AN ADAPTIVE FRAMEWORK FOR ENERGY-EFFICIENT EDGE AI: FROM CLASSIFICATION TO SYNTHESIS OF IMAGES AND 3D SHAPES

By

KANNAPPAN JAYAKODI NITTHILAN

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

DECEMBER 2022

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of KANNAPPAN JAYAKODI NITTHILAN find it satisfactory and recommend that it be accepted.

Janardhan Rao Doppa, Ph.D., Chair

Partha Pratim Pande, Ph.D.

Ananth Kalyanaraman, Ph.D.

Umit Ogras, Ph.D.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor Professor Janardhan Rao Doppa for his constant support, valuable advice and guidance during my PhD program. I gratefully acknowledge my PhD committee members, Professor Partha Pratim Pande, Professor Ananth Kalyanaraman and Professor Umit Ogras for their valuable feedback. I am also grateful to all my co authors and labmates specially Aryan Deshwal, Md Rakibul Islam, Syrine Belakaria, Shubhomoy Das, F A Rezaur Rahman Chowdhury, Mohammad Hossein Namaki, Taha Belkhouja, Alaleh Ahmadian, Anwesha Chatterjee and Wonje Choi for their help during my PhD.

Finally, I would like to thank all the funding agencies that supported my research work. I gratefully acknowledge the support from the National Science Foundation under Grant OAC-1910213 and Grant IIS-1845922, and in part by the U.S. Army Research Office under Grant W911NF-17-1-0485 and Grant W911NF-19-1-0162.

AN ADAPTIVE FRAMEWORK FOR ENERGY-EFFICIENT EDGE AI: FROM CLASSIFICATION TO SYNTHESIS OF IMAGES AND 3D SHAPES

Abstract

by Kannappan Jayakodi Nitthilan, Ph.D.
Washington State University
December 2022

Chair: Janardhan Rao Doppa

A large number of real-time artificial intelligence (AI) applications including robotics, self-driving cars, smart health and augmented (AR) / virtual reality (VR) are enhanced/boosted by deploying deep neural networks (DNNs). Currently, computation for most of these applications happens on the cloud due to huge compute, energy, and memory requirements. However, moving these applications to the edge platforms such as smartphones and AR/VR headsets reduce latency and improves user experience, accessibility, and data privacy. Existing solutions utilize high-performance and energy-efficient hardware accelerators and software solutions including sparsity and quantization of weights with a potential compromise on the prediction accuracy.

This dissertation proposes an adaptive framework for energy-efficient edge AI which complements all the previous solutions enhancing the performance on edge devices. We utilize the intuitive idea that easy inputs require simple networks and hard inputs require complex networks. This framework is based on three key ideas. First, we design and train a space of DNNs of increasing complexity (coarse to fine). Second, we perform an input-specific adaptive inference by selecting a DNN of appropriate complexity depending on the hardness of input examples. Third, we execute the selected DNN on the target edge platform using a resource management policy to save energy. We demonstrate the generalization of the proposed solution for three qualitatively different problem settings ranging from convolutional neural networks (CNNs) for simple image classification to structured generative adversarial networks (GANs) for photo-realistic unconditional image generation and graph convolutional networks (GCNs) for 3D shape synthesis. Our experiments on real-world applications on edge platforms demonstrate a significant reduction in energy and latency with little to no loss in prediction accuracy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AR	Augmented Reality
VR	Virtual Reality
DNN	Deep Neural Network
CNN	Convolutional neural networks
GAN	Generative adversarial networks
GCN	Graph convolutional networks
AI	Artificial Intelligence
C2F	Coarse to fine
EDP	Energy Delay Product
IoU	Intersection-over-Union
EIE	Energy Inference Engine

Dedication

This is dedicated to my parents Dr. S. Kannappan M.D and Mrs. K. Jayakodi M. A and my sister Mrs. S. Danabakiam. B. E, without whose sacrifice and encouragement it would never have been possible.

CHAPTER 1. INTRODUCTION

On one hand, we are witnessing an exponential growth of emerging edge applications enabled by the growth of powerful hardware platforms in small-form factors. Some prototypical examples of edge platforms [52] include mobiles, Internet of Things (IoT), wearables, robotics, AR/VR headsets, surveillance cameras, self-driving cars, and drones. On the other hand, we are also seeing the rise of data-driven AI applications based on deep neural networks (DNNs) trained from large scale data (e.g., text, images, speech, and sensor data). DNNs — a set of computational techniques to automatically extract patterns and useful features from raw data — achieve high prediction accuracy, but their large size makes them computationally expensive, consuming a lot of energy and memory. However, edge devices are inherently constrained by limited battery life, low compute resources, and small memory footprint. Edge AI is a sub-area of artificial intelligence (AI) that tries to bridge low-compute edge platforms with the complex DNN solutions.

Currently, most of the computation for these edge applications happens in the cloud, but there are many good reasons to perform this computation locally on edge platforms. First, it will increase the accessibility of AI applications to different parts of the world by reducing the dependence on the communication fabric and the cloud infrastructure. Second, many of these applications such as robotics and AR/VR require low latency. We may not be able to achieve their critical real-time requirements without performing the computation directly on the edge platform. Third, for many important applications such as mobile health, privacy and security are important concerns (e.g., medical data). By doing the inference on edge devices we avoid sharing data on the cloud. Since edge platforms are constrained by resources (power, computation, and memory), there is a great need for innovative solutions to realize the vision of practically useful Edge AI [52].

There is a growing body of literature to addresses the challenges to create energy-efficient Edge AI. These approaches include designing high-performance and energy-efficient hardware accelerators to perform DNN inference [18, 47, 19]. Important software methods include: reduced precision of weights and activations [85]; binary weights [22] and its variations like LightNN [33]; pruning and compression of large DNNs [51, 116]; exploiting sparsity of DNN models [18]; design of DNNs to reduce the computation and model size (e.g., MobileNet and SqueezeNet) and slimmable neural networks to allow varying number of channels for inference [118]. There are also simple hand-designed approaches to perform incremental/adaptive inference with CNNs for classification [95, 92, 111, 83]. One key limitation of the prior work is that all these approaches focus on the simple classification tasks. There are many edge AI applications such as AR/VR that involve predicting structured outputs (e.g., 3D shapes from color images). The structured output prediction problem is challenging due to the huge space of outputs for any given input. Further in these approaches the inference for *every* input example is made using a “fixed” computational process. They do not exploit the fact that hardness of inference varies from one example to another.

This dissertation proposes a general adaptive inference framework for energy-efficient edge AI which is applicable for both simple classification and complex structured output prediction tasks. The overall principle behind this framework is to be *adaptive* about 1) what to compute, and 2) how to compute on a target edge platform to save energy with little to no loss in accuracy. To address the challenge of what to compute, we exploit the fact that hardness of DNN inference varies from one input example to another. For example, we use simpler convolutional neural network (CNN) for simple images and complex CNN for harder images. Similarly, we use appropriate graph convolution network (GCN) to produce a 3D object shape in the form of a triangular mesh. Fig 1.1 illustrates this conceptual idea. To practically realize this approach, we automatically design a space of DNNs of increasing complexity from a given DNN architecture to allow reuse of computation from lower-levels to perform incremental computation at higher-levels. To perform input-specific adaptive DNN inference, we use some learnable parameters (one threshold for each level). To address the challenge of how to compute, we use data-driven runtime resource management techniques [87, 88, 26] to fully exploit the capabilities of the target hardware platform (e.g., ARM Big-Little architecture) to run any selected DNN for saving energy. The

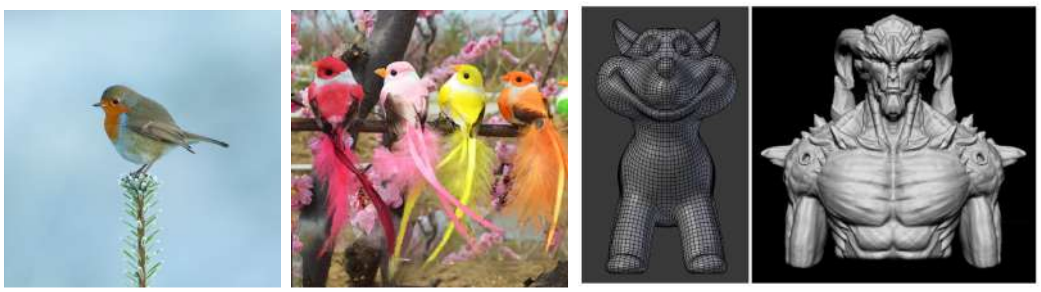


Figure 1.1: Example images to illustrate "easy" and "hard" inference problems.

Left Two: CNN based image classification,

Right Two: GCNs for producing 3D shapes with low and high polycount.

overall effectiveness of the co-design framework critically depends on the threshold parameters for different levels. Therefore, we use multi-objective Bayesian optimization (BO) algorithms [11, 12, 7, 122, 10] to configure the threshold parameters to achieve different trade-offs between accuracy, energy, and latency.

We provide three concrete instantiations of our proposed framework for CNN based adaptive inference for image classification, GCN based adaptive inference for producing 3D object shapes from color images to enable AR/VR applications, and GAN based adaptive inference for photo-realistic unconditional image generation (generate high-quality and diverse images with same visual content as the input image). Experiments on real-world datasets and a commercial hardware platform show the effectiveness of our adaptive inference framework for edge AI in achieving significant reduction in energy and execution time with little to no loss in prediction accuracy. We also list some future directions to push the boundaries of energy-efficient edge AI.

1.1 Technical Contributions

The main contribution of this dissertation is a general adaptive inference framework for energy-efficient edge AI for solving both classification and structured output prediction tasks. We provide effective instantiations of this framework for CNN based classification, GCN based 3D shape synthesis and GAN based image synthesis. However, for some deep learning models where the data flow is not linear (e.g., transformers), our framework may not be directly applicable and will require some changes to incorporateadaptability. Below we first provide a high-level overview of our adaptive inference framework and then describe the details of the key elements of the framework.

Overview of Adaptive Framework for Energy Efficient Edge AI. The framework tries to synergistically combine the structure in the DNN software program and benefits provided by the target hardware platform to perform *adaptive inference* optimized for each input example. One key advantage of this framework is that it is complementary to existing software approaches such as quantization, pruning, and knowledge distillation (can leverage the output network from these methods to further improve adaptive inference) and hardware approaches (can leverage to improve performance and energy-efficiency of inference).

While the state-of-the-art DNNs achieve very high accuracy, they are highly complex (deep) and consume a significant amount of energy. On the other hand, simple (shallow) networks have poor accuracy but are very energy-efficient. Our motivation is based on the fact that many input examples are *easy* and can be predicted correctly using simple DNNs and only a small fraction of *hard* inputs would need complex DNNs (illustrated in Figure 1.1). Furthermore, we observe that as we move from simple to complex DNNs, for small improvements of accuracy there is a significant growth in energy consumption, corroborating our hypothesis. Therefore, our goal is automatically select simple networks for easy inputs and complex networks for hard inputs. This mechanism will allow us to achieve high accuracy with significantly less energy consumption when compared to the baseline approach, where we make predictions for all input examples using the most complex DNN. After selecting the appropriate DNN for a given input example, we use an optimized resource management policy to execute the selected DNN on the target hardware platform to save energy. The effectiveness of the overall framework depends on the learnable parameters that select the appropriate complexity of the DNN model. We use a Bayesian optimization methodology to configure these parameters to trade-off objectives including accuracy, energy, and latency.

Key Elements. We explain the key elements of our edge AI framework below.

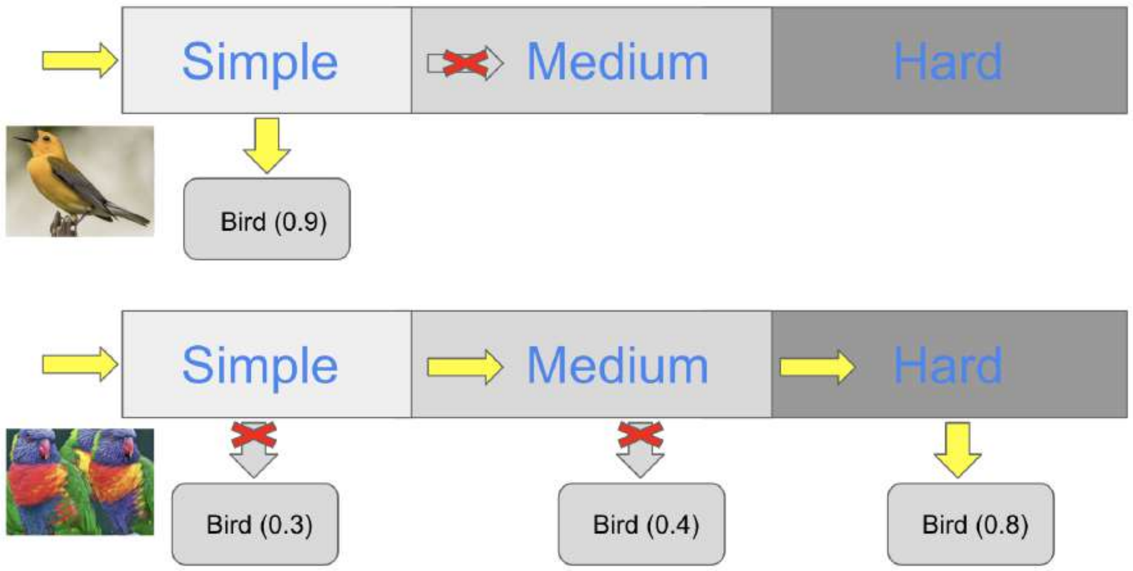


Figure 1.2: An adaptive framework intuition based on input complexity [64]

1. Design space of coarse-to-fine DNNs. Given a DNN architecture $\mathcal{M}$ (e.g., convolutional neural network) that is appropriate for solving a prediction task (e.g., image classification), we design space of coarse-to-fine DNNs with L levels of increasing complexity. Suppose $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_L$ correspond to the DNNs at the L levels respectively. Note that $\mathcal{M}_L$ corresponds to the most complex DNN and is exactly the same as $\mathcal{M}$. We have one confidence threshold for each of the L-1 levels, which determines the selection of the DNN for each input example. Suppose $T_1, T_2, \cdots, T_{L-1}$ correspond to the threshold parameters for adaptive inference.

2. Adaptive inference. Given a coarse-to-fine space of DNNs $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_L$ and confidence thresholds $T_1, T_2, \cdots, T_{L-1}$, we perform adaptive inference for an input example x as follows. We sequentially go through DNNs starting from $\mathcal{M}_1$ to make predictions. If the estimated confidence threshold at level i meets the threshold parameter T_i or we reach the final level L, we terminate and return the predicted output $\hat{y}$.

3. Hardware resource management policy. The resource management policy π's goal is to optimally execute the selected DNN model M_i for an input example x on the target hardware platform (e.g., ARM's Big/Little architecture) to save energy. The policy will select the appropriate hardware configuration by making resource management decisions such as the number of Big/Little cores to be ON, the frequency of Big/Little cores, and what task to run on which type of core. We can leverage recent machine learning methods for runtime resource management [87, 88, 26] to accomplish this goal.

4. Bayesian optimization for configuring thresholds. Our overall goal is to optimize the adaptive framework in a data-driven manner to trade-off relevant objectives including prediction accuracy, energy, and latency. Specifically, each configuration of the threshold parameters $T_1, T_2, \cdots, T_{L-1}$ gives us a particular trade-off between accuracy, energy, and latency. We want to find the threshold vectors corresponding to the optimal Pareto front. We can leverage multi-objective BO algorithms [11, 12, 7, 5, 8, 6, 9]. to solve this problem. BO algorithms can be seen as sequential decision-making processes that select the next threshold vector to be evaluated to quickly direct the search towards optimal Pareto sets by trading-off exploration and exploitation at each iteration. By selecting a sequence of candidate threshold vectors for evaluation and learning a statistical model based on the observed training data (input is threshold vector; and output is an objective vector including accuracy, latency, and energy), the BO approach can quickly move towards high-quality solutions. The overall BO methodology significantly reduces the number of expensive objective function evaluations for optimization.

1.2 Outline of Thesis

The remaining part of the dissertation is organized as follows. In Chapter 2, we discuss the related work on current solutions for edge AI applications considered in this research and motivate our proposed adaptive inference framework.

The next set of chapters describe the generalization of our framework by demonstrating it on multiple applications utilizing different DNN architectures of varying complexity. First we study the problem of computational efficiency of convolutional neural networks (CNNs) used for image classification task on edge devices. In Chapter 3, we design coarse-to-fine (C2F) network architecture which employs classifiers of varying complexity depending on the hardness of input examples. The key idea is to *learn* to select simpler (coarser) networks to classify "easy" examples and complex (finer) networks to classify "hard" examples. Figure 1.1 illustrates easy and hard inputs for image classification task.

Next, in Chapter 4, we propose *LEANets*, a second instantiation for image classification which takes advantage of the other dimension of the network complexity. While C2F Nets constructs progressively complex networks by adding more layers thereby increasing the network depth, LEANets achieve the same by adding more channels expanding along the network width. Both these architectures are stage-wise optimized to automatically configure for a specified trade-off between accuracy and energy consumption of inference on a target hardware platform. Further both the approaches are complementary to most of the existing methods in the sense that it can reuse a pre-trained DNN towards this goal. We perform comprehensive experiments and our optimized network provides a significant reduction in the Energy Delay Product (EDP) with no loss in accuracy when compared to the baseline solutions, where all predictions are made using the most complex network.

In Chapter 5, we demonstrate the framework for the task of 3D shape synthesis, especially as a surface mesh object, from single color images using Graph Convolutional Networks (GCN) [21, 53, 113]. Unlike image classification tasks that involve predicting a class label for each image, *3D shape generation is significantly complex* and involves the prediction of an entire mesh to approximate the shape. The accuracy of prediction to the actual 3D shape is a function of the number of triangles in the mesh measured in terms of *polycount*. Simple objects would require fewer polygons while complex objects would require more polygons to represent the 3D object shape. We propose a novel approach referred to as *PETNet* that *leverages pre-trained GCNs* for predicting 3D object shapes to automatically trade-off energy and Intersection-over-Union (IoU) score of inference at *runtime* on a target mobile platform. Comprehensive experiments on different object classes from the ShapeNet dataset to show the generality and effectiveness of our approach. Results show that PETNet achieves 20%-37% gain in energy for negligible loss (0.01-0.02) in IoU score.

In Chapter 6, we propose *SETGAN* for the task of photo-realistic image synthesis task using Generative Adversarial Networks (GAN). SETGAN leverages *single-image GANs* (smaller model size and faster training/inference time compared to multi-image GANs) for photo-realistic unconditional image generation to automatically trade-off energy consumption and accuracy of inference at *runtime* for a target mobile platform. SETGAN considers a multi-scale GAN model (GAN models of increasing complexity) that is amenable for embarrassingly parallel training procedures to reduce latency and results in more stable training for high-resolution images. For any given input image, this multi-scale GAN model is quickly trained on a server or cloud, and the trained model is used for repeated inferences (aka amortization of training cost) to enable image applications, e.g., image editing. Further, given a user-specified accuracy threshold, SETGAN automatically selects the smallest scale GAN that meets the user-threshold to save energy and improve inference time. Simple images can be generated with small-scale GANs to save energy and we only need higher-scale GANs for complex images.

Finally, we conclude the dissertation with the lessons learned and discuss important future directions in Chapter 7.

CHAPTER 2. BACKGROUND AND RELATED WORK

In this chapter, we review the related work on energy efficient edge AI problems. First we talk about generic solutions which could be used for any network configurations and edge AI applications. These solutions fall in the areas of hardware solutions, software solutions and hardware-software hybrid solutions. Next, since our framework uses a architecture similar to cascaded architectures, we discuss similar solutions in literature and their drawbacks. Most of these solutions are based on CNN based image classification which is relatively simple. It takes a image as input and classifies it into specific classes. Following which we discuss related work specific to the real-world edge AI applications, which demonstrate the generalization of our architecture. Specifically, we discuss about GCN based 3D shape synthesis and GAN based image synthesis. These two applications are structured in the sense that the input problem instances and output solutions are combinatorial structures. In 3D shape synthesis, the input is a 2d image grid and output is a 3D mesh shape object. The next application, GAN image synthesis, the input is a 2d image and output is a synthesized variation of similar images of different dimensions.

2.1 Deploying DNNs on Mobile Platforms

Main challenges for deploying deep learning applications on embedded systems include high resource requirements in terms of memory, computation, and energy.

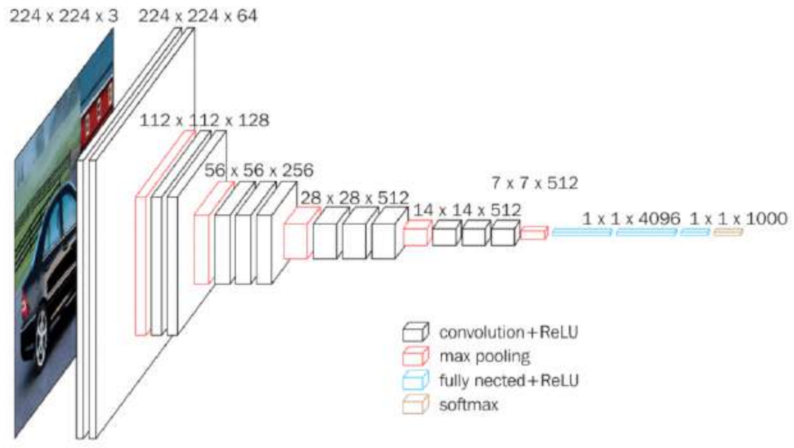


Figure 2.1: Image Classification

Prior work has addressed these challenges using methods based on hardware, software, and hardware/software co-design [43]. We discuss various approaches for these three categories below.

Hardware Approaches. Architecture level methods include design of specialized hardware targeting DNN computations using the data-flow knowledge [18], placement of memory closer to computation units [47], and on-chip integration of memory and computation [19]. Apart from the constraint of having an application-specific hardware system, most of these hardware technologies also have an overhead of analog to digital conversion [43].

Software Approaches. To address the challenges in general purpose CPUs and GPUs, following software-level approaches are studied. Reducing the precision of weights and activations is one such approach [38]. Fixed point representation of weights and activations dynamically varying across different layers is also employed to reduce the energy and area cost of the network [85]. In [79], authors show that using floating-point for weights and fixed-point for activations leads to reduction of power consumption by 50% and reduction in model size by 36%. Another software level approach is pruning and compression of large-scale DNNs [48]. The sparsity in features and activations after non-linear operations like ReLU is exploited to compress the models in [18]. To address significant redundancy in weights during training the network, the authors in [51] employ pruning technique on the weights based on their magnitudes. While magnitude-based pruning of weights reduces the computation cost, it does not directly address the energy cost. To make the model more energy efficient, an "energy-aware" pruning mechanism is proposed in [116], where pruning of weights is performed layer-by-layer based on the knowledge of energy hungry layers. This method is shown to be 1.74 times more energy-efficient when compared to the weight-magnitude based pruning. A finer-level software approach is to improve the network architecture. The inception module [40] performs a 2D-convolution considering two 1D convolutions considering the 2-D filters to be completely separable. The Xception [20] and Mobilenets [56] architectures also employ depth-wise separable convolutions similar to inception model, but perform an additional point-wise convolution, i.e., a 1x1 convolution at the end to combine the outputs of the depth-wise convolutions. This method helps in reducing the computation and model size. The

SqueezeNet [59] introduces a fire-module that squeezes the filter dimension followed by expansion to reduce the number of filter coefficients, which helps in achieving the accuracy of AlexNet with 50X fewer parameters.

Hardware and Software Hybrid Approaches. Many of these approaches involve design of hardware guided by software level optimization techniques. [85] proposes a compiler specific to FPGA that analyzes CNNs and generates modules to improve the throughput of the system. In [42], authors propose the design of Energy Inference Engine (EIE) that deploys pruned DNN models. In [41], the knowledge of compressed sparse-weights guides the hardware to read weights and perform MAC computations only for the nonzero activations, thereby reducing the energy cost by 45%. Slimmable neural networks (SlimNets) [119] is a very recent state-of-the-art approach that employs the adaptive computation graph technique. The approach behind SlimNets involve training a single neural network at different widths (number of channels). At runtime, we can adaptively execute SlimNet with one of the pre-defined widths as needed based on the resource constraints on mobile platform.

2.2 Cascaded and Conditional Inference Approaches

Cascaded computation is a general principle that has a long history in computer science and machine learning [46, 96]. The application of this general principle to a given problem is non-trivial, and the exact details and algorithmic procedures vary from one application to another including our co-design approach for inference with deep neural networks. Cascaded CNN architecture for face detection [83] progressively prune areas of the image that are not likely to contain a face. The key differences with our work are as follows: 1) In our architecture, the features computed in the previous level are reused in the next level to construct more complex features, whereas the CNNs used in [83] do not reuse any features; and 2) The approach proposed in [83] do not allow to optimize the cascade for a specified trade-off between speed and accuracy of face detection. Prior work on coarse-to-fine deep kernel networks [104] tries to combine multiple input kernels in a recursive manner to achieve a complex kernel and associated representation of the data. This paper makes very strict assumptions and incorporates these assumptions directly into the overall learning approach: 1) It can only work for binary classification tasks; 2) It assumes that the number of negative examples are significantly higher than positive examples. The learning approach in [104] for training coarser networks is tuned to binary classification and assigns very high-costs to misclassifying positive examples to incorporate this assumption; and 3) There are no thresholds to configure the architecture to trade-off speed and accuracy of prediction. Similarly, there is no optimization methodology to optimize the architecture for a specified trade-off. The high-level architecture of branch CNN (BCNN) [44] is similar in style to our C2F model, but there are some significant differences: 1) BCNN uses linear classifier after each layer, whereas we do not; and 2) The approach in [44] doesn't allow to configure the BCNN for any specified trade-off between speed and accuracy. This method is also very *ad hoc* as there is no well-defined optimization formulation and optimization methodology. The paper on conditional deep learning (CDL) model [95] is the closest related work to ours. However, there are some significant differences between our work and this paper: 1) CDL model employs a single threshold value at all the levels, whereas we employ different thresholds for different levels. When the number of levels are more, different thresholds for different levels will achieve better pareto-optimal solutions. As discussed later, our experimental results on CIFAR10 and CIFAR100 clearly show this difference; 2) In their training approach, they only train using a subset of the input examples that are not filtered by the previous level. This will likely cause over-fitting and may not work well on harder tasks. We train all the intermediate classifiers on all the training examples to improve their generalization. Additionally, the approach in [95], need to train all the different levels for each confidence threshold, which is very time-consuming. In our case, we train classifiers at different levels only once and tune the thresholds for different levels jointly conditioned on the trained features and classifiers; and 3) We look at the co-design problem of configuring a software application to run on a specific hardware architecture for a specified trade-off between accuracy and energy. Our optimization approach based on Bayesian Optimization principles to configure the C2F net to trade-off accuracy and energy is significantly different and allows us to achieve better pareto-optimal solutions.

There are also approaches that employ a sequence of networks to perform conditional inference depending on the input example to save computation and energy. Iterative CNN (ICNN) approach [92] makes multi-stage predictions for images, where a two-stage wavelet transform is applied to produce multiple small-scale images and train separate models to process them progressively to make predictions. Adaptive NN [111] employs two DNNs — small one on a mobile and large one on a remote server – and performs joint optimization. However, this approach is significantly sub-optimal: only allows two levels and does not reuse computation from small DNN. Work on conditional deep learning (CDL) model [95] also falls in this category.

2.3 Edge applications

Prior work to deploy DNNs on resource-constrained mobile platforms is primarily studied for simple classification tasks which uses CNN. *There is no prior work in this problem space for deploying GCNs or GANs to predict structured outputs.* In this following section, we discuss alternative solutions which are currently available in literature and discuss issues which make running complex architectures like GCN and GAN difficult to run on edge devices. We further enumerate how our proposed solution address them.

Multi-Image GANs. Initial attempts to apply GANs [49] suffered from being noisy and incomprehensible. However, growing both the generator and discriminator progressively starting from low-resolution images and adding new layers that introduce

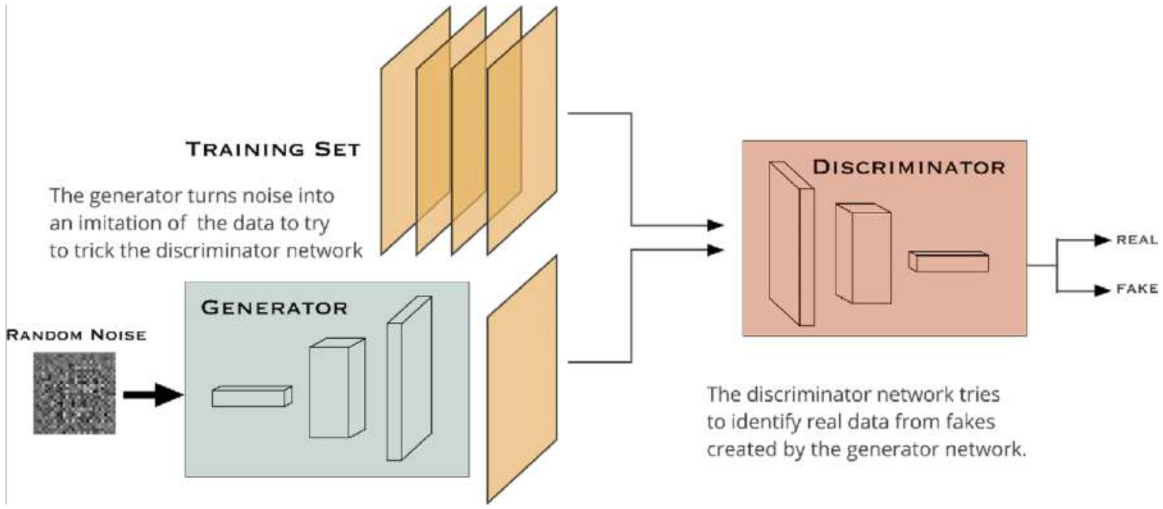

Training Set
The generator turns noise into an imitation of the data to try to trick the discriminator network
Generator
Random Noise
Discriminator
Real
Fake
The discriminator network tries to identify real data from fakes created by the generator network.

Figure 2.2: GAN Image synthesis

higher-resolution details as the training progresses, greatly improved the speed and stability of training leading to successful applications [75, 97] GANs trained on multiple images have been successfully applied for photo-realistic image generation applications including super-resolution (scaling a low-resolution image to a high-resolution image) [4], image-to-image translation [60], and paint2image [114]. However, GANs designed for these applications are very task-specific: models trained for a specific task cannot be used for other tasks without re-training or using different models. Additionally, these GANs work only for a specific class of image datasets (e.g., faces) for which the GAN has been trained for and fail in other cases (e.g., natural scenes). In contrast, our SETGAN model once trained can be applied to various image manipulation tasks including super-resolution, harmonization and paint2image. Since we train the model for every new image, our approach is applicable for diverse image types.

Single-Image GANs. Since multi-image GANs are trained using a large number of images from a specific dataset, they try to capture the characteristics of the whole dataset requiring large models making it difficult to run on edge platforms. This drawback is addressed by GANs which train on single images [100]. Several recent works fit a deep model to a single training image, but all these methods were designed for specific tasks (e.g., super-resolution [4]) and texture expansion [121]). Furthermore, unconditional single-image GANs have been explored only in the context of texture generation [16, 66]. In contrast, our SETGAN model is purely generative (i.e., maps noise to image samples) making it suitable for many image manipulation tasks.

Generation of 3D Objects from Images There are three major methods that address the problem of generating 3D object shapes from images. First extends 2D CNNs to 3D CNNs for producing 3D voxels as output [21].However, this method works on low-resolution 3D shapes due to memory constraints on modern GPUs. Further, 3D voxels representation is not popular in game and movie industry. Second method is based on point cloud representation [53]. However, this representation does not have connection information between points, which makes it hard to recover 3D mesh directly. Third method is based on 3D mesh generation [55, 67] [113]. Pixel2Mesh

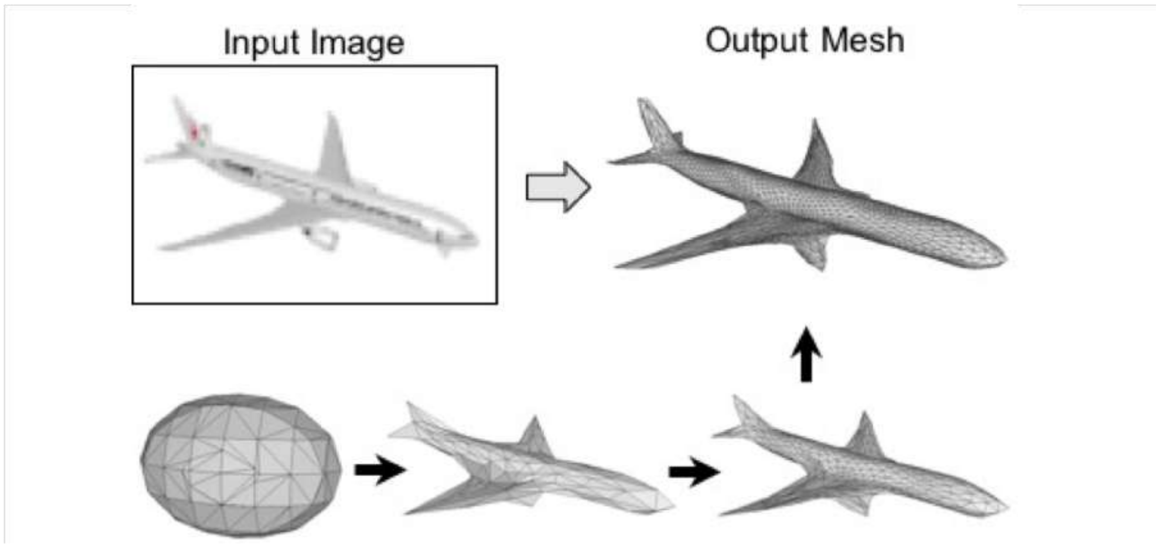


Figure 2.3: GCN 3D shape synthesis

(P2M) [113] is one such method based on GCNs [105] and has superior performance when compared to other two methods [55, 67]. Our PETNet approach builds on the P2M network to trade-off energy and accuracy for producing 3D object shapes from images on mobile platforms.

