OPTIMIZING LARGE-SCALE CAMERA ANALYTICS SYSTEMS WITH WIRELESS INSIGHTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Pan Hu

June 2021

This dissertation is online at: http://purl.stanford.edu/tq397gn7659

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Sachin Katti, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Balaji Prabhakar

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Zain Asgar

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Cameras are key enablers for a wide range of IoT use cases including smart cities, intelligent transportation, AI-enabled farms, and more. Many of them are connected to the Internet via wireless technologies like LPWAN(LoRaWAN, SigFox) and 5G. Conventional camera analytics systems treat the underlying network as a pipe, ignoring the fact that wireless links are lossy and wireless is a broadcast medium.

First, I present Starfish that is inspired by the lossy link insight. Starfish is a resilient image compression framework design for LPWAN that processes all the information loss in the application layer, thus simplifying the overall system design. To our best knowledge, it is the first DNN-based image compression framework that runs efficiently on low-cost AIoT hardware. It could be adapted to different task objectives and datasets, and exhibits graceful degradation in lossy scenarios without the need to design and tune source/channel coding. It uses neural network architecture search to generate DNN configuration thus reducing human labor and enable generalization to diverse AIoT hardware. Such end-to-end design reuses the DNN inference pipeline as in object detection/image classification tasks thus it could be easily adopted on various DNN accelerators.

Next, I present YOUO (You Only Upload Once) that leverages the broadcast insight. YOUO is an object re-identification system that shifts computation from cloud to devices for better privacy and lower network/computation cost. Each target object may have been captured multiple times by multiple cameras in large-scale camera networks. With the help of multicasting service provided in 5G eMBMS, YOUO enables efficient collaboration across cameras, thus significantly reduced network traffic by more than two orders of magnitude based on experiment results on a large-scale vehicle re-identification dataset. YOUO is 3x as efficient in bandwidth when compared with optimal single-camera filtering

with minimal accuracy loss while achieving 70%+ accuracy when uploading only once per object (the absolute minimum) for the entire camera network.

Previously Published Material

Chapter 3 revises a previous publication [52]: Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. "Starfish: resilient image compression for AIoT cameras." In Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pp. 395-408. 2020.

Acknowledgments

It has been a long journey from birth to Ph.D. I feel especially grateful to all the nice people I've met along the way, both in person and virtually. Thank you to my family, friends, and advisors from both the industry and academia.

I would miss the time working with Sachin Katti, my dissertation advisor. It is impossible for me to complete the dissertation without his brilliant insight, versatile skill set, and valuable feedback. I enjoy the luxury of pursuing research topics with little restriction while working with him. Also thanks for his patience in bearing with me. I'd like to thank Keith Winstein for all the help I've received. Keith has spent a considerable amount of time with me to improve the quality of this dissertation. He is my role model for doing good research, build useful systems, and convey ideas effectively.

I'd like to express my gratitude to my dissertation readers, Balaji Prabhakar and Zain Asgar. Thank you to Balaji for his insightful comments that inspired in-depth thinking of this work. It may be weird that I've received a great deal of help from Zain but we never actually met in person due to the pandemic. Thank you to Zain for help shaping the idea, design and evaluate the system for this dissertation. Special thanks to Haeyoung Noh for chairing the Oral Examining Committee. It's great to have your kindness and patience before and after the defense.

It is impossible for me to complete this dissertation without the help from my collaborators: Sandeep Chinchali, Rakesh Misra, Junha Im, and Tianshu Chu. Special thanks to Sandeep and Rakesh for guiding me through the early days of my Ph.D. I had the fortune of interned at two wonderful companies during the Ph.D. Special thanks to Yunfei Ma, Pengyu Zhang, Hongqiang Liu, and Rakesh Misra for the guidance and support in the industry, and for helping me make important life decisions.

Life at graduate school could be rough that alternating between stress and feeling lost. I'm fortunate to have received advice from Stanford SNSG members including Colleen Josephson, Eyal Cidon, Jenya Pergament, Samuel Joseph, Manikanta Kotaru. The conversations about research, teaching, and life helped me a lot to get through the rough time. Also, thank them for providing feedback on research and presentation during weekly meetings.

I'd like to thank so many wonderful people I've met on the Farm: Philp Levis, Hang Qu, Raejoon Jung, Sadjad Fouladi, Huizi Mao, Francis Yu Yan, Fei Xia, Shiyu Liu, Yilong Geng, Kai Zhang, Haihong Li, Yundong Zhang, Bisera More, Lancy Nazaroff and many others. Special thanks to Philp Levis for mentoring me during the first quarter at Stanford, I miss the happy days working in your group. Also, thank you to my "friends at large" that helped me escape from Ph.D. work on weekends.

I feel extremely grateful to have worked with Deepak Ganesan at UMass Amherst. He taught me how to shape ideas and carry out research agendas effectively. I wouldn't make it to the point without his generous support. Thanks for all the guidance from Ranveer Chandra, Bodhi Priyantha, and Anirudh Badam while working at MSR. I also would like to thank Xiaofang Jiang and Guobin Shen for mentoring me at MSRA and introducing me to the hall of science.

Lastly, I would like to thank my family for their unwavering love and support from my very first days to graduate school. I started the journey from the countryside to the heart of innovation because of my family. Words can't express my gratitude to them. Special thanks to my partner, Dan Zhang, for her hugs and tolerance. She encouraged me all the way during this Ph.D. This dissertation is dedicated to her and my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cameras deployed on a large scale over a wide area are the key enablers for a broad range of applications from transportation systems, business intelligence, AI(Artificial Intelligence)-enabled factory to precision agriculture [93]. Intelligent transportation systems leverage cameras for more efficient control of traffic lights, predicting subway traffic, and detect abnormal vehicles on the road, thus reducing congestion, increase safety and security [82, 17, 3]. Businesses install cameras for automated checkout in convenience stores, detecting safety hazards, and analyze customer flow for better planning in shopping malls. Oil refineries and farms use cameras to perform predictive maintenance and irrigation [91, 120] to reduce operation costs and increase production output.

Many cameras are connected to the Internet via wireless technologies like LPWAN (Low Power Wide Area Network, e.g. LoRaWAN/SigFox) and cellular links including 5G as they're lower cost and easier to deploy than wired ones. LPWAN cameras are typically battery-powered, comes with low cost and limited computation hardware. They upload data into the cloud that features powerful compute and massive storage via gateways. On the other hand, 5G cameras could be equipped with better computation and perform on-camera filtering to reduce network cost. They send data over 5G for multi-camera queries including unique object counting, re-identification, etc.

Conventional camera analytic systems treat the underlying network as a pipe, ignoring unique characteristics of wireless links such as wireless links are lossy, wireless is a

broadcast medium, and more. Although the pipe abstraction makes designing camera analytic systems easier, such abstraction often leads to inferior performance. A deep understanding of the underlying network could open up valuable opportunities for co-designing applications with the network, allowing the best use of the network and/or improved task performance.

1.1 Statement

In this dissertation, we attempt to answer the following question:

> *Should we incorporate the underlying wireless link characteristics, such as wireless links that are lossy and communicate over a shared medium, into the design of wide-area camera analytic systems?*

In the rest of this dissertation, we'll discuss how these insights contribute to the design of the camera analytics systems and how do we benefit from them.

We begin with the lossy link insight, focusing on TinyML(Tiny Machine Learning) cameras that send the image over LPWAN according to the results of DNN(Deep Neural Network). Instead of sending everything over the network, TinyML cameras use tiny, quantized DNNs for object detection/image classification to filter out irrelevant information. We reuse the TinyML hardware, build a lossy image compression framework that simplifying the design of hardware/software and leading to better task performance. We present details of this work in Chapter 3.

We then discuss the shared medium insight, focusing on 5G cameras and tasks that require collecting information from multiple cameras. With the help of shared medium insight, we design an efficient method of collaboration across cameras via unicast. It could significantly reduce network cost, especially in large-scale camera networks, while achieving comparable task performance. We present details of this work in Chapter 4.

We believe that a deeper understanding of the underlying wireless link could open up more opportunities in designing camera analytics systems, driving the innovation of new architecture and leading to substantially better performance. We discuss the characteristics

of more wireless links including mmWave and radios with multiple interfaces as well as how to build future camera analytics systems with these links in Chapter 5.

1.2 Contributions

The main contributions of this thesis are summarized below:
Resilient Image Compression for AIoT Cameras

- We propose STARFISH, a lossy image compression framework designed for LPWAN that processes all the information loss in the application layer, thus greatly simplify the system design. STARFISH reuses the DNN inference pipeline as in object detection/image classification tasks thus it could be easily adopted on various DNN accelerators.
- To our best knowledge, STARFISH is the first DNN-based compression framework that runs efficiently on low-cost AIoT (AI for IoT) devices. We generate DNN configuration with NAS automatically, making it future-proof and generalizable to a diverse set of DNN architectures and AIoT hardware.

Collaboration Across Cameras with Multicast

- We propose YOUO, a camera analytics system that enables efficient collaboration across cameras leveraging multicast service provided in 5G.
- YOUO is 3$\times$ as efficient in bandwidth when compared with optimal single-camera filtering with minimal accuracy loss while achieving 70%+ accuracy when uploading only once per object (the absolute minimum) for the entire camera network.
- Theoretically analysis and benchmark on synthesized dataset suggest the cost of YOUO stays the same as objects captured by an increasing number of cameras, demonstrating the potential of YOUO in large-scale camera deployment.

1.3 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we provide basic concepts about neural networks and the characteristics of wireless links for camera analytic systems. In Chapter 3 and 4, we present the details of *Resilient Image Compression for AIoT Cameras* and *Collaboration Across Cameras with Multicast* respectively. In Chapter 5, we discuss future research directions along the line, as well as an outlook for wireless and AI in camera analytic systems.

Chapter 2

Background

Building AI for IoT applications often requires understanding the characteristics of both the Neural Networks and the wireless links. In this section, we review basic concepts of Neural Networks, including Deep Neural Network (DNN), Convolutional Neural Network (CNN), and DNN/CNN accelerators. We then introduce different kinds of wireless links for IoT/Cameras including LPWAN and 5G for more insights on the underlying links.

2.1 Neural Networks

DNN: a Neural Network consists of an input layer, an output layer, and one or more hidden layers between the input and the output. In the forward pass of a fully-connected Neural Network, the value of each neuron is calculated as $y = f(\sum_i w_i x_i + b)$ where x_i are values of all neurons from previous layers or the input, w_i are the corresponding weights between them, and b is the bias value. Here f is a non-linear function that allows the neural network to compute non-linear, complex problems, typically referred as activation function. Some sample activation functions include sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, tanh $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and ReLu ($ReLu(x) = 0\ if\ x <= 0\ else\ ReLu(x) = x$). A DNN is a Neural Network with more than one hidden layers, leading to more weights and larger capacity. Typical DNN ranges from 11 layers (such as AlexNet [69]) to 152 layers (such as ResNet [48]).

CNN: we can build CNN from a fully connected Neural Network by replacing some of the fully-connected layers with a convolutional layer that consists of a set of filters (typically

Figure 2.1: Landscape of different DNN accelerators aiming at datacenter tasks to low-power, embedded applications.

2D with images as input). The activation is calculated by moving each filter across the input image or previous layer and compute the dot product between the filter weights and the input. A convolutional layer is preferred as it can greatly reduce the number of weights; it is also insensitive to locations of objects when applied on 2-D images. CNN has been widely used for various image tasks such as classification [116, 112, 48], object detection [101, 103, 102] as well as video tasks [62, 111, 59].

DNN/CNN accelerators: DNN/CNN relies on heavy computation [115] to achieve the desired performance. A number of accelerators have been build to improve the energy efficiency and/or throughput of DNN/CNN tasks. We show the landscape of major accelerator providers ranging from datacenter oriented to those who focus on embedded in Figure 2.1. Different accelerators come with very different process capabilities, cost, and power consumption; one may have to carefully design the neural network architecture so that it could be accelerated within the cost/power budget.

TinyML/AIoT: recent research on TinyML/AIoT aims at bringing artificial intelligence (especially DNN) to pervasive low-power embedded devices [126, 24]. However, squeezing computationally expensive DNN into low-power embedded devices requires innovations in both hardware and software. Aside from building special-purpose accelerators as mentioned in the previous paragraph, there are research directions for designing more

efficient software:

- Design special TinyML DNN architectures. MobileNet [51] is a notable example that replaces 3*3 convolution operators with more efficient 3*3 depth-wise and 1*1 convolution operators.
- Model quantization. Reducing the precision for DNN parameters and activation could significantly reduce the memory and computation requirement [47]. Instead of using 32/64-bit floating-point operation, one can reduce the precision to 8/4 bit, or even 1-bit [99] with acceptable accuracy.
- Network pruning and knowledge distillation. Removing weights that are close to zero and neurons that always output near-zero values could greatly simplify DNN with minimum performance degradation [73]. Large DNN models are easier to train but difficult to deploy. Knowledge distillation learns a small student model from a large teacher model, thus making training TinyML models easier [44].

2.2 Wireless Links

In this section, we introduce two representative wireless links, including LPWAN and 5G, for a better understanding of the underlying links used in low-power IoT and camera analytic systems.

LPWAN: as implied in the name, power consumption and range are the most important design considerations of LPWAN. LPWAN is suitable for battery-powered IoT devices deployed in the wild or areas that have poor wireless reception. As a result, LPWAN has to operate at a very low bitrate to ensure enough link budget to cover the path loss. Hardware cost is another consideration so LPWAN-enabled IoT devices could be deployed at scale. LPWAN typically operates in the unlicensed ISM band (Industrial, Scientific, and Medical radio band) that allows flexible deployment without paying for the cost of spectrum usage.

There are two major LPWAN techniques: LoRaWAN (Long Range Wide Area Network) and Sigfox:

- LoRaWAN is based on a special modulation scheme similar to the chirp spread spectrum. It could achieve a range of more than 20km under ideal scenarios between

LoRaWAN nodes and the gateway [88]. The physical layer supports several data rates from 0.3 to 27 kbps by changing the spreading factor. It is possible to build a private LoRaWAN if the user operates the gateway.

- Sigfox is based on the differential binary phase-shift keying (DBPSK) and the Gaussian frequency-shift keying (GFSK) modulation on the physical layer. Different from LoRaWAN, Sigfox is operated by the SigFox company that provides service in 72 countries covering a total of 5.8 million square kilometers[1]. The communication range is up to 40km but the maximum data rate is only 100 bps.

Both LPWAN techniques keep the radio off most of the time and run on battery for years when transmitting a small amount of data per day.

5G Multicast: wireless is a broadcast medium means it could support point-to-point communication (unicast) as well as point-to-multipoint communication (broadcast and multicast). Multicast could make better use of wireless network capacity when delivering the same content to multiple receivers. More than 40 cellular operators have deployed or will deploy multicast service in 5G, which is called 5G eMBMS(Evolved Multimedia Broadcast Multicast Services). It could be used in a single cell, or across multiple cells while using a common carrier frequency. A number of services have been proposed for 5G eMBMS including mobile video streaming, deliver software upgrades, and traffic information [19].

[1] https://www.sigfox.com/en/sigfox-story

Chapter 3

Resilient Image Compression for AIoT Cameras

Cameras are key enablers for a wide range of IoT use cases including smart cities, intelligent transportation, AI-enabled farms, and more. These IoT applications require cloud software (including models) to act on the images. However, traditional task oblivious compression techniques are a poor fit for delivering images over low-power IoT networks that are lossy and limited in capacity.

