work building on this dissertation should focus on further methods for improving CNN consistency and generating AoIs automatically, so that these methods can be more widely and easily adopted. Our code is available at GitHub: https://github.com/PurdueCAM2Project/focused-convolutions.

REFERENCES

[1] 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)*, 2022, pp. 11 976–11 986. DOI: 10.1109/CVPR52688.2022.01167.

[2] 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. DOI: 10.1109/TPAMI.2016.2577031.

[3] 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)*, IEEE, 2020, pp. 1–6. DOI: 10.1109/WF-IoT48130.2020.9221198.

[4] S. Alyamkin, M. Ardi, A. C. Berg, *et al.*, "Low-power computer vision: Status, challenges, and opportunities," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 411–421, 2019. DOI: 10.1109/JETCAS.2019.2911899.

[5] C. Tung, M. R. Kelleher, R. J. Schlueter, *et al.*, "Large-Scale Object Detection of Images from Network Cameras in Variable Ambient Lighting Conditions," in *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, Mar. 2019, pp. 393–398. DOI: 10.1109/MIPR.2019.00080.

[6] M. Piccardi, "Background subtraction techniques: A review," in *2004 IEEE international conference on systems, man and cybernetics*, IEEE, vol. 4, 2004, pp. 3099–3104. DOI: 10.1109/ICSMC.2004.1400815.

[7] 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)*, 2021, pp. 1611–1621. DOI: 10.1109/CVPR46437.2021.00166.

[8] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking," *arXiv:1504.01942 [cs]*, Apr. 2015, arXiv: 1504.01942. [Online]. Available: https://arxiv.org/abs/1504.01942.

[9] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft COCO: Common Objects in Context," en, in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2014, pp. 740–755, ISBN: 978-3-319-10602-1. DOI: 10.1007/978-3-319-10602-1_48.

[10] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2018, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2018.2844175.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 91–99. DOI: 10.1109/TPAMI.2016.2577031.

[12] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," in *2017 ICCV*, 2017, pp. 2980–2988. DOI: 10.1109/TPAMI.2018.2858826.

[13] W. Liu, D. Anguelov, D. Erhan, *et al.*, "SSD: Single Shot MultiBox Detector," en, in *Computer Vision ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 21–37, ISBN: 978-3-319-46448-0. DOI: 10.1007/978-3-319-46448-0_2.

[14] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in *2015 International Conference on Learning Representations*, Mar. 2015. [Online]. Available: https://arxiv.org/abs/1412.6572.

[15] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal Adversarial Perturbations," in *2017 IEEE CVPR*, 2017, pp. 1765–1773. DOI: 10.1109/CVPR.2017.17.

[16] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks," in *2016 IEEE CVPR*, 2016, pp. 2574–2582. DOI: 10.1109/CVPR.2016.282.

[17] T. d'Orsi, P. K. Kothari, G. Novikov, and D. Steurer, "Sparse PCA: Algorithms, Adversarial Perturbations and Certificates," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, ISSN: 2575-8454, Nov. 2020, pp. 553–564. DOI: 10.1109/FOCS46700.2020.00058.

[18] S. Gu and L. Rigazio, "Towards Deep Neural Network Architectures Robust to Adversarial Examples," *arXiv:1412.5068 [cs]*, Apr. 2015, arXiv: 1412.5068. [Online]. Available: https://arxiv.org/abs/1412.5068.

[19] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, Jun. 2016, pp. 1–6. DOI: 10.1109/QoMEX.2016.7498955.

[20] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004, Conference Name: IEEE Transactions on Image Processing, ISSN: 1941-0042. DOI: 10.1109/TIP.2003.819861.

[21] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "FSIM: A Feature Similarity Index for Image Quality Assessment," *IEEE Transactions on Image Processing*, vol. 20, no. 8, pp. 2378–2386, Aug. 2011, Conference Name: IEEE Transactions on Image Processing, ISSN: 1941-0042. DOI: 10.1109/TIP.2011.2109730.

[22] R. Zhang, "Making Convolutional Networks Shift-Invariant Again," en, in *International Conference on Machine Learning*, May 2019, pp. 7324–7334. [Online]. Available: https://proceedings.mlr.press/v97/zhang19a.html.