In conclusion, our adaptive framework automatically finds the optimized sequence of classifiers. Our proposed approach has several advantages over prior methods: **1)** It is complementary to methods including quantization, model pruning, hand-designed networks, and knowledge distillation. Specifically, we can consider our approach as a wrapper approach that takes the output network from these methods to automatically trade-off energy and accuracy. **2)** Our approach employs pre-trained DNN given as input and only trains a small number of classifiers. The classifier training overhead is negligible when compared to corresponding baselines. **3)** Next, our framework can achieve finer energy-accuracy trade-offs even with a fixed number of levels by changing the confidence threshold vectors employed to perform input-specific adaptive inference. On the other hand, other solutions with pre-defined levels can only achieve a coarse-grained trade-off. **4)** Finally, our solutions do not employ any extra network modules to perform dynamic inference.

CHAPTER 3. C2FNETS:COARSE-TO-FINE NETWORKS TO TRADE-OFF ACCURACY AND ENERGY FOR CLASSIFICATION

In this chapter, we discuss the effectiveness of adaptive inference demonstrating it using an instance of adaptive CNN inference, namely C2F Nets approach [64] that allows us to adaptively select a classifier of appropriate complexity depending on the hardness of the input example.

Convolution neural network (CNN) is the backbone for an image classification application (e.g., classifying images as cat vs. dog). The application takes an image x as input and classifies it as one of the candidate classification labels. A CNN is usually made up of two major blocks. *1) Feature transformation function* in the form of a network of layers that takes the features computed in previous level x_{i-1} as input and produces more complex features x_i. A feature transformer blocks $[MxN]$ has a series of convolution layers followed by a max-pooling layer. M and N represent the number of convolution layers and the number of filters in each layer. *2) Classifier* takes the features computed by feature transformation function x_i and produces a probability distribution over all candidate labels. The classifier block has a series of dense or fully connected layers followed by a soft-max layer.

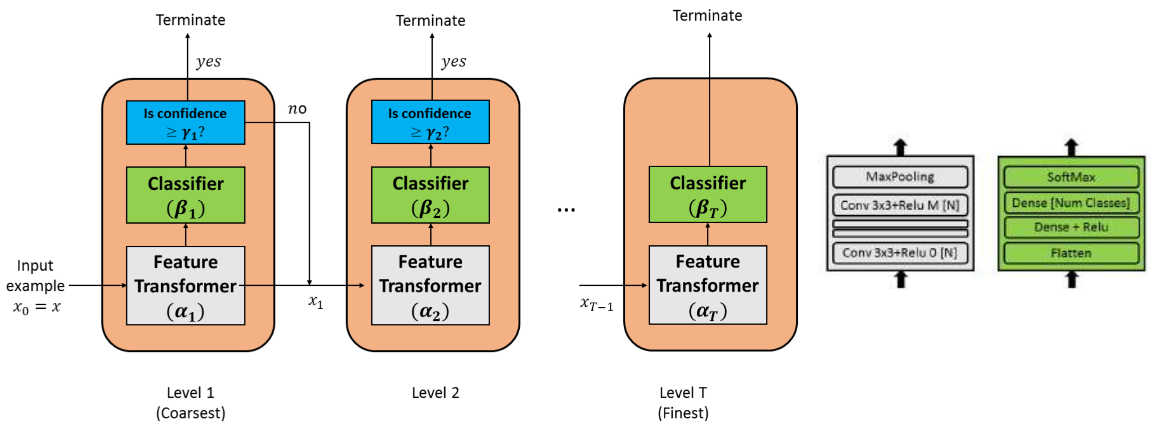

Terminate
yes
Is confidence $\geq \gamma_1$?
no
Classifier ($\boldsymbol{\beta}_1$)
Feature Transformer ($\boldsymbol{\alpha}_1$)
Input example $x_0 = x$
x_1
Terminate
yes
Is confidence $\geq \gamma_2$?
Classifier ($\boldsymbol{\beta}_2$)
Feature Transformer ($\boldsymbol{\alpha}_2$)
...
x_{T-1}
Terminate
Classifier ($\boldsymbol{\beta}_T$)
Feature Transformer ($\boldsymbol{\alpha}_T$)
Level 1 (Coarsest)
Level 2
Level T (Finest)
MaxPooling
Conv 3x3+Relu M [N]
Conv 3x3+Relu 0 [N]
SoftMax
Dense [Num Classes]
Dense + Relu
Flatten

Figure 3.1: Illustration of Coarse-to-Fine (C2F) networks model.An abstract C2F net model with T levels from coarsest to finest network, where each level consists of a feature transformer, a classifier, and a confidence threshold to decide when to terminate.

3.1 Coarse-to-Fine Networks (C2F Nets)

In this section, we describe the details of our coarse-to-fine networks (C2F Nets) model. In what follows, we provide the motivation, formal model of C2F Nets with some examples, and how to make predictions given a fully specified C2F Net model.

3.1.1 Motivation

Simple (shallow) DNNs are energy-efficient, but their accuracy is low. On the other hand, many state-of-the-art DNNs that achieve very high accuracy are highly complex (deep) and consume significant amount of energy. We are motivated by the fact that many input examples are *easy* and can be classified correctly using simple DNNs and only a small fraction of *hard* inputs would need complex DNNs. We illustrate this observation using two examples. First, Figure 1.1 shows bird images corresponding to easy and hard cases. Second, Figure 4.5 shows the accuracy of different DNNs with varying complexity. We can see that the accuracy improvement is small when we move from simple to complex DNNs, whereas their energy consumption grows significantly. This corroborates our hypothesis. Therefore, our goal is automatically select simple networks for easy inputs and complex networks for hard inputs. This mechanism will allow us to achieve high accuracy with significantly less energy consumption when compared to the baseline approach, where we make predictions for all input examples using the most complex DNN.

3.1.2 C2F Nets Model

A C2F Net model $\mathcal{NN}$ with T levels can be seen as a sequence of networks stacked in the order of increasing complexity as shown in Figure 3.1. Each level i of the model contains three key elements.

1) Feature transformation function in the form of a network of layers that takes the features computed in previous level x_{i-1} as input and produces more complex features x_i. Parameters of the feature function are denoted by α_i. Figure 3.1 and Figure 3.6 provide illustrative examples for feature transformation functions.

2) Classifier takes the features computed by feature transformation function x_i and produces a probability distribution over all candidate labels. Parameters of the classifier are denoted by β_i. Figure 3.1 provides an illustration of the classifier block.

3) Confidence threshold. Given a predicted probability distribution over classification labels, we can estimate the confidence of the classifier in a variety of ways [112] including: a) *Maximum probability*, where we take the probability of the highest score label; b) *Entropy* over the probability of all candidate labels; c) *Margin* denoted by the difference in probability scores between best and second-best label. For a given confidence type, the confidence threshold γ_i is employed to determine if the classifier is confident enough in its prediction to terminate and return the predicted label.

3.1.3 Inference in C2F Nets

Given a C2F Net with all its parameters (α, β, and γ) fully specified, inference computation for a new input example x is performed as follows (see Algorithm 1). We sequentially go through the C2F Net starting from level 1. At each level i, we make predictions using the feature transformation functions parameterized by α_i and intermediate classifiers β_i to compute the probability distribution of classification labels $\hat{P}_i(y)$. We estimate the confidence parameter t_i from $\hat{P}_i(y)$ and predicted output $\hat{y}$ as the label with highest probability score. If estimated confidence in prediction t_i meets the threshold γ_i or we reach the final level T, we terminate and return the predicted output $\hat{y}$ for the given input x.

3.2 Optimization Approach for C2F Nets

In this section, we describe a principled optimization algorithm to automatically configure C2F Nets for a specified trade-off between accuracy and energy on a target hardware platform.

Problem Setup. Without loss of generality, let us assume that we are given a C2F Net $\mathcal{NN}$ with T levels. The parameters of $\mathcal{NN}$ can be divided into three parts: 1) Parameters of feature transformation functions at different levels $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_T)$; 2) Parameters of intermediate classifiers at different levels $\beta = (\beta_1, \beta_2, \cdots, \beta_T)$; and 3) Confidence thresholds at different levels $\gamma = (\gamma_1, \gamma_2, \cdots, \gamma_{T-1})$ that decide the complexity of classifier employed for a given input example. We are provided with a training set D_{train} and validation set D_{val} of classification examples drawn from an unknown target distribution $\mathcal{D}$, where each classification example is of the form (x, y^*), where x is the input (e.g., image) and y is the output class (e.g., image label). We are also given a target hardware platform $\mathcal{H}$ for which the C2F model $\mathcal{NN}$ need to be optimized for. We can measure the energy consumption and accuracy of inference a candidate $\mathcal{NN}$ model using the hardware $\mathcal{H}$. Suppose $\mathcal{O}(\mathcal{NN}, \mathcal{H}, \lambda)$= $\mathbb{E}_{(x,y^*)\sim\mathcal{D}}$ $\lambda\cdot$ $\textsc{Error}(\mathcal{NN},\ \mathcal{H})$ + (1- λ)$\cdot\textsc{Energy}(\mathcal{NN},\ \mathcal{H})$ stands for the expected error and energy trade-off achieved over the target distribution of classification examples $\mathcal{D}$ when the coarse-to-fine network model $\mathcal{NN}$ is executed with parameters α, β, γ on the target hardware platform $\mathcal{H}$. For a specified trade-off parameter $\lambda \in [0, 1]$, our goal is to find the best parameters α, β, γ of coarse-to-fine network $\mathcal{NN}$ to achieve the specified trade-off between error and energy for the target platform $\mathcal{H}$.

$$(\alpha^*, \beta^*, \gamma^*) = \arg\min_{\alpha,\beta,\gamma} \mathcal{O}(\mathcal{NN}, \mathcal{H}, \lambda) \tag{3.1}$$

Since we do not have access to the distribution $\mathcal{D}$, we employ the training and validation set to find the best parameter values. For example, to evaluate a candidate parameter configuration over the validation set, we do the following. We compute the normalized error and normalized energy with respect to the setting where all predictions are made using the largest network over all the input examples in the validation set. We plug the normalized error and energy in the objective $\mathcal{O}$ to evaluate the candidate parameter configuration.

3.2.1 Stagewise Optimization Algorithm

The optimization problem posed in Equation 3.1 is extremely challenging to solve due to complex interactions between the parameter values (α, β, γ) on the optimization objective $\mathcal{O}(\mathcal{NN}, \mathcal{H}, \lambda)$. Generally, C2F Model $\mathcal{NN}$ would be trained using back-propagation via stochastic gradient descent (SGD) optimization [45]. Specifically, the Objective requires the energy consumption of coarse-to-fine model $\mathcal{NN}$ on the target hardware platform $\mathcal{H}$, where it is executed. Applying the standard back-propagation training algorithm would require us to estimate the energy for every iteration of the training step, which would make the training procedure impractical.

Algorithm 1 Inference in Coarse-to-Fine Networks

Input: x = input example; $\mathcal{NN}$ = C2F network with T levels; $\alpha_1, \alpha_2, \cdots, \alpha_T$ = parameters of feature transformers; $\beta_1, \beta_2, \cdots, \beta_T$ = parameters of intermediate classifiers; $\gamma_1, \gamma_2, \cdots, \gamma_{T-1}$ = confidence threshold values

```
1: $x_0 \leftarrow x$; and level $i \leftarrow 1$
2: TERMINATE $\leftarrow$ False
3: while TERMINATE == False do
4:     $x_i \leftarrow$ Transform-Features($x_{i-1}$, $\alpha_i$)
5:     $\hat{P}_i(y) \leftarrow$ Intermediate-Classifier($x_i$, $\beta_i$)
6:     Prediction $\hat{y} \leftarrow$ label with highest probability from $\hat{P}_i(y)$
7:     Compute confidence $t_i$ using $\hat{P}_i(y)$
8:     if $i == T$ OR $t_i \geq \gamma_i$ then
9:         TERMINATE $\leftarrow$ True
10:    else
11:        $i \leftarrow i + 1$ // Continue with next level
12: return predicted class label $\hat{y}$
```

To overcome these challenges, we leverage the structure in this optimization problem to develop an efficient stagewise optimization algorithm (see Algorithm 2), where the parameters α, β, and γ are optimized sequentially one after another. By splitting it into multiple stages, we decouple the dependency of the training procedure for α and β parameters from the energy measurement step. First, we learn the parameters α to optimize the feature transformers at different levels. Second, conditioned on the found α, we learn the parameters β to optimize the intermediate classifiers at different levels. In both the stages, we optimize only for the prediction accuracy independent of the energy. Third, conditioned on the found α and β, we find the best values of confidence thresholds γ to optimize the objective $\mathcal{O}$. We describe the details of these three optimization procedures in the following subsections.

Algorithm 2 Stagewise Optimization of C2F Nets

Input: D_{train} = training set of classification examples; D_{val} = validation set of classification examples; $\mathcal{NN}$ = C2F Network architecture with T levels; $\mathcal{H}$ = target hardware platform; λ = parameter to trade-off accuracy and energy;

```
1: $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_T) \leftarrow$ Train-Features($D_{train}$)
2: $\beta = (\beta_1, \beta_2, \cdots, \beta_T) \leftarrow$ Train-Classifiers($D_{train}$, $\alpha$)
3: $\gamma = (\gamma_1, \gamma_2, \cdots, \gamma_{T-1}) \leftarrow$ Optimize-Thresholds($D_{val}$, $\mathcal{H}$, $\lambda$, $\alpha$, $\beta$, $\mathcal{G}$), where $\mathcal{G}$
   is the joint space of $T$-1 confidence thresholds
4: **return** the optimized parameters of C2F Net $\alpha$, $\beta$, and $\gamma$
```

Algorithm 3 Training Feature Transformers

```
Input: $D_{train}$ = training set of classification examples
1: $\alpha_1, \alpha_2, \cdots, \alpha_T \leftarrow$ Train the finest (most complex) classifier in C2F Net to minimize the cross-entropy loss via back propagation algorithm
2: return parameters of feature transformers $\alpha_1, \alpha_2, \cdots, \alpha_T$
```

3.2.2 Optimizing Feature Transformers

To obtain the parameters α corresponding to feature transformation functions at different levels, we train the finest (most complex) C2F Net as follows. We minimize the cross-entropy loss over training data D_{train} using a SGD training procedure. Cross-entropy loss (aka log loss) measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. SGD is a stochastic approximation of the gradient descent optimization and is a iterative method for minimizing cross-entropy loss via back-propagation.

3.2.3 Optimizing Intermediate Classifiers

Our approach to optimize the parameters of intermediate classifiers (i.e., β) is inspired by the transfer learning approach employed in deep learning. The key idea is to leverage the pre-trained model for a source task to quickly learn a model for a relevant target task. In our case, the source task corresponds to learning parameters α for the finest classifier as described above and the target task corresponds to learning parameters β_i for intermediate classifier at level $i \in \{1, 2, \cdots, T\}$. Intuitively, we want to minimize the energy consumption as much as possible by making correct classification decisions with high confidence using simple (coarser) classifiers. Therefore, we fix the parameters $\alpha_1, \alpha_2, \cdots, \alpha_i$ for feature transformation functions and learn the parameters β_i by minimizing the cross-entropy loss over the training data D_{train}.

Algorithm 4 Training Intermediate Classifiers

Input: D_{train} = training set of classification examples; $\alpha_1, \alpha_2, \cdots, \alpha_T$ = parameters of feature transformers

```
1: for each level $i = 1, 2, \cdots, T$ of C2F Net do
2:     $\beta_i \leftarrow$ Train a classifier with feature transformers $\alpha_1, \alpha_2, \cdots, \alpha_i$ to minimize the cross-entropy loss
3: return parameters of intermediate classifiers $\beta_1, \beta_2, \cdots, \beta_T$
```

3.2.4 Optimizing Confidence Thresholds

In this section, we describe our approach for finding the best thresholds $\gamma = (\gamma_1, \gamma_2, \cdots, \gamma_{T-1})$ for the specified trade-off between accuracy and energy consumption of inference with C2F networks.

Problem Formulation. Suppose the domain of each confidence threshold γ_i at each level $i \in \{1, 2, \cdots, T-1\}$ is $[lb, ub]$. For example, if we employ probability of highest

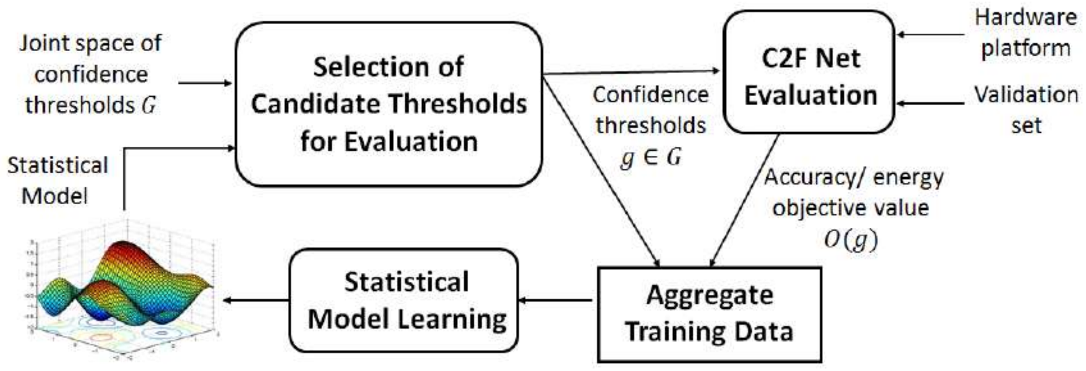

Joint space of confidence thresholds G
Selection of Candidate Thresholds for Evaluation
C2F Net Evaluation
Hardware platform
Validation set
Confidence thresholds $g \in G$
Statistical Model
Accuracy/ energy objective value $O(g)$
Statistical Model Learning
Aggregate Training Data

Figure 3.2: High-level overview of Bayesian Optimization approach for selecting the best confidence threshold values to configure a given C2F Net on a target hardware platform.

scoring label as our confidence parameter, then lb=0 and ub=1. Let $\mathcal{G}$ denote the joint space of candidate confidence threshold assignments, where $g \in \mathcal{G}$ is a candidate assignment to all the T-1 threshold parameters $\gamma_1, \gamma_2, \cdots, \gamma_{T-1}$. We can evaluate the objective $\mathcal{O}(\mathcal{NN}, \mathcal{H}, \lambda)= \mathbb{E}_{(x,y^*)\sim\mathcal{D}}\ \lambda\cdot$ Error($\mathcal{NN}$, $\mathcal{H}$) + (1- λ)$\cdot$Energy($\mathcal{NN}$, $\mathcal{H}$) for any $g \in \mathcal{G}$ by running the corresponding C2F net model $\mathcal{NN}$ on the validation set of classification examples D_{val} (i.e., expectation is approximated with a finite sample). Our goal is to find the best confidence thresholds $g^* \in \mathcal{G}$ that will lead to the highest objective value $\mathcal{O}(g)$.

Bayesian Optimization Approach. We propose to solve the above problem using the Bayesian Optimization (BO) framework [39] that is known to be very efficient in solving global optimization problems using black-box evaluations of the objective function (Figure 3.2). BO algorithms can be seen as sequential decision-making processes that select the next candidate input to be evaluated to quickly direct the search towards optimal inputs by trading-off exploration and exploitation at each search step. By iteratively selecting a candidate input for evaluation and learning a statistical model based on the observed input and output pairs, BO approach can quickly move towards high-quality inputs; this significantly reduces the number of objective function evaluations during the optimization process.

In what follows, we describe the three key elements of the general BO framework as applicable for our problem:

1) A Statistical Model of the true function $\mathcal{O}(g)$ by placing a *prior* over the space of functions. *Gaussian Process (GP)* [98] is the most commonly used prior due to its generality and uncertainty quantification ability. A GP over a space $\mathcal{G}$ is a random process from $\mathcal{G}$ to $\mathbb{R}$. It is characterized by a mean function $\mu : \mathcal{G} \rightarrow \mathbb{R}$ and a covariance or kernel function $\kappa : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$. *Radial kernels* are typically employed as prior covariance functions of GPs. Using a radial kernel means that the prior covariance can be written as $\kappa(g, g') = \kappa_0 \phi(\|g - g'\|)$ and depends only on the distance between g and g'. The scale parameter κ_0 captures the magnitude the function could deviate from μ. ϕ is a decreasing function with $\phi(0) = 1$. If a function F is sampled from $\mathcal{GP}(\mu, \kappa)$, then $\mathcal{O}(g)$ is distributed normally $\mathcal{N}(\mu(g), \kappa(g, g))$ for all $g \in \mathcal{G}$.

2) An Acquisition Function AF to score the utility of evaluating a candidate input based on the current statistical model. AF need to trade-off exploration and exploitation based on the predictions of the statistical model. Exploitation corresponds to selecting candidate inputs for which the statistical model is highly confident (high mean) and exploration corresponds to selecting candidate inputs for which the statistical model is not confident (high variance). We employ the popular Upper Confidence Bound (UCB) rule as acquisition function. UCB function is defined as $UCB_{t+1}(x) = \mu_t(x) + \sqrt{\kappa_{t+1}}\sigma_t(x)$, where μ_t and σ_t are the posterior mean and standard deviation of the GP conditioned on $\mathcal{D}_t$ (Algorithm 5). μ_t encourages exploitation, σ_t encourages exploration, and κ_{t+1} controls the trade-off.

3) An Optimization Procedure to select the the best scoring candidate input according to AF. We employ the popular DIvided RECTangles (DIRECT) approach to select the input $g \in \mathcal{G}$ to be queried in each iteration guided by the statistical model.

In each iteration, BO algorithm calls the optimizer to select the next input $g \in \mathcal{G}$ to evaluate the objective $\mathcal{O}(g)$, and update the statistical model using the aggregate training data from past evaluations (see Algorithm 5).

3.3 C2F Nets Design Methodology

In this section, we describe our methodology to design C2F networks from a given base network architecture (i.e., complex DNN). Although the methodology is

Algorithm 5 Finding Best Thresholds via Bayesian Optimization

Input: D_{val} = validation set of classification examples; $\mathcal{H}$ = target hardware platform; λ = parameter to trade-off accuracy and energy; $\alpha_1, \alpha_2, \cdots, \alpha_T$ = parameters of feature transformers; $\beta_1, \beta_2, \cdots, \beta_T$ = parameters of intermediate classifiers; $\mathcal{G}$ = joint space of confidence thresholds

```
1: $\mathcal{D}_0 \leftarrow$ small number of input-output pairs; and $t \leftarrow 0$
2: **repeat**
3:     Learn the GP model: $\mathcal{M}_t \leftarrow$ LEARN-GP($\mathcal{D}_t$)
4:     Compute the next input to query:
5: $g_{t+1} \leftarrow \arg max_{g \in \mathcal{G}}$ AF($\mathcal{M}_t, g$)
6:     Query objective function $\mathcal{O}$ at $g_{t+1}$ to get $\mathcal{O}(g_{t+1})$
7:     Aggregate the data: $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \cup \{(g_{t+1}, \mathcal{O}(g_{t+1}))\}$
8:     $t \leftarrow t + 1$
9: **until** convergence or maximum iterations
10: $g_{best} \leftarrow \arg max_{g_t} \mathcal{O}(g_t)$
11: **return** best found thresholds $g_{best}$=$\gamma_1, \gamma_2, \cdots, \gamma_{T-1}$
```

generally applicable to different neural network architectures, we use convolutional neural networks for image classification as a running example for illustrative purposes.

3.3.1 Hardware Setup

All our experiments were performed by deploying DNN models on an `ODROID-XU4` board [93]. `ODROID-XU4` is an Octa-core heterogeneous multi-processing system with ARM BigLittle architecture, which is very popular in current mobile devices. The BigLittle architecture consists of asymmetrical multi-core system, where cores are clustered into two groups based on the power and performance modes: 1) the power hungry (Big) mode; and 2) the battery saving slower processor mode (Little). The `ODROID-XU4` board employs ARM Cortex-A15 Quad 2GHz CPU as the Big Cluster and Cortex-A7 Quad 1.4GHz CPU as the Little Cluster. The board boots up Ubuntu 16.04LTS with ODROID's Linux kernel version 3.10.y. We execute DNN models with Caffe-HRT [17] that employs Caffe framework [68] with ARM Compute Library to speed up deep learning computations. We employ SmartPower2 [108] to measure power. We compute the average power over the total execution time.

3.3.2 Fine-Grained Energy Analysis of Base DNN

Base deep neural network (DNN) architectures. We consider deep convolutional networks for solving image classification task. These architectures consist of different layers including Convolution, Fully Connected (FC) or Dense, Max Pooling,

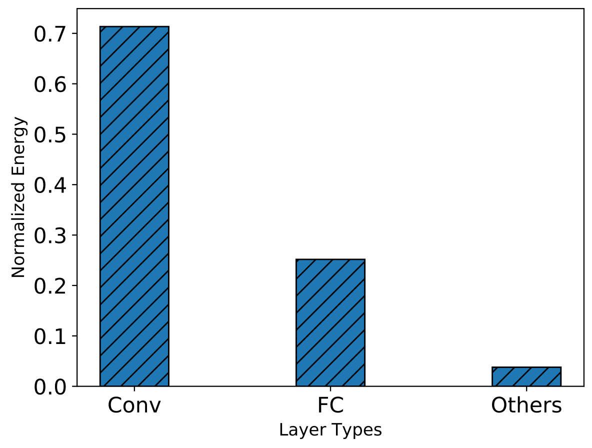


Figure 3.3: Energy distribution of different layers for base network A.

ReLU, Soft-Max, and Batch-Norm. For ease of understanding, we divide the network into *feature transformation block* and *classifier block*. The feature transformer block is the top part of the network and consists of convolution and max-pooling layers. It generates the input features for classifier block. The classifier block is the bottom part of the network and consists of fully connected layers followed by a soft-max layers. It predicts the final classification label for the input image. Refer Figure 3.1 for further details. We considered four different base DNNs with differing complexity to solve image classification task for our fine-grained energy analysis: *Base Network A and C* contains six convolution layers and *Base Network B and D* consists of ten convolution layers (a slight variant of VGG network [107]). However, we present our analysis and results for network A noting that the results are similar for all the other networks.

To understand the influence of a DNN architecture on the energy consumption of

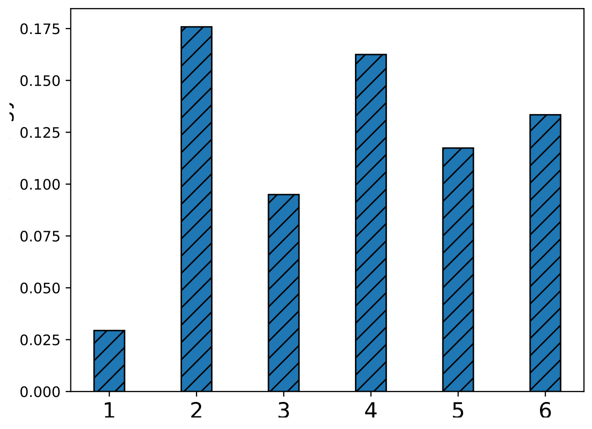


Figure 3.4: Energy consumption at different convolution layers for base network A. Layer 1 is the closest to the input layer. All energy values are normalized with respect to total energy consumed for inference of one image.

a real-world application, we analyzed the energy consumption per layer for all base networks to classify a single image. Our high-level energy analysis shows that convolution layers consume the most energy followed by fully connected layers. Figure 3.3 shows the distribution of energy consumed by convolution layers, fully connected layers, and all other layers for network A. In what follows, we perform fine-grained analysis for convolution and fully connected layers separately.

Energy analysis of convolution layers. The energy consumed by a convolution layer depends on the following elements: a) the size of the filter, b) image dimensions over which the filter is applied, and c) the number of filters. For all the base networks (A, B, C, and D), the number of convolution filters increases with the depth of the layer. For example, in base network A, the number of filters for increasing depths are

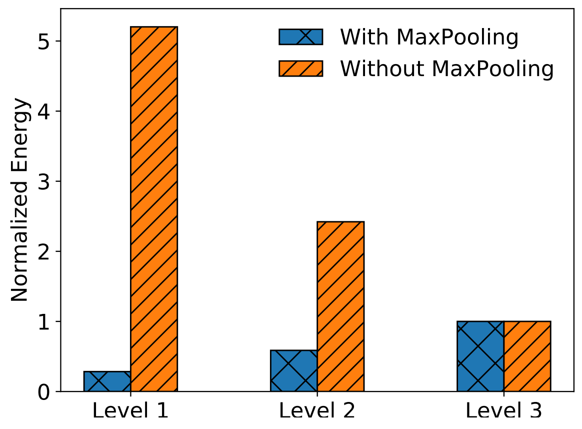


Figure 3.5: Energy consumption of FC layer for level 1, level 2, level 3 of C2F Net A. The effect of adding an extra max-pooling layer to reduce the energy. Energy values normalized w.r.t FC layer energy of the last level 3.

64, 128, and 192 respectively.

To design C2F Nets, we want to construct relatively simpler networks from base network for different trade-offs in accuracy and energy consumption. For these simpler (coarser) networks, we want to reduce the energy consumption by varying the values of the above-mentioned three key elements. We fix the size of the filter to 3x3 across all the layers for all the base networks. Therefore, our design of coarser networks is guided by reducing the number of convolution layers, number of filters per layer, and input image dimension. Figure 3.4 shows the distribution of energy consumption (normalized with respect to the total energy consumed by the base network) across different convolution layers of the base network A. Therefore, by progressively combining different number of convolution layers, we can generate different coarser networks of progressively increasing energy consumption.

Energy analysis of fully connected layers. Fully connected or dense layers consume the most energy after convolution layers. Dense layer is part of the classifier block of our base DNNs. The classifier block generally has two or more dense layers. Out of them, the first dense layer is usually the largest because it is obtained by flattening the features generated by the immediate convolution layer (just one Max Pooling layer) and is proportional to square of the image width. Hence, the first FC layer consumes significantly more energy than the rest.

3.3.3 Designing C2F Networks

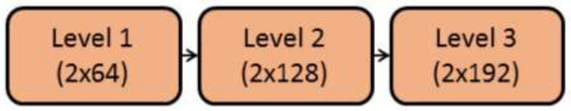
Our goal is to construct networks of varying complexity that show different trade-offs in accuracy and energy to be part of C2F Net.

Convolution layer. From the above discussion, we can construct networks that consume different amounts of energy by combining different number of convolution layers (acts as a feature transformer block for each level i) followed by a classifier for that level. For example, we construct a C2F Net for base network A with three levels as following. Level 1 (coarsest network) consists of two convolution layers as the feature block followed by a classifier block. Level 2 consists of four convolution layers (two extra convolution layers in addition to those from level 1). Level 3 corresponds to the full base network (finest network). Therefore, with increasing levels, we have progressively more convolution layers resulting in better accuracy at the expense of energy.

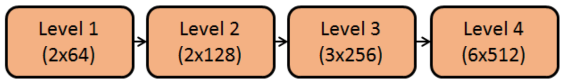
Fully connected layer. While designing C2F networks, we add classifiers at different depths of convolution layer for each level. At small depth, the image width is large (due to fewer max-pooling layers) for convolution. Hence, large number of features are fed to the classifier block. This increases the energy consumption of fully connected layers in the classifier blocks for coarser levels. Figure 3.5 shows the energy consumptions across the different levels of C2F Net corresponding to base network A as described above without an extra max-pooling layer.

We observe that the energy consumed by FC layer of level 1 and level 2 is significantly larger than the energy consumed by the FC layer of Finest Layer (level 3), which is not desirable. Therefore, to match the classifier block energy across different levels, we add an extra max-pooling layer for each of the classifier block for coarser networks (level 1 and level 2) such that the FC input dimension and the corresponding energy is similar across all the levels. Figure3.5 shows that the energy consumption with the extra max-pooling layer is comparable across all levels.

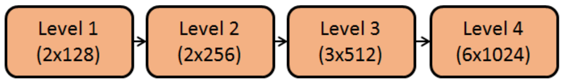
In summary, a practitioner can employ the above design principles to design coarse-to-fine networks that provide different trade-offs between accuracy and energy consumption.



(a) C2F Net A/Net C



(b) C2F Net B



(c) C2F Net D

Figure 3.6: C2F Nets definition in terms of the notation in figure 3.1.

3.4 Experiments and Results

In this section, we first describe the experimental setup and then discuss results along different dimensions of optimized C2F Nets.

3.4.1 Experimental Setup

Hardware Setup. We employ the hardware platform described in Section 5 to perform inference using different networks to measure the accuracy and energy consumption of inference process.