The key challenge is their brittleness against packet loss; they are highly sensitive to small amounts of packet loss requiring retransmission for transport, which further reduces the available capacity of the network. We propose Starfish, a design that achieves better compression ratios and is graceful with packet loss. In addition to that, Starfish features content-awareness and task-awareness, meaning that we can build specialized codecs for each application scenario and optimize for different task objectives, including objective/perceptual quality as well as AI tasks directly. We carefully design the DNN architecture and use an AutoML method to search for TinyML models that work on extremely low power/cost AIoT accelerators. Starfish is not only the first image compress framework that works on AIoT accelerators that costs for a few dollars, but also provide competitive results on image compression in the lossless scenario. It also features graceful and gradual performance degradation in the presence of packet loss.

3.1 Introduction

Recent advances in computer vision technologies have been a key enabler for the pervasive growth of vision-based IoT applications ranging from smart cities, intelligent transportation, AI-enabled factory to farms [93]. Cities and construction sites use cameras for intelligent transportation and monitoring to increase safety and security [82, 17, 3]. Oil refineries and farms use cameras to perform predictive maintenance and irrigation [91, 120]. The camera, being one of the most information-rich and versatile sensors maximizes its potential when coupled with powerful machine perception algorithms.

Although some computer vision applications can be performed onsite without cloud support, many applications need to leverage the vast compute and storage available on the cloud. These applications typically require heavy computation, perform computation on data from multiple camera streams, or store the raw data for archival purposes and future applications.

Transferring images efficiently to the cloud is essential for many battery-powered IoT cameras. Current solutions rely on LPWAN such as LoRaWAN and 5G IoT for communication that tends to have very limited bandwidth and high packet loss rate due to low transmit power, high path loss, and collision. Increasing the reliability of the network is possible. However, this comes at the cost of more complexity and retransmission of data, significantly reducing the total network capacity.

The highly limited network capacity in LPWAN call for extreme image compression. However, traditional image compression such as JPEG is a poor fit for most IoT applications; specifically, we make the following observations about the design objectives of JPEG: 1) designed to be used on reliable networks; 2) general-purpose compression applicable to a variety of images that is content and task oblivious. While JPEG is an excellent general-purpose image compression algorithm, it's not necessarily the best fit for IoT applications where the task objective and context are typically clear. Further, since many IoT devices use low-power networks with unreliable transport making JPEGs brittleness to any packet loss a substantial limitation.

If our goal is to send a lossy-compressed image over LPWAN, why do we need to

assume lossless transmission of compressed data? We present STARFISH, an application layer solution that works with intrinsically lossy wireless links. STARFISH avoids inefficient retransmission by addressing information loss and data loss altogether in the application space utilizing information about the context and task objectives. Inspired by recent advances in DNN hardware and software, STARFISH uses a tiny DNN to generate a loss-resilient, unstructured compressed representation that degrades gracefully in the presence of data loss. Unlike structured representation in JPEG, the information is distributed uniformly in DNN representation: reconstructing an image takes inputs from all bytes in representation rather than relying on specific bytes. As a result, image quality degrades gracefully as data loss occurs rather than completely damage part of the image.

The ability to tolerate packet loss in application space not only simplifies MAC(Medium Access Control) layer protocol but also brings significant benefits in energy consumption and network capacity. LPWAN nodes can send at higher bitrates without the need to wait during the back-off period before retransmission, or wait for an extended time if block-ack is used. Such ability is extremely powerful in LPWAN, where thousands to millions of nodes are connected to each base station or gateway, and nodes need to conserve energy as much as possible to extend battery life.

Nevertheless, our DNN-based design makes STARFISH aware of the content and objectives in sending compressed images over LPWAN. STARFISH generates application-specific codecs that are tailored for images from specific IoT application scenarios e.g. traffic cameras or property surveillance by training on similar image datasets, which differs from JPEG that is designed to compress versatile images. STARFISH also optimizes for a wide range of task objectives directly, including PSNR (Peak Signal to Noise Ratio), MS-SSIM(Multi-Scale Structure Similarity), perceptual quality, machine perception task performance, or a joint optimization objective.

One of the challenges we tackle is the design of a DNN that runs efficiently on low-cost, low-power IoT devices. AIoT (AI for IoT) accelerators are very different from desktop GPUs with various memory/storage/computation/energy constraints. We use AutoML techniques to generate tiny DNN that runs efficiently on AIoT devices based on a hardware-aware NAS(Network Architecture Search) accelerated by learning curve extrapolation. It improves NAS efficiency by rejecting DNN configurations that exceed the capability of

AIoT accelerators as well as those who underperform according to the learning performance predictor. We start with sampling a small fraction of the design space and train them until they converge. The training curves are used to train the learning performance predictor based on LSTM that predicts performance from the first a few epochs of the training curve. We then train each configuration in the design space for a few epochs and continue to train only if the predicted performance falls in the top percentile. We showcase our design and test a tiny DNN that runs on an AIoT accelerator that costs only a few dollars, as well as the more powerful Google Edge TPU.

Benchmark results on four large public datasets indicate that we can achieve the same task objective while using only a small fraction of the energy/bandwidth, thus significantly reducing the cost of operation and accommodating more AIoT cameras, and increase task performance. We summarize our contributions as follows:

- We propose STARFISH, a lossy image compression framework designed for LPWAN that processes all the information loss in the application layer, thus greatly simplify the system design, especially for a transmit-only link with an unknown loss rate. STARFISH reuses the DNN inference pipeline as in object detection/image classification tasks thus it could be easily adopted on various DNN accelerators.

- To our best knowledge, STARFISH is the first DNN-based compression framework that runs efficiently on low-cost AIoT (AI for IoT) devices. We generate DNN configuration with NAS automatically, making it future-proof and generalizable to a diverse set of DNN architectures and AIoT hardware.

Comparing against a more optimized codec: STARFISH is probably not a good choice for situations that an optimized codec is available, especially if the underlying link is lossless. A better mechanism may exist if the rateless-coding-based method could be conveniently implemented and tuned for lossy links. It is more for AIoT devices with hardware/software that heavily biased towards DNN accelerators and prefers the simplicity of STARFISH over rateless-coding based methods in transmit-only scenarios.

For example, we provide a different analysis with a more optimized codec than the previously published STARFISH paper [52] in section 3.9. The JPEG encoder and decoder

implementation in [52] are base on Pillow(Python Image Library [11] with IJG JPEG interface). We change the encoder configuration from `optimize=0 (default), subsampling=0` to `optimize=1, subsampling=2, smooth=100`. The updated configuration means the encoder should make an extra pass over the image to select optimal encoder setting; using 4:2:0 color sub-sampling rather than 4:2:2; set smoothing factor to 100 for better visual result[1]. We use Jpeg2png [6], a smart JPEG decoder that produces a smoother decoder image than Pillow decoder with more computation[2]. As a result, the performance of JPEG is significantly improved. However, not all AIoT devices support the updated encoder configuration, especially for those based on MicroPython. The decoder also costs more computation resources on the receiver side. It is recommended to investigate what are the constraints (such as computation resource, supported configurations) for applying both traditional image compression and DNN-based computation before putting STARFISH into use. Your mileage may vary.

3.2 Background and Motivation

We describe the background work motivating the design of our streaming framework in this section, starting with an analysis of network/compression and AIoT hardware, then show the challenges of using conventional image compression and our approach to deal with these limitations.

Characteristics of LPWAN: despite providing a connection to a massive number of nodes, each LPWAN cell has a limited total capacity. Further, LPWAN nodes have to tolerate high link loss due to long distance or obstacles. They tend to lack the support of an advanced modulation scheme due to power/cost limitation and the entire network operating inside a narrow spectrum. The estimated total uplink network capacity for a LoRaWAN gateway is less than 100kbps [43], which has to be shared by potentially thousands of nodes connected to that gateway. Similarly, 5G mMTC targets at supporting 300,000 IoT nodes per cellular station using only 10MHz spectrum [26]. This scarcity in network capacity

[1]Note that setting smoothing factor is not clearly stated in the Pillow documentation but it is actually implemented in the IJG JPEG binding interface.

[2]Takes an average of 1.57 seconds per 224*224 color image with default configuration, measured on an 8-core Google Cloud Machine. Not including the time for decoding the output PNG file

Table 3.1: Benefits of tolerating packet loss in LoRaWAN[1]

Scenario	Bitrate (bps)		Payload / Packet Size (Bytes)	
	Baseline	Lossy	Baseline	Lossy
Good	5,469	10,938[2]	242 / 255	242 / 255
Average	1,758	3,125	115 / 128	242 / 255
Bad	245	448	51 / 64	51 / 64

Scenario	Air-time (seconds) / Image[3]		TX Energy (uAh) / Image	
	Baseline	Lossy	Baseline	Lossy
Good	4.40	2.20	55.61	27.76
Average	14.89	7.78	188.25	97.16
Bad	160.97	90.32	1996.10	1127.20

[1] Assuming European 868 MHz ISM band, spreading factor=7-12, CRC=1, N_preamble=8, coding rate=2/3 or 4/5, with implicit header.

[2] Bandwidth=250kHz.

[3] Assuming an image size of 2.5kB.

calls for extreme compression to send image/video data over LPWAN.

Packet loss makes the problem even worse. We use LoRaWAN as an example. Packet loss is prevalent in LoRaWAN due to path loss and collision from other transmitters according to previous indoor and outdoor measurements [97, 114, 119]. Also, the study [85] shows that frequent retransmission is unrealistic due to the limited capacity of acknowledgments at the gateway, increased energy consumption for LoRa nodes as well as an increased probability of collision. As a result, previous applications prefer lower bitrate that has a lower packet loss rate. However, we observed that if the application could tolerate a packet loss rate of 20%, we can use a higher transmit bitrate [63], thus drastically increase the energy efficiency of LoRaWAN nodes and total link capacity, as shown in Table 3.1. The baseline method is the conventional mode that trying to have a small packet loss rate by choosing slower bitrates while lossy mode shows possible bitrates if a packet loss rate of 20% is acceptable. The result shows that we can reduce transmission air-time and energy consumption by half, thus double the total link capacity.

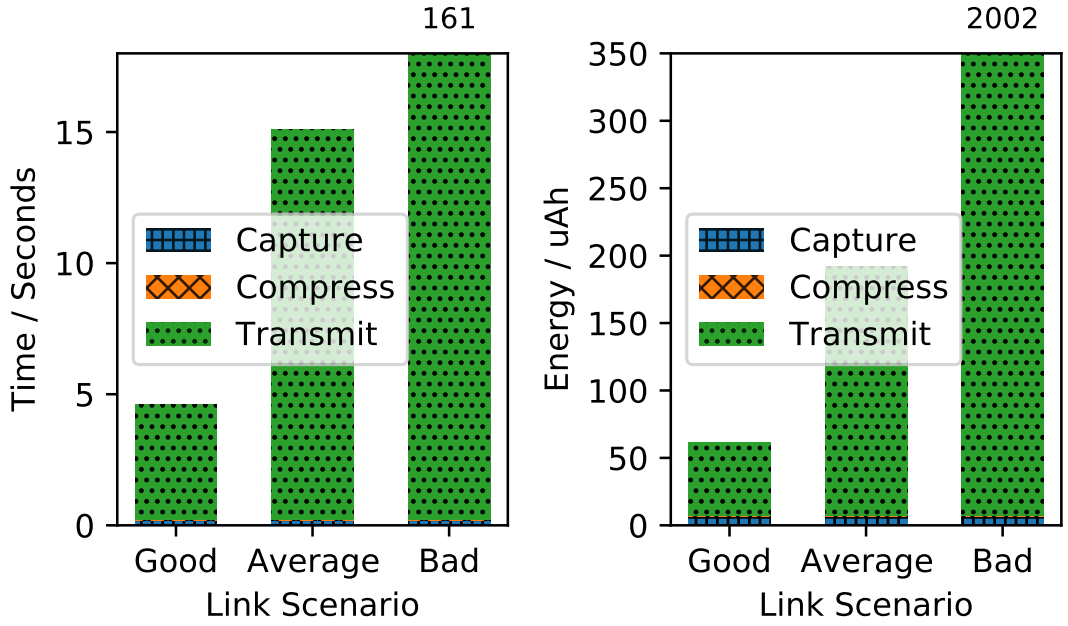

Figure 3.1: Time and energy consumption breakdown for image capture, compression, and transmission over LoRaWAN. The time and energy spent on compression are barely visible because JPEG compression takes only 16ms on average. Transmission over LoRaWAN dominates time/energy consumption in all scenarios and exceeds the range for both charts when the link condition is poor.

Energy/time consumption break-down: how much time/energy do we spend on communication in an LPWAN IoT camera system? To answer that question, we analyze the energy and time consumed on the image capturing, compression, and transmit data over LPWAN. A break-down of this analysis is in Figure 3.1. It is clear that communication dominates the time/energy cost; there is a severe imbalance as only a tiny amount of time/energy is allocated to compute a compressed representation of images. Can the overall system performance be improved by spending more on compute to improve the compression ratio, therefore, decreasing the overall network cost? To answer this question, we analyze current image compression algorithms on IoT devices and explore a novel design that works better for an LPWAN based IoT device.

Image compression on IoT devices: in this paper, we focus on lossy image compression as lossless compression algorithms typically generate images that are too large

Marker	Segment
0xFFD8	SOI(Start Of Image)
0xFFC0	SOF0(Start Of Frame 0)
0xFFDB	DQT(Define Quantization Table)
0xFFC4	DHT(Define Huffman Table)
0xFFDA	SOS(Start Of Scan)
0xFFD9	EOI(End Of Image)

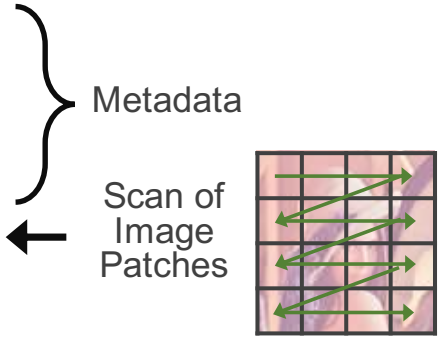

Figure 3.2: File structure of JPEG (JFIF standard with baseline profile, only showing essential segments for decoding).

for LPWAN. Although many lossy image codecs have been proposed like JPEG [122], JPEG2000 [98], WebP [25] and BPG [2], IoT devices tend to support JPEG only due to its simplicity and low resource requirement. Similarly, JPEG-compatible encoders like Guetzli [10] offer better performance but require significantly more RAM (300MB) and CPU time (1 min on desktop) so they're not feasible on IoT devices. In this paper, we use standard JPEG as our baseline. We also limit our scope to IoT cameras that send video as independent frames due to the high cost of inter-frame prediction in H.264 [127], as well as the fact that they send at a very low frame rate so the gain from inter-frame prediction may be limited.

JPEG compression consists of three steps: perform DCT (Discrete Cosine Transform) on patches of the input image, quantize DCT coefficients, then encode the quantized DCT coefficients with an entropy encoder (typically Huffman). Image data is stored in a structured manner with markers to indicate each segment, as shown in Figure 3.2. It starts with metadata that describes the type of algorithm and sampling scheme in SOI and quantization/Huffman compression table in DQT/DHT. The actual compressed data of each patch is stored sequentially after SOS.

Though JPEG works fine for a wide range of application scenarios, it does have several deficiencies in LPWAN IoT camera setting:

Table 3.2: Comparison of different AI computation platforms

	Desktop GPU (Nvidia K80)	**AIoT Accelerator (K210)**	**Arduino Uno (ATMEGA 328p)**
Cost	$900 (GPU only)	$8 ($3 chip only)	$20 ($2 for MCU only)
Power	300W, AC powered	300mW, battery-powered	225mW, battery-powered
Memory	24GB	2MB	2kB
Speed	13.45TFLOPS FP32	230GOPS INT8	8MOPS INT8
Software	TensorFlow[14]/ PyTorch[95, 96]	TensorFlow Lite[14]	TensorFlow Lite Micro[14]

- **Resiliency**: the structured storage format of JPEG requires reliable data transfer otherwise it fails abruptly: any loss in the metadata segment will block decoding of the entire image; a specially designed decoder may handle the loss of image patches, but it will still lose the relevant patches completely.