[23] K. Gu, B. Yang, J. Ngiam, Q. Le, and J. Shlens, "Using Videos to Evaluate Image Model Robustness," *arXiv:1904.10076 [cs]*, Aug. 2019, arXiv: 1904.10076. [Online]. Available: https://arxiv.org/abs/1904.10076.

[24] D. Zhang, J. Han, L. Yang, and D. Xu, "SPFTN: A Joint Learning Framework for Localizing and Segmenting Objects in Weakly Labeled Videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 475–489, Feb. 2020, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2018.2881114.

[25] D. Zhang, J. Han, L. Zhao, and D. Meng, "Leveraging Prior-Knowledge for Weakly Supervised Object Detection Under a Collaborative Self-Paced Curriculum Learning Framework," en, *International Journal of Computer Vision*, vol. 127, no. 4, pp. 363–380, Apr. 2019, ISSN: 1573-1405. DOI: 10.1007/s11263-018-1112-4.

[26] D. Kang, Y. Sun, D. Hendrycks, T. Brown, and J. Steinhardt, "Testing Robustness Against Unforeseen Adversaries," *arXiv:1908.08016 [cs, stat]*, Jun. 2020, arXiv: 1908.08016. [Online]. Available: https://arxiv.org/abs/1908.08016.

[27] Z. Yin, H. Wang, J. Wang, J. Tang, and W. Wang, "Defense against adversarial attacks by low-level image transformations," en, *International Journal of Intelligent Systems*, vol. 35, no. 10, pp. 1453–1466, 2020, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/int.22258, ISSN: 1098-111X. DOI: https://doi.org/10.1002/int.22258.

[28] Y. H. Yeu, M. I. Shapiai, Z. H. Ismail, and H. Fauzi, "Investigation on Different Color Spaces on Faster RCNN for Night-Time Human Occupancy Modelling," in *2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)*, Dec. 2019, pp. 118–121. DOI: 10.1109/ICSPC47137.2019.9068000.

[29] G. K. Thiruvathukal *et al.*, *Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence*, en. CRC Press, 2022. DOI: 10.1201/9781003162810.

[30] H. Wang *et al.*, "Structured Pruning for Efficient Convolutional Neural Networks via Incremental Regularization," *IEEE JSTSP*, vol. 14, no. 4, May 2020, Conference Name: IEEE Journal of Selected Topics in Signal Processing. DOI: 10.1109/JSTSP.2019.2961233.

[31] M. Courbariaux *et al.*, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv:1602.02830*, Mar. 2016, arXiv: 1602.02830. [Online]. Available: https://arxiv.org/abs/1602.02830.

[32] B. Peng *et al.*, "Correlation Congruence for Knowledge Distillation," in *2019 IEEE/CVF ICCV*, ISSN: 2380-7504, Oct. 2019, pp. 5006–5015. DOI: 10.1109/ICCV.2019.00511.

[33] M. Tan *et al.*, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *2019 IEEE/CVF CVPR*, ISSN: 2575-7075, Jun. 2019, pp. 2815–2823. DOI: 10.1109/CVPR.2019.00293.

[34] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," en, *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010, ISSN: 1573-1405. DOI: 10.1007/s11263-009-0275-4.

[35] M. Ren *et al.*, "SBNet: Sparse Blocks Network for Fast Inference," in *2018 IEEE/CVF CVPR*, ISSN: 2575-7075, Jun. 2018, pp. 8711–8720. DOI: 10.1109/CVPR.2018.00908.

[36] T. Verelst *et al.*, "Dynamic Convolutions: Exploiting Spatial Sparsity for Faster Inference," in *2020 IEEE/CVF CVPR*, ISSN: 2575-7075, Jun. 2020, pp. 2317–2326. DOI: 10.1109/CVPR42600.2020.00239.

[37] A. Skillman and T. Edso, "A technical overview of cortex-m55 and ethos-u55: Arms most capable processors for endpoint ai," in *2020 IEEE Hot Chips 32 Symposium (HCS)*, IEEE Computer Society, 2020, pp. 1–20. DOI: 10.1109/HCS49909.2020.9220415.

[38] P. Micikevicius, D. Stosic, N. Burgess, *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022. [Online]. Available: https://arxiv.org/abs/2209.05433.