Image classification task and Training data. We consider a image classification task with 10 different classes. We employ the CIFAR10, CIFAR100 and MNIST [24] [25] image dataset with a 4:1:1 split for training (40000), validation (10000), and testing (10000) to train and test our C2F Nets approach. The input image dimension is 32x32x3 for the CIFAR10 and CIFAR100 datasets while it is 32x32x1 for the MNIST data set. The number of classes to be predicted is 10 for CIFAR10 and MNIST while it is 100 for CIFAR100. We employed the CIFAR10 dataset to train our adaptive C2F nets A and B; MNIST dataset for C2F net C; and CIFAR100 dataset for C2F net D

Energy and accuracy trade-off objective $\mathcal{O}$. Recall that the training of C2F Nets is based on the objective $\mathcal{O}(\mathcal{NN}, \mathcal{H}, \lambda)$= $\mathbb{E}_{(x,y^*)\sim\mathcal{D}}$ $\lambda\cdot$ $\textsc{Error}(\mathcal{NN}, \mathcal{H})$ + (1-λ)$\cdot\textsc{Energy}(\mathcal{NN}, \mathcal{H})$ where $\mathcal{H}$ is the hardware platform and $\mathcal{NN}$ is the C2F Net. Since accuracy and error are complementary, we have presented all our results in terms of accuracy and energy for the ease of exposition. Accuracy is measured as the fraction of input images whose labels are predicted correctly. Energy consumption of C2F Nets is measured in terms of normalized energy delay product (EDP). EDP = $\sum$P Δt $\cdot$ T, where Δt is the time interval at which we record the power P and T is the total execution time. As power is measured at a regular interval we simply calculate EDP as EDP = $P_{avg} \cdot T^2$. This value is normalized with EDP of the base network. We vary the value of λ from 0 to 1 to get C2F Nets optimized for different trade-offs between accuracy and energy.

C2F Networks. We employ four C2F networks in our experimental evaluation. Figure 3.6 shows the high-level architecture of C2F Net A and C with three levels and C2F Net B and D with four levels using the notations introduced in Figure 3.6. With Net A and Net B we demonstrate the effect of different architectures on the same

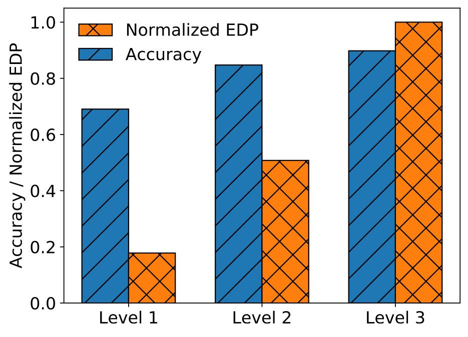
CIFAR10 dataset. With Net C and Net D we analyze the impact on the performance for a simpler and complex dataset like MNIST and CIFAR100 respectively.

Confidence threshold types. We experimented with all three types of confidence computation including *Max probability*, *Margin*, and *Entropy*. We observed that performance is almost same for all the three types. Therefore, we present all our results using Max probability as the confidence type.

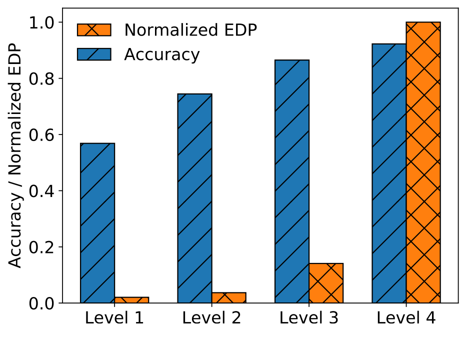
Offline training of C2F Nets. Recall that we need to find the parameters α, β, and γ for the specified trade-off objective using training and validation set of classification examples. For obtaining the parameters α and β, we employ the backpropagation training algorithm using RMS prop optimizer with learning rate of 0.0001 and a decay set to $1e^{-6}$. We employ a batch size of 128 and run training for 200 epochs with sufficient data augmentation including horizontal flips, width and height shifts. For obtaining the confidence thresholds, we employ the Bayesian Optimization (BO) approach with Gaussian kernel and UCB rule as the acquisition function. We use five evaluations of the objective $\mathcal{O}$ randomly for initialization. We run BO until convergence or a maximum of 100 iterations. We train C2F Nets for different values of λ to get different trade-offs between energy and accuracy.

3.4.2 Results and Discussion

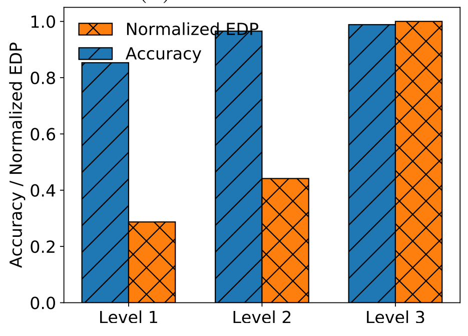
In this section, we present the results of our optimized C2F Nets and compare them along different dimensions.



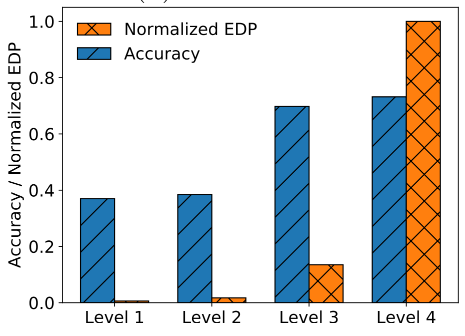
(a) C2F Net A



(b) C2F Net B



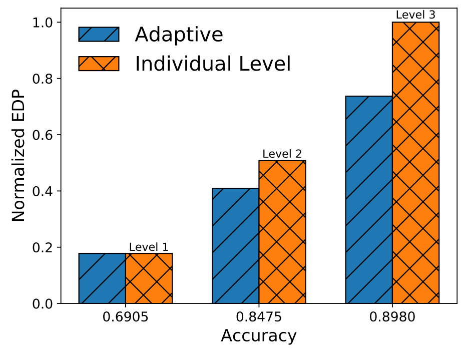
(c) C2F Net C



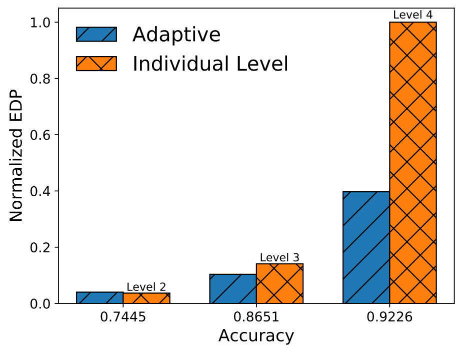
(d) C2F Net D

Figure 3.7: Accuracy and EDP when all predictions are made using an independently trained network at a particular level.

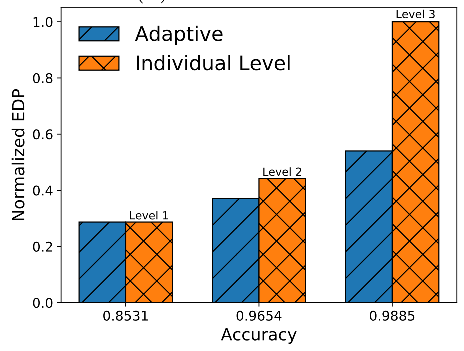
Accuracy and energy of networks at different levels. We train and test the networks at different levels that are part of C2F Net. Specifically, we make all the predictions using a single network for each level of C2F Net. This experiment will characterize the accuracy and energy trade-off achieved by different networks (coarsest to finest) that are part of C2F Net. Figure 4.5 shows the accuracy and energy metrics for networks at different levels for all the C2F Nets A, B, C and D. **C2F Net A:** From level 3 to level 2, we can see that accuracy drops by 5% and EDP reduces by



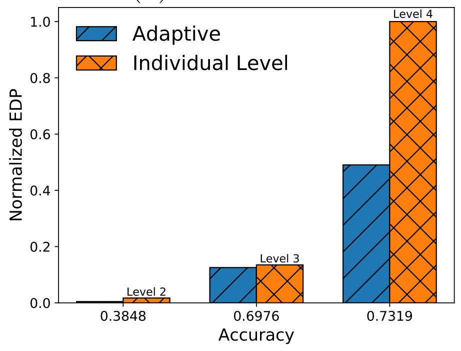
(a) C2F Net A



(b) C2F Net B



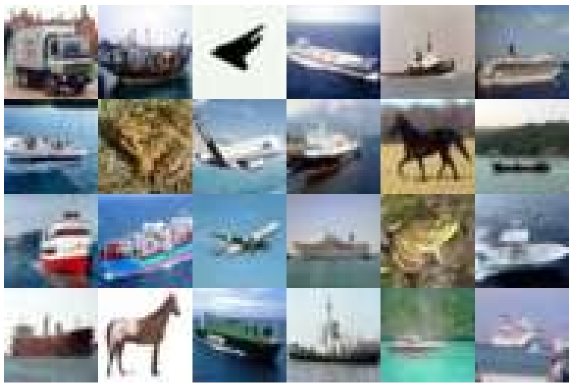
(c) C2F Net C



(d) C2F Net D

Figure 3.8: Energy consumption of adaptive C2F Nets optimized to achieve the accuracy of networks at different levels. Energy values are normalized w.r.t the most complex network.

50%. Similarly, from level 2 to level 1, accuracy drops by 15% and EDP reduces by 30% w.r.t level 3. **C2F Net B:** We achieve an accuracy of 92% when we train and test the base network architecture B (i.e., level 3 network) on CIFAR10 data. We see a 7% drop in accuracy and 85% gain in EDP when we move from level 3 to level 2. **C2F Net C and D:** We see the trend to be similar to Net A and B


(a) Images predicted at level 1 (coarsest network)

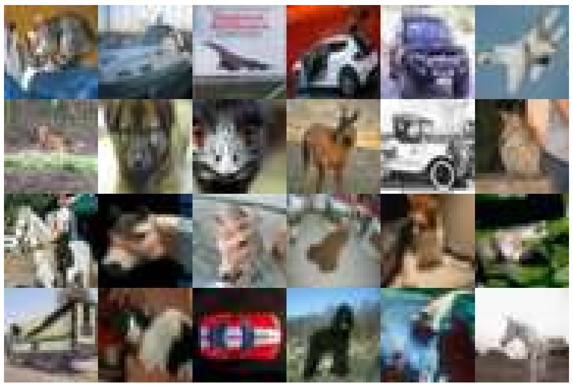
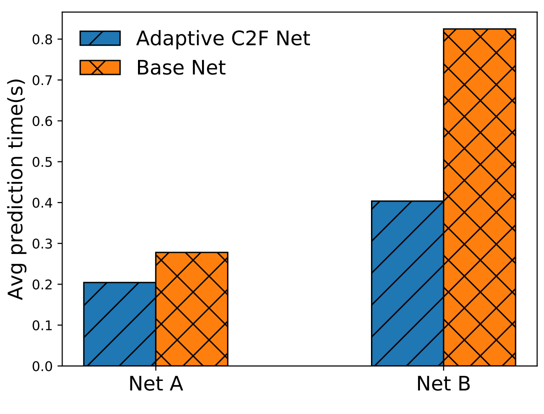

(b) Images predicted at Level 3 (finest network)

Figure 3.9: Predicted images at different levels for C2F Net A

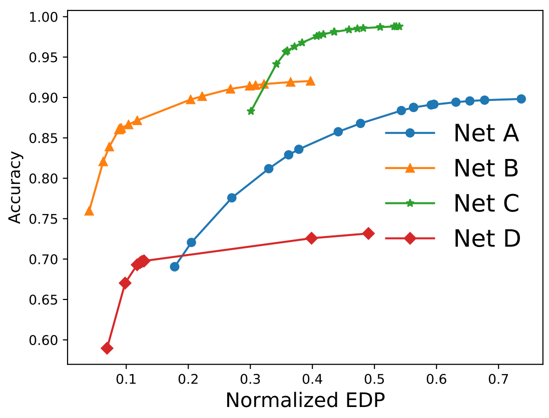
respectively. However, we observe higher accuracies with Net C because MNIST data set is simple and obtain lower accuracies in NET D because CIFAR100 data set is relatively complex.

In summary, we see a significant gain in energy with relatively small loss in accuracy when we move from finest to coarsest network in C2F Nets. This corroborates our hypothesis that we can save significant amount of energy if we are able to select coarser networks for large fraction of easy images and complex networks for hard images that are relatively small.

Optimized C2F Net for accuracy of different networks. We saw that fixed networks take significantly more energy for small improvement in accuracy. On the other hand, optimized C2F Nets can potentially reduce the energy consumption to



(a) Average prediction time in seconds for adaptive C2F Nets and finest (most complex) network. Adaptive C2F Nets are optimized to achieve the same accuracy as the finest network.



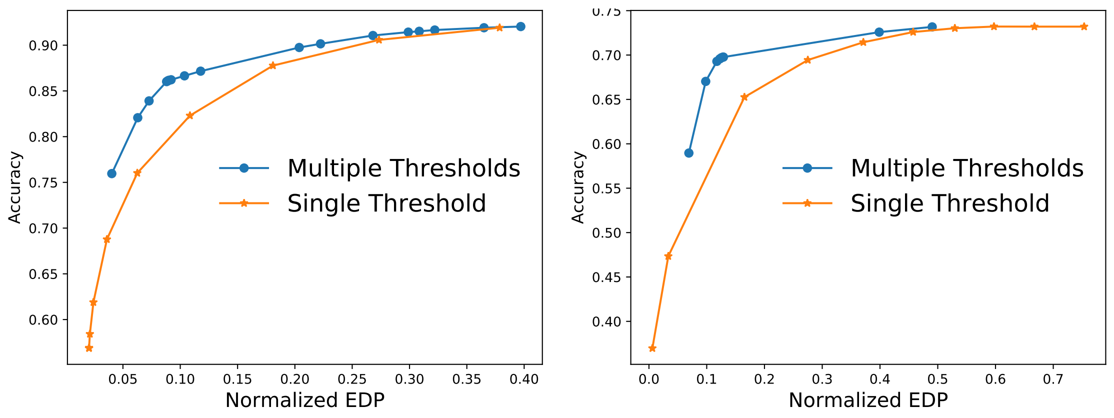
(b) Pareto-optimal curves obtained by varying λ value (trade-off between energy and accuracy) for different C2F Nets: A (CIFAR10), B (CIFAR10), C (MNIST), D (CIFAR100).

Figure 3.10: Prediction times and pareto performance of adaptive C2F nets

achieve the same amount of accuracy by performing input-specific adaptive inference. To test the effectiveness of adaptive C2F Nets, we trained C2F nets with different λ (trade-off) values to find the configurations that achieve the same accuracies as networks at different levels (Figure 3.8).

C2F Net A: Adaptive C2F Net improves the EDP by 26% and 20% to achieve the same accuracies when compared to the base networks at level 2 (89.8% accuracy) and level 1 (85% accuracy).

C2F Net B: We see significantly more improvement in energy gains when com-



(a) C2F Net B (b) C2F Net D

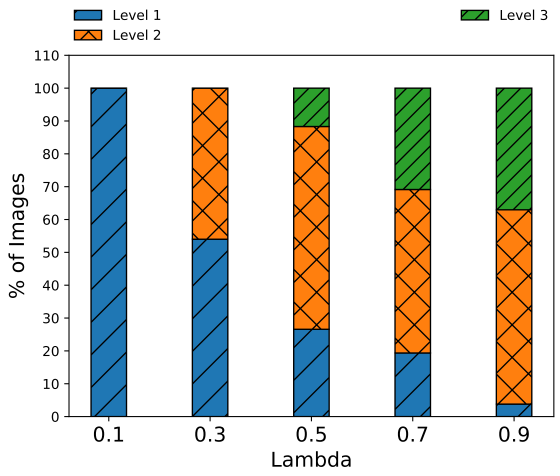
Figure 3.11: Comparison of pareto-optimal curves: single threshold for all levels vs. different thresholds for different levels.

pared to C2F Net A. For example, EDP improves by 60% to achieve 92% accuracy of level 3 (most complex network) and improves by 26% for achieving 85% accuracy of level 2 network.

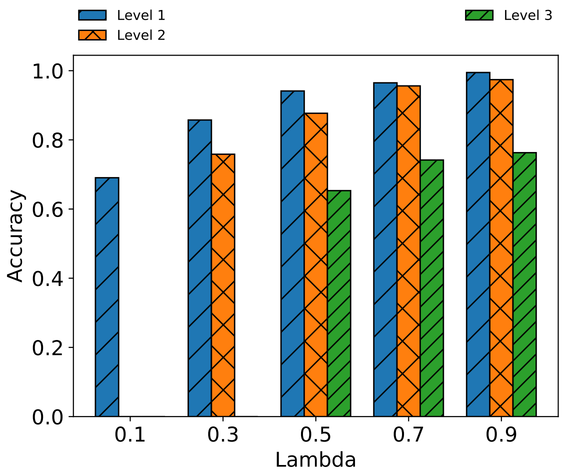
C2F Net C and D: Similar to the results for Net A and Net B, we observe that EDP improves by 46% and 51% respectively with respect to the base network.

In summary, our approach using adaptive C2F Nets can significantly reduce the energy consumption with negligible loss in accuracy when compared to base networks of different complexity that are part of C2F Net. Additionally, the energy gains increase significantly for complicated base networks (e.g. Net B and D).

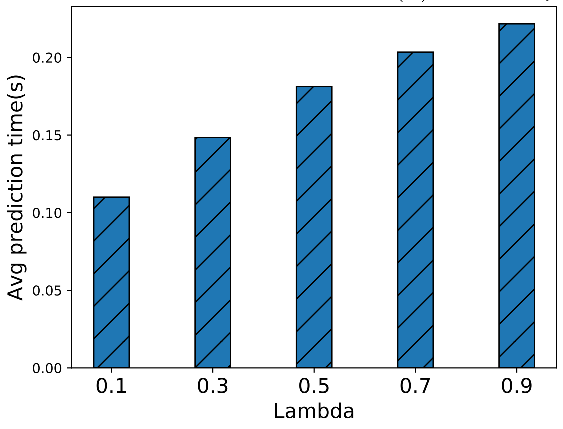
Prediction time of adaptive C2F Net vs. finest network. We compare the average execution time to make predictions using adaptive C2F Nets and the finest



(a) Num Images across Levels



(b) Accuracy across Levels



(c) Prediction time

Figure 3.12: Results for optimized C2F Net A by varying λ values.

(most complex) network, where C2F Nets are optimized to achieve the same accuracy as the finest network. Figure 3.10(a) shows these results. For network A, the average prediction time of base network and adaptive C2F net are ∼0.27 secs and ∼0.20 secs respectively, i.e., 26% reduction in prediction time. Similarly, for network B, the prediction time reduces from ∼0.82 secs to ∼0.40 secs leading to 52% reduction. In summary, our adaptive C2F networks perform better in terms of prediction time to achieve the same accuracy as the base network. Additionally, the improvement is much more for complex networks similar to energy gain results.

Fine-grained analysis of adaptive C2F Nets. We demonstrated that adaptive C2F nets can improve both energy and prediction time when compared to the finest network through the above presented results. We perform fine-grained analysis to understand how adaptive C2F nets achieve these gains. We present this analysis for C2F Net A noting that analysis for other C2F nets show similar trends.

C2F Net A was optimized for achieving the same accuracy (89.8%) as the finest network. The energy gains can be explained by understanding how many images out of the 10000 image testing set are classified at different levels. At level 1, 263 images are classified with 100% accuracy. At level 2, 4911 images are classified with with 98.7% accuracy. At level 3, 4826 images are classified with 80.2% accuracy. Therefore, 2%, 48%, and 49% of the total testing images are classified at levels 1, 2, and 3 respectively. From previous results, we see that the accuracy of 89.8% is obtained using 76% of the energy consumed by the finest network. Levels 1, 2, and 3 contribute 0.46%, 24.5% and 48.26% to this 76% respectively. Similarly, for C2F Net B with four levels, we see 0, 1339, 5517, and 3144 images classified with accuracies of 0, 99.7%, 97.17% and 79.73% and energy consumption of 0.0, 0.49%, 7.7% and 31.4%

respectively. In both cases, more than 50% of the images were predicted by a level other than the last level. Additionally, accuracy of the lower levels is closer to 100%. Therefore, we see huge improvements in energy and prediction time for adaptive C2F Nets.

Qualitative results. Figure 4.1 shows some sample images that are predicted at level 1 (coarsest) and level 3 (finest) using adaptive C2F Net A. We make following observations: 1) Images classified using level 1 are simpler with a single object and clear background; 2) Images classified using level 2 have overlapping or hidden objects with confusing backgrounds. These results demonstrate how adaptive C2F nets use simpler classifiers for easy images and complex classifiers for hard images.

Single threshold vs. Multiple threshold. To measure the quantitative difference between using single threshold for all levels of C2F net (as in CDL [95]) and different thresholds for all levels, we compare their pareto-optimal curves. Figure 3.11 shows the comparative results for adaptive C2F networks B and D. We make the following observations. When the number of levels are more, different thresholds for different levels achieve better pareto-optimal solutions when compared to the setting with single threshold for all levels. Our experimental results on CIFAR10 (Net B) and CIFAR100 (Net D) clearly show this difference. We also present the pareto-optimal solutions we obtained for all adaptive C2F Nets A, B, C and D in Figure 3.10(b) for completeness.

Behavior of optimized C2F Nets with different λ. We can vary the trade-off parameter to obtain optimized C2F nets for different accuracy and energy trade-offs. When λ is close to zero, there is more emphasis on saving energy without paying attention to accuracy. Similarly, when λ is close to one, adaptive C2F nets try to achieve best possible accuracy by minimizing the overall energy consumption.

We study the behavior of optimized C2F nets with different values of λ in terms of the number of images predicted at different levels, the prediction accuracy at different levels, and the average prediction time. Figure 4.11 shows these results for C2F Net A noting that results are similar for other C2F nets.

We make following observations from Figure 4.11(a). When λ=0, all the images are predicted at level 1 as coarsest network consumes the least energy. As the λ value increases, the number of images predicted by level 2 and level 3 slowly increases to achieve higher accuracy. Note that, even with the highest λ value, not all images are predicted by level 2 (finest network) to optimize the energy consumption to achieve high accuracy.

From Figure 4.11(b), we can see that as λ increases, the prediction accuracy of level 1 and level 2 increases proportionally and reach almost 100% accuracy. The overall accuracy of C2F net is primarily determined by the accuracy of level 2 classifier.

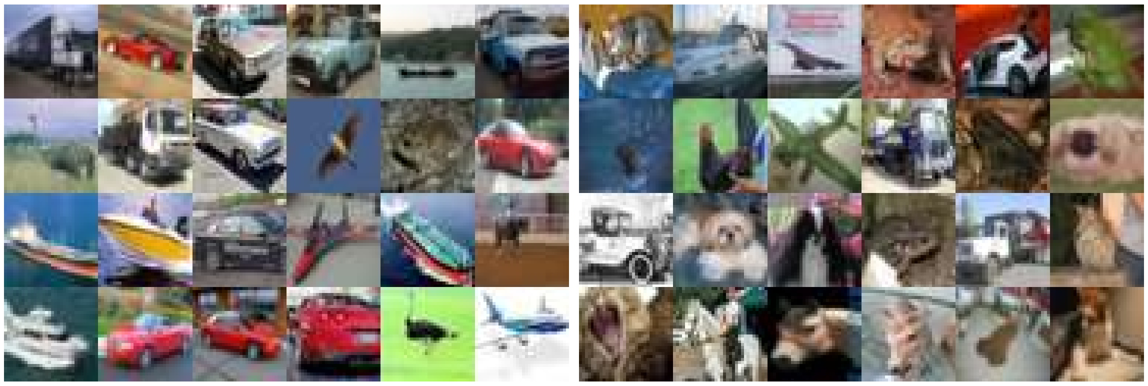
From Figure 4.11(c), we observe that the average prediction time gradually increases with λ, which is explained by more and more images getting predicted at higher levels. When compared to the base (finest) network, the prediction time at λ=0 is 63% lesser and at highest λ value, we get around 26% gain. Therefore, in scenarios where we require real-time predictions, we can optimize the accuracy of C2F nets for the target time constraints.

CHAPTER 4. LEANET: DESIGN AND OPTIMIZATION OF ENERGY-ACCURACY TRADE-OFF NETWORKS VIA PRETRAINED DEEP MODELS

In this chapter, we discuss another instance of adaptive CNN inference, namely LEANet approach [61] that allows us to adaptively select a classifier of appropriate complexity depending on the hardness of the input example. Complimentary to C2F Nets, the LEANets build adaptivity by scaling along the width of the CNN i.e. we increase the number of channels as the complexity of the CNN increases. In the following sections we first give an overview of the framework, followed by the design space and optimization methodology and finally present the experimentation setup and results.

4.1 Overview of LEANet Framework

In this section, we provide a high-level overview of the Learning Energy Accuracy Tradeoff Networks (LEANet) framework. First, we explain the key insights behind LEANet. Second, we formally describe LEANet model and the associated inference procedure.



(a) Images predicted at level 1 (b) Images predicted at level 5

Figure 4.1: Images predicted at lowest and highest levels for LEANet(MobileNet) w/ CIFAR10.

4.1.1 Motivation

Deploying large DNN models on mobile platforms requires lot of resources in terms of computation and energy. DNN architectures like MobileNet address the challenge of constrained-resources on mobile platforms by compressing the network with some loss in accuracy. Even though these architectures address the resource requirements for inference, they still involve a huge cost for training them from scratch for a new real-world (edge) application. Towards the goal of overcoming their drawbacks and for further improving the energy and computational-efficiency of state-of-the-art DNNs like MobileNet, we propose the LEANet model.

LEANet can be interpreted as a wrapper approach that takes a pre-trained DNN model as input and reuses the learned weights to automatically build a multi-level

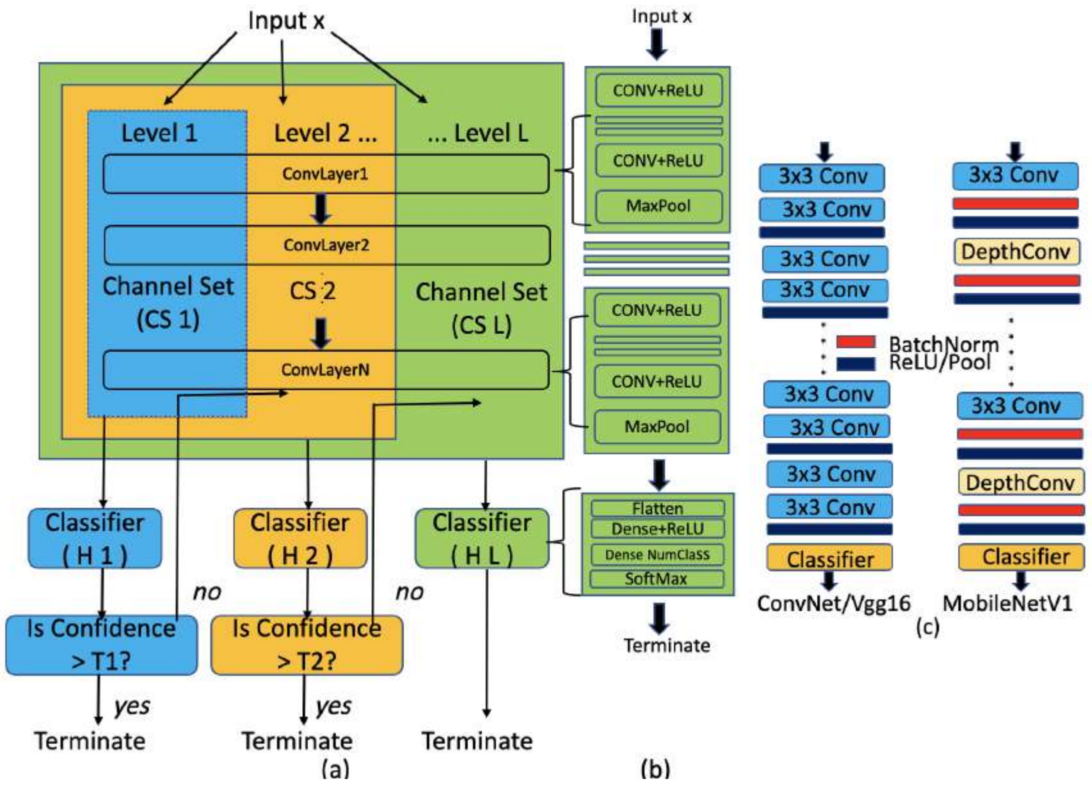


Figure 4.2: Illustration of LEANet model. Given a pre-trained network as shown in (b), we can split into multiple levels with reusable computation from lower levels as shown in (a). The nested boxes for different channel sets represent the reuse of computation from lower levels at higher levels. (c) Neural architectures of some popular DNNs including ConvNet, VGG16, and MobileNet.

DNN, where level number is proportional to the complexity of the corresponding classifier. The key idea is to slice the pre-trained DNN vertically into sets of filter-maps/channels and progressively go from simpler to complex classifier *only if needed* thereby reusing the computation of lower levels at higher levels to achieve upto 50%

reduction in energy-consumption and execution time with little to no loss in accuracy.

Figure 4.1(a) and (b) shows some sample images from our experiments that are predicted at level 1 and level 5 respectively using an instantiation of LEANet with five levels based on MobileNet. Images classified using level 1 are simpler with a single object and clear background while images classified using level 5 have overlapping, skewed or hidden objects with confusing backgrounds. These results provide evidence for the key intuition behind our proposed LEANet approach.

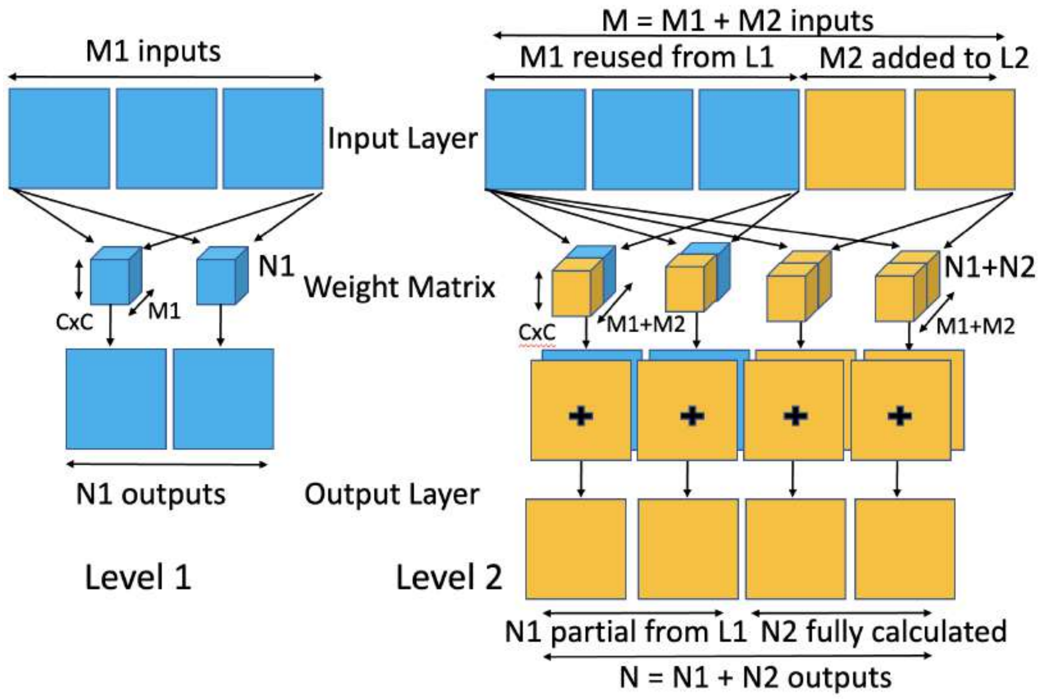


Figure 4.3: Illustration of a two-level LEANet model with one convolution layer. The blue part from level 1 is reused in level 2.

4.1.2 LEANet Model and Inference

A LEANet model $\mathcal{M}$ with L levels is a sequence of classifiers of growing complexity resulting from increased number of channels with growing levels as shown in Figure 4.2. There are three key components for each level in a LEANet model.

1) Channel sets: This comprises of a fraction of learned filter-maps/channels for each layer in the pre-trained DNN. Each successive level employs an increasing number of channels. The parameters (denoted by CS_i) of channel sets are reused from the given pre-trained DNN in the LEANet model. It takes the features computed in previous level L_{i-1} and input image (x) as input, and produces complex features L_i.

2) Classifier: A classification layer that takes the features computed by a given channel set and produces a probability distribution over all labels. In LEANet model, we only learn the parameters (denoted by H_i) of this classification layer.

3) Confidence threshold. Given a predicted probability distribution over candidate labels, we can estimate the confidence of the classifier [112] using the *maximum probability*, where we take the probability of the highest scoring label. The confidence threshold T_i is employed to determine if the classifier is confident enough in its prediction to terminate and return the predicted label.

Inference in LEANet model. Given a LEANet model $\mathcal{M}$ with all its parameters (CS_i, H_i, T_i for all levels $1 \leq i \leq L$) fully specified, inference computation for input example x is performed as follows. We sequentially go through the LEANet model starting from level 1. At each level i, the channel set CS_i takes the input x and reuses the features computed from the previous level CS_{i-1} to produce the features of CS_i. Then classifier layer H_i makes predictions using the features resulting from the given channel set CS_i. If the estimated confidence parameter $\hat{T_i}$ meets the threshold T_i or we reach the final level L, we terminate and return the predicted output $\hat{y}$.

4.2 Design Space of LEANet Models

In this section, we describe the design space of candidate LEANet models that can be constructed using a given pre-trained DNN (e.g., MobileNet). For the sake of understanding, we first explain the case of LEANet model with two levels and then generalize the space for more than two levels. We employ CNNs for image classification as a running example for illustrative purposes.

As shown in figure 4.2(a), we divide the convolution layers of the given pre-trained neural network (CNN) into multiple levels by using increased number of channel sets with growing levels. Any general CNN architecture is composed of several convolution layers as shown in Figure 4.2(b). Let a convolution layer consists of N channels.For a L level LEANet, our method selects n_i channels of each convolution layer for each level i, where $\forall\ j > i, \sum_{k=1}^{j} n_k > \sum_{k=1}^{i} n_i$, and $\sum_{k=1}^{n} n_k = N$. Intuitively, each successive level builds a classifier of increasing complexity resulting from the increased number of channels n_i in each layer.