- **Task agnostic**: JPEG encoder on many AIoT devices (such as K210, GAPuino) cannot directly optimize for task objectives like SSIM (structural similarity index) or image classification / object detection accuracy directly, especially more and more IoT images/videos will be consumed by machine perceptions models rather than humans.

- **Content agnostic**: JPEG encoder is designed to be agnostic of image content that accepts all possible images. However, we observe that many IoT cameras take a very similar image as input. As an example, traffic cameras will always see roads, cars, and pedestrians. An encoder specialized to compress these objects can do better as it does not need to care about other inputs.

As an alternative, we design STARFISH, a DNN-based compressed streaming framework that specially optimized for LPWAN IoT camera applications. It features:

- **Loss-resilient**: interference and collision in the wireless channel can cause packet loss and partial reception. Unless expensive retransmit operations are used, these

losses will lead to a failure to decode that part of the image if JPEG encoding is used. In contrast, our unstructured DNN can provide graceful quality degradation in such cases. The loss resiliency could also improve energy efficiency and network capacity by avoiding retransmission.

- **Directly optimize for task objective**: our DNN compression framework is end-to-end trainable, making it possible to directly optimize any differentiable objectives like PSNR (Peak Signal to Noise Ratio), MS-SSIM (Multi-Scale Structural Similarity index), or machine perception model accuracy.

- **Specialize to each AIoT camera applications**: we can generate a specialized DNN model by training on a similar image dataset for better results [53]. It is also possible to fine-tune the compression model as we collect more images from IoT cameras in the same application scenario.

However, running DNN on IoT devices is not as easy as on GPUs due to stringent limitations on computational power and energy budget. Although traditional micro-controllers like STM32 and MSP430 support tiny neural networks with low dimension input for keyword spotting [137] and gesture recognition with accelerometer [13], they're ill-suited for running CNN (Convolutional Neural Network) with images as an input. These general-purpose microcontrollers have instruction sets optimized for control logic focusing on branch prediction and data prediction. However, the majority of CNN inference tasks are *deterministic* and compute-heavy, utilizing *massive* matrix multiplications. Recently, a number of AI accelerators for IoT devices have enabled low-power and low-cost acceleration of CNN inference as shown in Figure 3.3.

We compare the specs of different AI platforms to better understand the performance of AI accelerators for IoT as shown in Table 3.2. The cost and power consumption of an IoT AI accelerator is comparable to an Arduino while providing several orders of magnitude more memory and compute power. Their performance, however, still falls far behind the traditional desktop-grade GPUs. The limited compute and memory available on these accelerators make designing a DNN that performs image compression a challenging task. We further discuss the limitations of the AIoT DNN accelerator and how it affects DNN design in Section §3.3.1.

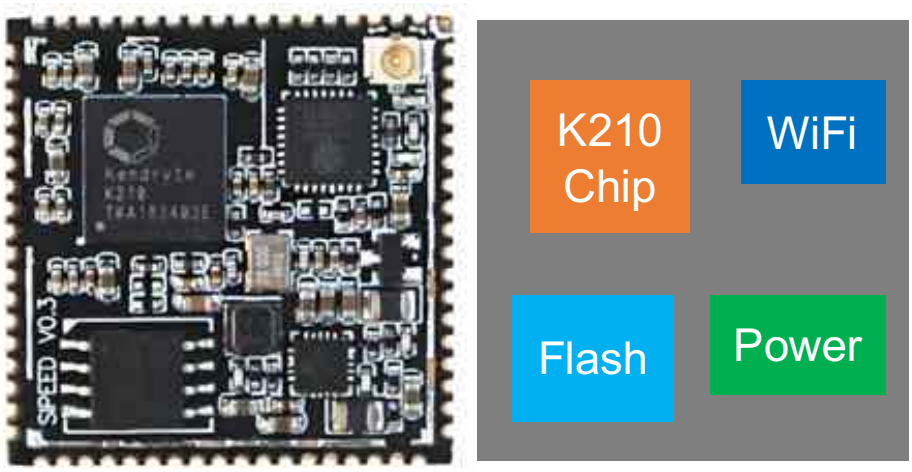

Figure 3.3: An $8 K210 Kendryte AI accelerator module. The corresponding system layout is shown on the right. It includes K210 chip (with RISC-V CPU and AI inference accelerator), an ESP8266 WiFi module, 16MB Flash storage and power management chip.

3.3 System Design

We present the design of our AIoT streaming framework in this section. As an overview, we first present the basics of image compression using a DNN, followed by our design of a DNN based image compressor, including hardware-aware network architecture search, design of application-specific codec, and how we quantize model and intermediate representation. Lastly, we present our design to improve the resiliency of the compression DNN.

3.3.1 Design of Compression DNN

To perform DNN-based image compression, we typically use a DNN encoder $\mathbf{E}$ to generate an intermediate representation $\mathbb{R}^N$ with length of N on input image $\mathbb{I}^{(H,W,C)}$ with H, W, C as the height/width/channel respectively. The intermediate representation is then quantized with $\mathbf{Q}$ that maps floating point values into low-precision integers $\mathbb{Z}^N$. They are further compressed with an entropy encoder $\mathbf{C}$ before sending over the network. On the receiver side, we decode the compressed data using corresponding entropy decompression and run through a DNN decoder $\mathbf{D}$ to output recovered image $\mathbb{I}'^{(H,W,C)}$. The entire process is shown

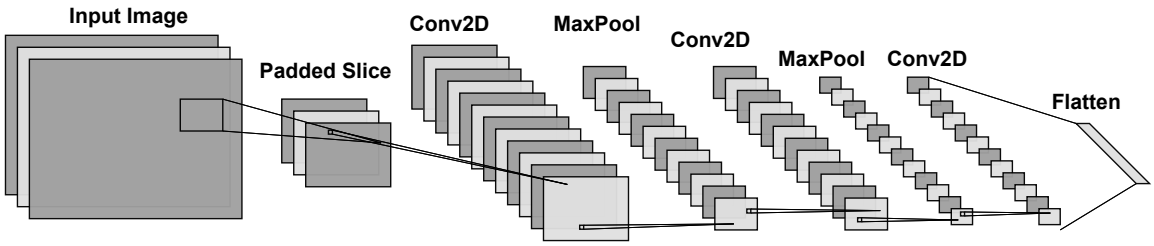

Figure 3.4: Architecture of encoder DNN. Images are optionally sliced into patches, then zero-padded as quality degrades on edges of decoded images. Padded patches are processed multiple stages of 2D convolution (stride=1 with 3*3 kernel) followed by 2*2 max-pooling layer to reduce spatial resolution, then flattened to generate binary representation.

in Equation 3.1:

$$\underbrace{\mathbb{I}^{(H,W,C)} \underset{\mathbf{E}}{\rightarrow} \mathbb{R}^N \underset{\mathbf{Q}}{\rightarrow} \mathbb{Z}^N \underset{\mathbf{C}}{\rightarrow} \mathbb{B}^L}_{\text{Sender/Encoder}} \underbrace{\underset{\mathbf{C}^{-1}}{\rightarrow} \mathbb{Z}^N \underset{\mathbf{D}}{\rightarrow} \mathbb{I}'^{(H,W,C)}}_{\text{Receiver/Decoder}} \tag{3.1}$$

Design and implement DNN on AIoT accelerators is not as straight-forward as on general purpose GPU. As we've shown in Table 3.2, low-cost AIoT accelerators pose strict limitations on the DNN architecture used, including RAM size, computation power, and supported operators. Only very basic operators include 2D convolution, max pooling, and softmax are supported due to hardware limitation[3]. Limitations exist even within the supported operator. For example, the convolution filter size has to be 1*1 or 3*3, the stride has to be 1 or 2 with no more than 1024 filter channels. These limitations prevent us from using existing DNN designs directly. For example, RNN (Recurrent Neural Network) is a very popular method to extract spatial correlation over images but the underlying operators are not supported yet.

We carefully designed our DNN to use only the supported operators. The architecture of encoder DNN is shown in Figure 3.4. We omit the architecture of decoder DNN as it basically mirrors the encoder. The decoder DNN takes the quantized binary stream as input, uses nearest-neighbor scaling for up-sampling, and has the same 2D convolutional layers. We use ReLU [92] activation for most layers in encoder and decoder DNN, except: 1)

[3]Notice that these limitations are not unique to our platform. Other low-cost AIoT accelerators and even more powerful Edge AI accelerators like Google Edge TPU have similar limitations. We expect that our design choices can be applied to a wide variety of AIoT platforms.

`Sigmoid` activation at the output of decoder DNN to generate images with values between [0, 1]; 2) no activation/nonlinearities for the layer before Flatten layer (to generate binary representation) and after Flatten layer so the binary representation uses the full data range (rather than throw away the negative half in ReLU). We avoid using batch normalization during training compression DNN is more sensitive to internal noise, thus normalization leading to poor performance [23].

Compress image in full vs. patches: low-cost AIoT accelerators have very limited memory that can be used to hold activations and weights. Though it is possible to feed full-resolution image in to the AIoT accelerator so the compression can be done in one pass, the performance is poor due to limited number of filter channels. There is a trade-off between the patch size used and the number of filter channels available. The size of activations and trainable weights of a vanilla convolutional layer is shown in Equation 3.2:

$$\begin{aligned} N_{activation} &= h \times w \times C_{out} \\ N_{weights} &= k \times k \times C_{in} \times C_{out} + C_{out} \end{aligned} \tag{3.2}$$

Where h, w are height, width of output activation, k is size of filter kernel and C_{in}, C_{out} are number of input/output filter channels respectively. With C_{in}, C_{out} set to 36 we get very close (224*224*36=1.81 MBytes) to the 2MB memory limit but having a small number (11.7k) of filter weights. The limited weights prohibit encoder-decoder from working effectively. Instead, if we divide input into four patches, we can quadruple filter channels and trainable weights are 16$\times$ as original configuration.

Network Architecture Search

The design space for image compression DNN is large, ours consists of $3 * 64^3 = 786432$ possible configurations as shown in Table 3.3. It's infeasible to perform a brute-force search for the best possible parameter set: training a single compression DNN takes about \$30 and 10 hours[4], so the cost of searching through all possible combinations of

[4]Based on a VM with one Nvidia V100 GPU and 8 vCPU in Google Cloud: https://cloud.google.com/compute/all-pricing

Table 3.3: Design Space of Image Compression DNN

Design Dimension	**Range**	**Choices**
Number of Image Patches	1,4,16	3
Number of Conv. Layers	2,3,4	3
Number of Channels of each Layer	[4, 256]	64

design space is prohibitively expensive. Alternatively, we adopt a principled search method that is not only generalizable to a diverse set of AIoT hardware, but can also evolve as new hardware becomes available.

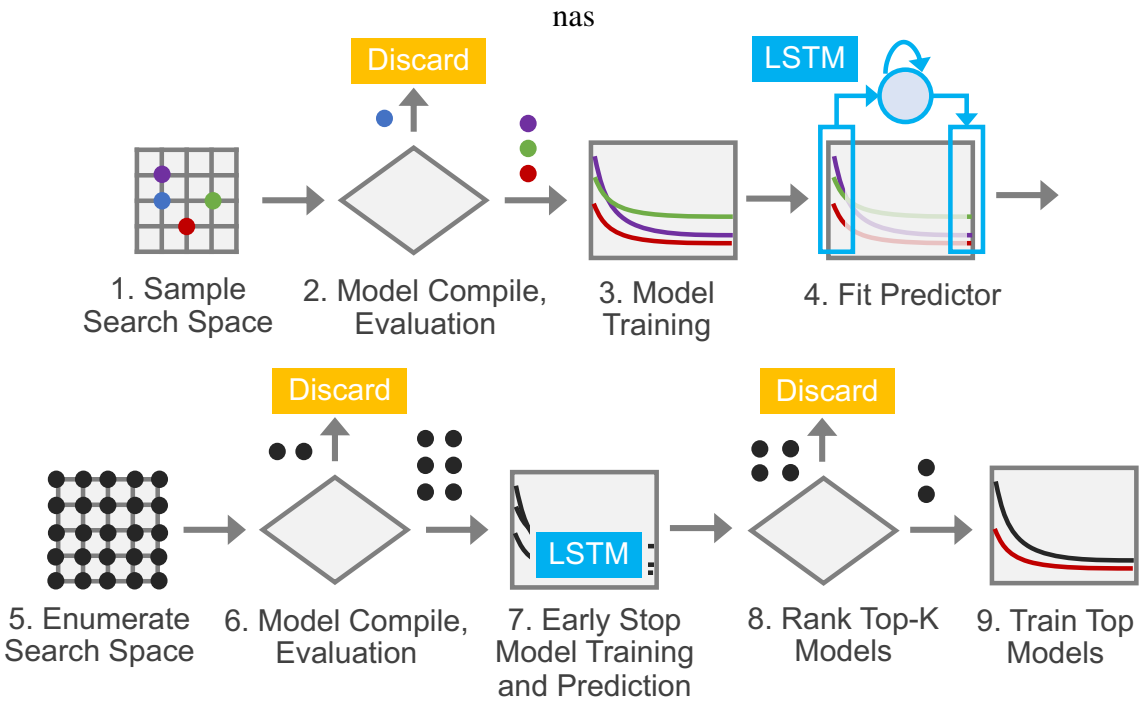

Figure 3.5: Workflow of hardware-aware NAS with learning curve extrapolation.

While previous methods focus on the trade-off between accuracy and inference time, our DNN has a strict limitation on RAM and model size. We build on top of previous NAS with learning curve extrapolation by adding a model compilation and evaluation stage to filter out configurations that are infeasible to run on our target devices. The workflow of our hardware-award NAS is shown in Figure 3.5, and consists of the following steps:

1. Get a few random sample configurations from the design space according to the

computation budget.

2. Build model according to sample configurations, export (untrained) model and weights, then compile the model/weights using the AIoT compiler. Discard configurations that require more buffer in RAM or Flash storage than AIoT hardware can offer, or those severely underutilize hardware.

3. Train the model on a subset of the dataset for a fixed number of epochs or until it converges. Save training history that contains validation loss.

4. We fit a simple LSTM-based learning curve predictor using the training data we've collected. The predictor includes only one LSTM unit, followed by a dense layer, so the total number of trainable parameters is only 14 (12 for LSTM, 2 for the dense layer). We deliberately keep it simple and lightweight to avoid over-fitting. Input to the predictor is the validation loss values of the first a few training epochs, as shown in Figure 3.6. A more detailed mini-benchmark is shown in Table 3.4. Experiment results show that our predictor could achieve much better results in selecting top configurations than random selection, while only taking first 10 or 20 epochs as input. In this paper we take first 10 epochs as input as it could outperform random by more than $2\times$ in selecting top 5% configurations (16.4% v.s 5%). Taking more epochs (e.g. 20) as input could further increase accuracy at the cost of additional time for profiling.

5. After fitting the predictor we enumerate the entire search space (but reject configuration trained in previous steps).

6. Similar to step 2, we reject configurations that over/under-utilize the AIoT hardware.

7. We train the remaining model on a subset of the dataset for 10 epochs and predict the final loss value using the trained predictor.