[39] T. Dubhir, M. Mishra, and R. Singhal, "Benchmarking of quantization libraries in popular frameworks," in *2021 IEEE International Conference on Big Data (Big Data)*, IEEE, 2021, pp. 3050–3055. DOI: 10.1109/BigData52589.2021.9671500.

[40] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *2019 International Conference on Machine Learning (ICML)*, PMLR, 2019, pp. 6105–6114. [Online]. Available: https://arxiv.org/abs/1905.11946.

[41] A. Goel, S. Aghajanzadeh, C. Tung, S.-H. Chen, G. K. Thiruvathukal, and Y.-H. Lu, "Modular neural networks for low-power image classification on embedded devices," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 1, pp. 1–35, 2020. DOI: 10.1145/3408062.

[42] J. Hostetler, "Toward runtime-throttleable neural networks," *arXiv preprint arXiv:1905.13179*, 2019. [Online]. Available: https://arxiv.org/abs/1905.13179.

[43] K. Bhardwaj, J. Ward, C. Tung, *et al.*, "Restructurable activation networks," *arXiv preprint arXiv:2208.08562*, 2022. [Online]. Available: https://arxiv.org/abs/2208.08562.

[44] K. Bhardwaj, G. Li, and R. Marculescu, "How does topology influence gradient propagation and model performance of deep networks with densenet-type skip connections?" In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 13 498–13 507. DOI: 10.1109/CVPR46437.2021.01329.

[45] C. Tung, A. Goel, X. Hu, *et al.*, "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)*, IEEE, 2022, pp. 340–343. DOI: 10.1109/AICAS54282.2022.9870012.

[46] X. Hou and L. Zhang, "Saliency detection: A spectral residual approach," in *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Ieee, 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383267.

[47] K. Gauen *et al.*, "Comparison of Visual Datasets for Machine Learning," in *2017 IEEE IRI*, Aug. 2017, pp. 346–355. DOI: 10.1109/IRI.2017.59.

[48] S. Song *et al.*, "Semantic Scene Completion From a Single Depth Image," in *2017 IEEE/CVF CVPR*, 2017, pp. 1746–1754. DOI: 10.1109/CVPR.2017.28.

[49] R. Ranftl *et al.*, "Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer," *IEEE TPAMI*, pp. 1–1, 2020, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.3019967.

[50] A. Anderson *et al.*, "Low-memory GEMM-based convolution for deep neural networks," *arXiv:1709.03395*, 2017. [Online]. Available: https://arxiv.org/abs/1709.03395.

[51] E. Wang *et al.*, "Intel Math Kernel Library," en, in *High-Performance Computing on the Intelő Xeon Phi: How to Fully Exploit MIC Architectures*, E. Wang, Q. Zhang, B. Shen, *et al.*, Eds., Cham: Springer International Publishing, 2014, pp. 167–188, ISBN: 978-3-319-06486-4. DOI: 10.1007/978-3-319-06486-4_7.

[52] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in *2020 IEEE/CVF CVPR*, 2020, pp. 10 781–10 790. DOI: 10.1109/CVPR42600.2020.01079.

[53] M. Sandler *et al.*, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF CVPR*, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[54] G. K. Thiruvathukal, Y.-H. Lu, Y. Chen, J. Kim, and B. Chen, *Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence* (CRC Computer and Information Science), 1st. Chapman & Hall, 2022. DOI: 10.1201/9781003162810.

[55] 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)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[56] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556.

[57] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, Springer, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2.

[58] 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)*, Ieee, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[59] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision (ECCV)*, Springer, 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48.

[60] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. DOI: 10.5555/3295222.3295349.

[61] S. Teerapittayanon and B. McDanel, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, IEEE, 2016, pp. 2464–2469. DOI: 10.1109/ICPR.2016.7900006.

[62] M. Figurnov, M. D. Collins, Y. Zhu, *et al.*, "Spatially adaptive computation time for residual networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1039–1048. DOI: 10.1109/CVPR.2017.194.

[63] 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. DOI: 10.1145/292395.292412.

[64] 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*, 2017, pp. 5687–5695. DOI: 10.1109/CVPR.2017.643.

[65] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," *ArXiv*, Dec. 2015, arXiv:1512.04150 [cs]. [Online]. Available: https://arxiv.org/abs/1512.04150.