Two-Level LEANet. For illustration, we construct a two level LEANet. Consider a convolution layer with a filter of width (C), height (C), and depth (M) convolving over an input layer (I) of M Channels to produce an output layer (O) of N channels. We describe splitting at two levels using M_1 input channels and N_1 output channels for level L_1, and $M_1 + M_2 = M$ input channels and $N_1 + N_2 = N$ output channels for level L_2. In level L_1, first N_1 output channels (O_n) are obtained from the first M_1 input channels (I_m) as given by:

$$O_n^{L_1}[a, b] = \sum_{m=1}^{M_1} \sum_{i=-C/2}^{C/2} \sum_{j=-C/2}^{-C/2} I_m[i + a, j + b] \cdot w_{mn}[i, j] \tag{4.1}$$

$$\forall n \in 1, 2 \ldots N_1$$

, where w_n represents filter weights. The above equation can be re-written as:

$$\begin{aligned} O_n^{L_1}[a, b] &= \sum_{m=1}^{M_1} C_m^*[a, b] \quad \forall n \in 1, 2 \ldots N_1 \\ C_m^*[a, b] &= \sum_{i=1}^{C} \sum_{j=1}^{C} I_m[i + a, j + b] \cdot w_n[i, j] \end{aligned} \tag{4.2}$$

, where C_m^* represents the convolution operation.

In level L_2, we consider all N output channels. The equation for the first N_1 output channels is as follows:

$$O_n^{L_2}[a,b] = \sum_{m=1}^{M_1+M_2} C_m^*[a,b] \quad \forall n \in 1, 2 \ldots N_1$$

$$O_n^{L_2}[a,b] = \sum_{m=1}^{M_1} C_m^*[a,b] + \sum_{p=M_1+1}^{M_2} C_p^*[a,b] \tag{4.3}$$

, where the first part in equation (3) can be reused from level L_1. This reuse of computation is illustrated in Figure 4.3. Furthermore, the remaining N_2 new output channels used in L_2 can be computed similar to equation (1).

For a numerical illustration, consider a 2 level LEANet setup for a convolution layer with $M = 48$ input channels and $N = 96$ output channels. Let $M_1 = 32$, $N_1 = 64$, $M_2 = 16$ and $N_2 = 32$. The computation in L_2 is proportional to MN of which we are reusing M_1N_1 part from L_1. Therefore, the reuse ratio is

$$\frac{M_1 \cdot N_1}{M \cdot N} = \frac{M_1 \cdot N_1}{(M_1 + M_2) \cdot (N_1 + N_2)} = \frac{32 \cdot 64}{48 \cdot 96} = 0.44$$

The illustrative demonstration of convolutional layer with two levels show how much we are reusing the computation from previous level to the next level. Equation 3 corresponds to this demonstration without pooling and non-activation layers. Specifically, we are not reusing the solutions from pooling and non-activation layer operations. Instead, the partial output of convolutional layer from previous level i-1 and the new convolutions at level i are *jointly* passed through the pooling and non-activation layers (negligible computational overhead) to compute the input for the next convolutional layer for level i. We reuse the computation from the previous level in the next level to generate more features. Importantly, *these features are not necessarily equivalent to the features generated from scratch at each level in the base DNN*. The classifier used at the end of each level can compensate for the difference in the features generated (exact vs. approximate) by learning appropriate weights to be able to make correct predictions. Hence, learning classifiers tuned for approximate features allow us to trade-off energy and accuracy of inference in our LEANet approach.

Multi-Level LEANet. We can easily generalize the idea of LEANet with two levels to multiple levels ($L > 2$). As illustrated in Figure 4.3, while moving across successive levels, LEANet models reuse the computation from previous levels. To describe the general expression for this computational reuse factor consider a multi-level LEANet model with $L > 2$ levels. Without loss of generality, consider a convolutional layer which is split into M_i input channels and N_i output channels corresponding to each level $1 \leq i \leq L$. Extending equation (5), the reuse of computation going from level i to $i+1$ is given by:

$$\frac{\sum_{j=1}^{i} M_j \cdot \sum_{j=1}^{i} N_j}{\sum_{j=1}^{i+1} M_j \cdot \sum_{j=1}^{i+1} N_j} \tag{4.4}$$

Though most networks use normal convolution, as shown in Figure 4.2 (c), networks like MobileNet employ different types of layers. We now extend the LEANet splitting criterion to other layer types. *Pointwise 2D Convolution* is similar to a $C \times C$ convolution layer with $C = 1$. Thus the reuse factor remains same since M and N does not change. *Depth-wise 2D Convolution* is another special case where the input channels ($M = 1$). Therefore, we have a single parameter N output layers in LEANet. This makes the reuse factor $\frac{\sum_{j=1}^{i} N_j}{\sum_{j=1}^{i+1} N_j}$ *Batch Normalization* too has $M = 1$ like Depth-wise 2D convolution. This layer has four different parameters given by gamma weights, beta weights, moving mean (non-trainable), and moving variance (non-trainable). Thus, based on the above details, while constructing multiple levels, we carefully map the appropriate input channels to appropriate output channels across levels to maximize the computational reuse, thereby minimizing the energy consumption of the overall LEANet model.

4.3 Design Optimization Methodology

In this section, we describe the details our optimization methodology to find the best LEANet model in terms of the Pareto front to trade-off energy and accuracy of inference.

Design Space Definition. As explained in Section 4.2, each candidate LEANet model in our design space depends on three variables: number of levels (L), fraction of channels in channel set CS_i of each level given by $(F_1, F_2, \cdots, F_L)$, and confidence thresholds $(T_1, T_2, \cdots, T_{L-1})$ for each classifier H_i. Each F_i is equal to N_i/N where N_i is the number of channels in level i and N is the total number of channels. This is a conditional search space in the sense that the size of fraction-of-channels vector and confidence thresholds vector depend on the number of levels L.

Optimization problem. Each candidate LEANet model (configuration of the three variables mentioned in the design space definition) corresponds to a accuracy and energy pair over a given evaluation set of classification examples. Given a pre-trained DNN $\mathcal{N}$, training set TSR, and validation set VA of classification examples, our goal is to find the LEANet model configuration, which converges to the optimal energy and accuracy trade-off Pareto front for a given target mobile platform $\mathcal{MP}$.

Optimization methodology. As described in Algorithm 6 to obtain the optimal Pareto front, we iterate over the number of levels L starting from L=2 until convergence. We determine the fraction of channels at each level i=1 to L as follows. The final level L corresponds to the input pre-trained DNN $\mathcal{N}$ that will achieve the highest accuracy. The fraction of channels at level one (F_1) is based on the *minimum energy budget* of target mobile platform $\mathcal{MP}$, which we want to use to be able to perform inference using an appropriately granularity of DNN. The highest level (level L) will employ all the channels (F_L=1.0x): consumes the maximum energy to provide highest accuracy. The fraction of channels at levels ($i > 1$) are obtained by allocating the remaining channels equally for all the remaining L-2 levels. For example, if the minimum energy budget of mobile platform allows us to run at 0.5x of the base DNN, it would correspond to the lowest level. For 3 levels, fraction of channels in channel sets would be [0.5x, 0.75x, 1.0x] and for 4 levels, fraction of channels in channel sets would be [0.5x, 0.675x, 0.875x, 1.0x].

Given the number of levels L and fraction of channels in each level $(F_1, F_2, \cdots, F_L)$, we train the classifiers $H_1, H_2, \cdots, H_{L-1}$ with the training data $\textsc{TSr}$ using stochastic gradient descent and back-propagation. H_L corresponds to pre-trained DNN. Subsequently, we determine a set of confidence threshold vectors $\mathcal{T}_L$ based on the target mobile platform $\mathcal{MP}$ (to compute energy consumption) and the validation data $\textsc{Va}$ (to compute accuracy) using a novel multi-objective Bayesian Optimization (BO) approach referred as *UnPAc*. Each threshold vector $(T_1, T_2, \cdots, T_{L-1}) \in \mathcal{T}_L$ corresponds to a particular trade-off between energy and accuracy. We observed diminishing returns as we increase the number of levels and Pareto fronts converge in at most 5 levels. Our overall optimization procedure including identifying the number of levels, training the classifiers at all levels, and finding the set of confidence threshold vectors corresponding to the energy and accuracy trade-offs is executed *offline*. At *runtime*, we can select the appropriate confidence threshold vector for the required energy and accuracy trade-off.

Algorithm 6 LEANet Design Optimization

```
Input: $\mathcal{N}$ = Pre-trained deep model; TSr= Training set; Va= Validation set; and $\mathcal{MP}$ = Target mobile platform
1: Initialization: number of levels $L$=2
2: while Convergence do
3:     Set channel fractions $F_1, F_2, \cdots, F_L$ via domain knowledge
4:     Train classifiers $H_1, H_2, \cdots, H_{L-1}$ using TSr
5:     Threshold vectors $\mathcal{T}_L \leftarrow$ UnPAc($\mathcal{N}, \mathcal{MP}$, Va) // Find the optimal Pareto front of energy and accuracy
6:     $L \leftarrow L + 1$ // Increase the number of levels
7: return optimized LEANet model with $L$ levels, classifiers at all levels, and confidence threshold vectors $\mathcal{T}_L$ to trade-off energy and accuracy
```

4.3.1 *UnPAc Algorithm to Estimate Thresholds*

For a given number of levels L and the trained intermediate classifiers of a candidate LEANet model, our goal is to find a set of threshold vectors $\mathcal{T}_L$ that correspond to the Pareto front of energy and accuracy trade-off. There are two key challenges in solving this problem: 1) The energy and accuracy objective functions are unknown and we need to perform experiments to evaluate each candidate threshold vector $(T_1, T_2, \cdots, T_{L-1}) \in \mathcal{T}$. In our specific problem, evaluation of both the objectives is expensive: evaluation of the energy objective on target mobile platform $\mathcal{MP}$ is much more expensive than the accuracy objective; and 2) The objectives are conflicting in nature and they cannot be optimized simultaneously. Therefore, we need to find the *Pareto optimal* set of solutions. A solution is called Pareto optimal if one objective cannot be improved without compromising other objectives. The overall goal is to approximate the true Pareto set by minimizing the overall number of energy and accuracy evaluations.

Multi-Objective Bayesian Optimization Formulation. We formulate the above problem in the framework of multi-objective Bayesian Optimization (BO): the input space $\mathcal{T}$ of all candidate threshold vectors is $[0,1]^{L-1}$; and energy and accuracy over the validation set VA and target mobile platform $\mathcal{MP}$ as the two unknown objective functions. BO algorithms learn cheap surrogate models (e.g., Gaussian Process) from training data obtained from past evaluations of objective functions. They judiciously select the next candidate input $(T_1, T_2, \cdots, T_{L-1}) \in \mathcal{T}$ for evaluation by trading-off exploration and exploitation to quickly direct the search towards approximating the true Pareto front. Acquisition functions are defined in terms of the statistical models to score the candidate inputs and guide this search process. PESMO [54] is the state-of-the-art approach to solve multi-objective BO problems based on entropy optimization. PESMO reduces the problem to single-objective optimization. It iteratively selects the input that maximizes the information gained about the true Pareto set. Unfortunately, this choice is sub-optimal as it is hard to capture the trade-off between multiple objectives and can potentially lead to aggressive exploitation behavior. Indeed, our experiments demonstrate the inefficiency of PESMO. Motivated by the need for a more sample-efficient optimization approach, we propose a novel algorithm referred as *Uncertainty Reduction via Portfolio Acquisition optimization (UnPAc)*.

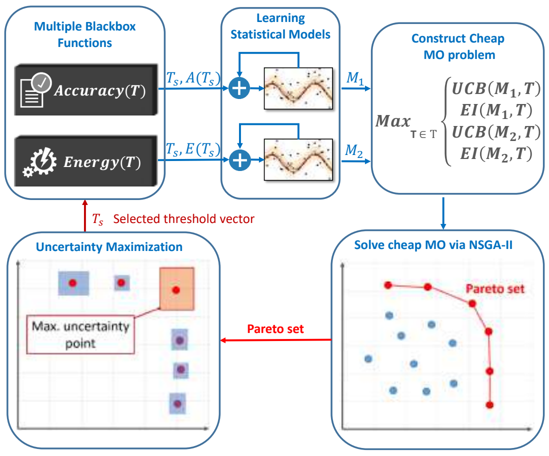

Multiple Blackbox Functions
Accuracy(T)
Energy(T)
Learning Statistical Models
$T_s, A(T_s)$
$T_s, E(T_s)$
M_1
M_2
Construct Cheap MO problem
$Max_{T \in \mathrm{T}} \begin{cases} UCB(M_1, T) \\ EI(M_1, T) \\ UCB(M_2, T) \\ EI(M_2, T) \end{cases}$
T_s Selected threshold vector
Uncertainty Maximization
Max. uncertainty point
Pareto set
Solve cheap MO via NSGA-II
Pareto set

Figure 4.4: Overview of the UnPAc algorithm.

UnPAc Algorithm. As shown in Figure 4.4, UnPAc is a iterative algorithm which involves four key components as described below. The energy and accuracy objective functions are considered as the *two blackbox functions* to be optimized.

1) Learning statistical models. We build statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$ for both unknown objective functions *accuracy* and *energy* from the training data in the form of past function evaluations. In each iteration of UnPAc, the learned statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$ are employed to select the next candidate threshold vector $(T_1, T_2, \cdots, T_{L-1}) \in \mathcal{T}$ for evaluation.

2) Constructing cheap MO problem. We select a set of promising candidate inputs $\mathcal{T}_p \subset \mathcal{T}$ by solving a cheap multi-objective (MO) optimization problem defined using the statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$. Specifically, we employ two acquisition functions for each statistical model: upper confidence bound (UCB) and expected improvement (EI) as defined below.

$$UCB(x) = \mu(x) + \beta^{1/2}\sigma(x) \tag{4.5}$$

$$LCB(x) = \mu(x) - \beta^{1/2}\sigma(x) \tag{4.6}$$

$$EI(x) = \sigma(x)(\alpha\Phi(\alpha) + \phi(\alpha)) \tag{4.7}$$

, where $\alpha = \frac{\tau - \mu(x)}{\sigma(x)}$; $\mu(x)$ and $\sigma(x)$ correspond to the mean and standard deviation of the prediction from statistical model, and represent exploitation and exploration scores respectively; β is a parameter that balances exploration and exploitation , which is automatically specified based on the iteration number t based on theoretical analysis of convergence from Srinivas et al. [110]; τ is the best uncovered input; and Φ and ϕ are the CDF and PDF of normal distribution respectively.

3) Solving the cheap MO problem. We use the popular NSGA-II algorithm [**?**] to solve the cheap MO poblem to obtain the Pareto set $\mathcal{T}_p$ representing the most promising candidate inputs for evaluation.

$$\mathcal{T}_p \leftarrow \max_{T \in \mathcal{T}} \left(\text{UCB}(\mathcal{M}_1, T), \text{EI}(\mathcal{M}_1, T), \text{UCB}(\mathcal{M}_2, T), \text{EI}(\mathcal{M}_2, T)\right) \tag{4.8}$$

The acquisition functions for each unknown function tells us the utility of evaluating a candidate threshold vector. For example, a threshold vector might have a high utility for accuracy, but may have a lower utility for optimizing energy consumption. Therefore, the Pareto set found by solving cheap MO problem captures the trade-off between the utility for both unknown objective functions.

4) Uncertainty maximization. From the promising candidate set$\mathcal{T}_p$ obtained by solving the cheap MO problem, we need to select the best threshold vector such that it will guide the overall search towards the goal of quickly approximating the true Pareto set. We employ a uncertainty measure defined in terms of the statistical models $\mathcal{M}_1, \mathcal{M}_2$ to select the most promising candidate input for evaluation. We define the uncertainty measure as the volume of the uncertainty hyper-rectangle.

$$U_{\beta_t}(T) = VOL(\{(LCB(\mathcal{M}_i, T), UCB(\mathcal{M}_i, T)\}_{i=1}^{2}) \tag{4.9}$$

, where LCB$(\mathcal{M}_i, x)$ and UCB$(\mathcal{M}_i, T)$ represent the lower confidence bound and upper confidence bound of the statistical model $\mathcal{M}_i$ for threshold vector T as defined in equations 4.6 and 4.7. We measure the uncertainty volume measure for all threshold vectors $T \in \mathcal{T}_p$ and select the one with maximum uncertainty for evaluation: $T_s = \arg max_{T \in \mathcal{T}_p} U_{\beta_t}(T)$.

Dataset[DNN]	Fraction of channels(Width)
CIFAR10[Conv/VGG/MobileNet]	0.55x 0.625x 0.75x 0.875x 1x
MNIST[Conv/VGG/MobileNet]	0.5x 0.625x 0.75x 0.875x 1x
SVHN[Conv/VGG/MobileNet]	0.5x 0.625x 0.75x 0.875x 1x

Table 4.1: The number of levels L and the fraction of channels (width) at different levels when LEANet optimization approach converges.

Finally, this selected threshold vector T_s is used for evaluation to get the corresponding energy $E(T_s)$ and accuracy $A(T_s)$. The next iteration starts after the statistical models $\mathcal{M}_1$ and $\mathcal{M}_2$ are updated using the new training example: input is T_s and output is $(E(T_s), A(T_s))$. Unlike prior methods including PESMO that reduce to single-objective optimization, UnPAc follows the above procedure to select better candidates for evaluation in each iteration. Recent work on multi-objective optimization [12] showed that output space entropy search is more effective than input space entropy search. It would be interesting future work to see how UnPAC compares to this approach.

4.4 Experiments and Results

In this section, we first describe the experimental setup and then discuss results of LEANet along different dimensions.

4.4.1 Experimental Setup

Hardware Setup. All our experiments were performed by deploying DNN models on an `ODROID-XU4` board [93]. `ODROID-XU4` is an Octa-core heterogeneous multi-processing system with ARM Big-Little architecture, which is very popular in current mobile devices. The `ODROID-XU4` board employs ARM Cortex-A15 Quad 2GHz CPU as the Big Cluster and Cortex-A7 Quad 1.4GHz CPU as the Little Cluster. The board boots up Ubuntu 16.04LTS with ODROID's Linux kernel version 3.10.y. We execute DNN models with Caffe-HRT [17] that employs Caffe framework [68] with ARM Compute Library to speed up deep learning computations. We ran the software programs for our LEANet approach, full DNN, and SlimNets baseline on the mobile platforms to measure the energy consumption. Therefore, all the presented results are relative to the common software execution scheme. Studying optimized execution schemes for LEANet models to minimize memory overhead is part of our immediate future work. We employ SmartPower2 [108] to measure power. We compute the average power over the total execution time.

Energy objective. Energy consumption of LEANet models is measured in terms of the normalized energy delay product (EDP). EDP = $\sum$P Δt $\cdot$ T, where Δt is the time interval at which we record the power P and T is the total execution time. As power is measured at a regular interval, we simply calculate EDP as EDP = $P_{avg} \cdot T^2$. This value is normalized with EDP of the pre-trained DNN. We use EDP and energy interchangeably in the discussion of our results. However, all our presented results are for EDP unless specified otherwise.

Image classification task and training data. We demonstrate the general applicability of our LEANet based approach using three different pre-trained DNNs as shown in Figure 4.2 (c): 1) Conv, a VGG style neural network with 6 convolution layers; 2) VGG, a more complex network with 10 convolution layers; and 3) MobileNet. We ran extensive experiments using MNIST, CIFAR10, and SVHN image datasets to study the performance gains with classification tasks of varying difficulty. We used the standard 4:1:1 split ratio for training, validation, and testing (10000) set respectively. Accuracy is measured as the fraction of input images whose labels are predicted correctly. We present all our results in terms of accuracy and normalized EDP for the ease of exposition. To demonstrate the effectiveness of LEANets on large datasets, we ran experiments using the ImageNet dataset [101]. ImageNet has a training set of roughly 1.2 million images and around 50000 validation and test images. Each image has 224×224×3 dimensionality and maps to one of the 1000 candidate class labels. We present the results for ImageNet in terms of top-5 accuracy (commonly employed metric in the literature) and normalized EDP.

Classifier Training. We employ a multi-layer Perceptron to learn classifiers at intermediate levels (H_i) as shown in Figure 4.2(b). In this multi-layer Perceptron, input corresponds to the output of the final convolution layer and number of output class labels is same as that of the classification task. The hidden layer is given a fixed value as a function of the size of the input feature vector. In practice, half the size of input feature vector gives consistent results in all our experiments. To train the parameters of classifier (H_i) at different levels, we employ backpropagation using RMS prop optimizer with learning rate of 0.0001 and a decay set to $1e^{-6}$. We use a batch size of 128 and run training for 200 epochs with sufficient data augmentation including horizontal flips, width, and height shifts. As shown in Fig. 4.9, our design optimization algorithm converges at five levels in all cases. Therefore, for each pre-trained DNN, we have to train four classifiers: 200 training epochs for only classifier layer, which has a very negligible overhead (around two percent of the computational resources needed to train the full network).

Multi-objective BO for LEANet models. For obtaining the confidence threshold vectors to trade-off energy and accuracy, we employ different mutli-objective BO algorithms. We compare our UnPAc algorithm with the state-of-the-art PESMO method [54].

We employ the PESMO code from the BO library Spearmint[1]. We use a GP based statistical model with squared exponential (SE) kernel in all our experiments. The SE kernel is defined as $\kappa(x, x') = s\cdot \exp(\frac{-\|x-x'\|^2}{2\sigma^2})$, where s and σ correspond to scale and bandwidth parameters. These hyper-parameters are estimated after every

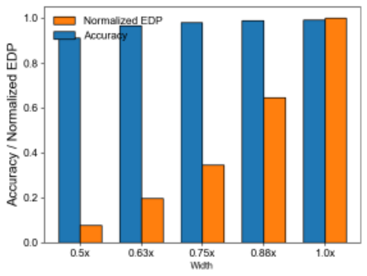
[1]https://github.com/HIPS/Spearmint/tree/PESM

10 function evaluations. We initialize the GP models for both objective functions by sampling initial points at random from a Sobol grid. This initialization procedure is same as the one in-built in the Spearmint library. We run for a maximum of 200 iterations.

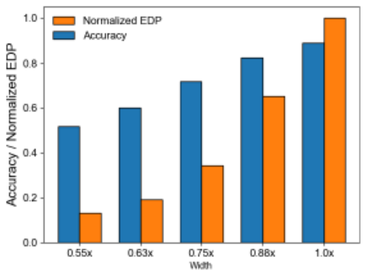
Pareto hypervolume (PHV) metric. PHV is a commonly employed metric to measure the quality of a given Pareto front [123]. PHV is defined as the volume between a reference point and the given Pareto front. After each BO iteration t (or number of function evaluations), we report the difference between the hypervolume of the ideal Pareto front ($\mathcal{T}^*$) and hypervolume of the estimated Pareto front ($\mathcal{T}_t$) by a given algorithm: $PHV_{diff} = PHV(\mathcal{T}^*) - PHV(\mathcal{T}_t)$. Since PHV_{diff} represents a regret function, when comparing algorithms, the smaller the better (i.e., the estimated Pareto front is closer to the ideal Pareto front).

4.4.2 Results and Discussion

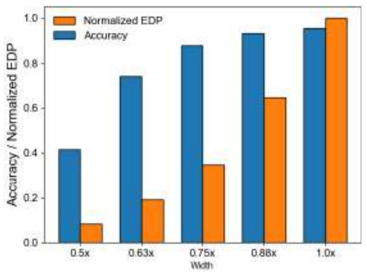
In this section, we first present the performance of a variant of state-of-the-art method based on SlimNets followed by a discussion of improvements achieved by optimized LEANets. Subsequently, we explain the results of our design optimization methods including its convergence behavior and comparison of different multi-objective BO algorithms (UnPAc vs. PESMO). Finally, we present fine-grained analysis of optimized LEANet models.



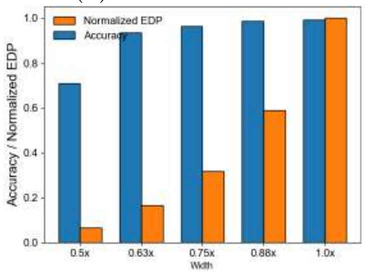
(a) Conv-MNIST



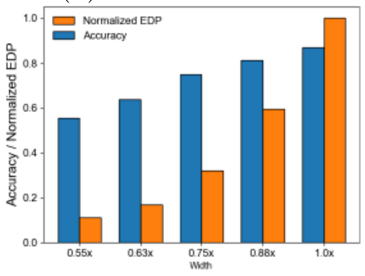
(b) Conv-CIFAR10



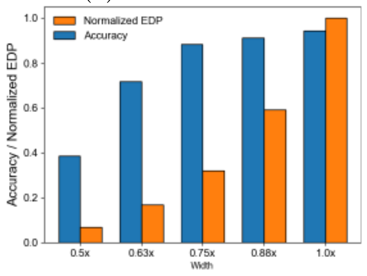
(c) Conv-SVHN



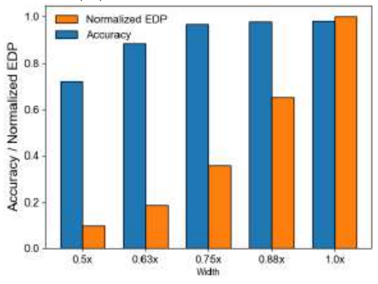
(a) VGG-MNIST



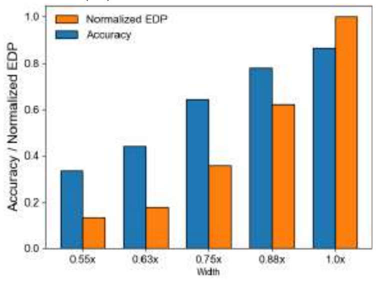
(b) VGG-CIFAR10



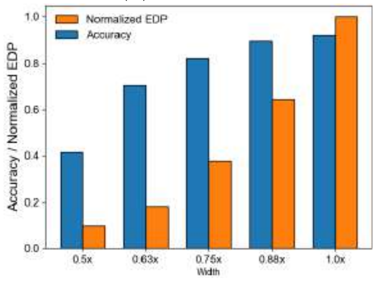
(c) VGG-SVHN



(a) MobileNet-MNIST



(b) MobileNet-CIFAR10



(c) MobileNet-SVHN

Figure 4.5: Energy and accuracy of SlimNets with 5 different widths for each DNN and dataset. For CIFAR10, we use [0.55x 0.625x 0.75x 0.875x 1x] and for SVHN/MNIST, we use [0.5x 0.625x 0.75x 0.875x 1x]

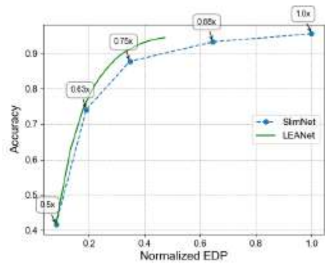
Energy and accuracy trade-offs with varying fraction of channel. We employ each classifier at different levels of LEANet as shown in Table 4.1 to obtain the corresponding energy and accuracy values by performing *static* inference over all the testing examples. This gives us one energy and accuracy values pair for each level. It is easy to see that this setting is conceptually similar to slimmable neural networks (SlimNet) with fixed width values *without* the corresponding training to optimize its accuracy averaged for all widths [119]. For example, SlimNet(0.5x) corresponds to neural network with a fraction of 50% channels. Figure 4.5 shows the energy and accuracy trade-offs obtained using SlimNets with different widths. In this case, the trade-off is obtained at five discrete points as per the widths shown in Table 4.1 for different datasets. We present the energy and accuracy trade-off patterns for different datasets and pre-trained DNN models. **SlimNet(Conv/SVHN):** From width 1.0x to 0.875x, we can see that accuracy drops by 2% and energy reduces by 35%. Similarly, from width 0.875x to 0.75x, accuracy drops by 8% and energy reduces by 65% w.r.t SlimNet(1.0x). **SlimNet(MobileNet/SVHN):** We see a 2.5% drop in accuracy and 35% gain in energy when we move from width 1.0x to 0.875x. **SlimNet(MobileNet/CIFAR10):** For a difficult dataset like CIFAR10, from width 1.0x to 0.875x, we can see that accuracy drops by 8.5% and energy reduces by 38%. Similarly, from width 0.875x to 0.75x, accuracy drops by 22% and energy reduces by 65% w.r.t SlimNet(1.0x). SlimNet(VGG) and dataset MNIST also shows similar

DNN/Dataset	Percentage of evaluations
VGG [SVHN/MNIST/CIFAR10]	1.25%
MobileNet [SVHN/CIFAR10]	1.25%
MobileNet [MNIST]	6.25%

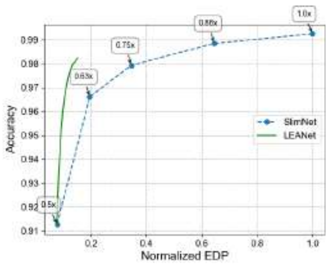
Table 4.2: Results of UnPAc algorithm in terms of the percentage of candidate threshold vectors (w.r.t total evaluations by scalarization-based BO) evaluated to reach convergence.

trends. In summary, we see significant gain in energy and relatively smaller loss in accuracy as we decrease the width in SlimNet irrespective of the difficulty of dataset.

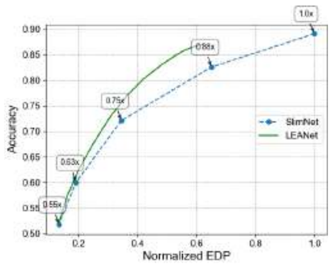
Optimized LEANet vs. SlimNets. Figure 4.6 shows the comparison of optimized LEANet and SlimNets as per the five level widths shown in Table 4.1. We compare them in terms of the Pareto front for energy and accuracy trade-offs. We can see that optimized LEANet outperforms SlimNets in terms of the achievable trade-offs for all cases. When compared to SlimNets, LEANets achieve a more fine-grained trade-off between energy and accuracy by varying the threshold vectors employed to make classification decisions. We observe that LEANets reach the same accuracy values as SlimNet(1.0x) with a significant gain in energy. On the SVHN dataset, LEANet(MobileNet) and LEANet(VGG) achieve around **35%** and **45%** gain in en-



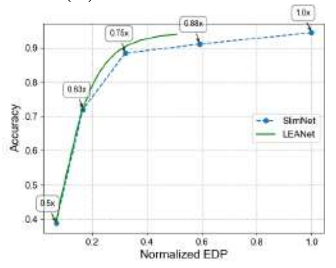
(a) Conv-SVHN



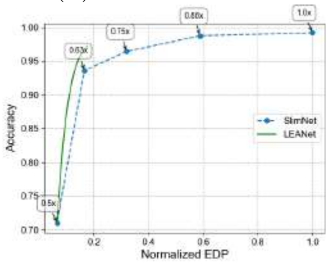
(b) Conv-MNIST



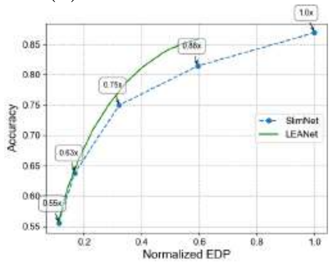
(c) Conv-CIFAR10



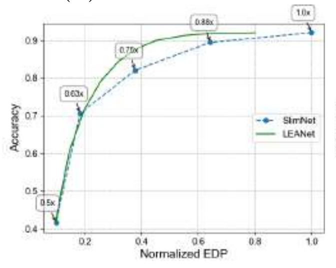
(d) VGG-SVHN



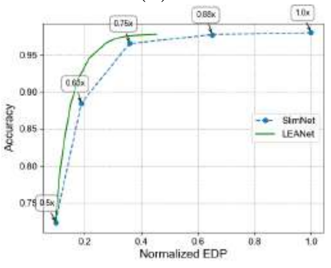
(e) VGG-MNIST



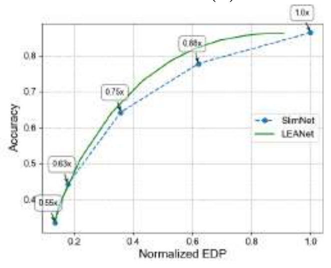
(f) VGG-CIFAR10



(g) MobileNet-SVHN



(h) MobileNet-MNIST

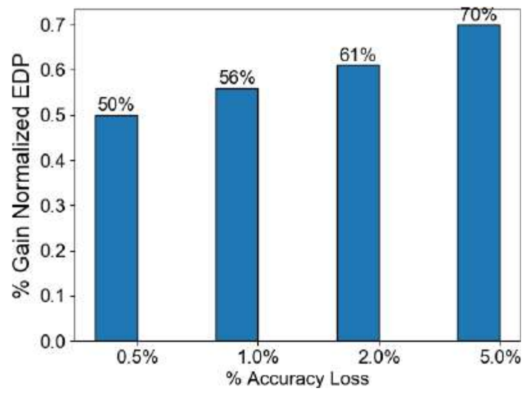


(i) MobileNet-CIFAR10

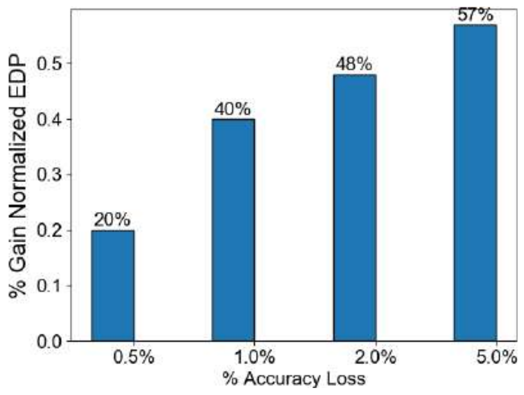
Figure 4.6: Energy and accuracy trade-off results comparing optimized LEANet and SlimNets of different widths.

ergy for achieving the maximum possible accuracy, i.e., SlimNet(1.0x). For CIFAR10, the corresponding energy gains are around **10%** and **40%** respectively. We see a similar trend with LEANet(ConvNet) and MNIST datasets. We observe that the energy gains we get with MNIST and SVHN are higher than those with CIFAR10. The significant energy gains using optimized LEANet is explained by the fact that many *easy* images are predicted at lower levels and only *hard* images are classified at higher levels. Remarkably, our LEANet based approach is able to get energy savings even on MobileNet, which is hand-designed to save energy. In figure 4.10, we show the gains in inference time obtained using optimized LEANets in comparison with SlimNets baseline and observe a similar trend as energy.