8. We then rank top k model configurations according to the computation budget and discard the other configurations.

9. Train the models with these configurations on the entire dataset until converge and save the model with the best validation loss.

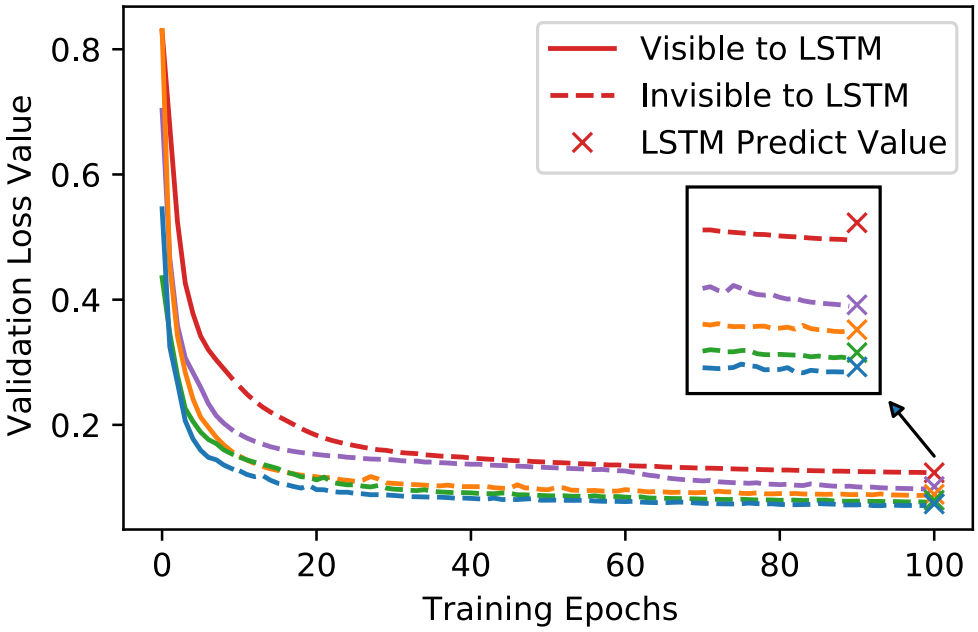

Figure 3.6: Predict final loss value with LSTM-based learning curve predictor. Different colors indicate different DNN configuration. The predictor takes first loss value of first 10 epochs as input. Loss values after that shown as dashed line are invisible to the predictor. A zoom-in of the LSTM predicted values shows we can predict final loss value accurately by only look at first 10 epochs.

Application-specific Codec

Using DNNs allows us to encode content-specific information as represented by the training dataset (i.e., content-awareness) as well as optimizing for application goals directly (i.e., task-awareness). Combining content-awareness and task-awareness enables us to build application-specific codecs. Since training DNN to be content-aware could be done

Table 3.4: Benchmark of the Learning Curve Predictor

LSTM Input	RMSE Mean/STD	Accuracy		
		Top 5%	Top 10%	Top 20%
First 5 epoch	0.062/0.0076	4.8%	9.4%	20.3%
First 10 epoch	0.053/0.0077	16.4%	22.4%	28.2%
First 20 epoch	0.051/0.0084	23.2%	28.6%	32.0%

rather straightforwardly using standard train/test split, we focus on how to train DNN to be task-aware by designing appropriate loss functions.

Loss function: we train the encoder and decoder together by minimizing $\mathcal{L}(\mathbb{I}^{(H,W,C)}, \mathbb{I}'^{(H,W,C)})$ where $\mathcal{L}(\cdot)$ is loss function. We utilize three major kinds of loss functions where traditional/perceptual quality are optimized for human eyes and AI task objective optimized for task accuracy:

- Traditional quality estimator: traditional estimators include L_1 distance, L_2 distance / PSNR, SSIM [124]/MS-SSIM [125]. We choose MS-SSIM as representative quality estimator as it performs the best [125] among them.

- Perceptual quality estimator: recent studies suggest perceptual loss [61] that calculate metrics using DNN features to be "unreasonably effective" [136] in approximating human perception. Perceptual loss is defined as $||\mathbf{DNN}(\mathbb{I}^{(H,W,C)}), \mathbf{DNN}(\mathbb{I}'^{(H,W,C)})||_2$ where $\mathbf{DNN}(\cdot)$ means feature map output of perceptual DNN with corresponding image as input. Common choices for perceptual DNN include Alexnet [69] and VGG [112]. In this paper we use the output of last layer before Softmax of VGG16 pretrained on Imagenet [37] to calculate perceptual loss, as it is more powerful and up to date than Alexnet.

- AI task objective: image classification, object detection and segmentation are most typical AI tasks with image as input data. In this paper we focus on image classification, the most fundamental one as detection/segmentation could build on top of

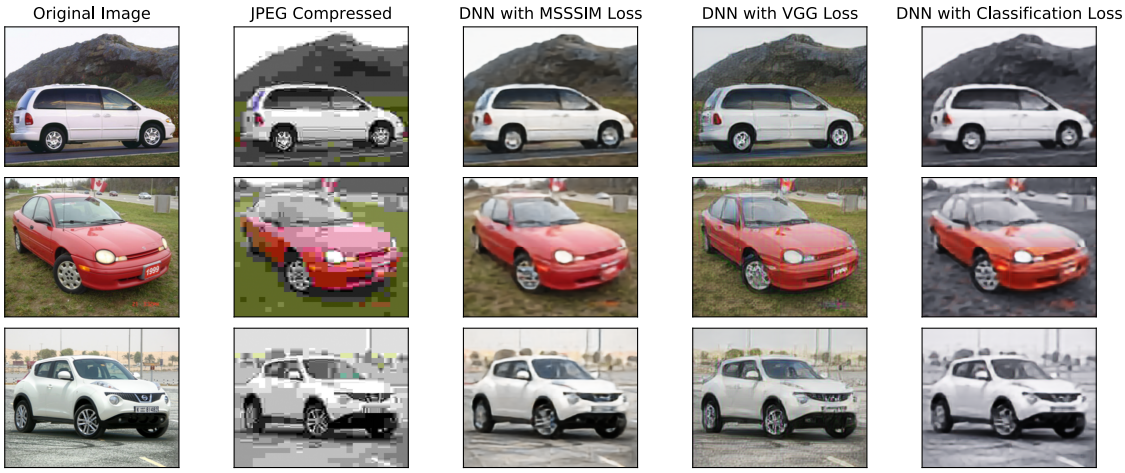

Figure 3.7: Visual quality comparison with different loss functions. All compressed images have a similar size of about 2.5kB. JPEG has a lot of block artifacts while DNN optimized for objective MS-SSIM loss tends to produce a more smoothed image; DNN optimized for perceptual quality with VGG Loss preserves details well but adding color noise; the image quality is good even if we optimize for classification tasks, though the color of images looks strange suggesting that the classification DNN might be less sensitive to color changes. We present numerical results in Section 3.5.1.

classification. We use Categorical Cross-Entropy loss for image classification tasks:

$$\mathcal{L}_{cross\_entropy} = -\sum_{i=1}^{C} d_i log(d'_i) \tag{3.3}$$

Where d_i and d'_i are classification output with $\mathbb{I}^{(H,W,C)}$ and $\mathbb{I}'^{(H,W,C)}$) as input respectively.

In real-world applications, we can use a weighted version of different loss functions if the image will be used for different purposes.

We demonstrate visual quality with different loss functions in Figure 3.7. Different loss functions lead to different visual styles but all of them are much better than JPEG of similar size. We present numerical comparisons across different datasets in Section 3.5.

Model/IR Quantization

Quantization of models allows more efficient DNN inference as 32-bit floating-point multiplication consumes 18× more energy and takes 27× more silicon space to implement [50] than 8-bit integer operation. Both K210 and Edge TPU requires INT8 quantization.

All models are trained with FP32 then quantized to INT8 with TFLite so the model could be accelerated by AIoT accelerator. Notice that this is *different* from quantizing intermediate representation $\mathbf{Q}(\cdot)$: while model quantization quantizes DNN weights into 8 bits to accelerate computation, intermediate representation quantize into finer bins that do not have to be an exponent of 2 (e.g. 72 bins) to save LPWAN bandwidth. Similar to JPEG, we use Huffman [57, 5] coding to further compress the data. Another version without Huffman coding, which is more suitable for lossy networks is described in section 3.3.2.

Notice that intermediate quantization function $\mathbf{Q} : \mathbb{R}^{\mathbb{N}} \rightarrow \mathbb{Z}^{\mathbb{N}}$ is not differentiable so we add uniform noise as described in [22, 56] during training to simulate the effect of quantization. We can vary the scale of noise according to the quantization step and range of activations.

In Figure 3.8 we illustrate the file size and image quality for model/intermediate representation quantization. The result indicates that model quantization with TFLite leads to minimal performance degradation when converting the floating-point model into integer, while compressed intermediate quantization allows for flexibility on size vs. quality trade-off. In STARFISH we select quantization bits according to target compressed size. Visualization of image quality with model/weights quantization is shown in Figure 3.9. There is no strong visible distortion (as those in JPEG compressed images) even after aggressive quantization.

3.3.2 Loss-tolerant DNN compression

In the previous section, we demonstrated that DNN in STARFISH provides graceful degradation as the numerical accuracy of compressed intermediate representation decreases. We

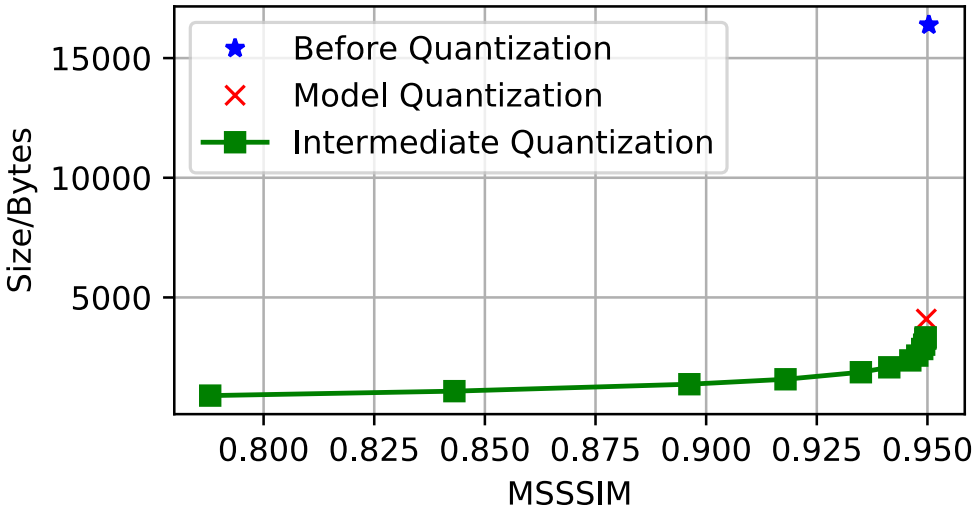

Figure 3.8: Average image MS-SSIM vs. file size for model/intermediate quantization. There is minor SSIM loss when quantize model with TFLite but results in a 75% smaller file size. Quantization of intermediate representation allows flexible size vs. MS-SSIM trade-off. It is possible to significantly decrease the file size without significant loss in accuracy or compression with intermediate quantization.

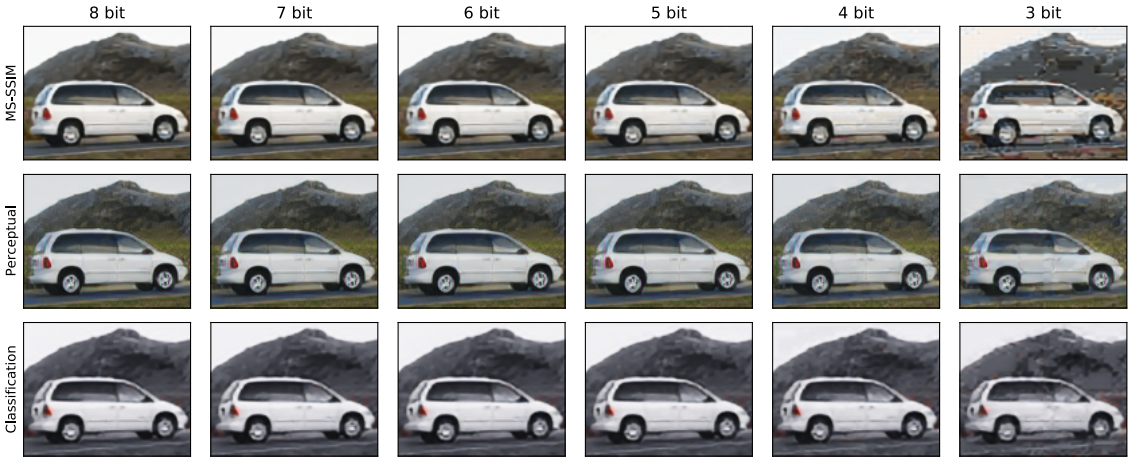

Figure 3.9: Visual quality comparison showing the effect of intermediate representation quantization with different precision. The first row are result of DNN trained with MS-SSIM loss, and second/third row are trained with VGG and classification loss.

further enhance the ability to tolerate data loss in DNN by removing Huffman coding and train the DNN use Dropout. We simulate packet loss during DNN training in a way similar to the Dropout layer that randomly sets some of the weights to zero to avoid over-fitting: $\mathbb{B}^L \leftarrow \mathbb{B}^L \cdot \mathbb{M}^L \in \{0, 1\}$ where $\mathbb{M}$ is a randomly generated binary mask with the same length as the intermediate representation. The intermediate representation is shuffled so the data loss could be distributed over the entire image and recover with nearby data using DNN. The pseudo-random shuffling also helps distribute the error evenly if the the real-world data loss pattern is different from uniform, for example, bursty loss caused by collision in transmission or interference.

There are two methods for adapting our compression DNN to be loss-tolerant: 1) train from scratch and 2) fine-tuning based on weights saved from un-adapted DNN. Our experiment result indicates that fine-tuning could achieve the same performance while using only 30% of the GPU time for training.

We visualized how data loss affects the visual quality of reconstructed images in Figure 3.10. Visualization results indicate that adapted DNN could mask data loss while un-adapted DNN suffers from incorrectly reconstructed spots that are typically visually stand out in the image. The degradation pattern of adapted DNN due to spatial data loss is similar

Figure 3.10: Visual quality comparison of DNN before (top) and after (bottom) adaption to packet loss. Both models are trained with MS-SSIM loss. DNN with adaption is trained with packet loss rate of 0.1. Adapted DNN could mask reconstruction errors due to packet loss and its image quality degrades more gracefully, even if the actual loss rate deviates from the trained rate.

to those due to quantization (as shown in Figure 3.9).

3.4 Implementation

Our implementation spans across three hardware platforms: Google Cloud VMs with Nvidia V100/T4 GPU for training, Google Coral Dev board with Edge TPU using TFLite for inference, and Kendryte K210 AIoT platform for deployment. It takes 500+ GPU-hours for training and evaluation with different objectives and datasets. The training code is written in Python with TensorFlow 2.2 and Keras [34] as the deep learning framework except for model quantization with TFLite, which is done in TensorFlow 1.15 due to compatibility issues. Modeling training is done with Nvidia GPUs. We export Tensorflow/Keras model to TFLite after training. TFLite cannot utilize Nvidia GPU and is not optimized for x86 CPUs so we use Coral Dev board to speed up inference. The final trained model is then compiled for the Kendryte K210 AIoT platform with NNcase [9]. After flashing model to the AIoT platform, we develop MicroPython code in MaxiPy IDE [7] for benchmarking image capturing and DNN inference.