VITA

Caleb Tung is an energy-efficient artificial intelligence researcher. Caleb came to West Lafayette, Indiana to attend Purdue University in 2015. He began performing research under his advisor, Prof. Yung-Hsiang Lu, in Summer 2017 as a Purdue Summer Undergraduate Research Fellow, winning the Top Research Talk award at the SURF Symposium. He graduated as a Trustees Scholar from the John Martinson Honors College in 2019 with his BS in Computer Engineering, With Distinction. Caleb then enrolled under Prof. Lu as a Ross Fellow to complete his Electrical and Computer Engineering PhD degree in 2023. During his academic career, Caleb has worked as an engineering intern at SEP, Inc. and USAA, and he has completed computer vision research internships at Southwest Research Institute, Latent AI, and Arm. Caleb's current research interests involve low-power computer vision systems for embedded devices and verifiable general artificial intelligence models.

PUBLICATIONS

A. Goel, **C. Tung**, N. Eliopoulos, X. Hu, G.K. Thiruvathukal, J.C. Davis, and Y.-H. Lu.
"Directed Acyclic Graph-based Neural Networks for Tunable Low-Power Computer Vision".
2022 ACM/IEEE International Symposium on Low Power Electronics and Design

C. Tung, A. Goel, N. Eliopoulos, X. Hu, G.K. Thiruvathukal, V. Chaudhary, and Y.-H. Lu.
"Irrelevant Pixels are Everywhere: Find and Exclude Them for More Efficient Computer Vision".
2022 IEEE International Conference on Artificial Intelligence Circuits and Systems

A. Goel, **C. Tung**, X. Hu, G.K. Thiruvathukal, J.C. Davis, and Y.-H. Lu.
"Efficient Computer Vision on Edge Devices with Pipeline-Parallel Hierarchical Neural Networks".
2022 Asia and South Pacific Design Automation Conference

C. Tung, A. Goel, F. Bordwell, N. Eliopoulos, X. Hu, G.K. Thiruvathukal, and Y.-H. Lu.
"Why Accuracy Is Not Enough: The Need for Consistency in Object Detection".
2022 IEEE Multimedia

S. Allcroft, M. Metwaly, Z. Berg, I. Ghodgaonkar, F. Bordwell, X. Zhao, X. Liu, S. Chakraborty, V. Banna, A Chinnakotla, A. Goel, **C. Tung**, G. Kao, W. Zakharov, D. Shoham, G.K. Thiruvathukal, and Y.-H. Lu.
"Observing Human Mobility Worldwide during COVID-19".
2022 IEEE Computer

A. Goel, **C. Tung**, X. Hu, H. Wang, J.C. Davis, G.K. Thiruvathukal and Y.-H. Lu.
"Low-Power Multi-Camera Object Re-Identification using Hierarchical Neural Networks".
2021 ACM/IEEE International Symposium on Low Power Electronics and Design

A. Goel, **C. Tung**, S. Aghajanzadeh, S.-H. Chen, G.K. Thiruvathukal, Y.-H. Lu.
"Modular Neural Networks for Low-Power Image Classification on Embedded Devices".
2020 ACM Transactions on Design Automation of Electronic Systems

A. Goel, **C. Tung**, Y.-H. Lu, G.K. Thiruvathukal.
"A Survey of Methods for Low-Power Deep Learning and Computer Vision".
2020 IEEE World Forum on Internet of Things

A. Goel, **C. Tung**, S. Aghajanzadeh, I. Ghodgaonkar, S. Ghosh, G.K. Thiruvathukal, Y.-H. Lu.
"Low-Power Object Counting with Hierarchical Neural Networks".
2020 ACM/IEEE International Symposium on Low Power Electronics and Design

S. Aghajanzadeh, R. Naidu, S.-H. Chen, **C. Tung**, A. Goel, Y.-H. Lu, G.K. Thiruvathukal.
"Camera Placement Meeting Restrictions of Computer Vision".
2020 IEEE International Conference on Image Processing

C. Tung, M.R. Kelleher, R.J. Schlueter, B. Xu, Y.-H. Lu, G.K. Thiruvathukal, Y.-K. Chen, Y. Lu.
"Large-Scale Object Detection of Images from Network Cameras in Variable Ambient Lighting Conditions".
2019 IEEE International Conference on Multimedia Information Processing and Retrieval