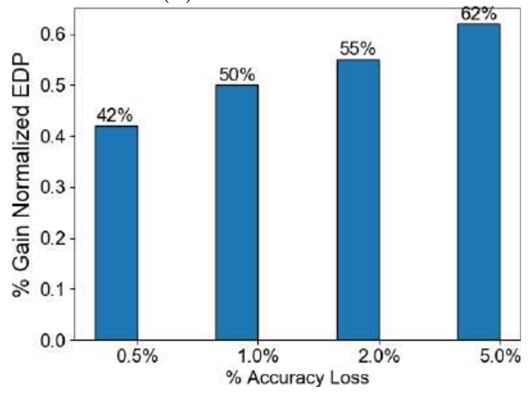
Energy and accuracy trade-off with optimized LEANets. Optimized LEANets can achieve significant energy gain for small loss in accuracy. Figure 4.7 shows the configurations that achieve 0.5%, 1.0%, 2.0%, and 5.0% accuracy loss for VGG and MobileNet to determine their energy gains for SVHN and CIFAR10 datasets. For SVHN, a simple dataset, we see energy gain of 56% and 50% with accuracy loss of 1% for LEANet(VGG) and LEANet(MobileNet) respectively. For CIFAR10, a more complex dataset, the corresponding energy gains are 40% and 22% respectively. Similarly, for a 2% loss of accuracy, we see around 31% to 61% gain in accuracy for LEANets across different DNN and dataset combinations. The energy gains range



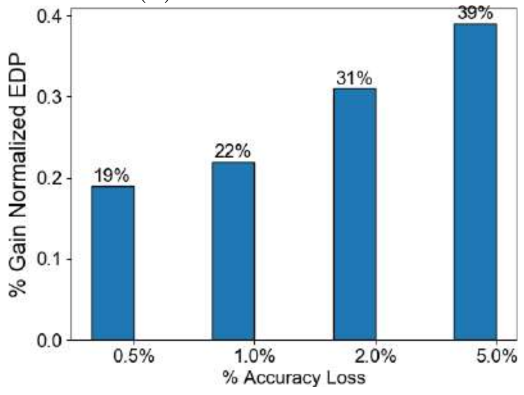
(a) VGG-SVHN



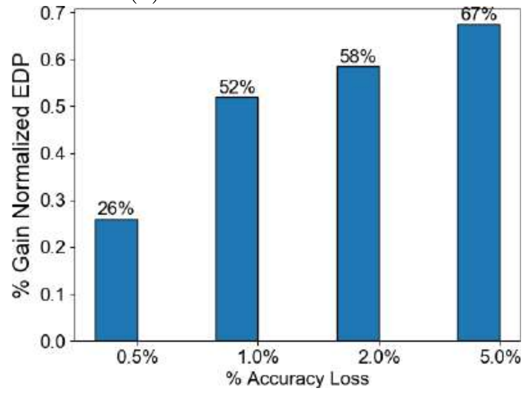
(b) VGG-CIFAR10



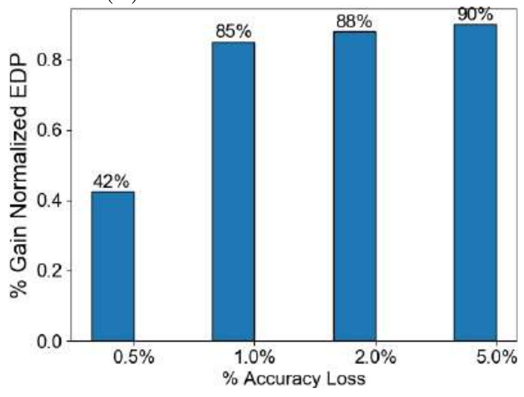
(c) MobileNet-SVHN



(d) MobileNet-CIFAR10



(e) Conv-SVHN

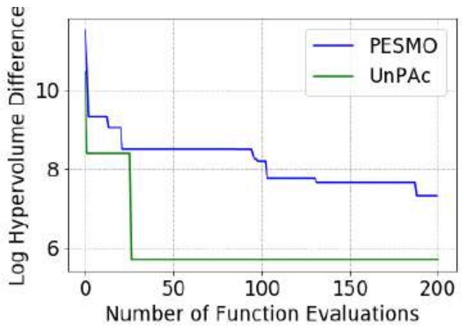


(f) Conv-MNIST

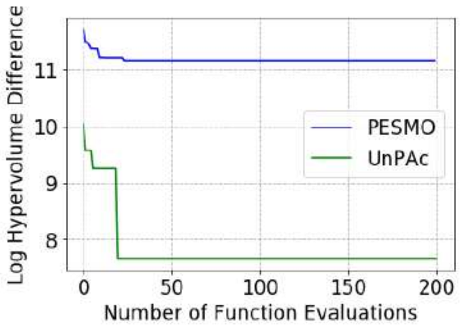
Figure 4.7: Energy gain with different amounts of accuracy loss (0.5%, 1%, 2%, 5%) for optimized LEANet models.

from 39% to 70% for a 5% loss in accuracy. In summary, our approach using LEANet significantly reduces the energy consumption with small loss in accuracy when compared to pre-trained DNNs, i.e., SlimNets(1.0x).

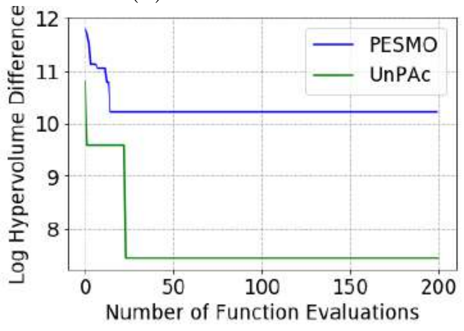
Optimized LEANet performance on ImageNet dataset. To demonstrate the performance of LEANets on large datasets, we compare the MobileNet performance on ImageNets. In Figure 4.12 (a), we present the energy and accuracy value pairs for each level for the fractions [0.5x 0.625x 0.75x 0.875x 1x]. Similar to small datasets, from width 1.0x to 0.875x, the accuracy drops by 2% resulting in a energy gain of 35%. From width 0.875x to 0.75x, the accuracy drops by another 2.5% for a energy gain of 30%, thus producing significant gain in energy for relatively small loss in accuracy. In Figure 4.12 (c), we show the comparison of optimized LEANet and SlimNets as per the five level widths shown in Table 4.1 in terms of Pareto front of the energy and accuracy trade-offs. Similar to other datasets, optimized LEANets not only offer fine-grained trade-off between energy and accuracy, but also reaches the same accuracy values as SlimNet (1.0x) with significant gain in energy. In Figure 4.12 (d), we show the gains in inference time obtained using optimized LEANets and observe a similar trend as energy. Finally, in Figure 4.12 (b), we present the results for significant energy gain for small loss in accuracy.



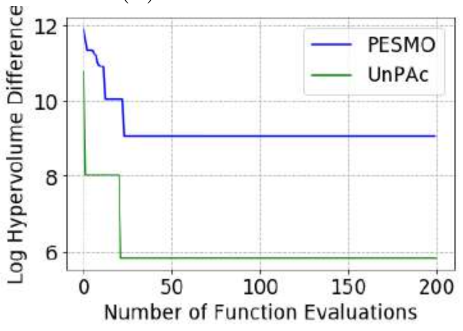
(a) VGG-SVHN



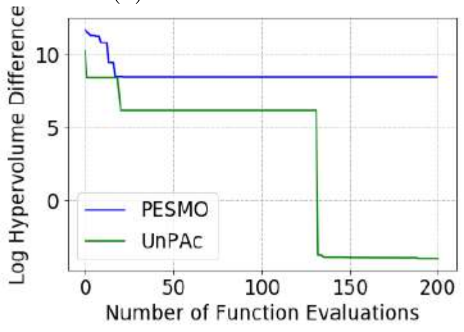
(b) VGG-MNIST



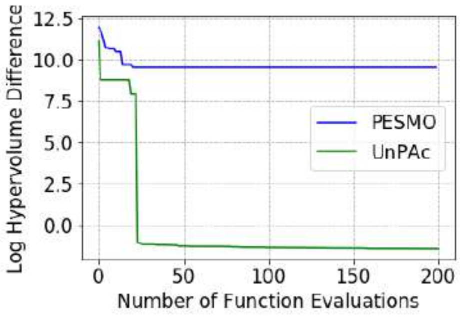
(c) VGG-CIFAR10



(d) MobileNet-SVHN

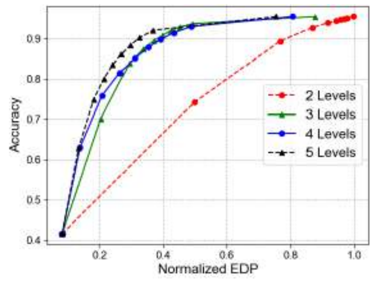


(e) MobileNet-MNIST

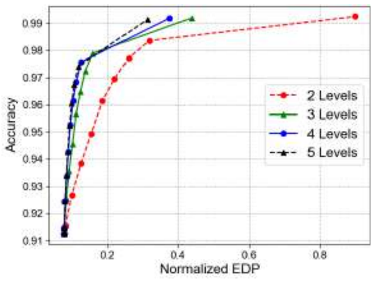


(f) MobileNet-CIFAR10

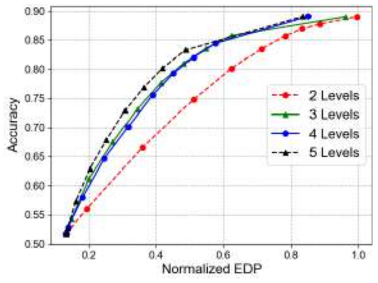
Figure 4.8: Results comparing multi-objective BO algorithms UnPAc and PESMO to find the Pareto set of threshold vectors.



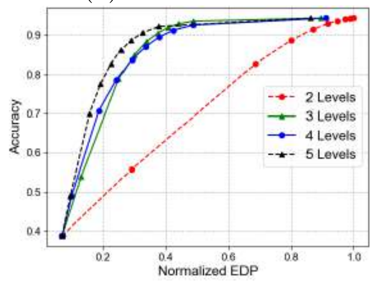
(a) Conv-SVHN



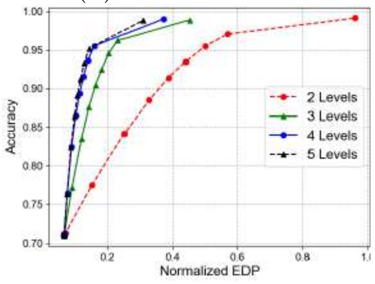
(b) Conv-MNIST



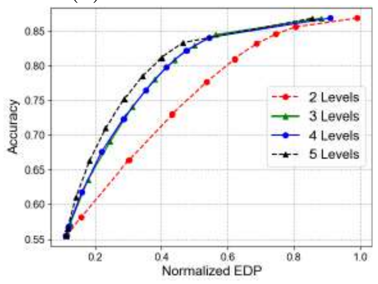
(c) Conv-CIFAR10



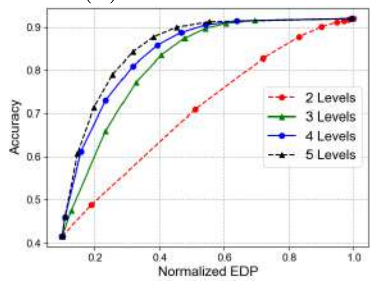
(d) VGG-SVHN



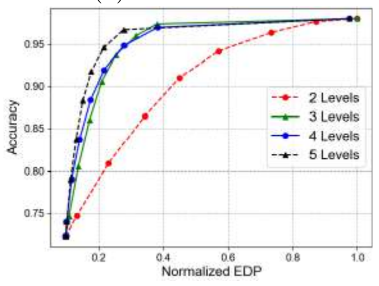
(e) VGG-MNIST



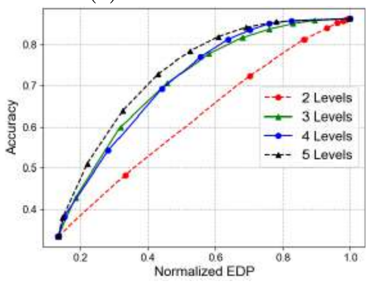
(f) VGG-CIFAR10



(g) MobileNet-SVHN



(h) MobileNet-MNIST



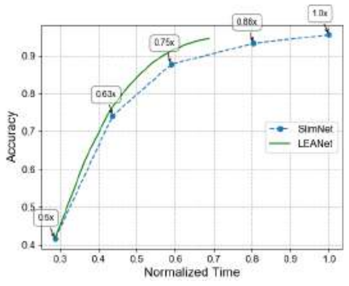
(i) MobileNet-CIFAR10

Figure 4.9: Results of optimised LEANet with 2 levels [0.5x 1.0x], 3 levels [0.5x 0.75x 1x], 4 levels [0.5x 0.625x 0.875x 1x], 5 levels [0.5x 0.625x 0.75x 0.875x 1x] for different DNN and dataset combinations.

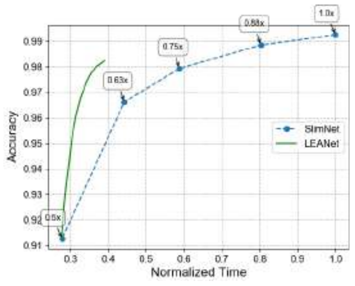
Estimating classifier thresholds. We first compare the efficiency of multi-objective BO algorithms with a scalarization based single-objective BO approach. We combine the energy and accuracy into a single objective using a scalarization parameter ($\alpha \in [0, 1]$); and run single-objective BO algorithm with ten different values of α in the increments of 0.1 until convergence. To achieve the same Pareto front, our multi-objective BO algorithm (UnPAc) takes only 2-7% of the total evaluations of candidate threshold vectors taken by single-objective BO as shown in Table 4.2.

We now compare UnPAc with PESMO, the state-of-the-art multi-objective BO method. Figure 4.8 shows the PHV difference as function of the number of threshold vectors evaluated by the algorithms. We make the following observations: 1) UnPAc converges within 25 evaluations for all DNN and dataset combinations, except for MobileNet and MNIST, where it converges in 125 evaluations. 2) UnPAc convergences to a smaller regret value when compared to PESMO, resulting in better Pareto front for energy and accuracy trade-off. Since UnPAc is consistently shown to be better than PESMO, we will show all our results using UnPAc algorithm.

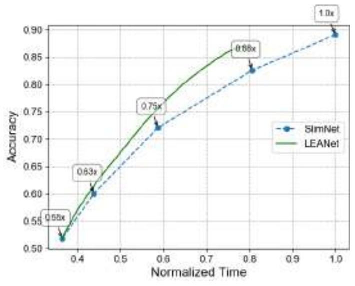
Memory requirement for optimized LEANets. Since inference in LEANets occur incrementally across levels until the confidence threshold to make classification decision, we need to store intermediate computations to be reused across levels. In Table 4.3, we present the total memory required by the two optimized LEANets,



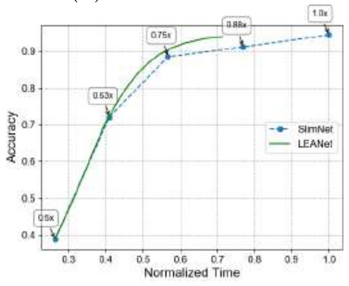
(a) Conv-SVHN



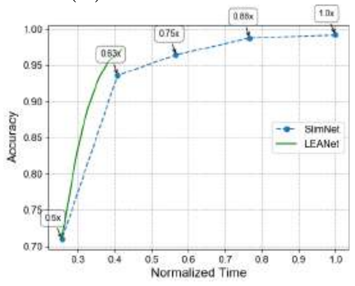
(b) Conv-MNIST



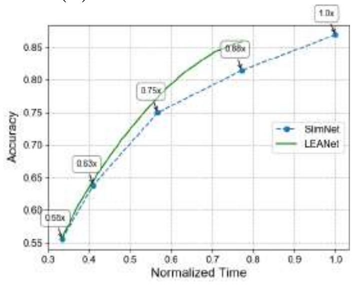
(c) Conv-CIFAR10



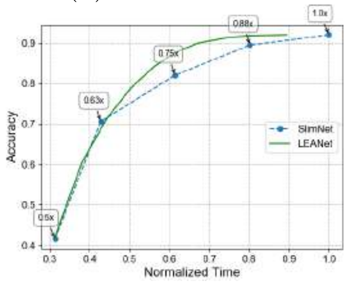
(d) VGG-SVHN



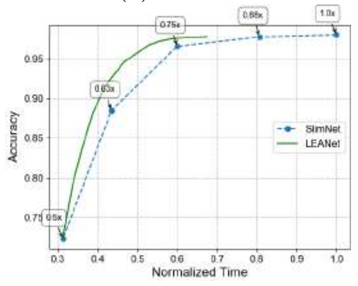
(e) VGG-MNIST



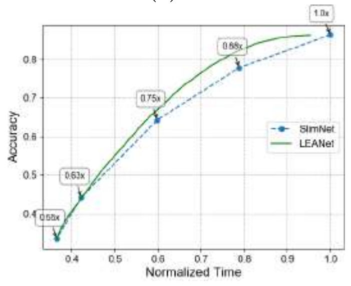
(f) VGG-CIFAR10



(g) MobileNet-SVHN



(h) MobileNet-MNIST



(i) MobileNet-CIFAR10

Figure 4.10: Time and accuracy trade-off results comparing optimized LEANet and SlimNets of different widths.

LEANets	Memory (MB)	Num Layers
VGG19	22	19
MobileNet	42	28

Table 4.3: Memory requirement of optimised LEANet inference on Odroid board for ImageNet dataset.

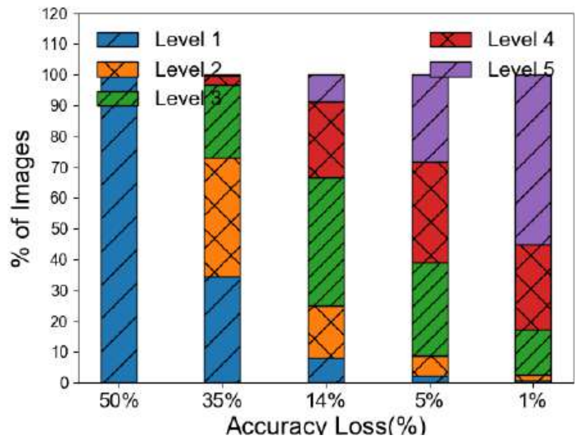
namely, VGG19 and MobileNet, to perform inference for ImageNet dataset on Odroid board. An Odroid board has a 2GB DDR memory. The presented memory results correspond to the setting when the prediction for a given input example is made at the last level of the optimized LEANet (*worst case*). From the results, we can see that the memory requirement is proportional to the number of layers in the network. A 19 layer VGG network consumes 21MB of memory and the 28 layer MobileNet consumes 41 MB of memory respectively. Recall that optimized LEANets make adaptive predictions depending on the hardness of the input examples: easy examples will be classified at lower levels (large fraction) and only a small number of hard examples will classified at higher levels (small fraction). Therefore, the memory overhead will be significantly smaller than the presented results for a large fraction of input examples. Overall, the memory overhead of LEANets to improve energy-efficiency and inference time is negligible.

Convergence of LEANets optimization. As explained in Section 4.3, our iterative optimization approach considers LEANet with increasing levels and stops at convergence. We employ the Pareto hypervolume indicator (PHI) metric to define the convergence criterion: the difference between PHI of Pareto fronts of two consecutive levels l and $l+1$ is very small. Table 4.1 shows the number of levels L and the fraction of channels at different levels when LEANet optimization approach converges for each pre-trained DNN and dataset pair. Figure 4.9 shows the convergence phenomenon of LEANets for all combinations of DNN and dataset pairs. LEANet approach is parameterized by the number of levels. For a fixed number of levels (say L), a candidate threshold vector for each of the L-1 classifiers gives a concrete prediction engine. By executing this prediction engine on a set of input images, we can get the corresponding accuracy and energy consumed: a point on the normalized EDP vs. accuracy plot. For a fixed number of levels (say L), by varying the threshold vectors, we get different accuracy and energy values. The Pareto curve corresponds to the dominant solutions in the energy and accuracy trade-off space obtained by the UnPAC algorithm. Therefore, we get one Pareto curve for a fixed number of levels L. We make the following observations: 1) Going from two to three levels of LEANet, we see a significant improvement in the achievable energy and accuracy trade-off. 2) As we move beyond level three, the improvements are reduced and converge in at most five levels. This is a practically beneficial result that shows that we can apply our design optimization approach to large-scale DNNs.

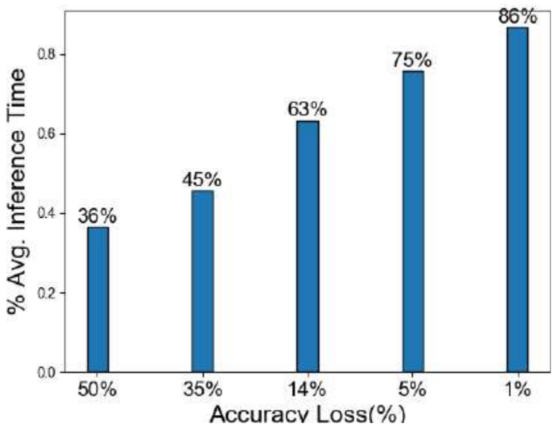
LEANet with varying accuracy. We perform fine-grained analysis to under-

Accuracy	Norm. EDP	T4	T3	T2	T1
0.864	0.998	1.0	1.0	1.0	1.0
0.859	0.851	0.985	0.880	1.0	1.0
0.849	0.742	0.734	1.0	0.836	0.999
0.836	0.706	0.787	0.981	0.690	0.578
0.801	0.607	0.416	0.922	0.971	0.868
0.758	0.556	0.058	0.952	0.947	0.500
0.725	0.442	0.067	0.457	0.967	0.967
0.595	0.366	0.016	0.958	0.817	0.256
0.477	0.212	0.862	0.849	0.206	0.749
0.442	0.177	0.017	0.958	0.014	0.736
0.334	0.133	0.991	0.856	0.985	0.065

Table 4.4: Results of Pareto front obtained using LEANet(MobileNet) with five levels on CIFAR10 dataset. We show the candidate threshold vectors for variables $T_1, T_2, T3$, and T_4 obtained by UnPAC to achieve different energy and accuracy trade-offs.



(a) % images predicted w/ levels

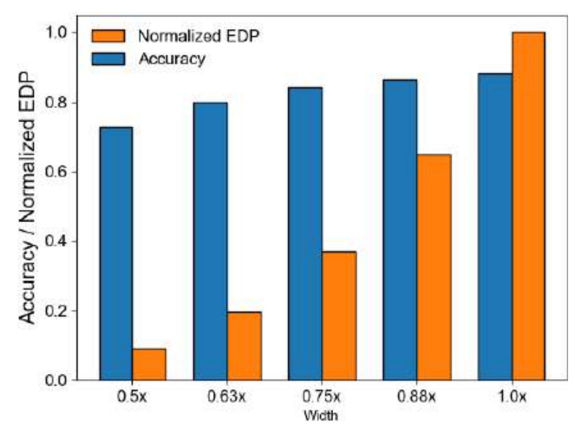


(b) Prediction time

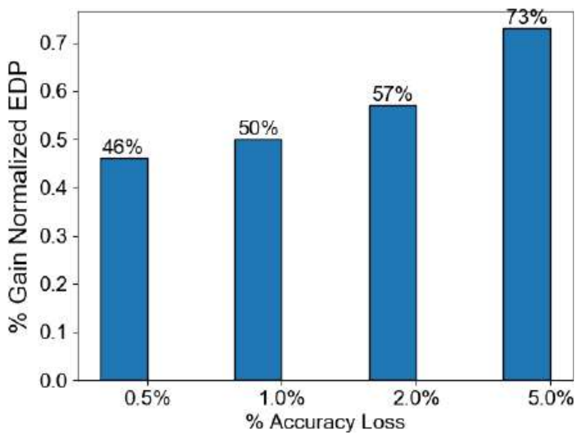
Figure 4.11: Results for LEANet(MobileNet/CIFAR10) w/ varying accuracy loss.

stand how LEANet achieved the above shown energy gains. We present this analysis for LEANet(MobileNet/CIFAR10) noting that analysis for other cases show similar trends.

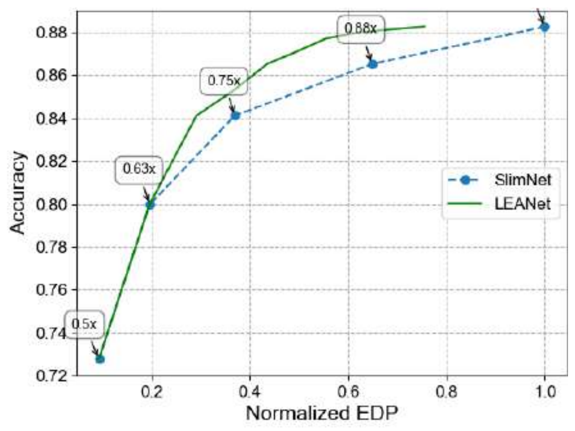
The energy gains can be explained by understanding how many images from the testing set are classified at different levels. We make the following observations from Figure 4.11(a). With accuracy loss of 50%, almost all the images are predicted at level 1 as the simplest classifier consumes the least energy. As the accuracy loss decreases, the number of images predicted by level 2, 3, 4, and 5 slowly increase to achieve better accuracy. The key observation is that with accuracy loss of 1%, unlike SlimNet (1.0x) where all the images get predicted at Level 5, we see that the 10000 images are distributed as 5512, 2768, 1453, 193, and 74 images across Levels 5, 4, 3,



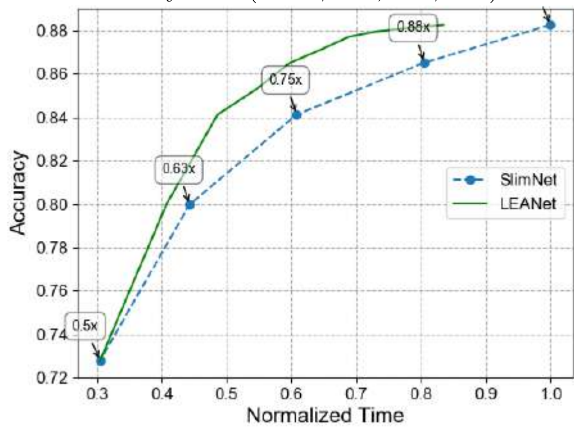
(a) Energy and accuracy of SlimNets with 5 different widths



(b) Energy gain with different amounts of accuracy loss (0.5%, 1%, 2%, 5%)



(c) Energy and accuracy trade-off optimized LEANet and SlimNets



(d) Time and accuracy trade-off optimized LEANet and SlimNets

Figure 4.12: Optimized LEANets performance for large DNN MobileNet using ImageNet dataset.

2, and 1 respectively. Since roughly 45% images are predicted using a lower level, optimized LEANet models are able to achieve significant energy gains.

From Figure 4.11(b), we observe that the average prediction time gradually increases with decrease in accuracy loss, which is explained by more and more images getting predicted at higher levels. With accuracy loss of 35%, the prediction time of inference is 55% lesser when compared to the pre-trained DNN and with a accuracy loss of 5%, we save around 25% in prediction time. Therefore, in scenarios, where we require real-time predictions, we can trade-off the accuracy of LEANet for the target time-bound constraints.

Table 4.4 shows a sample Pareto front of LEANet(MobileNet) with five levels on CIFAR10 dataset. It shows the candidate threshold vectors for variables $T_1, T_2, T3, T_4$ obtained by UnPAC algorithm.

CHAPTER 5. PETNET: POLYCOUNT AND ENERGY TRADE-OFF NETWORKS FOR PRODUCING 3D OBJECTS FROM IMAGES

In this chapter, we discuss an instance of adaptive GCN inference, namely PETNet approach [63] which extends the pre-trained Pixel2Mesh network [113]. Below we explain the background on 3D object shape prediction from color images application and use of GCNs, how the PETNet adaptive inference is optimized, and present some experimental results for the PETNet approach.

Figure 5.1: Virtual (left) and Augumented (right) reality applications

5.1 Background and Preliminaries

In this section, we provide the background on graph convolution networks applied to 3D object shape generation.

3D Mesh is the structural descriptor of a 3D object consisting of polygons/faces in 3D computer graphics and animation. It employs reference points in X, Y, and Z axes to define shapes with height, width, and depth. The mesh is a collection of vertices, edges, and polygons/faces. A vertex is a single point in a 3D coordinate space. An edge is a straight line segment connecting two vertices. A face/polygon is a flat surface enclosed by edges (typically triangles) that describes the shape of a 3D object. Polycount is defined as the total number of polygons/faces found in a 3D object shape. In Fig. 5.2 (left: a), we see a sample 3D model in the form of a 3D ellipsoid object represented as a mesh using triangles. The ellipsoid objects (a) and (c) have 156 and 628 vertices respectively and show that finer details are captured better with increased polycount.

Graph convolution network layer. As mentioned above, 3D mesh is a collection of vertices, edges, and faces that defines the shape of a 3D object. It can be represented by a graph $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V}$ is the set of N vertices in the mesh, $\mathcal{E} = \{e_i\}_{i=1}^{E}$ is the set of M edges, and $\mathcal{F} = \{f_i\}_{i=1}^{N}$ are the feature vectors attached to vertices. Fig 5.2 (right) illustrates a graph convolutional layer defined on graph with vertices A, B, C, and P as $f_p^{l+1} = w_0 f_p^l + \sum_{q \in \mathcal{N}(p)} w_1 f_q^l$, where $f_p^l \in \mathbb{R}^{d_l}, f_p^{l+1} \in \mathbb{R}^{d_{l+1}}$

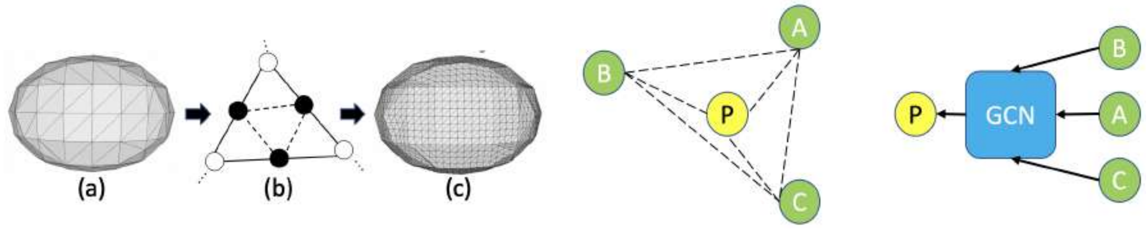


Figure 5.2: Left: Graph Unpooling operation (a) Input ellipsoid object (156 vertices), (b) Edge based unpooling operation, and (c) Output unpooled ellipsoid object (628 vertices). Right: Graph convolution operation on vertex P with neighboring vertices A, B, and C.

are the feature vectors for vertex P before and after the convolution operation, and $\mathcal{N}(p)$ is the set of neighboring vertices of P, namely A, B, and C; w_0 and w_1 are the learnable parameter matrices of size $d_l \times d_{l+1}$ that are applied uniformly to all vertices. Note that w_1 is *shared* for all edges, thus the above-mentioned convolution operation works on nodes with different vertex degrees. In our case, the feature vector f_p associated with vertex P is the concatenation of the 3D vertex coordinates, features encoding 3D shape, and features learned from the input color image (if they exist). Running convolution operation updates the features, which is equivalent to performing a deformation on a mesh.

G-ResNet. The GCN layer predicts the new location and 3D shape features for each vertex, which requires an efficient exchange of information between vertices. In this work, we employ a deep network with shortcut connections called G-ResNet

[74]. Each mesh deformation block in the proposed PETNet architecture (Fig. 5.3) contains a G-ResNet block that employs 14 graph residual convolutional layers with 128 channels and each layer producing a new 128-dimensional 3D feature vector. In addition to output features, there is a branch that applies an extra graph convolutional layer to features of the last layer and outputs the 3D coordinates of a vertex. In short, G-ResNet blocks are computationally expensive.

Graph unpooling layer starts from a mesh with few vertices and adds more vertices to increase the capacity of the neural model to capture finer details of 3D object shapes. A vertex is added at the center of each edge and is connected with both end-points of the edge (Fig 5.2 [left:a]). The three new vertices are also connected to form a triangle (dashed lines in Fig 5.2 [left:b]). Consequently, we create four new triangles for each triangle in the original mesh, and the number of vertices are increased by the number of edges in the original mesh. This edge-based unpooling operation uniformly up-samples the vertices as shown in Fig. 5.2 [left:b]. In the PETNet architecture (Fig. 5.3), the number of vertices increases with the level number. Unpooling layer and G-ResNet block increases the computational complexity across the levels of PETNet.

Accuracy metric. Since we do not have the reference 3D mesh object during inference, we project the 3D object generated by PETNet back to a 2D image. Subsequently, we predict the closeness of this projected image to the input image using *Intersection over Union (IoU)* metric. IoU is defined as the ratio of the area of intersection to the area of the union over the two input binary masks. We employ IoU score to measure the accuracy of generated 3D object shapes by PETNet.