Image dataset: we use well-known, public datasets for our benchmark instead of collecting a private dataset to emulate a wide range of IoT camera applications and facilitate

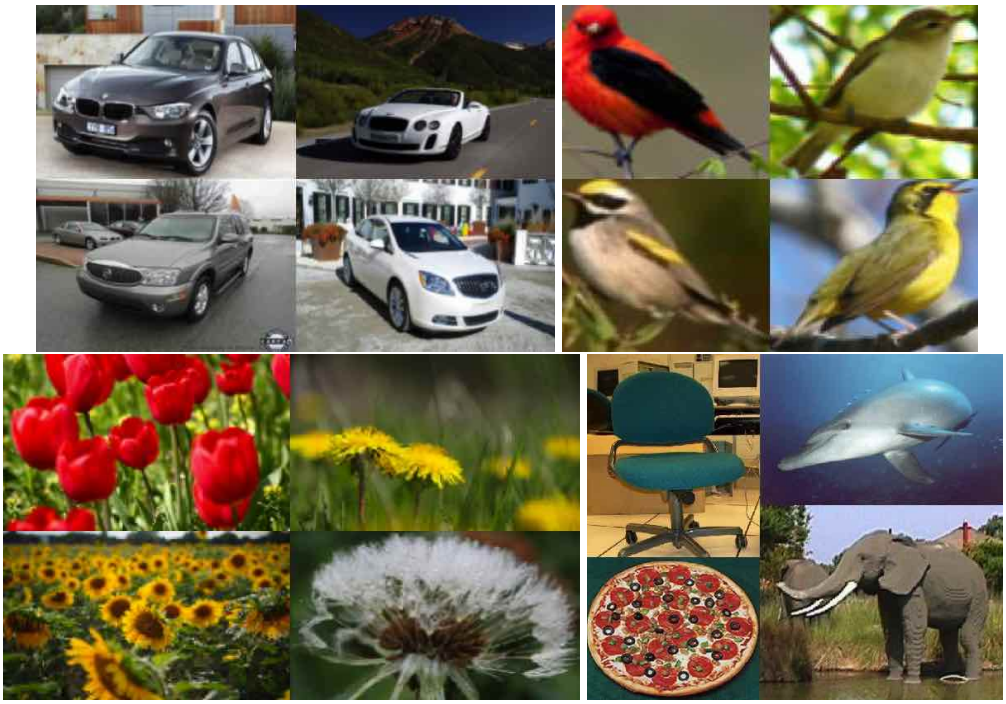

Figure 3.11: Sample images from Stanford Cars, Caltech Birds, Tensorflow Flower and Caltech101 dataset.

reproducing. We summarize the datasets we've used in Table 3.5. Only labeled images are used to evaluate the performance of compression that randomly split into train/test dataset with 80%:20% ratio. We use a data augmentation method that shift, rotate and flip training images to avoid over-fitting. We also show some sample images from each dataset in Figure 3.11. Stanford Cars, Caltech Birds and TensorFlow Flowers are domain-specific datasets that exist similarities between images inside the dataset while Caltech 101 is more general. We re-sample all images into 224×224 with Lanczos algorithm [70], which is a standard resolution for many DNN models pre-trained on ImageNet [37].

DNN training: all models are trained with `Adam` optimized with learning rate of 0.001. Implementation of the MS-SSIM and perceptual loss are straight-forward with TensorFlow API `tf.image.ssim_multiscale` and `tf.keras.application.VGG16(weights='imagenet', include_top=False)`. The classification loss is implemented with

Table 3.5: Summary of Image Datasets Used

Dataset	Labeled Images	Classes
Stanford Cars [68]	8144	196
Caltech Birds 2011 [121]	11788	200
TensorFlow Flowers [117]	3670	5
Caltech 101 [42]	9144	102

Table 3.6: Properties of Compression DNN

Property	Result
Image Patches	4
Channels of each Conv2D Layer	64,64,64,8
Total Number of Parameters	78k
Quantized Model Size	83 KB
Total Number of Ops	126 M
Per Image Running Time on K210	0.94 second
Per Image Energy consumption	0.10 Joules or 7.81 uAh for 3.7V battery

`tf.keras.applications.ResNet50`. Classification model are pre-trained on ImageNet [37] dataset and we then train them on each dataset with `categorical_crossentropy` loss and obtained an validation accuracy of 97.97%, 71.57%, 93.05% and 93.27% for Stanford Cars, Caltech Birds 2011, TensorFlow Flowers and Caltech101 respectively. Classifying birds is a considerably harder task than the rest.

DNN parameters: A summary of the NAS result and DNN properties is shown in Table 3.6. We believe that there is significant headroom for optimizing AIoT compilers and the inference pipeline as the average inference speed is only 134 MOPS when running compression DNN, which is several orders of magnitude lower than specified 230 GOPS theoretical speed performance of the K210 chipset. The compression DNN could be as fast as JPEG if the average inference speed is 7.9 GOPS (so it will complete in 16ms).

3.5 Evaluation

We evaluate STARFISH from these perspectives for a better understanding of the system:

- **Compressed size vs. quality**. We benchmark the compressed file size between

STARFISH and JPEG for various task objectives, and evaluate how much do we benefit from each design considerations.

- **Resiliency to packet loss**. We benchmark the performance of STARFISH in the presence of various packet loss rates.
- **System-level benchmark**. We present a system-level simulation to evaluate how much do we benefit from STARFISH in LPWAN that has a lot of transmitting nodes.

3.5.1 Compressed size vs. quality

Size-quality benchmark: similar to rate-distortion curve, in Figure 3.12 we show the rate-quality curve with different loss functions on the Stanford Cars dataset. Experimental results show that STARFISH can achieve significantly better bandwidth efficiency when compared against JPEG: STARFISH is up to 2.7× as efficient with MS-SSIM objective, and more than 2.9× for VGG perceptual objective and 3.0× for classification tasks.

Operation range of STARFISH: We noticed that the improvement gets smaller for higher quality metrics due to the limited capacity of our DNN model; it is possible to have DNN models with broader working range and better performance [22, 104] but exceed the capability of AIoT hardware. STARFISH could easily take advantage of newer AIoT accelerators with its automated pipeline. Nevertheless, we believe the current working range from less than 1KB to 3kB is a great fit for LPWAN that has very limited bandwidth and total capacity.

In Figure 3.13 we benchmark STARFISH on different datasets in terms of compression efficiency with regard to JPEG. The result indicates that STARFISH significantly outperforms JPEG consistently for all quality metrics across all datasets. STARFISH performs slightly different across datasets: we achieved the lowest performance gain on Caltech101 dataset in terms of MS-SSIM and VGG Loss objective, as images subjects tend to be more versatile in that dataset; we also achieved higher gain on Tensorflow Flowers dataset for classification tasks due to the fact that there are significantly fewer classes for that dataset.

How much do we benefit from content-aware and task-aware: being aware of the content and task are unique advantages of DNN over JPEG. We try to understand how much

do we benefit from that. We first benchmark the performance of DNN models trained on different training datasets and evaluates them on the Stanford cars test dataset. The result indicates that there is a significant gap between the content-aware one that is trained on Stanford Cars dataset and the rest. Similarly, we benchmark the performance of DNN trained with different loss functions and evaluate them with VGG loss (for perceptual quality). Results reveal that task-aware DNN outperforms others in terms of max, mean, and min compression efficiency significantly.

Time and Energy saving from DNN compression: STARFISH does not only reduce network traffic, thus making it possible for the gateway to server more IoT cameras, but can also provide transmit time and energy savings. we calculate time and energy saving with STARFISH to better understand the performance gain. We compare STARFISH and JPEG in Figure 3.16. Results suggest that we can get improve time/energy efficiency by up to 2.5$\times$ for the link with bad condition.

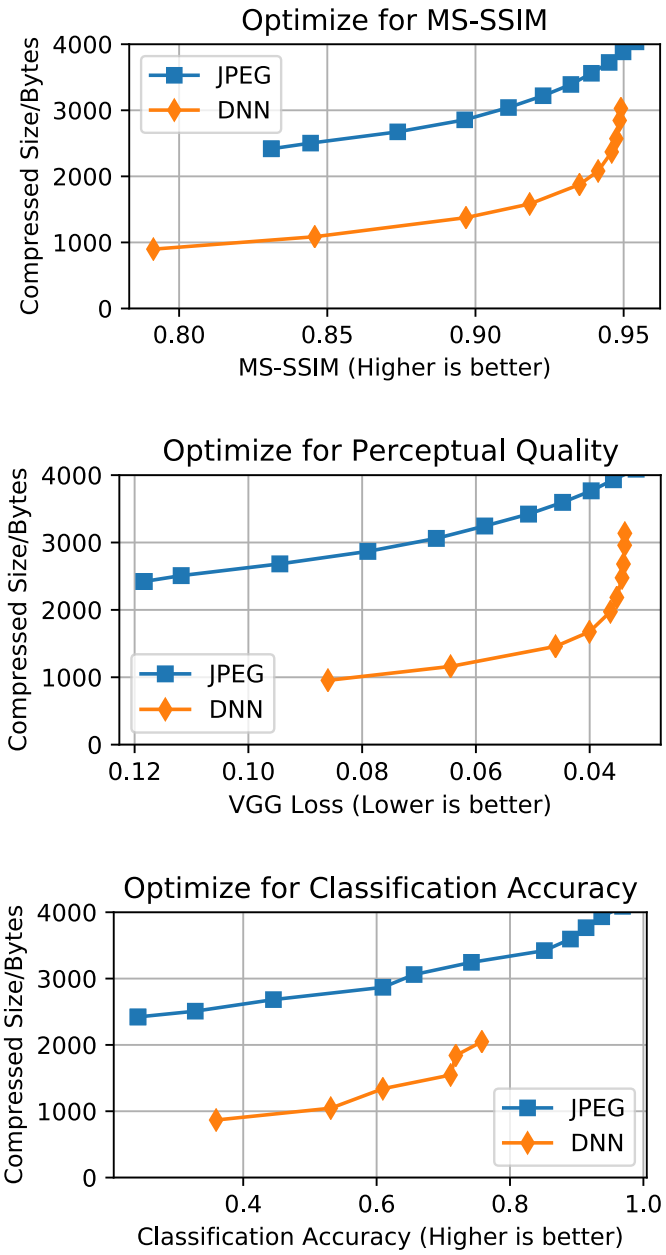

Figure 3.12: Size-quality benchmark for DNN in STARFISH and JPEG with different optimization objectives. Compressed File Size is the smaller the better for a given quality metric. JPEG compressed files could be up to 2.7× as large as DNN with the same quality for MS-SSIM metric, and about 3× as large for perceptual quality and classification accuracy.

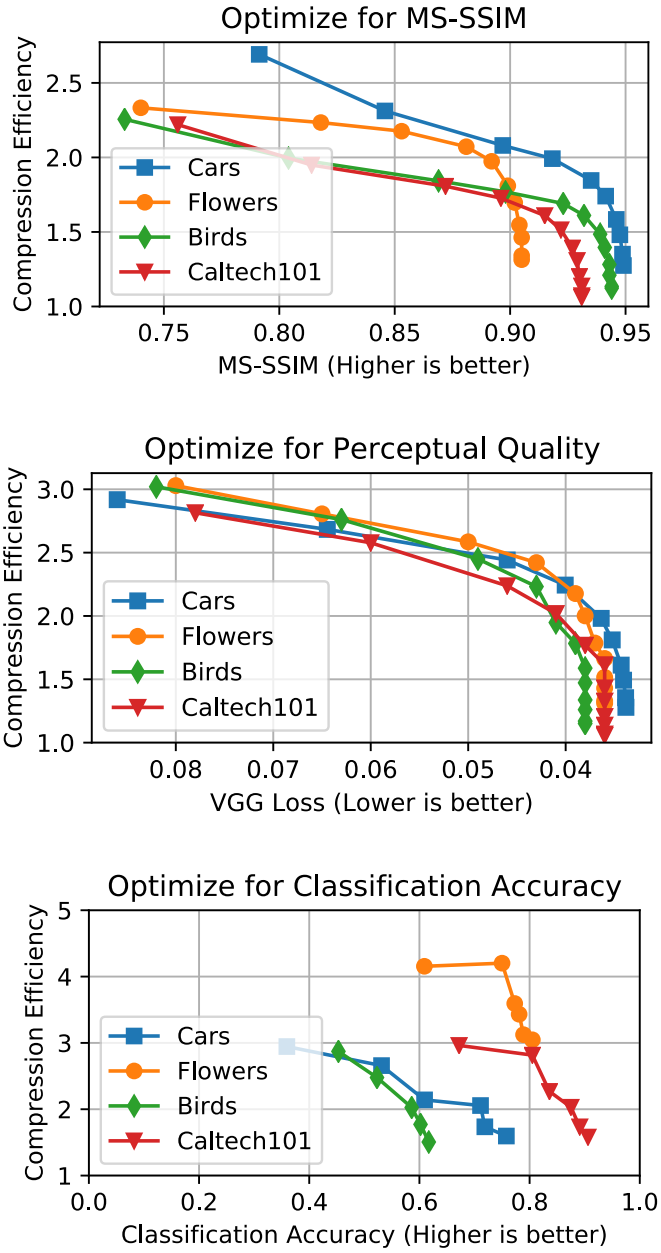

Figure 3.13: Compression efficiency of STARFISH on different datasets with different optimization goals relative to JPEG. The compression efficiency of JPEG is defined as 1, so a compression efficiency of 2.5 means DNN uses only 40% of the bandwidth when compared with JPEG. DNN significantly outperforms JPEG on all datasets with all optimization goals.

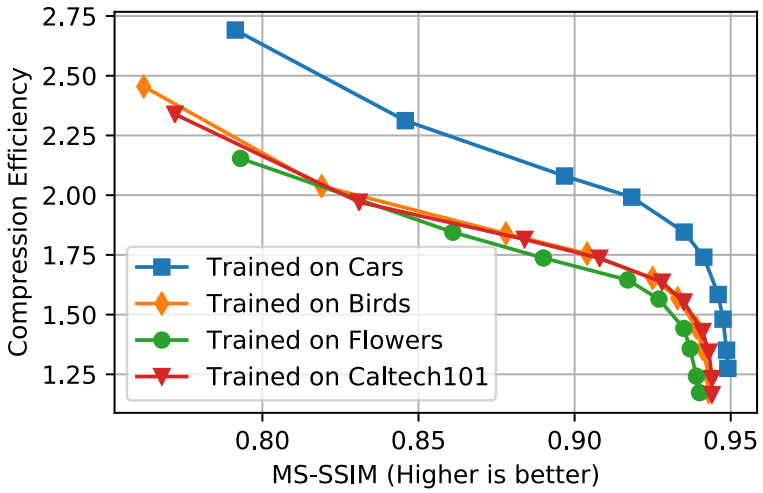

Figure 3.14: Content-aware benchmark by comparing models trained on different datasets and evaluate on Stanford Cars dataset. The model that train and test on the same dataset clearly outperforms models trained on a different one. However, all models still outperform JPEG by wide margins.