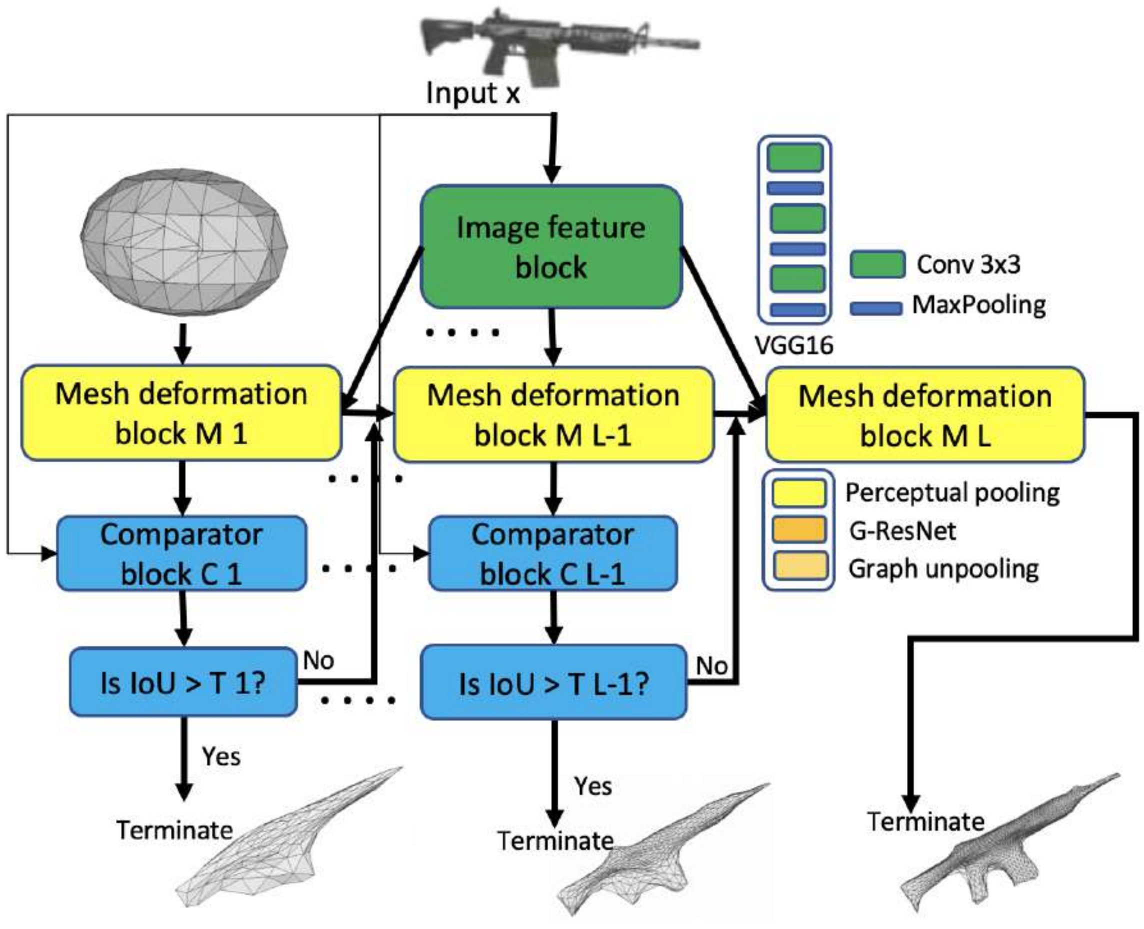


Figure 5.3: High-level overview of PETNet architecture.

5.2 PETNet Architecture

In this section, we provide a detailed description of our proposed *Polycount-Energy Trade-off Network (PETNet)* architecture. First, we provide the motivation behind the PETNet approach. Second, we formally describe the PETNet model and the associated inference procedure. Third, we describe the optimization procedure for configuring PETNet to achieve particular energy and IoU score trade-off.

5.2.1 Motivation

Unlike DNN architectures (e.g. MobileNet[1]) which addresses the problem of image classification on mobile platforms, there are no existing solutions to port GCNs used for AR/VR applications on mobile platforms. As shown in Fig 5.9, simple 3D objects have smooth boundaries and would require less number of polygons to approximate the object, whereas complex objects have sharp edges and concave hulls requiring a large number of polygons. Our main motivation is to design a network architecture such that simple objects would use a smaller GCN model while complex objects would require a larger GCN model. PETNet can be interpreted as a wrapper approach that takes a pre-trained P2M model as input and reuses the learned weights to automatically build a multi-level GCN, where level number is proportional to the complexity of the corresponding model. The key idea is to progressively go from simpler to complex model only if needed thereby reusing the computation of lower levels at higher levels to achieve significant reduction in energy-consumption and execution time with negligible loss of accuracy. Figure 5.9 (a) and (b) shows some sample images from our experiments that are predicted at level 2 and level 3 respectively using PETNet. 3D objects generated at level 2 have simpler shapes like cuboid, whereas those generated at level 3 have complex shapes with sharp curvatures. These results provide evidence for the key intuition behind our proposed PETNet approach. However, formalizing this intuition poses significant algorithmic challenges for predicting structured outputs (e.g., 3D shapes).

5.2.2 PETNet Model

As illustrated in Fig. 5.3, a L level PETNet extends the pre-trained P2M network [113]. It takes an input RGB image and predicts a progressive 3D mesh object in camera coordinates of increasing polycount vis-a-vis vertices. The architecture consists of the following components:

(1) Image feature block takes an input RGB image as input and produces a perceptual feature vector of 1280 dimensions. It employs a 2D Convolutional Neural Network (CNN) based on VGG16 [107]. This perceptual feature vector acts as an input to mesh deformation block at all L levels.

(2) Mesh deformation block employs a graph convolution network (GCN) based ResNet, which progressively deforms an ellipsoid mesh object into the desired 3D object. The deformation block takes as input the structured output and perceptual features from the previous block. This layer is discussed in detail in the P2M network [113]. Each deformation blocks M_i is made of *(a) Perceptual pooling* maps the 1280 dimensional feature vector to each graph vertex to produce 1408 features; *(b) G-ResNet* deforms the input mesh to produce a 128 dimensional feature vector; and *(c) Unpooling layer* increases the number of vertices across L levels.

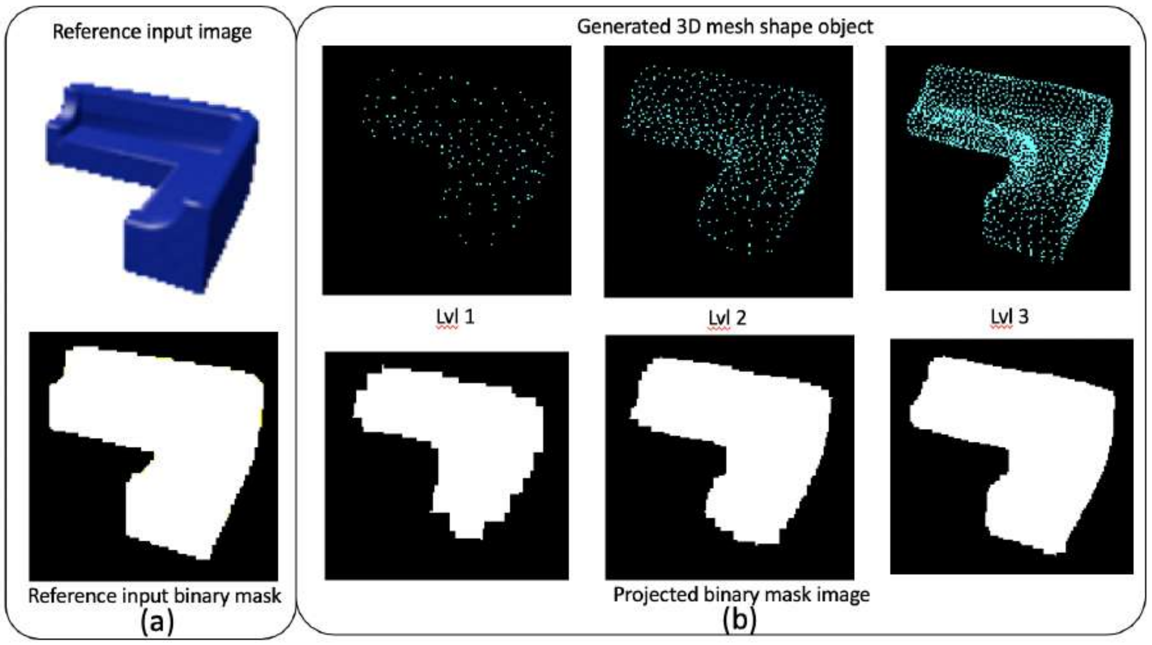

Reference input image
Generated 3D mesh shape object
Lvl 1
Lvl 2
Lvl 3
Reference input binary mask
Projected binary mask image
(a)
(b)

Figure 5.4: Illustration of comparator procedure. (a:top) is the input image; (b:top) shows the generated 3D mesh shape object at level 1, 2, 3; (a and b:bottom) shows the binary mask generated using the input image and the generated 3D mesh shape object at levels 1, 2, 3 respectively. Comparator module at each level takes the input image and generated mesh (top) as input and generates the binary mask, and employs it to calculate the IoU score

(3) Comparator block takes the 3D mesh object generated at each level of the previous block and compares it with the input reference image to calculate the IoU score. In an image classification setting the predicted output value is representative of how close it is to a particular class. For a structured output case, this is not available directly. The IoU score calculated by the comparator block denotes the similarity between the 3D object and the reference input image and is used as a threshold value to terminate early. For a L level network, there are L-1 comparator blocks. Each comparator block C_i has a threshold value T_i, which determines whether to terminate at this particular level based on the IoU score.

Below we describe the various steps involved in calculating the IoU score. **Inputs:** The comparator block receives two inputs. The generated 3D mesh object (In1) from the mesh deformation block at each level [Fig. 5.4(b:top)]. It consists of the (x, y, z) coordinates of all the vertices that make up the mesh. The input reference image ((In2) [Fig. 5.4(a:top)]. Each comparator block does the following operations to calculate the IoU score using the above inputs. **Step 1:** The input (In1) is converted from 3D coordinates to a 2D image by projecting the (x, y, z) values using camera parameters. **Step 2:** The projected 2D image is converted to a binary mask by filtering the image using a morphological closing operator. This operator merges two nearby projected points to form a single region to create a binary mask of the projected points as shown in Fig. 5.4 (b:bottom). **Step 3:** The input (In2) is passed through a simple threshold-based segmentation operator to produce the reference binary image as shown in Fig 5.4 (a:bottom). **Step 4:** The two binary images produced in steps 2 and 3 are then used to calculate the IoU score. **Step 5:** If the IoU score at level i is greater than the threshold T_i, we terminate the deformation and produce the output 3D shape object. Otherwise, we proceed to the next level at $i+1$.

Multi-level inference. Given a PETNet with all its parameters fully specified, inference computation for a new input image x is performed as follows. We sequentially go through the PETNet starting from level 1. At each level i, we generate a 3D mesh object using the mesh deformation block M_i. Then the intermediate comparator block C_i computes the IoU score IoU_i from the generated 3D mesh object and the reference image x. If the IoU score for prediction meets the threshold T_i or we reach the final level of L, we terminate and return the 3D mesh object for the given input image x.

Multi-objective Bayesian optimization formulation. We set the number of levels L as the number of graph unpooling layers in the base P2M network. We formulate the problem of estimating the threshold vector $(T_1, T_2, \cdots, T_{L-1}) \in \mathcal{T}$ in the framework of multi-objective Bayesian Optimization (BO): the input space $\mathcal{T}$ of all candidate threshold vectors is $[0,1]^{L-1}$; and energy and IoU score over the validation set $\textsc{Va}$ and target mobile platform $\mathcal{MP}$ as the two unknown objective functions. USeMO [11] and MESMO [12] are state-of-the-art methods to solve multi-objective BO problems. They iteratively select the input $(T_1, T_2, \cdots, T_{L-1}) \in \mathcal{T}$ to approximate the true Pareto set for energy and IoU score objectives.

5.3 Experiments and Results

In this section, we first describe the experimental setup and then discuss the results of PETNets along different dimensions.

5.3.1 Experimental Setup

Hardware setup. All experiments were performed by deploying GCN models on an `ODROID-XU4` board, which is an Octa-core heterogeneous platform that is popular in current mobile devices. However, our methodology can be employed for other platforms such as Samsung Exynos 5422 MPSoC. We execute GCN models using the Python Tensorflow framework. We employ SmartPower2 to measure power and compute the average power over total execution time.

Dataset and GCN training We test PETNet based approach as shown in Fig. 5.3 using the pre-trained Pixel2Mesh (P2M) network, which contains three graph unpooling layers. Therefore, we consider a PETNet architecture with three levels, generating a mesh of 156, 628 and 2466 vertices respectively. We ran extensive experiments using the ShapeNet dataset [2] for the performance evaluation of PETNets. This dataset contains rendering images of 50k 3D shape objects belonging to 13 object categories. For a fair comparison, we employ the same training/validation/testing split as in Choy et al.[21].

Energy objective. Energy consumption of PETNet models is measured in terms of the normalized energy-delay product (EDP). EDP = $\sum$P Δt $\cdot$ T, where Δt is the time interval at which we record the power P and T is the total execution time. As power is measured at a regular interval, we simply calculate EDP as EDP = $P_{avg} \cdot T^2$. This value is normalized with EDP of the pre-trained P2M network. We use EDP and energy interchangeably in the discussion of our results. However, all our presented results are for EDP unless specified otherwise.

Accuracy objective. The accuracy of 3D shapes from the PETNet method is measured using the IoU score.

Multi-objective BO for PETNet models. For obtaining the threshold vectors to trade-off energy and IoU score, we run the state-of-the-art method USeMO [11] for 200 iterations.

5.3.2 Results and Discussion

In this section, we first present the performance of the state-of-the-art method based on Pixel2Mesh (P2M) followed by a discussion of improvements achieved by optimized PETNets across different dimensions. We also show qualitatively how optimized PETNet performs across different levels. Finally, we present a fine-grained analysis of optimized PETNet models.

IoU score, energy, and inference time across PETNet levels. We test the pre-

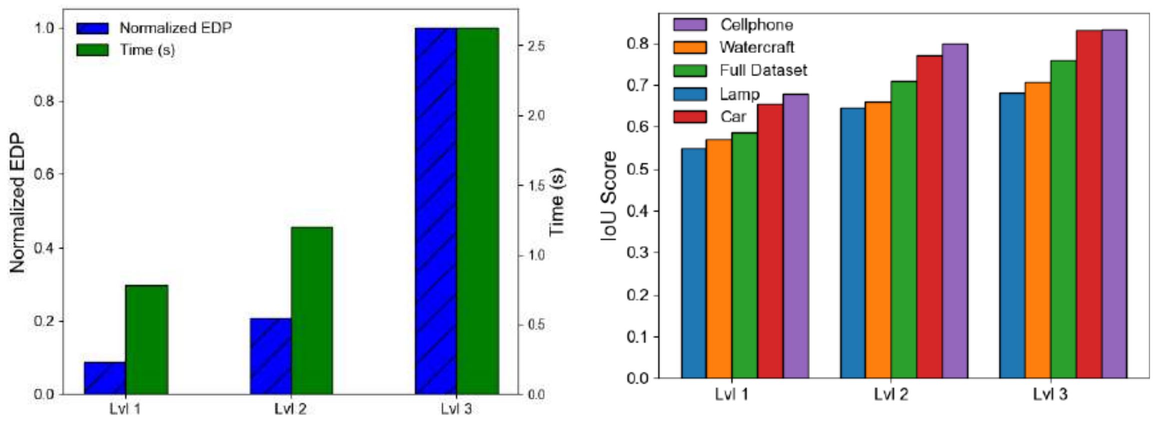


Figure 5.5: Left: Energy and inference time of PETNet at levels 1, 2 and 3. Right: IoU score of different 3D objects namely, lamp, watercraft, car, and cellphone generated at levels 1, 2 and 3 respectively.

trained P2M network at levels 1, 2 and 3 producing 156 (simple), 628, 2466 (complex) vertices respectively, i.e. we make all the predictions at each level of PETNet. This experiment will characterize the IoU and energy/time trade-off achieved by PETNet at each level. The performance of the base P2M network is characterized by the PETNets finest level (i.e., level 3). In Figure 5.5 (a) we show the energy and time metrics of PETNet at levels 1, 2 and 3 respectively. Figure 5.5 (b) shows the respective IoU score for different object types like a lamp, watercraft, car, and cellphone. **Lamp:** From level 3 to level 2, we can see that IoU score drops by 0.05 and EDP reduces by 80%. The inference time drops by 56%. Similarly, from level 2 to level 1, IoU score drops by 0.1% and EDP reduces by 15% w.r.t level 3. The inference time drops by 16%. **Watercraft, car, and cellphone:** we achieve similar EDP and inference time gains for an IoU score drop of 0.048, 0.06, and 0.033 respectively from level 3 to level 2. From levels 2 to 1 we observe an IoU score drop of 0.09, 0.11 and 0.12. On average, the IoU score drops 0.059 from levels 3 to 2 and 0.12 from levels 2 to 1.

In summary, we see a significant gain in energy and inference time with a relatively small loss in IoU score when we move from level 3 to level 1 in PETNets. This corroborates our hypothesis that we can save a significant amount of energy if we are able to select simpler networks for large fraction of easy 3D objects and complex networks for hard 3D objects that are relatively small. We also observe that simple object shapes like cars and cellphones have better IoU scores in comparison to complex object shapes like lamps and watercraft.

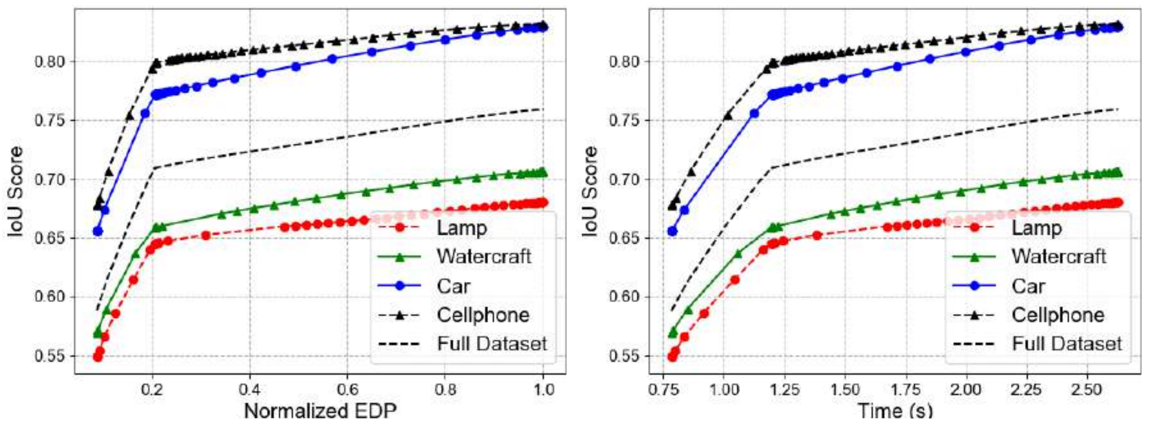


Figure 5.6: Optimized PETNet energy (left) and inference time (right) trade-off with IoU score by varying the threshold vector across different 3D objects

Optimized PETNet for IoU score. In the above results, we saw that across levels, PETNet takes significantly more energy for a small improvement in the IoU score. Based on this observation, optimized PETNets obtain a more fine-grained trade-off between EDP and IoU score by varying the threshold vectors T employed to make the exit decisions. Unlike the P2M network which just has a single IoU-energy value (no input-specific adaptive predictions), in Figure 5.6 (left) we show the performance of optimized PETNet in terms of the Pareto front for EDP and IoU trade-offs.

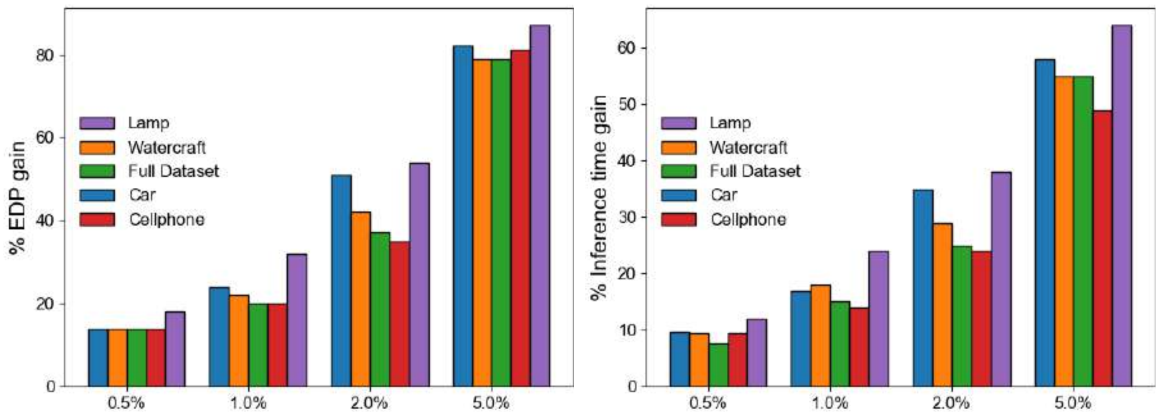


Figure 5.7: Energy (left) and inference time (right) with different amounts of IoU score loss [0.5%, 1.0%, 2.0%, and 5.0%] for optimized PETNet models.

Furthermore, in Figure 5.6 (left), we compare the performance across different 3D object shapes. Simple objects like cars and cellphones exhibit a better Pareto front when compared to complex objects like lamps and waterfront. The IoU energy trade-off using optimized PETNet is explained by the fact that many *easy* 3D object shapes terminate at lower levels and only *hard* objects are passed to higher levels.

In Figure 5.6 (right), we show the gains in inference time obtained using optimized PETNets across different object types. In this case, too, we observe a similar trend as energy.

EDP / inference time and IoU score trade-off with optimized PETNet. In comparison to the fixed performance of the P2M network, optimized PETNet can achieve significant energy gain for a small loss in the IoU score. Figure 5.7 (left) shows the configurations that achieve 0.5%, 1.0%, 2.0%, and 5.0% IoU score loss to determine their EDP gains for different object types of the ShapeNet dataset. For a negligible loss of 0.005 IoU score, we obtain 14%-18% EDP gain across different object types. This increases to 20%-54% for an IoU loss of 0.01 to 0.02 respectively. The EDP gain values shoot up to 80%-87% for an IoU score loss of 0.05. Figure 5.7 (right) shows the inference time gains obtained for similar IoU score losses. We get around 8%, 15%, 25%, and 55% inference time speedup for IoU score losses of 0.005, 0.01, 0.02, and 0.05 respectively on the full ShapeNet dataset. In this case, too, we observe similar patterns like energy for different object types. In summary, our approach using PETNet significantly reduces energy consumption with negligible loss in the IoU score.

PETNet with varying IoU score loss. We perform fine-grained analysis to understand how PETNet achieved the above shown energy gains. The energy gains can be explained by understanding how many images from the testing set are predicted at

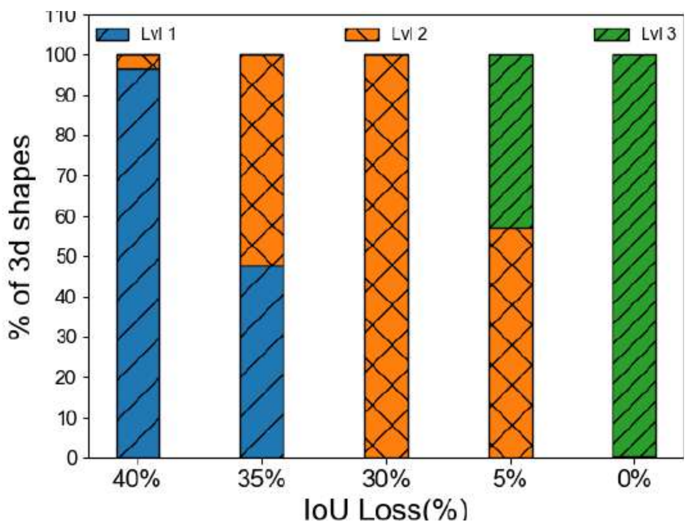


Figure 5.8: 3D shapes predicted at levels 1, 2, and 3 by varying IoU score loss.

different levels. We make the following observations from Figure 5.8. With IoU score loss of 0.4, almost all the images are predicted at level 1 as the smallest polycount consumes the least energy. As the IoU loss decreases, the number of images predicted by level 2 and 3 slowly increase to achieve better accuracy. The key observation is that with accuracy loss of 0.05, we see that the total 210050 objects are distributed as 120342 and 89708 objects across level 2 and 3 respectively. Since roughly 57% objects are predicted using a lower level, optimized PETNets model is able to achieve significant energy gains.

Qualitative Analysis. Figure 5.9 shows some sample 3D object shapes that are predicted at level 2 and level 3 using optimized PETNet. We make following observations: 1) Objects terminating at level 2 are simple with basic shapes like cuboid; 2) Objects terminating at level 3 are complex with concave shapes and sharp edges;

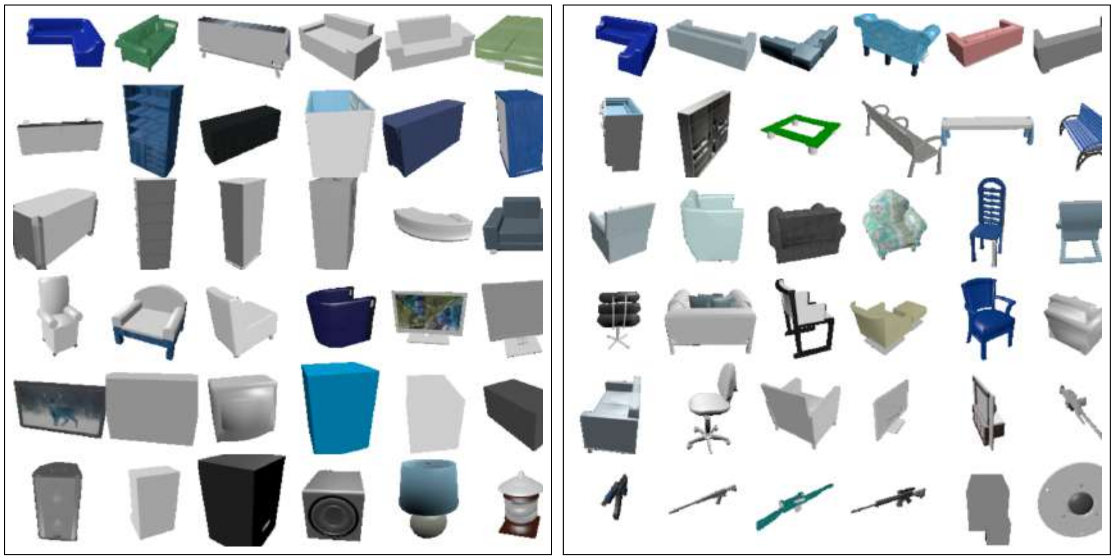


Figure 5.9: 3D objects predicted at level 2 (left) and 3 (right) using PETNet.

These results demonstrate how adaptive PETNets use smaller polycount models for simple objects and larger polycount models for complex objects.

CHAPTER 6. SETGAN: SCALE AND ENERGY TRADE-OFF GANS FOR IMAGE APPLICATIONS

In this chapter we discuss an instance of adaptive GAN inference, namely SETGAN [62] across different scales operating over a client-server architecture for image-editing applications. The SETGAN approach allows us to update the end-user image editing application progressively as and when better models are available. Below we explain the background on application tasks and GANs, how the SETGAN adaptive inference is optimized, and present some results to demonstrate the effectiveness of SETGAN.

6.1 Background and Preliminaries

Generative Adversarial Networks. A GAN for unconditional image synthesis involves a generator G which takes a noise vector z as input and outputs a synthetic image that closely resembles the image from a real data set. As shown in figure 6.1, a GAN architecture primarily is made of two networks: a) Generator network, and b) Discriminator network. The generator G learns to generate plausible data by taking a random noise vector z, and outputs synthetic data $G(z)$; a discriminator D learns to distinguish the generator's fake data $G(z)$ from real data. The discriminator penalizes the generator for producing implausible results. It takes an input x or $G(z)$

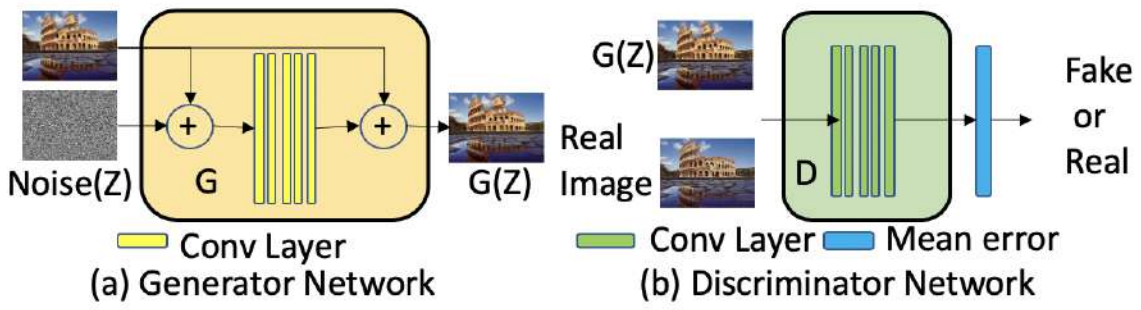


Figure 6.1: General structure of a Generative Adversarial Network, where the generator G takes a noise vector z as input and outputs a synthetic sample $G(z)$, and the discriminator D takes the synthetic input $G(z)$ or the real sample X as inputs and predict whether they are real or fake.

and outputs a probability $D(x)$ or $D(G(z))$ to indicate whether it is synthetic or from the true data distribution, as shown in Figure 6.1.

Training GAN Models. The principle to train the generator and discriminator is to form a two-player min-max game, where the generator G tries to generate realistic data to fool the discriminator and discriminator D tries to distinguish between real and fake data. The value function to be optimized is as follows: $\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[log \ (D(x))] + E_{z \sim p_z(z)}[log \ (1 - D(G(z)))]$, where $p_{data}(x)$ denotes the true data distribution and $p_z(z)$ denote the noise distribution. When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake. As training progresses, the generator gets closer to producing samples that can fool the discriminator. Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

Discriminator: The discriminator in the GAN model is a classifier. It tries to distinguish real data from fake data created by the generator. The discriminator's training data comes from two sources: a) Real data samples, which act as positive examples during training; and b) Fake data samples created by the generator, which act as negative examples during training. The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator's loss and just uses the discriminator loss. The discriminator classifies both real data and fake data from the generator. The discriminator's loss penalizes the discriminator for misclassifying a real sample as fake or a fake sample as real. The discriminator updates its weights from the discriminator's loss.

Generator: The generator part of a GAN learns to create fake samples to make the discriminator classify its output as real. Generator's training requires tighter integration between the generator and the discriminator. The portion of the GAN that trains the generator includes: a) random input; b) generator network, which transforms the random input into a data sample; c) discriminator network, which classifies the generated data as real or fake; and d) discriminator output generator loss, which penalizes the generator

Structural Similarity Index Metric . For predicting the perceived quality of synthesized images in SETGAN approach, we employ SSIM score that measures the similarity between two images. The $SSIM$ between two images x and y is given by equation $SSIM(x,y) = \frac{(2\mu_x\mu_y+C_1)+(2\sigma_{xy}+C_2)}{(\mu_x^2+\mu_y^2+C_1)(\sigma_x^2+\sigma_y^2+C_2)}$ where μ_x, μ_y are the means and σ_x, σ_y are the variances of images x, y; and σ_{xy} is the covariance of x and y.

6.2 SETGAN Framework

Motivation. Image editing applications such as Paint2Image and super-resolution are time-sensitive and utilize an image synthesis operation as part of their workflow. Deploying large GAN models on mobile platforms requires a lot of resources in terms of computation, energy, and memory. Multi-image GANs such as PGAN [75] have a) a huge number of parameters, and b) take a long time to train for new image datasets (usually in days). Single-image GANs such as SinGAN [100] have relatively less number of parameters, but they still take a long time (in order of hours) to train a given input image.

In the SETGAN framework, we propose a single-image multi-scale GAN model, which *automatically* adapts the number of scales based on the complexity of input image in a client-server architecture. The key idea is to use a small number of scales for a simple image (which consists of small structures and textures) while using more number of scales for complex images (which has large objects). We were able to achieve 56% gain in energy for a loss of 3%-12% SSIM accuracy. Furthermore, we propose a novel training methodology to reduce the training time by a factor of 4X. We adapt the SinGAN model [100] since it has a relatively small number of parameters making it an ideal candidate for image editing applications in a client-server architecture. In the following section, we describe our solution in detail. First, we describe the SETGAN model including generator, discriminator, training procedure, and inference methodology. Second, we discuss the client-Server architecture including the different modes of operation of the GAN model for image editing applications. Finally, we instantiate our SETGAN framework for diverse image editing application workflow.