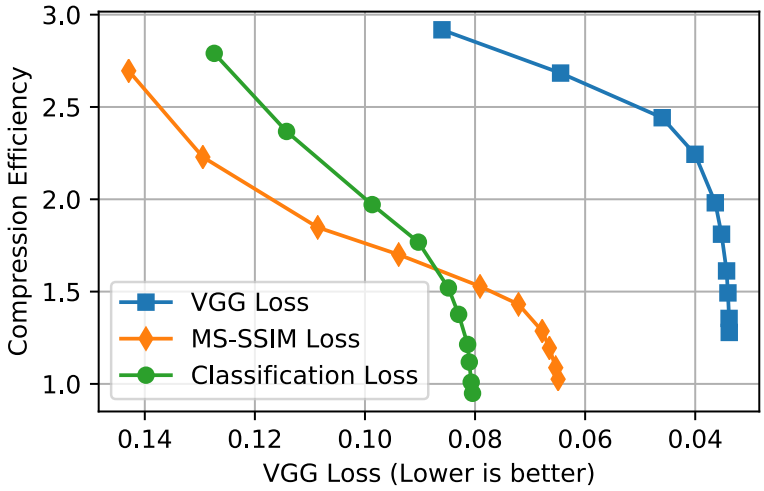

Figure 3.15: Task-aware benchmark by comparing models trained with VGG/MS-SSSIM/Classification loss and evaluated with perceptual quality using VGG loss. Despite the significant shift in working region, both the max and mean efficiency drops for DNN trained with MS-SSIM and classification loss, with efficiency degrades close or even worse than JPEG at small VGG loss values.

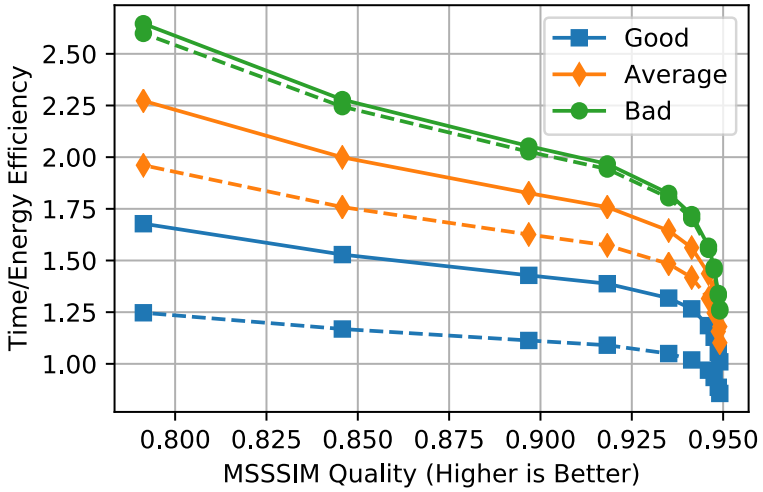

Figure 3.16: Time and energy efficiency of STARFISH under different link conditions. Solid lines showing time efficiency and dashed lines show energy efficiency. Two times as efficient means we need to spend only half of the time/energy for the image when compared against JPEG

3.5.2 Resiliency to packet loss

So far, we assume the link is reliable (using confirmed traffic in LoRaWAN). Now let's benchmark the performance of STARFISH in the presence of packet loss. We trained DNN with different dropout rates from 0.05 to 0.40 and found a dropout rate of 0.1 achieves the best compression efficiency. It even works better at a higher loss rate than DNN trained exactly on that rate, potentially due to the fact that DNN cannot learn well with a loss rate that is too large.

We benchmark the compression efficiency for loss resilient DNN against JPEG *assuming no packet loss* in Figure 3.17. Results indicate that could outperform JPEG over a wide range of target MSSSIM quality and loss rates. For example, at MSSSIM value of 0.85, STARFISH could outperform JPEG even with 20% packet loss. Compression efficiency for perceptual quality and classification tasks shows a similar pattern. DNN in STARFISH exhibits graceful degradation across a wide range of packet loss rates thus it is more favorable than FEC (Forward Error Correction) which needs to know actual loss rate to determine redundancy. Nevertheless, STARFISH is also much more efficient: LoRa supports a coding rate of 4/7 which means 4 bits of data and 3 bits of parity check to correct 1 bit of error, thus there is 75% overhead to just recover 25% data error.

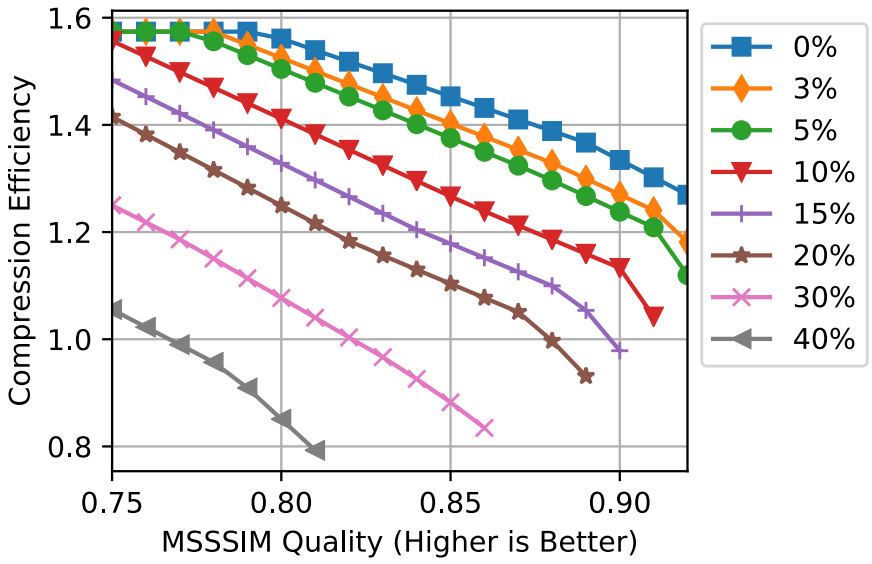

Figure 3.17: Compression efficiency of resilient DNN in STARFISH against JPEG under different packet loss rates ranging from 0% to 40%. Overall performance gain is lower than non-resilient DNN as we have to remove Huffman coding. Result shows that STARFISH degrades gracefully in the presence of data loss. STARFISH outperforms JPEG even under heavy loss.

3.5.3 System-level benchmark

The benefit of loss tolerance for a single link shown in Table 3.1 does not take collision into account. However, there may have hundreds of nodes connected to a gateway in LPWAN, and collision happens frequently. In this section, we use LoRaSim [27] to understand how the loss resiliency of STARFISH translates into performance in large scale deployment. We simulate 100 to 1000 transmit nodes that are randomly placed up to 3km away from the gateway[5]. All nodes are configured with fixed transmit power that is optimized based on the distance to the gateway. The channel loss model is log-distance and the path-loss exponent is set to 2.08. Aside from simulating frequency/spreading-factor/power/timing collision, full collision check is enabled that not only considers packets arrived at the same time, but also includes the "capture effect" where the relatively stronger signal could be received based on receive power and relative timing. We run the simulation for 1 hour. We vary the average sending interval to simulate different traffic density with a total of 2400 different combinations. Notice that we count the number of packets rather than emulate actual image data due to the implementation of the simulator we used. A large-scale real-world deployment or more authentic emulators that provide an interface to real data would be good future directions for this work.

The simulated total throughput and energy efficiency are shown in Figure 3.18. We can find the data deliver rate drops significantly as total throughput increases due to increased collision of packets. The throughput is only 219 KBytes per hour at 95% data delivery rate (for JPEG) when compared with 825 kBytes per hour at 80% data delivery rate (for STARFISH). There is significant benefit for applications to tolerate data loss: STARFISH could deliver 3.7$\times$ as many images as JPEG does while providing better image quality, even ignoring the additional overhead of re-transmission for the 5% packet loss for JPEG. We only observe minor energy degradation by sending more aggressively.

[5]The command we use is: "`python3 loraDir.py N_nodes Interval_ms Exp_mod=3 Sim_ms=3600000 Collision=1`" where `N_nodes`$\in$ $[100, 200, 500, 1000]$ and `Interval_ms`$\in$ $[2000, 120000]$ `to` $[20000, 1200000]$.

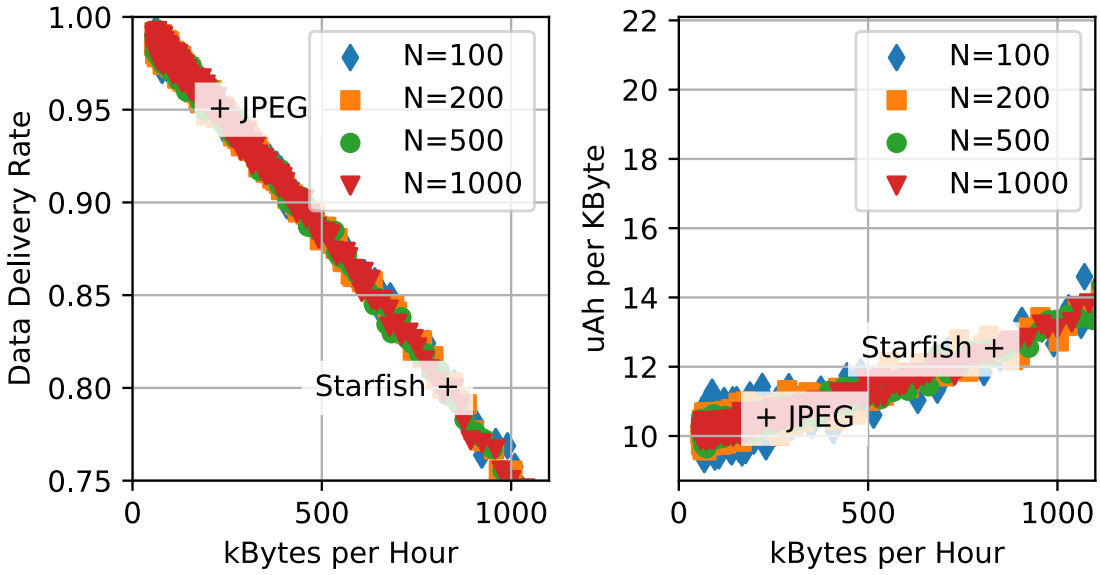

Figure 3.18: Simulated LoRaWAN total throughput (left) and energy efficiency (right) for a single gateway with 100 to 1000 IoT devices in the network. Data deliver rate drops sharply as total traffic increases, thus tolerating data loss brings huge throughput gain.

3.6 Robustness of STARFISH

The performance of DNN in STARFISH is sensitive to various hyper-parameters, including learning rate, choice of the optimizer, size of the dataset, and the number of training epochs when training for a new task or on a different dataset. For example, the optimizer will not converge if we magnify the learning rate by 10×, or use the Adagrad optimizer (instead of Adam) with the same learning rate on the Stanford Cars dataset while optimizing for MS-SSIM metric, as shown in Table 3.7. We observe degraded performance when using a subset of the training dataset. The number of training epochs matters as well. The normalized performance for training for 10% of the epochs is 86% and gradually increases to 98% while trains for half of the total epochs, as shown in Figure 3.19.

We believe AutoML(Automated Machine Learning)-based HPO(Hyper-Parameter Optimization) could help to reduce the effort of finding the best hyperparameters. The most widely used HPO approaches include grid search, random search, and Bayesian optimization. A fully automated pipeline that combines NAS and HPO could be a good future

Table 3.7: Normalized performance of STARFISH under different experiment settings

	Increase learning rate	Use Adagrad instead of Adam	Train on 50% of data	Train on 10% of data
Normalized Performance	16.54%	18.63%	97.39%	86.47%

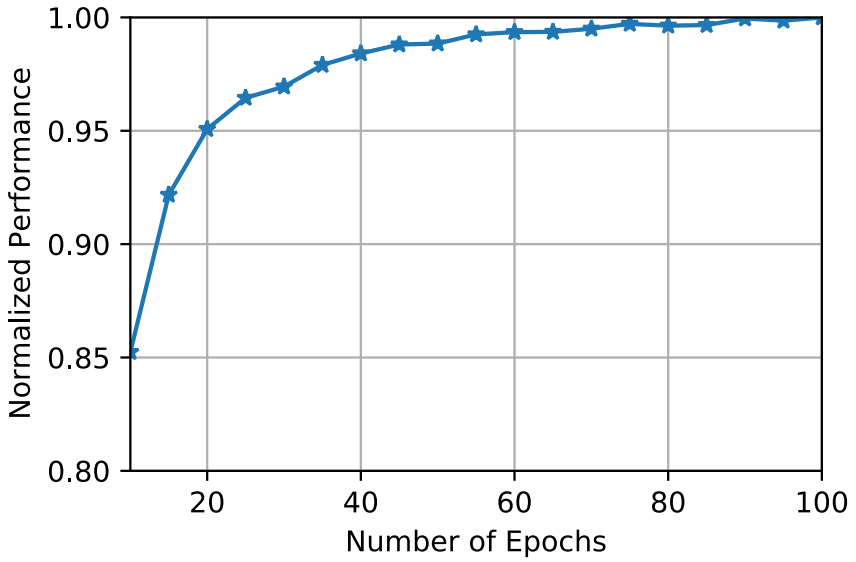

Figure 3.19: Relationship between normalized performance and number of training epochs.

research direction to improve the robustness and making STARFISH more practical.

STARFISH is robust to different content types and weather/lighting conditions. We test model trained on the Stanford Cars dataset and evaluate on the Cartoon Set dataset [4] that consists of cartoon faces, and the Multi-class Weather dataset [8] with images under sunny/cloudy/rainy/sunset conditions. The experiment results are shown in Figure 3.20 for the Cartoon dataset and Figure 3.21 for the weather dataset. We find that experiment results are very similar to Figure 3.14 that significantly outperforms the baseline. Experiment results suggest that STARFISH could be generalized to unseen images, even with a quite different genre.

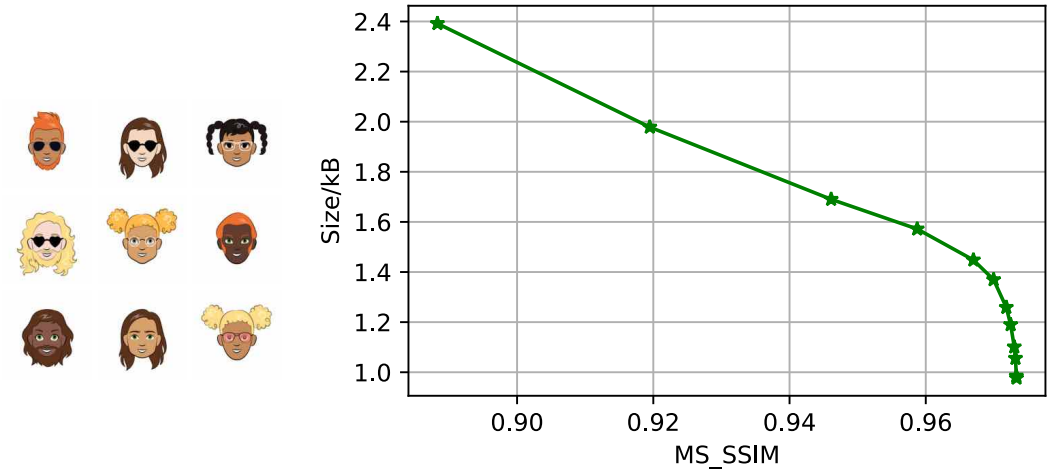

Figure 3.20: A visualization of the Cartoon dataset (on the left) and performance of model trained on Stanford Cars and tested on this dataset (on the right) .

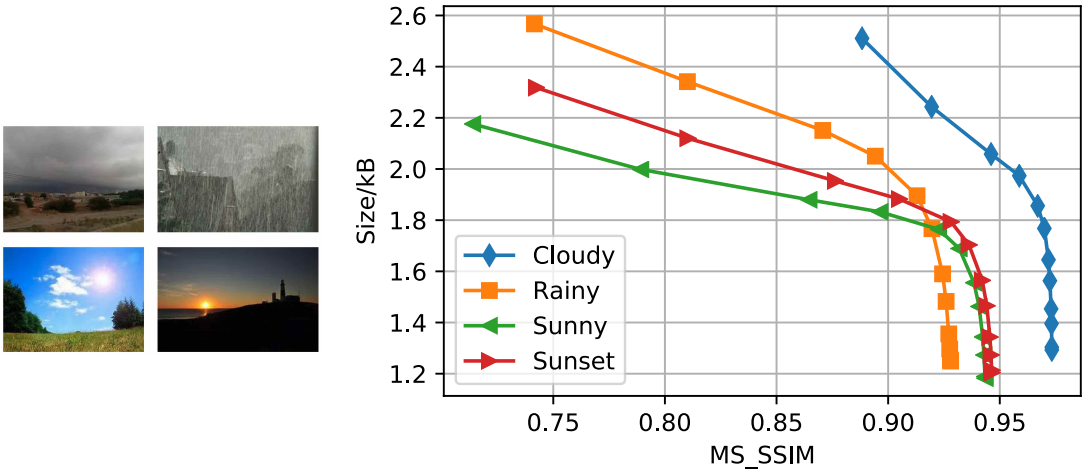

Figure 3.21: A visualization of the Weather dataset (on the left) and performance of model trained on Stanford Cars and tested on this dataset (on the right)

3.7 Related Work

LPWAN and DNN-based image compression are hot topics in their respective research communities. Our work is inspired by a great deal of previous work in DNN architecture design, error correction in LPWAN, and IoT camera systems.

DNN for image compression: there has been significant prior works on DNN-based image compression techniques that have inspired our design: end-to-end optimized [22], recurrent neural network [118], context-adaptive [71], residual coding [72], content-weighted [74], conditional probability models [89], GAN(Generative Adversarial Networks) [16, 23], and real-time compression on desktop-grade GPUs [104]. However, most of them focus on reducing compressed size except for the last one. None of them are designed to run on AIoT accelerators with a few MB of RAM and GOPS-level computation power. In addition, STARFISH differs from them fundamentally by being aware of the underlying lossy wireless link, featuring loss resiliency in the application layer. There are works on optimizing JPEG with DNN by learning quantization table, color space transformation, and other hyper-parameters of JPEG codec [131, 81], but the performance gain is significantly limited by the JPEG framework and cannot tolerate data loss.

Network architecture search: works that optimize search speed in NAS usually falls in to four categories [40]:

1. *Learning curve extrapolation*[21, 65] that extrapolate learning curve after a few episodes of training.

2. *Lower fidelity estimates*[106, 100] that training on partial or down-sampled data.

3. *Weight inheritance*[39, 29] that warm-start new models by inheriting weights from parent model.

4. *One-shot model*[30, 130] that trains only one model and use its sub-graph to generate smaller models.

In STARFISH we combine the first two methods to speed up finding the best architecture as they're easier to train and implement. However, other methods are worth trying in future works.

Error correction for LoRaWAN: Collision and perturbation of wireless channels cause frequent packet error. Here are recent works on recovering error for LPWAN: DaRe [85] is an application layer coding that recovers data from redundancy and outperforms repetition coding by 21% percentage; NScale [119] tries to resolve concurrent transmission leveraging subtle inter-packet time offsets; FTrack[128] uses a parallel decoding method to decode concurrent LoRa symbols. STARFISH could benefit from these works as combining them would allow more aggressive bitrate and more concurrent transmission leading to higher energy efficiency and overall network capacity. Whereas, STARFISH provides an alternative approach for error correction, featuring loss resiliency in the application layer and provides more significant performance gain.

Distributed inference: instead of sending images to the cloud, distributed inference provides an alternative solution by splitting DNN inference between the edge and cloud [41, 33, 87]. However, these methods are not designed for application scenarios where human-readable images are required for additional verification or the data should be repurposed for other AI tasks. Also distributed inference tends to put more pressure on the network side, which is not desired in LPWAN.

3.8 Theoretical Analysis

Source-channel coding theorem [108]: Source coding is the process that compresses the input to remove redundant information, and in channel coding, we use an error-correcting code to re-introduce redundancy that allows for reconstruction in the presence of noise. It is proven that for a Digital Memoryless Communication channel, as the input size goes to infinity, it is optimal to apply source coding and channel coding separately. Let V^n be inputs to the encoder and $\hat{V}^n$ be outputs of the decoder. $H(\mathcal{V})$ be the entropy of $\mathcal{V}$ and C be the communication channel capacity, we have:

> **Theorem** (Source-channel coding theorem). If V_1, V_2, ... V_n satisfies Asymptotic Equipartition Property and $H(\mathcal{V}) \leq C$, there exist a source-channel code with $p(\hat{V}^n \neq V^n) \to 0$. Conversely, if $H(\mathcal{V}) > C$, the probability of error is bounded away from 0.

However, STARFISH is a joint source-channel coding technique based on two real-world observations: the input length is not infinity, and the application could tolerate data loss. Previous work [67, 66] has shown that joint source-channel coding brings considerable performance gain for lossy compression with finite blocklength from the information theory perspective. However, there is no theoretical guarantee on STARFISH, or other DNN-based compression techniques are close to the optimal bound. We hope future work on explainable DNN could proving the upper/lower bound of DNN-based joint coding.

Distributed source coding [132]: it is defined as the coding of two or more dependent sources with separate encoders and a joint decoder. STARFISH could be viewed as coding with side information, a special case for distributed source coding that requires one communication channel only: information about the input images has been shared to the receiver side as decoder DNN weights. Previous work [53] shows that DNN weights are a capable medium for side information about the inputs.

Slepian-Wolf bound [113] could help us understand how and by how much could we benefit from side information in STARFISH: let R_X and R_Y be the coding rate for two i.i.d. random sequence X and Y, Slepian-Wolf theorem defines the achievable coding rate region must satisfy the following conditions:

$$\begin{aligned} R_X &\geq H(X|Y) \\ R_Y &\geq H(Y|X) \\ R_X + R_Y &\geq H(X,Y) \end{aligned} \tag{3.4}$$

Where $H(X)$ and $H(Y)$ are entropies of X and Y. The implication is that, as long as the total rate is larger than the joint entropy and none of the sources is encoded with a rate larger than its entropy, an arbitrarily small error probability could be achieved. Although STARFISH is a lossy compression method, we do observe the side information should be sufficiently large (DNN weights at the receiver side) and a closer distribution (smaller $H(X,Y)$) between the training and evaluation dataset leads to better performance as suggested by the Slepian-Wolf theorem.

Fountain codes / Rateless erasure codes: fountain codes refer to codes that do not have a fixed coding rate. They generate arbitrary long sequences of encoding symbols at the

source, and source information could be recovered from any subset of the encoding symbols. LT codes [84] and Raptor codes [110] are the most representative implementation of fountain codes that have been used in optical communication, wireless broadcasting, and deep-space communication. We further discuss fountain codes / rateless codes in section 3.9.

Progressive Coding [109, 38, 86]: progressive coding, especially for images has been studied extensively and was incorporated in the JPEG standard [122]. The image quality keeps increasing as more data coming in progressive JPEG while baseline JPEG renders the image in a block-by-block, top-down fashion. Progressive JPEG has two coding features: 1) spectral selection that sends low-frequency components first. 2) successive approximation that sends the most significant bits of the coefficient first.

Progressive JPEG seems to be a better choice than STARFISH for LPWAN, but it suffers from two shortcomings: 1) resiliency. Progressive JPEG relies on the previous decoded output to render the image, making it susceptible to data loss. 2) efficiency. Study [12] shows that for small images (<10kB), progressive JPEG files tend to be (a 75% chance) larger than baseline JPEG, which exactly images that could send over LPWAN efficiently.

Rate-distortion [108, 36]: originally proposed by Shannon, the rate-distortion function determines the minimum channel capacity required to transmit the source input as a function of desired average distortion. STARFISH could adapt intermediate representation length (which may require fine-tuning of the DNN) and quantization depth to achieve different rate-distortion curves. There are recent works [46, 22] on end-end optimized, DNN-based image compress that optimized for rate-distortion performance directly by including expected coding cost and reconstruction error in the loss function. We expect STARFISH to perform better by optimizing for rate-distortion if there is enough computation power.

Theoretical analysis on DNN for image compression: while DNNs have been wildly successful in various computer vision tasks and beyond, we actually know little about how/why they work [129, 35, 107]. The deeply nested nonlinearity, coupled with the huge number of parameters turns DNN into a perfect blackbox.

Alternatively, we can try to understand how DNN works by performing visual analysis. We show a comparison between two-dimensional DCT used in JPEG [6] and first layer

[6]DCT image is in public domain: https://commons.wikimedia.org/wiki/File:DCT-8x8.png

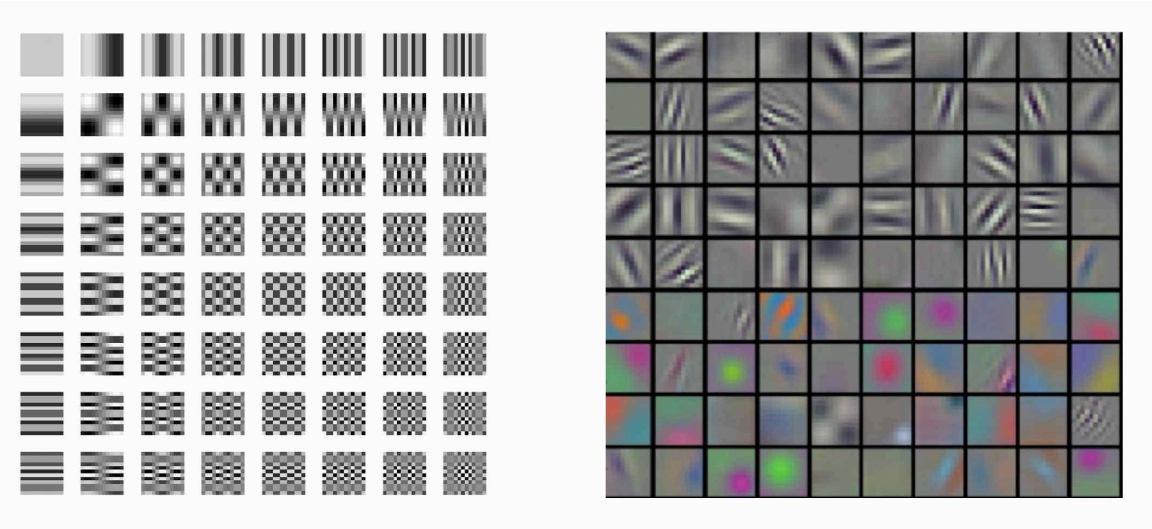

Figure 3.22: Visualization of two-dimensional DCT used in JPEG (left) and first layer filter weights of AlexNet

filter weights of AlexNet [135] in Figure 3.22. As we mentioned earlier, JPEG compression involves three basic steps: convert image into frequency domain with DCT, quantize DCT coefficients, and entropy coding. From Figure 3.22 we can find that the pattern of the convolutional filter visually resembles DCT, but the weights are learned rather than handcrafted, making it possible to adapt to certain images and tasks. Similarly, we also apply quantization on intermediate representations of DNN as applying quantization table on DCT coefficients. However, how does the DNN learns entropy coding remains an open question, and we welcome any future work that tries to open up the blackbox.

3.9 Limitations of STARFISH

STARFISH is not meant to be a replacement for general image compression techniques. It is best for TinyML Cameras that run DNN tasks and send images of interest over LPWAN accordingly. In these cases, a hardware accelerator is a prerequisite for DNN-based image classifications or object detection tasks. STARFISH thus conveniently reuses the pre-existing DNN accelerator hardware and software pipeline for transmitting images over the lossy, or even transmit-only link. Under certain conditions STARFISH could be used over

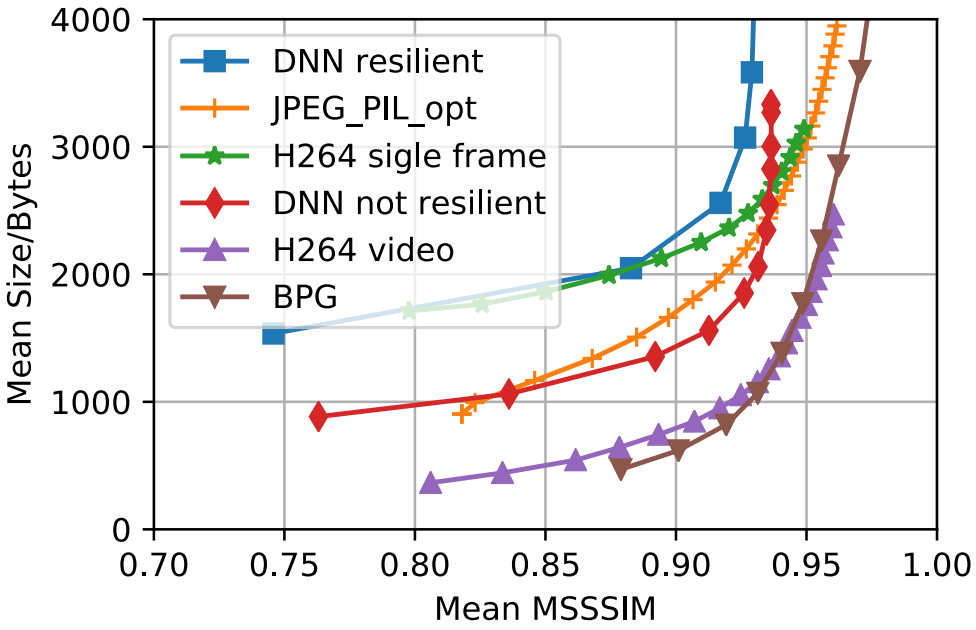

Figure 3.23: Comparing STARFISH DNN with other alternative image/video codecs

a reliable link if the native, non-DNN image compression is absent or performs poorly.

We discuss alternatives to STARFISH that may yield better performance for both lossless and transmit-only lossy links.

Over lossless link: Many image/video coding methods, include optimized JPEG could outperform STARFISH if the link is lossless (with the help of acknowledgment/retransmission). We compare the performance of DNN in STARFISH (in both resilient and non-resilient mode), optimized JPEG from PIL, H264 that encodes each image into a single frame or 128-images video and BPG, as shown in Figure 3.23. The result indicates that BPG and H264(video) clearly dominates the comparison, generating files that almost half the size as not resilient DNN in STARFISH. Well-optimized JPEG also performs as good or even better than STARFISH. DNN resilient and single-frame H264 performs considerably worse than other methods.

We also re-run visual quality comparison (Figure 3.24) and size-quality benchmark (Figure 3.25) for the optimized JPEG. The performance of JPEG improved quite a lot in both experiments.

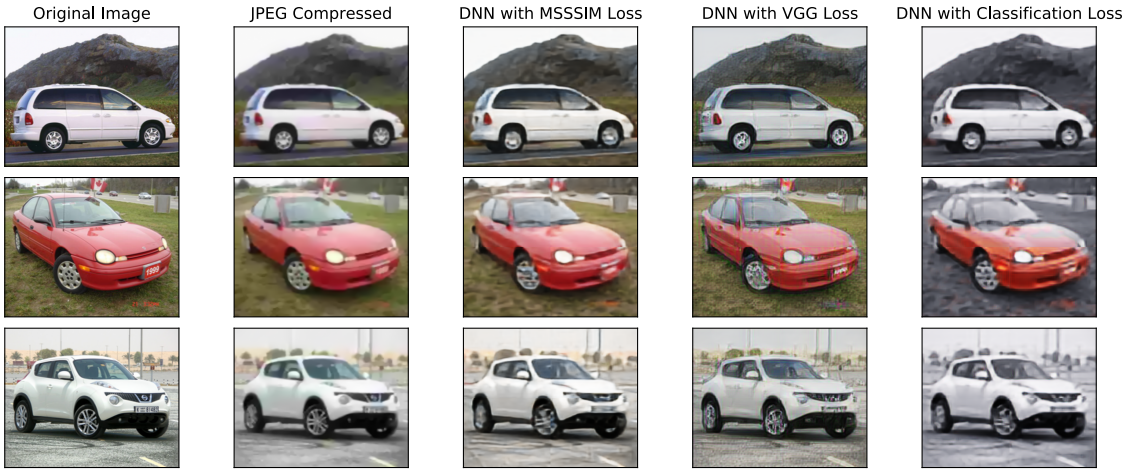

Figure 3.24: Visual quality comparison between optimized JPEG and DNN with different loss functions. Notice that the optimized JPEG looks much better than in Figure 3.7

As a result, we encourage the users to carefully check the performance of non-DNN based image compression and explore possibilities of improvements before considering STARFISH under lossless link scenarios. STARFISH is more likely to work if the hardware/software is heavily biased towards the DNN accelerators. The users could also consider adding a high-performance external codec (like BPG) for best performance.