6.2.1 SETGAN Model, Training, and Inference

Multi-scale SETGAN Model. Our model tries to capture the internal statistics of the given single-image X for training. The unconditional image generation task is conceptually similar to the conventional GAN setting. However, instead of the whole image, we train the model using patches of a single image. As shown in the Figure 6.2, we create a pyramid of images from the given input X: $X_0, \cdots, X_N$, where X_i is a down-sampled version of X (X_0 being the coarsest image) by a factor r^{N-i}, for some $r > 1$. The multi-scale SETGAN model consists of a list of generators for each down-sampled version of X, $G_0, \cdots, G_N$, a associated list of discriminators

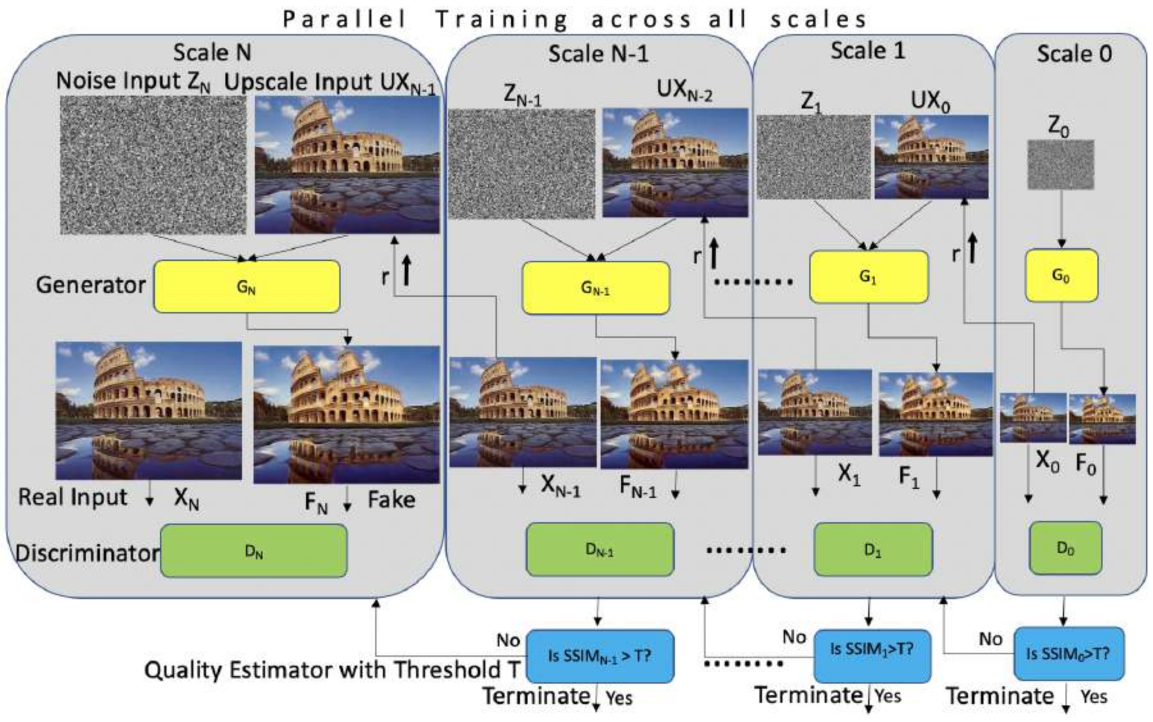


Figure 6.2: Illustration of SETGAN approach with multi-scale GAN model. An abstract model with N scales from coarsest-scale GAN 0 to finest-scale GAN N. Each scale consists of a generator G_i, giscriminator D_i, and a quality estimator $SSIM_i$ to decide when to terminate. Generator G_i takes a noise vector Z_i and up-scaled real input X_{i-1} and generates a fake image F_i. Discriminator D_i takes F_i or real input X_i as input and classifies them as fake or real. $SSIM_i$ checks whether SSIM between the generated fake image F_i and the original input image X is greater than the user-specified threshold T.

$D_0, \cdots, D_N$ and Quality estimators $SSIM_0, \cdots, SSIM_{N-1}$.

Parallel Training Algorithm. Image-editing applications are time-sensitive. However, the SinGAN model based on which we have developed our SETGAN model employs a serial training procedure, which is very expensive. Starting from the coarsest-scale, SinGAN uses the fake image F_{i-1} generated at the end of training of the previous stage as input for the next generator G_i's input. *In contrast, in SETGAN, we employ the scaled (real) input image X_{i-1}, which allows us to perform embarrassingly parallel training of models at all scales.* The key reason we are able to do this is as follows: F_{i-1} is an approximation of X_{i-1} and this approximation helps us in achieving 4x reduction in training time with negligible loss in quality of generated images. Since the goal of image editing tasks is for the generated image to be similar and not exact to the original input image, this negligible loss in quality is not noticeable.

Training Procedure. Given an input image X and a target SSIM threshold for early exit T, we perform the training of models at all N scales in parallel. Each scale i is made of a generator G_i, discriminator D_i, and a quality estimator $SSIM_i$. They take as input 1) X_i, the image the generator has to match for this scale i; 2) Up-scaled X_{i-1}, the image from a coarser scale X_{i-1} up-scaled by a factor r; 3) Gaussian noise vector Z_n. Each generator G_i is responsible for producing realistic image samples w.r.t the patch distribution in the corresponding image X_i. As described in the background

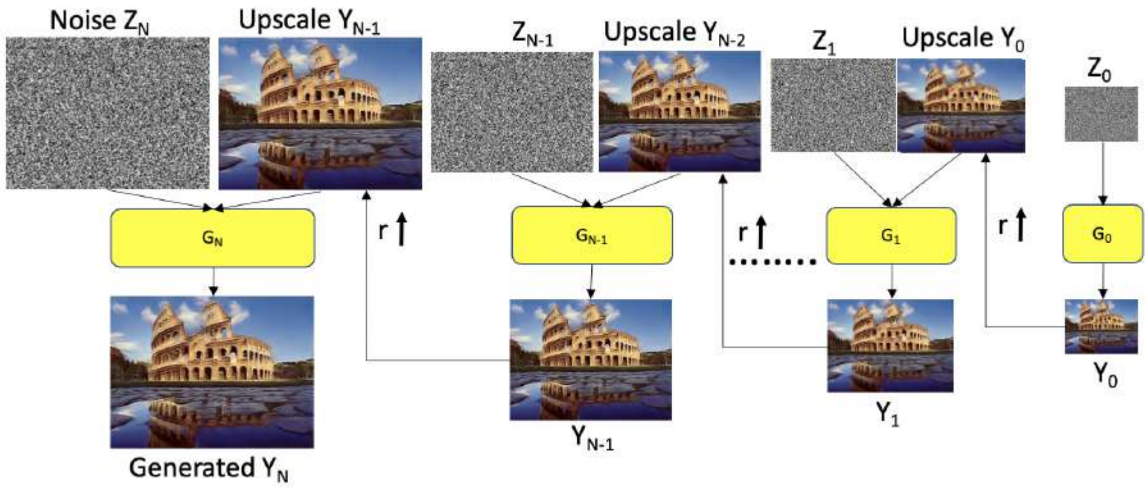


Figure 6.3: SETGAN inference: The unconditional image synthesis starts at coarsest scale G_0 taking noise vector Z_0 as input and passes through all generators upto finest scale G_N.

section, the generator achieves this goal via adversarial training, where G_i learns to fool an associated discriminator D_i, which attempts to distinguish patches in the generated samples from patches in X_i. At the end of adversarial training, quality estimator $SSIM_i$ checks whether SSIM between the generated fake image F_i and the original input image X is greater than the early exit threshold T or not. If this condition is satisfied, then we terminate the training procedure at this scale and use the models from $0, 1, \cdots, i$ for our inference.

a) Generator: The generator at scale 0 is purely generative, i.e., G_0 maps Gaussian noise Z_0 to an image sample F_0. However, in all other scales, G_i maps the Gaussian noise Z_i to produce the incremental image from X_{i-1} to that at X_i. All the generators have a similar architecture as shown in figure 6.1. Each generator is a fully-convolutional network with 5 convolution blocks. Each block is made of three layers namely (3x3) Conv layer, BatchNorm followed by a ReLU, except for the last block whose activation is Tanh. It starts with 32 kernels per block at the coarsest-scale and increases by a factor of 2 for every 4 scales. The noise vector is added to the scaled image prior to being fed to the convolution blocks, ensuring that the GAN does not disregard the noise.

b) Discriminator: Each generator G_i works with a corresponding Markovian discriminator D_i that classifies the overlapping input patches as real or fake. A WGAN-GP loss [50] is used to increase the training stability. The architecture of D_i is similar to that of the generator as shown in figure 6.1. However, the last block at each scale includes neither normalization nor activation. It is followed by a mean error block which averages the error over entire image.

$$\min_{G_n} \max_{D_n} L_{adv}(D_n, G_n) + \alpha * L_{rec}(G_n) \tag{6.1}$$

Equation 6.1 describes the training loss of the ith scale GAN model. L_{adv} represents the adversarial loss.

Inference Algorithm. As shown in figure 6.3, the image generation starts at the coarsest-scale generator G_0 which takes input as noise vector Z_0 and produces a generated image Y_0. At each scale $i > 0$, generator G_i takes the image generated at a coarser scale Y_{i-1} and noise vector Z_i as input and produces a finer image Y_i. This procedure passes through all the generators upto the scale determined by the training algorithm to meet the quality threshold. However, as shown in figure 6.3, during the inference stage, we do not use the real images. Because the generators are fully convolutional, we can generate images of arbitrary size and aspect ratio at test time (by changing the dimensions of the noise maps).

6.2.2 Client-Server Architecture

The training procedure of GAN models involve thousands of iterations of forward and backward propagation steps, which are very compute-intensive and are usually executed using a GPU.

Hence, training GANs on mobile platforms is impractical. Furthermore, any image manipulation task involves repeated editing of the given input image until the user is satisfied (i.e., repeated inference calls). The **Naive client-server architecture** involves sending the image to be edited to the server and for every image edit, the server sends the edited image back to the client: *all training and inference calls are executed on the server.* This causes network latency to the user after every edit and increases the client-server memory bandwidth for every edit operation. Hence, we propose a modified client-server architecture to efficiently use SETGAN for image-editing applications and discuss three different modes of operation.

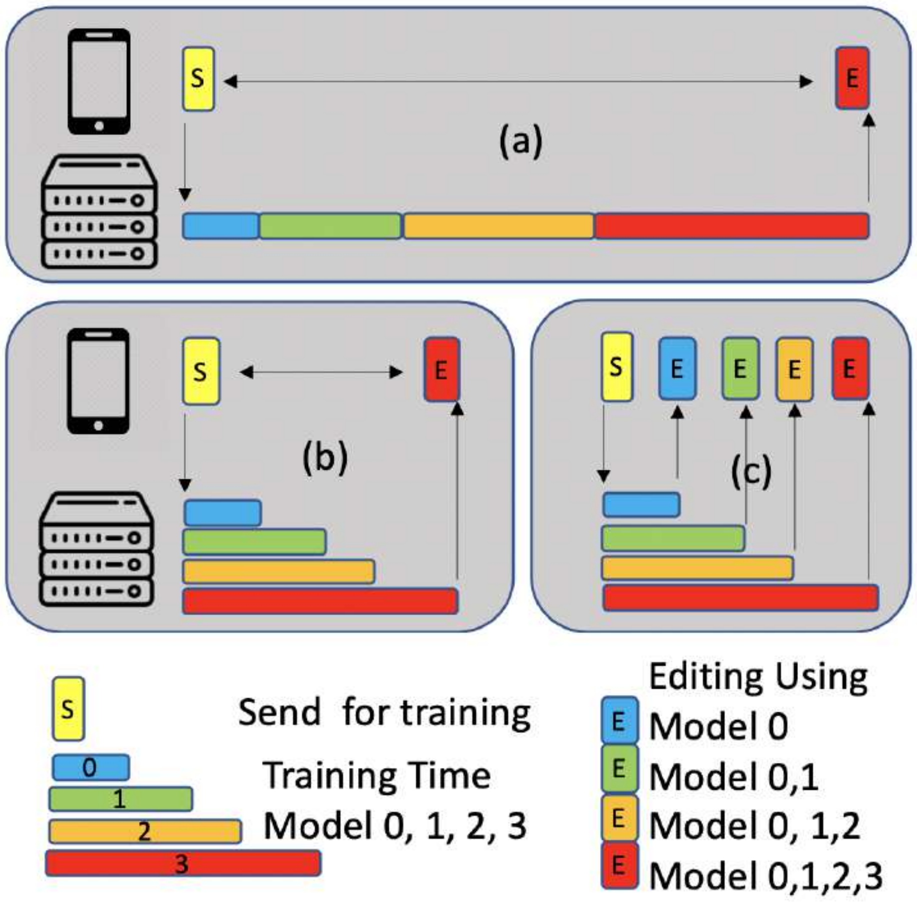


Figure 6.4: Client-server architecture: Describes the three modes of operation for the SETGAN Framework (a) SinGAN Baseline, (b) Parallel training, and (c) Progressive update.

[Mode 1] SinGAN Baseline: As shown in figure 6.4 (a), we employ the existing SinGAN model for our training. This is our baseline mode. In SinGAN, the training happens sequentially from the coarsest scale G_0 to the finest scale G_N for any given input image. Once the training is complete, all the models are sent together in a one-shot to the edge device.Unlike the default mode, since the entire model is available on the mobile device, repeated inference happens without the network latency and does not increase the client-server memory bandwidth for every edit operation

[Mode 2] Parallel Training: As shown in figure 6.4 (b), in this mode of operation, we employ SETGAN model for our training. Since the SETGAN models at all the scales are trained in parallel and the *best* scale to meet the quality threshold is determined automatically, it takes significantly less time to train SETGAN for a given input image. Once training is complete, all the models from G_0 to G_{best} (best $\leq N$) are sent in a one-shot to the edge device.

[Mode 3] Progressive Update: As shown in figure 6.4 (c), in this mode of operation, we use SETGAN model for our training similar to mode #2. However, we send the trained model to the edge device as and when it's training is complete. Since we do not wait for the full training to be complete, we can start using our image-editing application with the available models which provide a coarser output. The generated images would be refined as and when finer-scale models are available. This mode reduces the time to start the editing application and results in a better user experience.

6.2.3 SETGAN for Image Applications

In this section, we discuss the use of the SETGAN framework for various image-editing applications. Recall that a trained SETGAN generator model is capable of producing images with the same patch distribution as that of the single training image. Different image manipulation tasks can be performed by injecting (a down-sampled version of) an image into the pyramid of generators at some scale $i \leq N$. When this image passes through the various generators, it produces an output that matches the patch distribution to that of the training image. We can perform diverse image manipulations by using an appropriate scale at which we introduce the image.

Key Steps. On receiving an input image for editing, the following sequence of steps are performed: 1) The image is sent to the server; 2) Using the input image, the SETGAN model is trained for the optimal number of scales (say *best*); 3) The trained SETGAN generator models $(G_0, \cdots, G_{best})$ are sent to the edge device; and 4) Using the input image and the received SETGAN model, we can repeatedly perform various image-editing operations on the edge device. We instantiate SETGAN for three diverse image-editing applications that will be employed in our experimental evaluation.

Super-resolution: Given a low-resolution (LR) input image, we want to scale it up by a factor s to produce a high-resolution image as shown in Fig 6.10. To support a scale factor s, we train SETGAN for N scales such that $r = \sqrt[k]{s}$. During inference, we up-sample the LR image by a factor r. This along with the noise vector is fed to the last generator G_N. We repeat k times to obtain the final high-resolution output.

Paint2Image: Given a clip-art image as input, we want to generate a photo-realistic image as output (illustrated in Fig 6.12). The clip-art image is an LR representation of the output image we need to generate. Hence, we downsample the clip-art image and feed it to one of the coarse scales like G_1 or G_2. This preserves the global structure of the painting while texture and high-frequency information matching the original image generates realistic photos.

Harmonization: As shown in Fig 6.11, the Eiffel tower object is pasted onto to image which looks like a painting. Harmonization is the process of realistically blending this pasted object with the background image. In this case, we inject a down-sampled version of the composite image at the inference stage. Unlike paint2image that works from the coarse scales, we inject the image at the finer scales of the generator in this application. Scales $G_{N-1}, G_{N-2}, G_{N-3}$ generally provide a good balance between preserving the object's structure and transferring the background texture.

6.3 Experimental Setup

Hardware setup. All our experiments were performed by deploying SETGAN on an `ODROID-XU4` board, [93]. We employ SmartPower2 [108] to measure power.

GAN training on server. We set the minimal dimension at the coarsest-scale to 25px, and choose the number of scales N such that the scaling factor r is as close as possible to 4/3. For all the results (unless mentioned otherwise), we re-sized the training image to the maximal dimension 256 px. Thus, the maximum number of scales N is 9. At each scale, the weights of the generator and discriminator are initialized randomly. We train each scale for 2000 iterations. In each iteration, we alternate between three gradient steps for the generator and three gradient steps for the discriminator. We employ the Adam optimizer with a learning rate of 0.0005 (decreased by a factor of 0.1 after 1600 iterations), and momentum parameters β_1= 0.5 and β_2= 0.999. The weight of the reconstruction loss α is set to 10, and the weight of the gradient penalty in the WGAN loss is set to 0.1. All Leaky-ReLU activations have a slope of 0.2 for negative values. The training uses an RTX 2080Ti GPU.

Datasets. We employed the Berkeley Segmentation Database (BSD) [90] and Places [120] datasets consisting of 500 and 10 million images respectively. Our approach works with one image at a time.

Energy objective. Energy consumption of SETGAN is measured in terms of the normalized energy-delay product (EDP). EDP = $\sum$P Δt $\cdot T$, where Δt is the time

GAN Type	Memory (MB)	Training Time (Hours)
PGAN	3343	24.00
SinGAN	98	1.00
SETGAN	98	0.25

Table 6.1: Training time and memory required for optimized SETGAN and state-of-the-art GAN models.

interval at which we record the power P and T is the total execution time. As power is measured at a regular interval, we simply calculate EDP as EDP $= P_{avg} \cdot T^2$. This value is normalized with EDP of SETGAN with all scales (0 to N).

Accuracy objective. The accuracy of the unsupervised photo-realistic image generation from the SETGAN method is measured using the Structural Similarity Index Metric (SSIM) score which measures the quality of the generated images similar to how human perceive images unlike metrics such as Peak Signal-to-Noise Ratio.

6.4 Results and Discussion

In this section, we compare the performance of SETGAN with state-of-the-art GAN models to demonstrate its effectiveness.

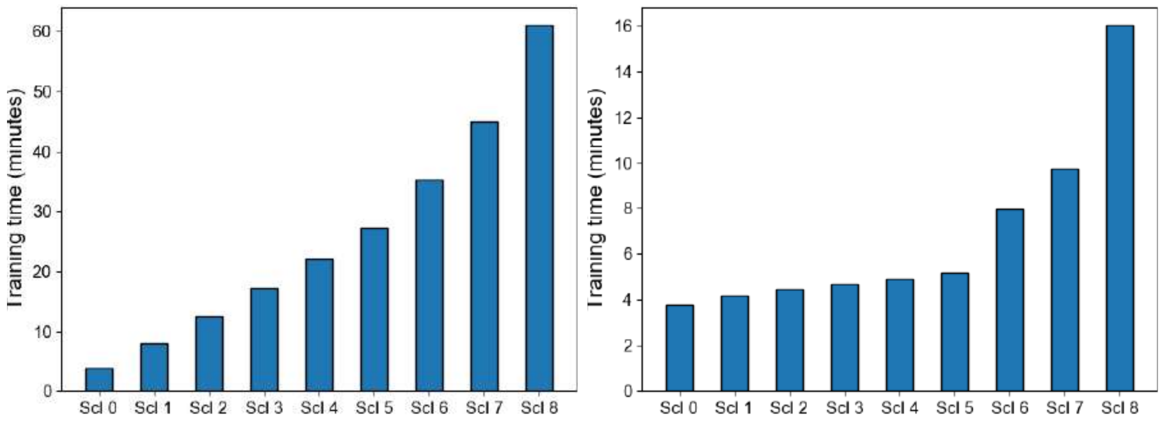
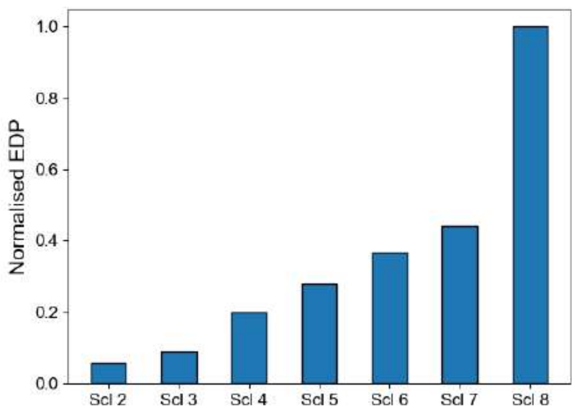


Figure 6.5: Effect of training time with the number of scales for a single image GAN: (Left) Baseline SinGAN training time across scales and (Right) Optimized SETGAN training

6.4.1 SETGAN vs. PGAN and SinGAN

We compare the memory and training time of SETGAN in comparison to state-of-the-art GAN models: PGAN, a multi-image GAN [75], and SinGAN, a single-image GAN [100].

Results for memory and training time. PGAN uses multi-scale GAN models and is trained on specific datasets with thousands of images. SinGAN performs serial training using a single image. Table 6.1 shows the results comparing SETGAN with PGAN and SinGAN. SETGAN and SinGAN models consume 34x less memory than the PGAN model. This is expected because PGAN needs to store the information regarding different kinds of images used for training, whereas SETGAN and SinGAN



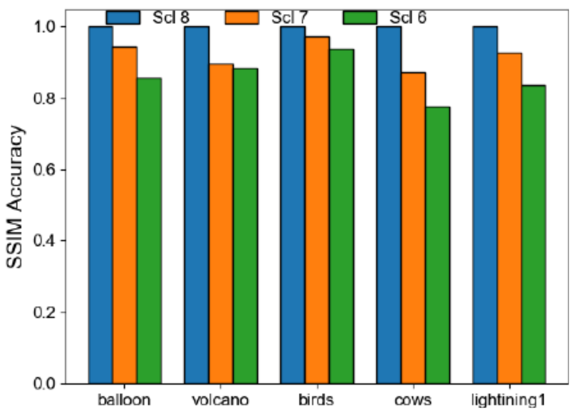


Figure 6.6: Energy and accuracy trade-off of SETGAN for unconditional photo-realistic image synthesis on mobile platform: (Left) Normalised EDP consumed at different scales of SETGAN generator network and (Right) SSIM Accuracy of different images across scales 6, 7, and 8.

architectures are intended to be trained only on a single image. Furthermore, the SETGAN memory can be further optimized with progressive update mode as discussed later. The average training times of PGAN, SinGAN, and SETGAN are 24 hours, 1 hour, and 15 minutes respectively. Thus, PGAN and SinGAN are not suitable for time-sensitive image editing applications. The fast training time of SETGAN is due to our proposed parallel training.

Effect of parallel training. In Fig. 6.5 (left), we show the time required to train the baseline SinGAN model of scales 0 to 8. We can see that training time increases gradually with the number of scales (4 mins for the coarsest scale and 60 mins to train for all scales) due to serial training. On the other hand, due to parallel training,

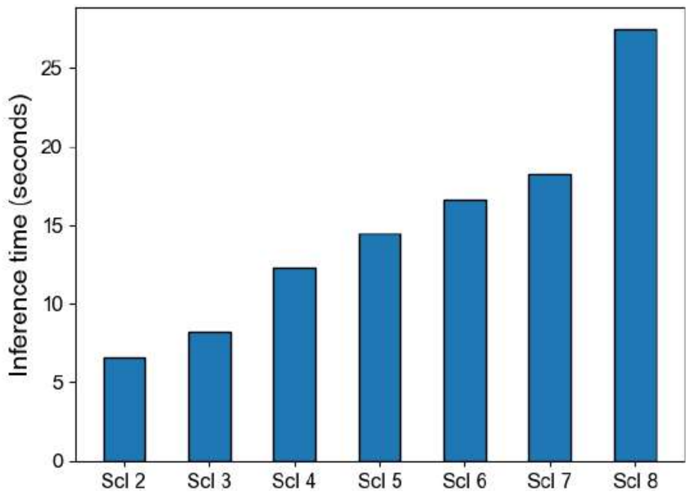


Figure 6.7: Inference time with SETGAN for photo-realistic image synthesis on mobile platforms: Variation of inference time from scales 2 to 8.

in fig. 6.5 (right) SETGAN can train all nine scales in 15 minutes resulting in 4X improvement over SinGAN. All these results demonstrate that SETGAN is much better suited for image-editing applications

6.4.2 SETGAN Inference across Scales

The SETGAN model trained on the server is sent to the client (mobile device). In this section, we discuss the results of SETGAN inference on mobile platforms.

Accuracy of Generator vs. Normalised EDP. As shown in Fig. 6.6, we compare the accuracy and energy consumed by the generator network of SETGAN at different scales. Recall that we can control the overall compute of SETGAN using the quality estimate threshold of T. By varying this threshold, we can exit early reducing the

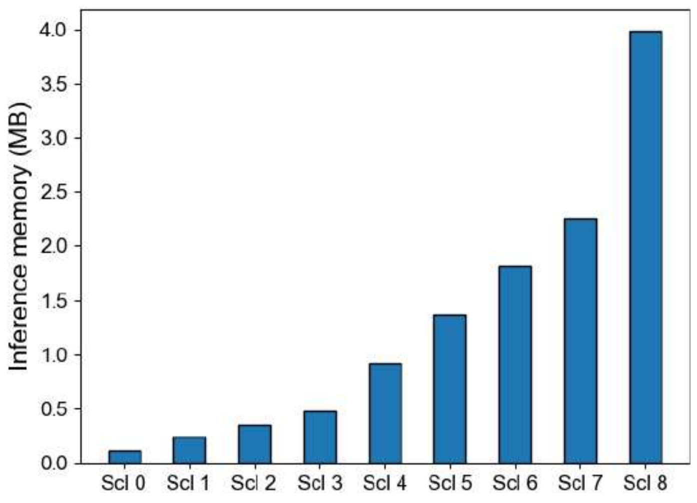


Figure 6.8: Performance of SETGAN with client-server architecture. Variation of inference model memory across scales from 0 to 8.

number of scales used by the image-editing application. We compare the normalized EDP and SSIM accuracy trade-off by a varying threshold of T. In figure 6.6(left), we plot the normalized EDP consumed for different scales by averaging across all images in the dataset. We observe that there is a 56% reduction in energy from scale 8 to scale 7. For the same reduction in energy, in fig. 6.6 (right), we see a drop of 6%, 10%, 3%, 13%, 7% in SSIM accuracy for different images such as balloon, volcano, birds, cows, and lightning. Similarly, the energy drops by another 8% form scale 7 to 6 for a drop of 10%, 1%, 4%, 9%, 8% in SSIM accuracy respectively. We see that for simple images such as balloons and birds (contains smaller structures), we see a very small drop in accuracy across scales, thereby use a fewer number of scales and consumes 56% lesser energy. GANs are generally evaluated for the capability to generate varied images and finding good metrics is still a open problem within ML community. Hence, qualitative results in the form of generated images are shown. Although we use SSIM to measure the closeness with the input image, the drop in SSIM accuracy does not reflect the true performance of GANs. Therefore, similar to prior work, we will present qualitative results for our SETGAN in subsequent sections as user-experience is more important for image-editing applications.

Inference Time. In this part, we discuss the time taken for running the SETGAN generator model across different scales. As shown in fig 6.7, we see that there is a 30% drop in inference time from scale 8 to scale 7. We also see another 10% drop in inference time from scale 7 to scale 6. From these results, we make the observation that by reducing the number of scales of a generator, we will be able to get faster feedback for different image-editing applications.

6.4.3 Performance of Client-Server Architecture

In this section, we study the properties of the SETGAN model which enable the different modes of the client-server architecture. The latency for the client-server architecture depends on the memory of the SETGAN generator model across different scales sent from the server to the mobile device. We discuss the impact of memory size of the model for the different modes of the SETGAN architecture.

Parallel training (Mode #2). In this mode, after training the SETGAN model, the entire model is sent to the mobile device. In Fig. 6.8, we show the inference

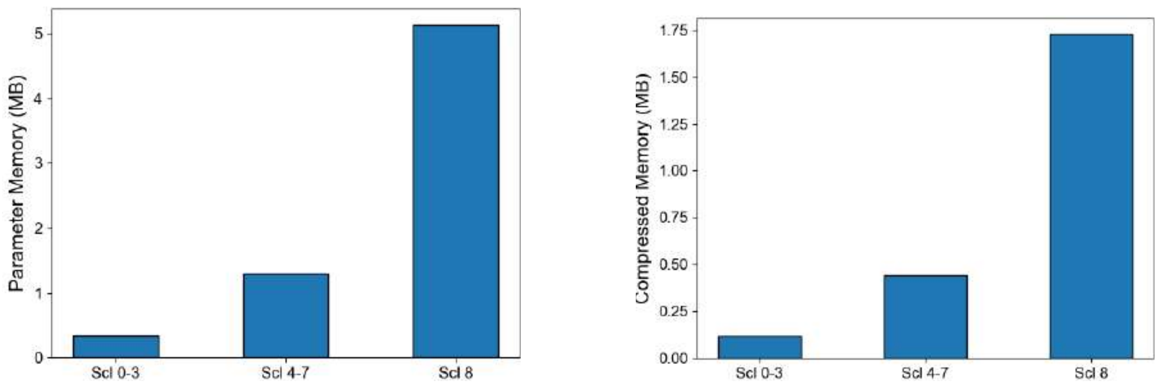


Figure 6.9: Memory consumed by SETGAN generator at each scale: (Left) Memory size of parameters and (Right) Compressed memory size.

memory required for the optimal number of scales of the SETGAN generator model. The generator for the finest-scale (8) takes 4MB of memory. The latency to transfer a 4MB model is minuscule when compared to the training time. From the Fig. 6.8, we see a 44% decrease in memory of the SETGAN generator model from scale 8 to scale 7 and another 19% decrease in memory when we go from scale 7 to scale 6

Progressive update (Mode #3). is a mode where we use the image-editing application as and when a model is available for use. The latency also depends on the model size. In fig. 6.9 (Left), we show the memory consumed by the SETGAN generator at each scale. Generator model at scale 0-3 consumes the smallest memory across all scales (0-8). This is because the first four scales of the network consist of 32 filters. The number of filters doubles for every four scales, i.e., scales 4-7 consists of 64 filters and scale 8 consists of 128 filters. This explains the memory consumed at the respective scales. Also, the compressed memory consumed at each scale reduces the memory size by a factor of 3x as shown in Fig. 6.9 (Right). The latency for a model to be used on the mobile device depends on the training time as shown in figure 6.5 and the time it takes to transfer the trained model to the mobile. For the first four scales, the models will be trained at around the same time of 4-5 minutes and the size of the model is 120KB each, which could be transferred in negligible time. For a progressive update mode of operation, we could start using the generator model from 8 minutes at scale 6, and successively use the scale 7 model at 10 minutes and use all the 8 scales at 16 minutes. Thus, the image-editing application could be improved adaptively to further reduce the overall usage performance by another 50%. Hence, the overall running time from "send for training" to "Editing" depends on three major components: a) The Training time ranges between 4-16 minutes based on scale; b) The model transfer time which is negligible due to small model and image size; and c) The Inference time (from the previous section) of around 15-25 seconds for each edit operation. However, the user interaction depends only on the inference time since the user starts editing only after the model is received on the edge device.

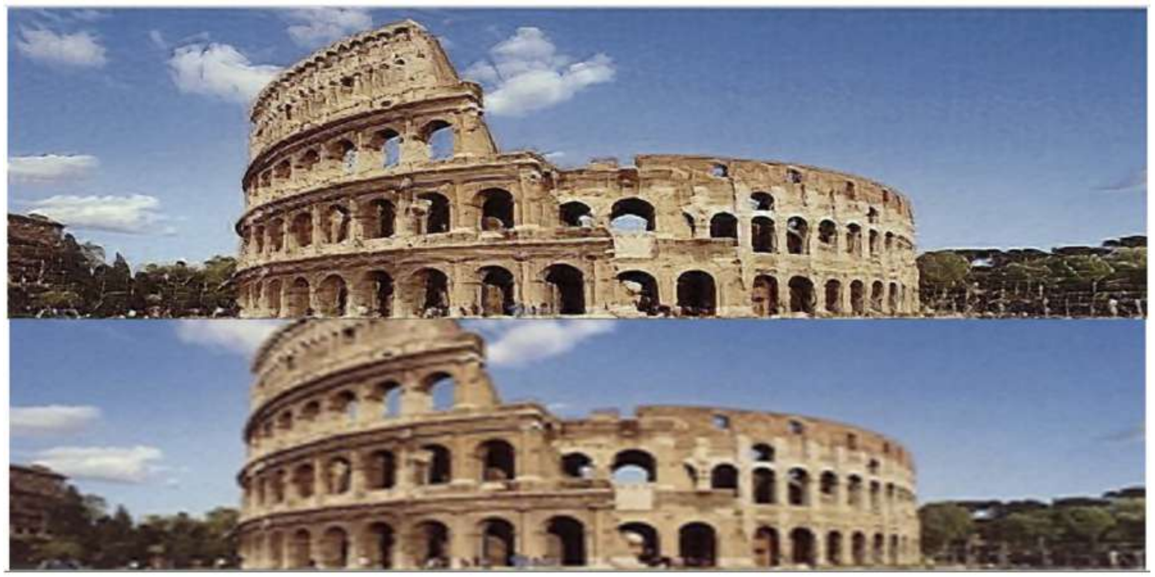


Figure 6.10: Super-resolution: Colloseum image scaled by a factor of 4x generated using (top) SETGAN model and (bottom) Bilinear interpolation.

6.4.4 Image-Editing Applications

We now discuss the quantitative and qualitative results of SETGAN for four diverse image-editing applications.

Super-resolution. We train a Colosseum image using the SETGAN model for super-resolution applications. Using the trained SETGAN model, we would be able to scale the low-resolution image for different scales such as 2x, 4x, 8x, etc. In Fig. 6.10, we scale the 256x256 image by 4x factor to produce 1024x1024 output images using SETGAN model (top) and bilinear interpolation (bottom). Since the bilinear interpolation is not a generative model, the output image is blurred. However, the

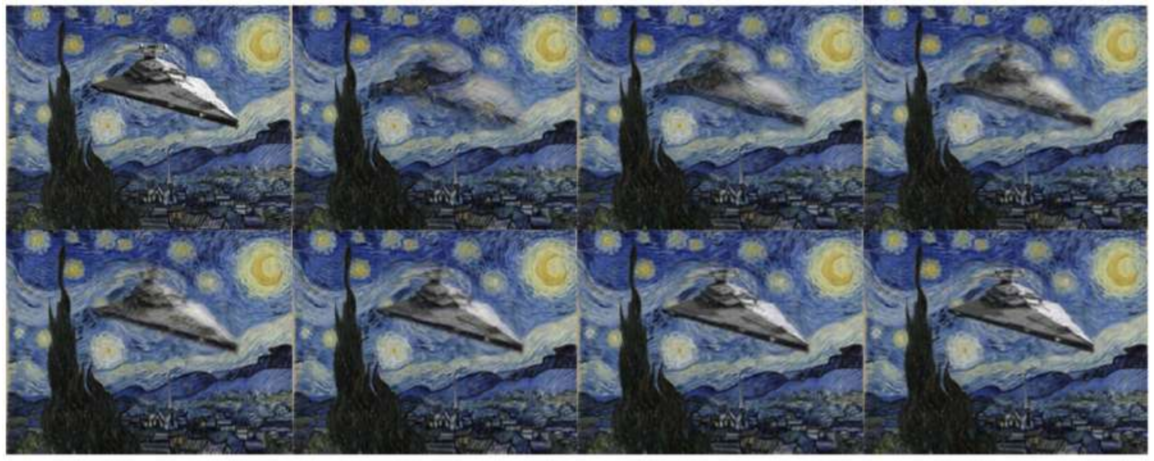


Figure 6.11: Harmonization: Starry-night image with space ship object overlayed. From top-left to bottom-right: (1) Input overlayed image, and (2)-(8) Images generated by injecting image at scales (2) to (8) in the SETGAN model.

image generated using SETGAN has sharp and crisp features since it uses the features at lower-resolution to generate features at higher-resolution. The same effect is seen even at higher scale factors of 8x and 16x.

Harmonization. In figure 6.11, we show the results of using a trained SETGAN model in harmonizing (blending) a spaceship object overlayed on a starry-night image. The images (2)-(8) are generated by injecting images at scales (2) to (8) of the SETGAN model. Based on the scale at which the image is introduced, we find that the degree of blending between the foreground and background image varies. An image generated from coarser scales (2)-(4), both the shape and the texture of the foreground spaceship is matched with the background starry-night painting. How-

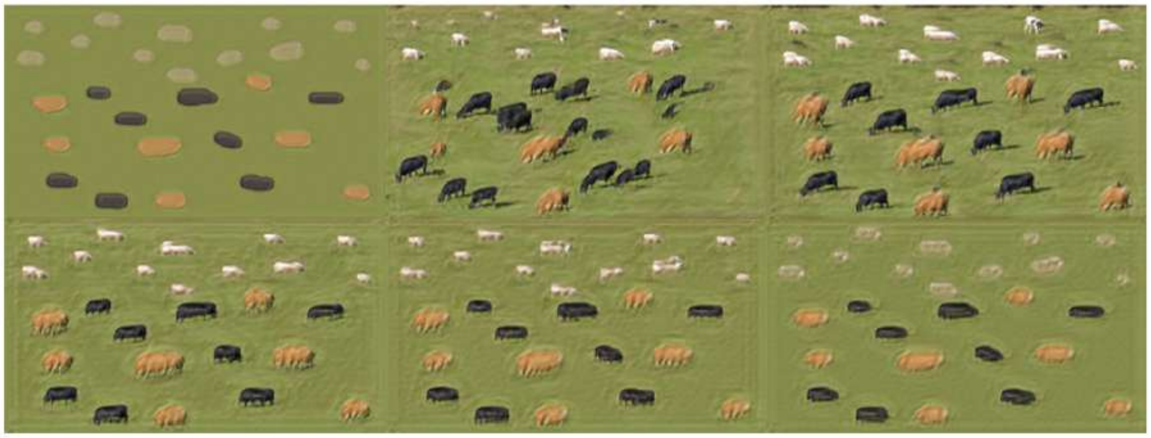


Figure 6.12: Paint2image: Cows image used for clip-art based editing. From top-left to bottom right: (1) Input clip-art image, (2)-(6) Images generated by injecting image at scales (1) to (5) in the SETGAN model.

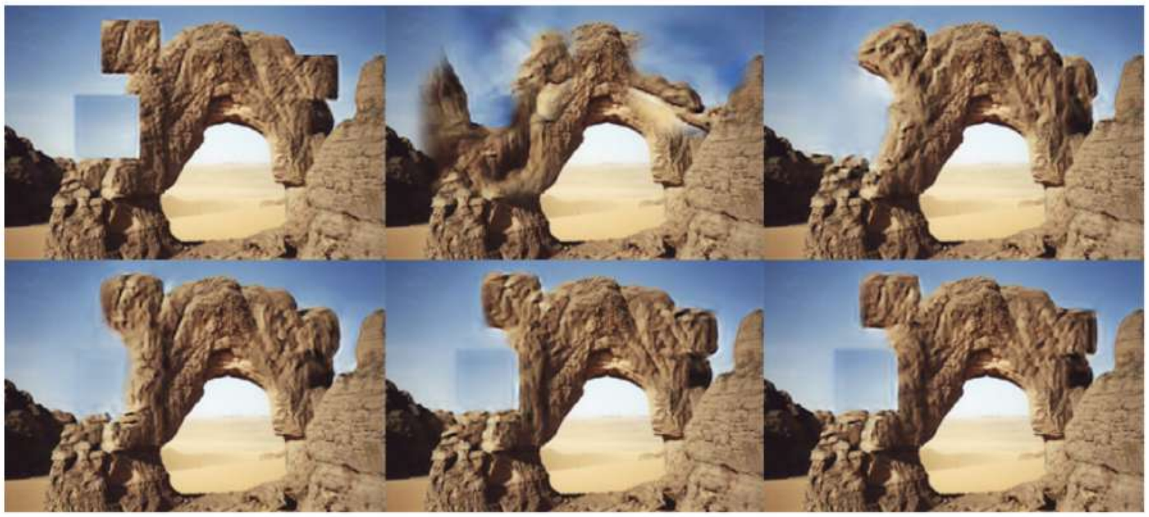


Figure 6.13: Editing: Stones image used for image editing. From top-left to bottom right: (1) Input edited image, (2)-(6) Images generated by injecting image at scales (1) to (5)

ever, for the finer scales (5)-(7), the shape of the spaceship is maintained while the texture of the spaceship alone is matched to the background. The energy consumed by the editing process also scales accordingly based on the scale at which the image is injected. Furthermore, in this application, the generative process is done only around the region of the spaceship and the energy consumed is reduced accordingly only for that smaller part of the image.

Paint2Image. In figure 6.12, we show the results of feeding the clip-art image for the cows' image trained on the SETGAN model. We feed the clip-art image from scales (1) to (5) of the SETGAN model. We can see high-quality features generated by the SETGAN model when injected at coarser scales (1) and (2).

Editing In Fig. 6.13, we show the results of feeding the edited stone image trained on the SETGAN model. We feed the stone image from scales (1) to (5) of the SETGAN model. For an editing application, feeding the image at the coarsest scale (1) creates artifacts. This is because, at the coarse-scale, it loses the local information. However, higher scales (4)-(5) does not have enough detail. Scales (2)-(3) seems to be the optimal scale for the right set of details.

From the above results, we can observe that we need to choose the appropriate scale based on the image manipulation task. Fig. 6.6 (top) shows the energy consumed by SETGAN across different scales. The energy spent by different applications depends on the scale at which these applications operate. A Paint2Image task

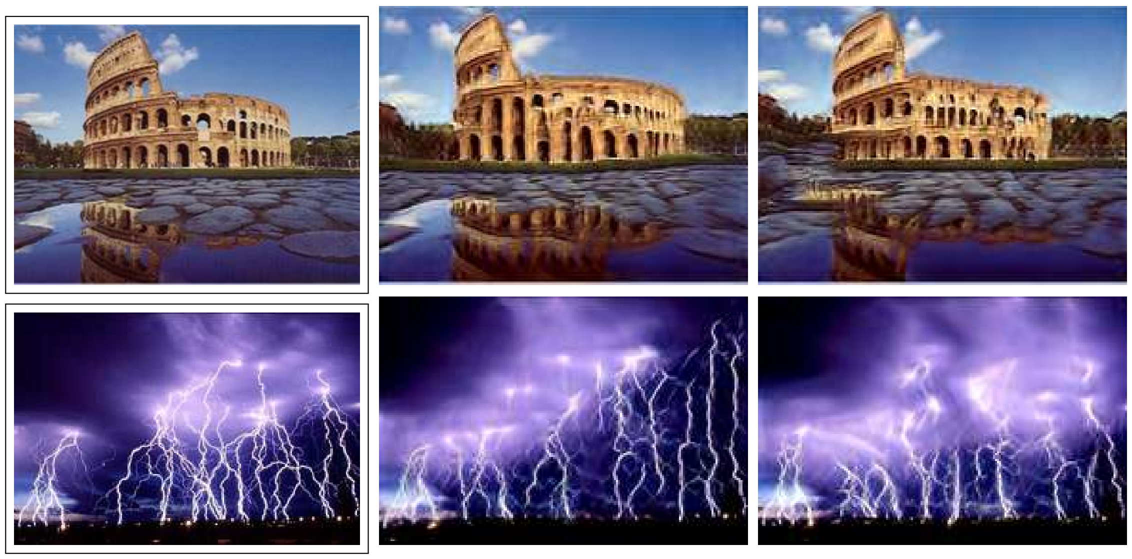


Figure 6.14: Random images generated using SETGAN model with early exit and parallel training: (Top) Colosseum and (Bottom) Lightning. Leftmost image is the input source image while the next two are the random images.

operating at the coarsest scale spends the largest energy while the Harmonization task operating at the finer scales consumes the least energy. Since the editing task works at the intermediate scale, it consumes energy between the above two tasks. The energy spent also depends on the size of the image on which the application acts on. Applications like Harmonization and Editing are localized and hence, spend less energy when compared to paint2image and super-resolution which work on the whole image.

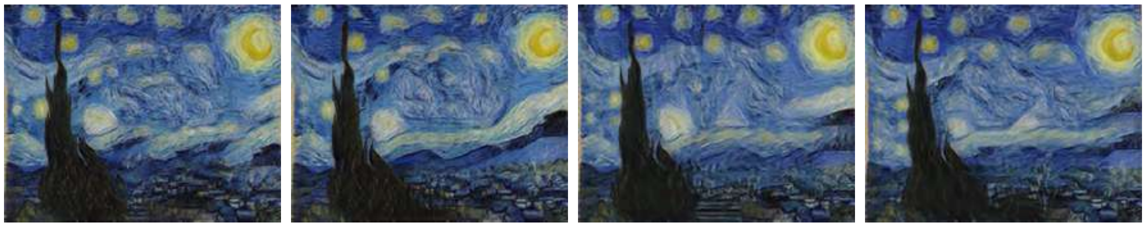


Figure 6.15: Comparing random images of Starry Night generated using SinGAN (left two) and SETGAN (right two) model

6.4.5 *Qualitative Analysis*

Fig. 6.14 shows some random images generated using SETGAN model. The leftmost image is the source image while the images at center and right are the random images generated using SETGAN model. The overall characteristics of the source image are maintained by our model, but still, the images are different. This can be seen from the number and shape of the windows of the Colosseum image. However, similar to the source image, we see the reflections of the building in the water in all the three images. A similar observation could be made in the number of stars and the swirls in the starry-night image. Fig. 6.14 shows the random images generated using the early exit with a reduced number of scales. Even with the reduced number of scales, we see that the images maintain the same characteristics as the un-optimized SETGAN while providing around 56% reduction in energy. Fig. 6.15 compares images generated using our baseline SinGAN and those generated with SETGAN. The image characteristics are similar between SinGAN and SETGAN generated images.

CHAPTER 7. CONCLUSIONS

Edge AI is an enabling technology for many emerging real-world applications by using deep neural networks for automated predictions and decisions. This dissertation highlighted the various challenges to develop energy-efficient Edge AI, and discussed a adaptive framework to achieve a desired trade-off between accuracy, energy, and latency. The key insight is performing adaptive inference depending on the hardness of input examples by solving what to compute and how to compute in a data-driven manner [65]. We illustrated the generality of this framework by providing three qualitatively different instantiations namely, convolutional neural networks (CNN), graph convolutional networks (GCN), and generative adversarial networks (GAN). The results demonstrated the effectiveness of the framework to achieve significant savings in energy and inference time with little to no loss in prediction accuracy. However, for some deep learning models where the data flow is not linear (e.g., transformers), our framework may not be directly applicable and will require some changes to incorporate adaptability. Future work includes studying the design optimization [37, 77, 72, 32, 28, 27, 34] of Resistive random-access memory (ReRAM)-based processing-in-memory accelerators [117, 3, 70] and monolithic 3D integration [91, 69, 23, 81, 82] for Edge AI ; studying more general multi-level inference where the routing paths need not be serial between different levels; and cross-layer co-design solutions to exploit additional opportunities to develop energy-efficient and secure Edge AI.

7.1 Summary of Contributions

In this section we enumerate summary of each instantiation of our framework and how we could extend these ideas in the future

- First, motivated by the challenges associated with performing inference using deep neural networks on resource-constrained embedded systems, we studied a adaptive solutions based on the formalism that allows us to employ classifiers of varying complexity depending on the hardness of input examples. Our proposed optimization algorithm to automatically configure pre-trained network for a specified trade-off between accuracy and energy consumption on a target hardware platform is very effective. Results on four different C2F Nets for image classification show that using optimized C2F Net we can significantly reduce the overall energy with no loss in accuracy when compared to a baseline solution, where predictions for all input examples are made using the most complex network. Though we have demonstrated the method for four nets this idea could be extended to other architectures like MobileNets [56]. Future directions include evaluation of our framework on deep networks such as

MobileNets [56], studying automated approaches for C2F nets design, heterogeneous architectures for embedded systems in the context of inference using deep networks, machine learning based approaches for software and hardware co-design using C2F Nets, and dynamic power management to further improve the overall energy-efficiency.

- Next, we introduced a novel approach based on the LEANets formalism to trade-off energy and accuracy of inference on mobile platforms at runtime. We also propose a principled algorithm to find optimized LEANet configurations by reusing pre-trained neural networks. Our LEANet based approach can be used as a complementary wrapper for other existing techniques to deploy deep neural networks on mobile devices. Results with optimized LEANets constructed from pre-trained networks including VGG and MobileNet on image classification datasets including ImageNet show significant energy gains with minimal loss in accuracy. Future work includes applying dynamic resource management techniques [76] [86] and task mapping optimization based on machine learning [109, 73, 32] to further improve the power and thermal trade-offs in heterogeneous mobile SoCs; and generalizing the LEANets formalism from simple classification tasks to structured output prediction tasks [35, 36, 80, 31, 84] including semantic segmentation and scene understanding that commonly arise in robotics and autonomous driving applications.

- Then, we introduced and evaluated a novel approach referred as PETNets to generate 3D object shapes from color images using graph convolution networks on resource-constrained mobile platforms. Our results show that optimized PETNets can achieve 20 to 37 percent gain in energy for negligible loss in IoU score when compared to SOTA Pixel2Mesh networks. Future work includes applying dynamic resource management techniques [102, 99] to further improve the power and thermal trade-offs for deployment on heterogeneous SoCs.
- Finally, we propose and evaluate a new SETGAN framework to trade-off energy and scale of inference using single-image GANs for image manipulation tasks on resource-constrained mobile platforms. The key novelty includes a multi-scale GAN model that can be efficiently trained for each input image using a parallel algorithm and allows efficient inference by automatically selecting the appropriate scale to achieve a user-specified quality threshold. Our results show that optimized SETGAN models can achieve a 56 percent gain in energy for negligible loss in SSIM accuracy and 4x speedup in training when compared to state-of-the-art SinGAN models. Future work includes applying dynamic resource management techniques [102, 99, 103, 89, 26] to further improve the energy-efficiency

Future work includes extending our framework to more complex deep learning models such as transformers and applications for time-series domain [13, 14, 15, 57].

One could consider improving our current instantiations of the adaptive inference framwork such as improving the performance of 3D shape generation by using multiple images as input or extend image synthesis by using stable diffusion models. From the hardware perspective, we could complement our implementation by using approximate computing, dynamic resource management, and energy harvesting [58, 115] to further improve the power and thermal trade-offs which we did not address in our experiments. We can also consider designing hardware accelerators [78] for the specific application workloads [106] using BO [29, 30] and consider hardware-aware model pruning methods [71, 94] to improve performance and energy-efficiency.

BIBLIOGRAPHY

[1] A. Howard et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[2] Angel X. Chang et al. ShapeNet: An information-rich 3D model repository. *CoRR*, abs/1512.03012, 2015.

[3] Aqeeb Iqbal Arka, Biresh Kumar Joardar, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. ReGraphX: NoC-enabled 3D heterogeneous ReRAM architecture for training graph neural networks. In *DATE*, 2021.

[4] Michal Irani Assaf Shocher, Nadav Cohen. ”zero-shot” super-resolution using deep internal learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[5] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Information-theoretic multi-objective Bayesian optimization with continuous approximations. *CoRR*, abs/2009.05700, 2020.

[6] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Max-value entropy search for multi-objective Bayesian optimization with constraints. *CoRR*, abs/2009.01721, 2020.

[7] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Multi-fidelity multi-objective Bayesian optimization: An output space entropy search approach. In *AAAI conference on Artificial Intelligence (AAAI)*, 2020.

[8] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Uncertainty aware search framework for multi-objective Bayesian optimization with constraints. *CoRR*, abs/2008.07029, 2020.

[9] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Output Space Entropy Search Framework for Multi-Objective Bayesian Optimization. *JAIR*, 2021.

[10] Syrine Belakaria et al. Machine learning enabled fast multi-objective optimization for electrified aviation power system design. In *IEEE Energy Conversion Congress and Exposition*, 2020.

[11] Syrine Belakaria et al. Uncertainty-aware search framework for multi-objective Bayesian optimization. In *AAAI Conference on Artificial Intelligence*, 2020.

[12] Belakaria et al. Max-value entropy search for multi-objective Bayesian optimization. In *NeurIPS*, 2019.

[13] Taha Belkhouja and Janardhan Rao Doppa. Analyzing deep learning for time-

series data through adversarial lens in mobile and iot applications. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):3190–3201, 2020.

[14] Taha Belkhouja and Janardhan Rao Doppa. Adversarial framework with certified robustness for time-series domain via statistical features. *Journal of Artificial Intelligence Research (JAIR)*, 73:1435–1471, 2022.

[15] Taha Belkhouja, Yan Yan, and Janardhan Rao Doppa. Training robust deep models for time-series domain: Novel algorithms and theoretical analysis. In *Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, pages 6055–6063. AAAI Press, 2022.

[16] Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial GAN. *CoRR*, abs/1705.06566, 2017.

[17] Caffe-HRT. https://github.com/OAID/Caffe-HRT, 2018. Online; Accessed 29 March,2018.

[18] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Using dataflow to optimize energy efficiency of deep neural network accelerators. *IEEE Micro*, 2017.

[19] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, and Ling Li. Dadiannao: A machine-learning supercomputer. In *Proceedings of MICRO*, pages 609–622, 2014.

[20] François Chollet. Xception: Deep learning with depthwise separable convolutions. 2016.

[21] Christopher Bongsoo Choy et al. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *ECCV*, 2016.

[22] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[23] Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. Monolithic 3D-enabled high performance and energy efficient network-on-chip. In *ICCD*, pages 233–240, 2017.

[24] CIFAR10 Dataset. https://www.cs.toronto.edu/~kriz/cifar.html, 2009. Online; accessed 29 March 2018.

[25] MNIST Dataset. http://yann.lecun.com/exdb/mnist/. Online; accessed 29 March 2018.

[26] Aryan Deshwal, Syrine Belakaria, Ganapati Bhat, Janardhan Rao Doppa, and Partha Pratim Pande. Learning pareto-frontier resource management policies for heterogeneous socs: An information-theoretic approach. In *(DAC)*, 2021.

[27] Aryan Deshwal, Syrine Belakaria, and Janardhan Rao Doppa. Bayesian optimization over hybrid spaces. In *ICML*, 2021.

[28] Aryan Deshwal, Syrine Belakaria, and Janardhan Rao Doppa. Mercer features for efficient combinatorial Bayesian optimization. In *AAAI*, 2021.

[29] Aryan Deshwal, Syrine Belakaria, Janardhan Rao Doppa, and Dae Hyun Kim. Bayesian optimization over permutation spaces. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 6515–6523. AAAI Press, 2022.

[30] Aryan Deshwal and Janardhan Rao Doppa. Combining latent space and structured kernels for bayesian optimization over combinatorial spaces. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 8185–8200, 2021.

[31] Aryan Deshwal, Janardhan Rao Doppa, and Dan Roth. Learning and inference for structured prediction: A unifying perspective. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6291–6299. ijcai.org, 2019.

[32] Aryan Deshwal, Nitthilan Kannappan Jayakodi, Biresh Kumar Joardar, Ja-

nardhan Rao Doppa, and Partha Pratim Pande. MOOS: A multi-objective design space exploration and optimization framework for NoC enabled many-core systems. *ACM TECS*, 2019.

[33] Ruizhou Ding, Zeye Liu, R. D. Shawn Blanton, and Diana Marculescu. Quantized deep neural networks for energy efficient hardware-based inference. In *Proceedings of ASP-DAC*, pages 1–8, 2018.

[34] Janardhan Rao Doppa. Adaptive experimental design for optimizing combinatorial structures. In *IJCAI*, pages 4940–4945, 2021.

[35] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Hc-search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research*, 50:369–407, 2014.

[36] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured prediction via output space search. *Journal of Machine Learning Research*, 15(1):1317–1350, 2014.

[37] Janardhan Rao Doppa, Justinian Rosca, and Paul Bogdan. Autonomous design space exploration of computing systems for sustainability: Opportunities and challenges. *IEEE Design and Test*, 36(5):35–43, 2019.

[38] Aojun Zhou et al. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv:1702.03044*, 2017.

[39] Bobak Shahriari et al. Taking the human out of the loop: A review of bayesian optimization. In *Proceedings of the IEEE*, volume 104, pages 148–175, 2016.

[40] Christian Szegedy et al. Going deeper with convolutions. 2014.

[41] J. Albericio et al. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of ISCA*, pages 1–13, 2016.

[42] Song Han et al. Eie: Efficient inference engine on compressed deep neural network. In *Proceedings of ISCA*, pages 243–254, 2016.

[43] Vivienne Sze et al. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[44] Wenzheng Hu et al. Fast branch convolutional neural network for traffic sign recognition. In *IEEE Intelligent Transportation Systems Magazine*, volume 9, pages 114–126, July 2017.

[45] Y. LeCun et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[46] Francois Fleuret and Donald Geman. Coarse-to-fine face detection. In *International Journal of Computer Vision*, volume 41, pages 85–107, 2001.

[47] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. Tetris: Scalable and efficient neural network acceleration with 3d memory. *SIGARCH Computer Architecture News*, 2017.

[48] Jeff Dean Geoffrey Hinton, Oriol Vinyals. Distilling the knowledge in a neural network. *NIPS Deep Learning Workshop*, 2014.

[49] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[50] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *NIPS*, pages 5767–5777, 2017.

[51] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[52] Cong Hao, Jordan Dotzel, Jinjun Xiong, Luca Benini, Zhiru Zhang, and Deming Chen. Enabling design methodologies and future trends for edge AI: specialization and codesign. *IEEE Des. Test*, 38(4):7–26, 2021.

[53] Haoqiang Fan et al. A point set generation network for 3D object reconstruction from a single image. In *CVPR*, 2017.

[54] Daniel Hernandez-Lobato, Jose Miguel Hernandez-Lobato, Amar Shah, and Ryan P. Adams. Predictive entropy search for multi-objective bayesian optimization. In *ICML*, pages 1492–1501, 2016.

[55] Hiroharu Kato et al. Neural 3D mesh renderer. In *CVPR*, 2018.

[56] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[57] Dina Hussein, Taha Belkhouja, Ganapati Bhat, and Janardhan Rao Doppa. Reliable machine learning for wearable activity monitoring: Novel algorithms and theoretical guarantees. In *Proceedings of 41st International Conference on Computer-Aided Design (ICCAD)*, 2022.

[58] Dina Hussein, Ganapati Bhat, and Janardhan Rao Doppa. Adaptive energy management for self-sustainable wearables in mobile health. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 11935–11944. AAAI Press, 2022.

[59] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[60] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[61] Nitthilan Kanappan Jayakodi, Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models. *ACM Trans. Embedded Comput. Syst.*, 19(1):4:1–4:24, 2020.

[62] Nitthilan Kanappan Jayakodi, Janardhan Rao Doppa, and Partha Pratim Pande. SETGAN: Scale and energy trade-off gans for image applications on mobile platforms. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[63] Nitthilan Kanappan Jayakodi, Janardhan Rao Doppa, and Partha Pratim Pande. PETNet: Polycount and energy trade-off deep networks for producing 3D objects from images. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[64] Nitthilan Kannappan Jayakodi, Anwesha Chatterjee, Wonje Choi, Janard-

han Rao Doppa, and Partha Pratim Pande. Trading-off accuracy and energy of deep inference on embedded systems: A co-design approach. *IEEE TCAD*, 37(11):2881–2893, 2018.

[65] Nitthilan Kannappan Jayakodi, Janardhan Rao Doppa, and Partha Pratim Pande. A general hardware and software co-design framework for energy-efficient edge AI. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*, pages 1–7. IEEE, 2021.

[66] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. Texture synthesis with spatial generative adversarial networks. *ArXiv*, abs/1611.08207, 2016.

[67] Jhony K. Pontes et al. Image2Mesh: A learning framework for single image 3D reconstruction. In *ACCV*, 2018.

[68] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[69] Biresh Kumar Joardar, Aqeeb Iqbal Arka, Janardhan Rao Doppa, and Partha Pratim Pande. 3D++: Unlocking the next generation of high-performance and energy-efficient architectures using M3D integration. In *DATE*, 2021.

[70] Biresh Kumar Joardar, Aryan Deshwal, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. High-throughput training of deep CNNs on ReRAM-based heterogeneous architectures via optimized normalization layers. *IEEE TCAD*, 2021.

[71] Biresh Kumar Joardar, Janardhan Rao Doppa, Hai Li, Krishnendu Chakrabarty, and Partha Pratim Pande. Realprune: Reram crossbar-aware lottery ticket pruned cnns. *CoRR*, abs/2111.09272, 2021.

[72] Biresh Kumar Joardar, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Learning-based application-agnostic 3D NoC design for heterogeneous manycore systems. *IEEE Transactions on Computers*, 68(6):852–866, 2018.

[73] Biresh Kumar Joardar, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Learning-based application-agnostic 3d noc design for heterogeneous manycore systems. *IEEE Transactions on Computers*, 68(6):852–866, 2018.

[74] Kaiming He et al. Deep residual learning for image recognition. In *IEEE CVPR*, pages 770–778, 2016.

[75] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive

growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

[76] Ryan Gary Kim, Wonje Choi, Zhuo Chen, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Imitation learning for dynamic VFI control in large-scale manycore systems. *IEEE Transactions on VLSI Systems (TVLSI)*, 25(9):2458–2471, 2017.

[77] Ryan Gary Kim, Janardhan Rao Doppa, and Partha Pratim Pande. Machine learning for design space exploration and optimization of manycore systems. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, page 48. IEEE, 2018.

[78] Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Machine learning and manycore systems design: A serendipitous symbiosis. *Computer*, 51(7):66–77, 2018.

[79] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Deep convolutional neural network inference with floating-point weights and fixed-point activations. *CoRR*, abs/1703.03073, 2017.

[80] Michael Lam, Janardhan Rao Doppa, Sinisa Todorovic, and Thomas G. Dietterich. HC-search for structured prediction in computer vision. In *IEEE Con-*

ference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, pages 4923–4932, 2015.

[81] Dongjin Lee, Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. Performance and thermal tradeoffs for energy-efficient monolithic 3D network-on-chip. *ACM TODAES*, 23(5):60:1–60:25, 2018.

[82] Dongjin Lee, Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. Impact of electrostatic coupling on monolithic 3D-enabled network on chip. *ACM TODAES*, 24(6):62:1–62:22, 2019.

[83] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceeding of CVPR*, pages 5325–5334, 2015.

[84] Chao Ma, F. A. Rezaur Rahman Chowdhury, Aryan Deshwal, Md Rakibul Islam, Janardhan Rao Doppa, and Dan Roth. Randomized greedy search for structured prediction: Amortized inference and learning. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5130–5138. ijcai.org, 2019.

[85] Yufei Ma, Naveen Suda, Yu Cao, Jae sun Seo, and Sarma Vrudhula. Scalable

and modularized rtl compilation of convolutional neural networks onto fpga. In *Proceedings of FPL*, pages 1–8, 2016.

[86] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2842–2854, Dec 2019.

[87] Sumit Mandal, Ganapati Bhatt, Chetan Arvid Patel, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Ogras. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE TVLSI*, 2019.

[88] Sumit K. Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Ümit Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM TODAES*, 25(3), 2020.

[89] Sumit K. Mandal, Ümit Y. Ogras, Janardhan Rao Doppa, Raid Zuhair Ayoub, Michael Kishinevsky, and Partha Pratim Pande. Online adaptive learning for runtime resource management of heterogeneous socs. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020.

[90] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and

measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[91] Shouvik Musavvir, Anwesha Chatterjee, Ryan Gary Kim, Dae Hyun Kim, Janardhan Rao Doppa, and Partha Pratim Pande. Power, performance, and thermal trade-offs in M3D-enabled manycore chips. In *DATE*, 2020.

[92] Katayoun Neshatpour, Farnaz Behnia, Houman Homayoun, and Avesta Sasan. Icnn: An iterative implementation of convolutional neural networks to enable energy and computational complexity aware dynamic approximation. In *Proceedings of DATE*, 2018.

[93] ODRRIOD-XU4. https://wiki.odroid.com/odroid-xu4/hardware/hardware, 2017. Online; Accessed 29 March 2018.

[94] Chukwufumnanya Ogbogu, Aqeeb Iqbal Arka, Biresh Kumar Joardar, Janardhan Rao Doppa, Hai Helen Li, Krishnendu Chakrabarty, and Partha Pratim Pande. Accelerating large-scale graph neural network training on crossbar diet. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 41(11):3626–3637, 2022.

[95] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *Proceeding of DATE*, 2016.

[96] Slav Petrov. Coarse-to-fine natural language processing. *PhD Thesis, Electrical Engineering and Computer Sciences, UC Berkeley*, 2009.

[97] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[98] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

[99] R.G. Kim et al.,. Imitation learning for dynamic VFI control in large-scale manycore systems. *TVLSI*, 25(9), 2017.

[100] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *Computer Vision (ICCV), IEEE International Conference on*, 2019.

[101] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[102] S. Mandal et al.,. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *TVLSI*, 27(12), 2019.

[103] S. Mandal et al.,. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM TODAES.*, 25(3):28:1–28:26, 2020.

[104] Hichem Sahbi. Coarse-to-fine deep kernel networks. In *ICCV Workshops*, pages 1131–1139, 2017.

[105] Scarselli et al. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009.

[106] Harsh Sharma, Sumit K. Mandal, Janardhan Rao Doppa, Ümit Y. Ogras, and Partha Pratim Pande. SWAP: A server-scale communication-aware chiplet-based manycore PIM accelerator. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 41(11):4145–4156, 2022.

[107] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[108] SmartPower2. https://wiki.odroid.com/accessory/power_supply_battery/smartpower2, 2018. Online; Accessed 29 March,2018.

[109] Das Sourav, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. Design-space exploration and optimization of an energy-efficient and reliable 3-d small-world network-on-chip. *IEEE Transactions*

on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 36(5):719–732, 2017.

[110] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*, pages 1015–1022, 2010.

[111] Dimitrios Stamoulis, Ting-Wu Chin, Anand Krishnan Prakash, Haocheng Fang, Sribhuvan Sajja, Mitchell Bognar, and Diana Marculescu. Designing adaptive neural networks for energy-constrained image classification. *CoRR*, abs/1808.01550, 2018.

[112] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. Cost-effective active learning for deep image classification. In *Proceeding of IEEE Transactions on Circuits and Systems for Video Technology*, 2017.

[113] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single RGB images. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI*, pages 55–71, 2018.

[114] Wenqi Xian, Patsorn Sangkloy, Varun Agrawal, Amit Raj, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Texturegan: Controlling deep image syn-

thesis with texture patches. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[115] Nuzhat Yamin, Ganapati Bhat, and Janardhan Rao Doppa. DIET: A dynamic energy management approach for wearable health monitoring devices. In *2022 Design, Automation & Test in Europe Conference & Exhibition, DATE 2022, Antwerp, Belgium, March 14-23, 2022*, pages 1365–1370. IEEE, 2022.

[116] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of CVPR*, 2017.

[117] Xiaoxuan Yang, Syrine Belakari, Biresh Kumar Joardar, Huanrui Yang, Janardhan Rao Doppa, Partha Pratim Pande, Krishnendu Chakrabarty, and Hai (Helen) Li. Multi-objective optimization of ReRAM crossbars for robust DNN inferencing under stochastic noise. In *Proceedings of the 40th International Conference on Computer-Aided Design (ICCAD)*, 2021.

[118] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. Slimmable neural networks. In *ICLR*, 2019.

[119] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. Slimmable neural networks. In *ICLR*, 2019.

[120] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 487–495, Cambridge, MA, USA, 2014. MIT Press.

[121] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *ACM Trans. Graph.*, 37(4), July 2018.

[122] Zhiyuan Zhou et al. Design of multi-output switched-capacitor voltage regulator via machine learning. In *Design, Automation and Test in Europe*, 2020.

[123] Eckart Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*, volume 63. Ithaca: Shaker, 1999.