Over transmit-only lossy link: We summarize representative works in image/video compression lossy, transmit-only link in Table 3.8. Overall there are two major design strategies:

- Rateless image coding: in this scheme, the input representation (either raw pixels or DCT coefficients) is first grouped according to their importance. Proportional representation is then generated from the distortion group with rateless coding so that image quality degrades gracefully as the loss rate increases. Generally, more redundancy is allocated to important groups (such as most significant bits).

- Error concealment: in this scheme, the input representation is partitioned into independently decodable segments, thus compatible with entropy coding. A whole image could be recovered from a subset of all segments sent with controlled quality degradation.

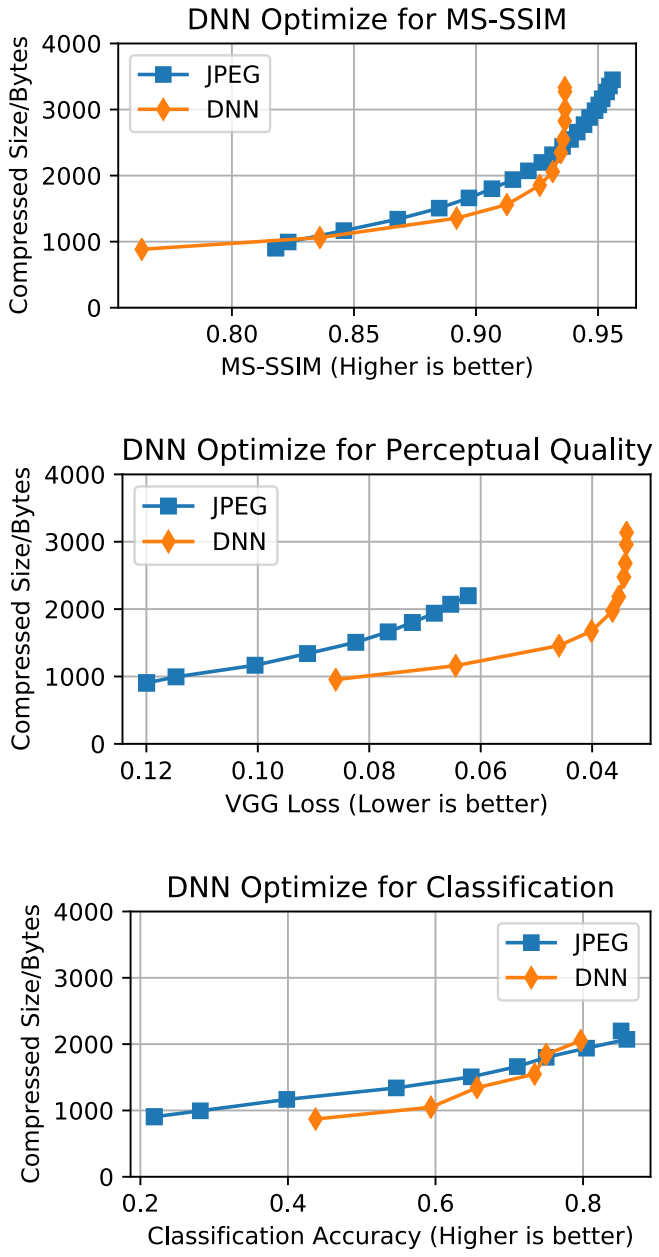

Figure 3.25: Size-quality benchmark for optimized JPEG and DNN in STARFISH with different optimization objectives. Also notice that JPEG performs much better than in Figure 3.12

However, both methods require human intervention in designing each stage and trade-off between protection levels and efficiency. In contrast, STARFISH is an end-to-end learned method that leverages existing AIoT software/hardware pipeline, making it easy to implement in the embedded setting. Still, we encourage users to check whether the performance of STARFISH could satisfy their needs in this scenario, and compare against alternative methods if could be conveniently implemented.

We've identified scenarios that STARFISH shines, as well as those scenarios that better solutions may exist:

A good scenario for STARFISH typically satisfying the following conditions:

- AIoT Devices with DNN accelerators that already using DNN for object/event detection tasks, so DNN could be used for the entire pipeline that includes image compression and channel coding;
- AIoT Devices that do not have highly optimized encoder;
- AIoT Devices that transmit over a lossy, severe bandwidth-limited wireless network with high cost associated with retransmission;
- Application scenarios that are possible to collect a dataset of sample images.

In many other scenarios, STARFISH might not be the best choice. Here are a few examples:

- Devices that come with highly optimized encoder, or do not have DNN accelerator;
- Cheaper to transmit than compress (e.g. RFID/backscatter [54, 55] network), or the cost of retransmission is low;
- No prior knowledge about target images; JPEG and many other codecs do not require that information.

We envision STARFISH could still have a large impact given the aforementioned application constraints, especially considering the prevalence of DNN for various image tasks

and the sheer number of AIoT/TinyML devices that are going to be deployed[7]. Both the design of DNN for compression and the underlying accelerator hardware are evolving at a fast pace, and newer compression methods that do not require DNN are being proposed. We encourage practitioners to try both methods and pick the one that suits their needs best.

[7]ABI Research predicts global shipment of TinyML Devices to reach 2.5 Billion by 2030: https://www.prnewswire.com/news-releases/global-shipments-of-tinyml-devices-to-reach-2-5-billion-by-2030–301123076.html

Table 3.8: Summary of works on image compression for lossy, transmit-only link

Method	**Input**	**Distortion Grouping**	**Rateless Code**	**Entropy Code**	**Proportional Representation**	**Error Concealment**	**Complexity**	**Erasure Channel?**	**Remark**
Fixed Length Coding [94]	Image	8*8 sub-blocks of image	Repetition	None	No, treat each sub-block equally	None	Very low	Yes	Use fixed quantization scheme
ULP [90]	Gray-scale image	Group wavelet coefficient trees into sets with SPIHT	Variable length Reed-Solomon	None	Orders groups by the highest bit plane of the magnitude	None	Not mentioned	Yes	Better than PET, FLIIT, PZW
ICER [64]	Image	None	None, but support progressive compression	Fast interleaved entropy coding	None	Partition wavelets	On par with JPEG2000	Yes	For Mars Rover mission
FlexCast [15]	DCT Coefficients from MPEG	DCT MSB to LSB	LDPC+LT	None, compression gain w/ BP decoding	Depending on importance and number of bits	Not mentioned, use MPEG	1500fps decoding on Core i7	No; Depends on SoftPHY	Close to omniscient MPEG
QLIC [28]	Image	None	Joint coding and unequal error protection	Joint coding	Included in coding	None	Linear	No, AWGN	Follow-up of ICER, map codes to symbols directly

3.10 Future works

We believe DNN is an especially powerful tool when combined with domain knowledge, and our work is just the beginning for a wide range of cross-layer, application-specific network stacks that achieve better capacity and computation/communication trade-off:

- *More efficient DNN design with novel architecture*. We believe the performance of DNN in STARFISH could be further improved with the state-of-the-art residual and recurrent architecture if the AIoT compiler and hardware support them, which may happen in the near future as AIoT hardware is evolving at an extremely fast pace. Another direction of research would be more aggressive quantization like XNOR-Net [99] or Dorefa-net [140] with customized implementation on FPGA devices to further push the limit of energy consumption.

- *Coordination across AIoT cameras* [60, 77]. Physically proximate cameras might see similar images and information/computation could be shared between them with less overhead than over a long-range link between cameras and gateway. With intelligent coordination, cameras could process more data locally that increases the battery life of AIoT cameras and network capacity.

- *DNN-based video streaming for AIoT cameras*. Recent works [105, 83, 32] show promising results on compressing videos with DNN and some of them have outperformed popular video encoding standards like H.264/H.265. Though these methods require massive computation power (2fps@640×480 resolution on Nvidia V100 GPU) or only work for extremely low resolution (32×32), we believe certain application like stationary cameras could benefit from this work as there is a lot of similarity across frames.

3.11 Conclusion

To sum up, we propose STARFISH, a DNN-based image compression framework for AIoT cameras connected by LPWAN that has limited capacity. STARFISH features the insight that solving link loss in the application layer could replace the need for reliable transmissions, thus brings simplifies system design and potential performance gain. We carefully design and implement STARFISH on a low-cost AIoT accelerator with strict resource constraints. Although the result is promising, we want to emphasize that STARFISH is not the panacea for many image compression problems in AIoT. We urge the practitioners to carefully review the requirements of STARFISH and weigh the pros and cons before putting it in use.

Chapter 4

Collaboration Across Cameras with Multicast

Large-scale camera networks that generate data at an unprecedented rate is entering our life, from security, to transportation and business intelligence. Police authorities in Chicago and London are analyzing data from tens of thousands of cameras in real time [58]. Cameras are widely used in ITS(Intelligent Transportation Systems) for counting, trajectory tracking, and anomaly detection. Businesses also install cameras to analyze customer trajectory, detecting queue length and occupancy for better planning and investment decisions [1].

However, as more and more cameras being deployed, there is an increasing redundancy of information, especially for cameras that are close to each other. Cameras are information-rich sensors that are capable of generating data at several Mbps, causing huge pressure over the network and computation resources. Enabling cross-camera collaboration in an efficient manner could eliminate such redundancy, thus reducing the operation cost of large-scale camera analytic systems.

We present YOUO, short for **Y**ou **O**nly **U**pload **O**nce. YOUO is an object re-identification system that shifts computation from cloud to cameras, thus enabling efficient collaboration between cameras for lower network and computation cost. We illustrate how YOUO works with a car counting example, as shown in Figure 4.1. The target object (car) is captured multiple times by multiple cameras, which is common in large-scale camera networks. Usually, each time a camera sees an object, it will upload the object to the cloud

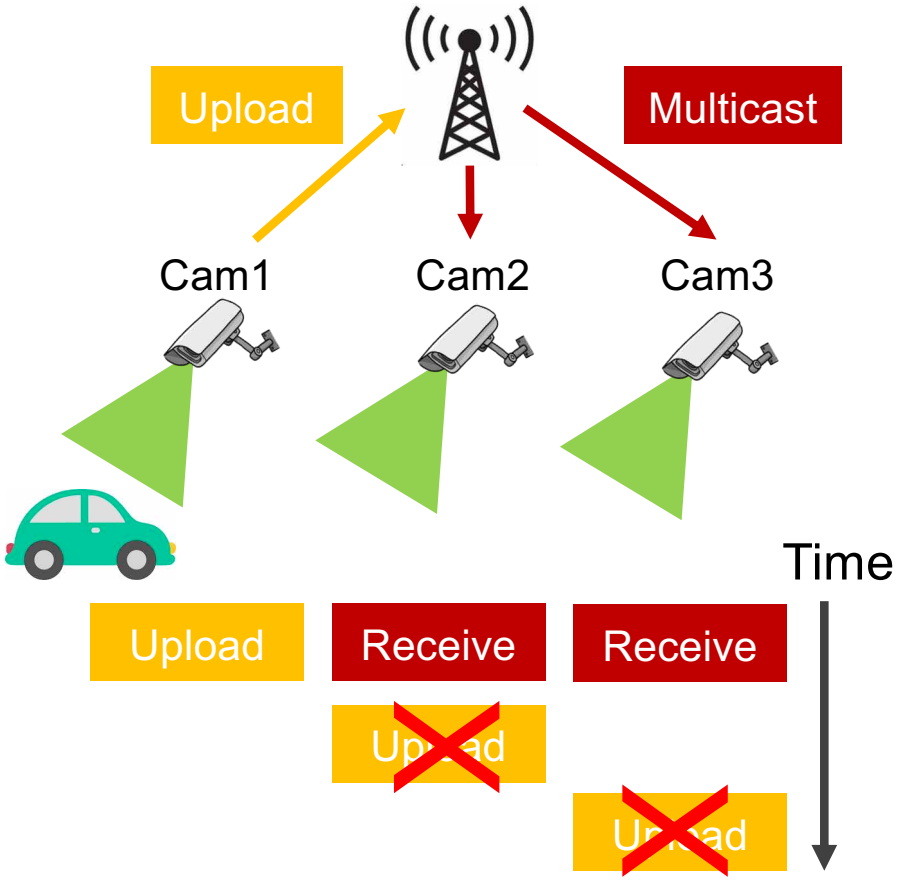

Figure 4.1: Illustration of YOUO. Each target object may have been captured multiple times by multiple cameras in large-scale camera networks. With the help of multicasting service provided in 5G eMBMS, YOUO enables efficient collaboration across cameras, thus significantly reduce total network cost.

for analysis, creating duplicated uploads. However, YOUO avoids duplicated uploads by sharing the information across cameras: if Cam1 detects an object, it uploads the fingerprint of the target object, and the fingerprint will be shared to nearby cameras via multicast, thus Cam2 and Cam3 do not need to upload the same object again and again, reducing the amount of network and computation resource needed.

YOUO achieves efficient collaboration across cameras with multicast. When compared with the baseline method that uploads all images, YOUO could reduce the network traffic by more than two orders of magnitude based on experiment results on a large-scale vehicle

re-identification dataset. YOUO is very efficient in bandwidth when compared with optimal single-camera filtering and maintains high accuracy when uploading only once per object, indicating its cost is sub-linear or even constant as the number of cameras increases.

4.1 Background

We provide an introduction on object re-identification, and related works on re-identification and on-camera filtering on AI-enabled cameras in this section.

Object re-identification task: re-identification is a process that matches the appearances of an object across different frames, especially if there are multiple cameras. It is the key to a number of camera analytic tasks including counting, tracking, and retrieval. In the re-identification task, we have an image of the target object or fingerprint generated from the object as query input, and try to find occurrences among all the cameras. For example, we're trying to find a van with a green covering as shown in Figure 4.2. All the cameras will calculate the distance between the query input and objects stored on the camera. After comparing the distances, Cam 2, 7, 8 return images or fingerprints that are determined to be a match as the distance is smaller than a predefined threshold, and camera 3, 18 and 14 return false as the most similar images is still not close enough as the query input.

Re-identification was done with hand-crafted metrics for describing images and distant metrics before DNN was widely used [139]. For example, re-identification using the histogram on different color channels and texture filters on the luminance channel is described in [45]. [138] improves the feature extraction algorithm with dense color histogram and SIFT(Scale-invariant feature transform). DNN for re-identification picks up after the huge success of Alexnet on the ImageNet task [69]. Re-identification with siamese network[1] to determine where two input images contain the same object or not directly are described in [134, 75]. An end-to-end re-identification is described in [133] that jointly modeling people's commonness and person uniqueness. In this paper, we focus on DNN-based re-identification as it leads to the highest performance on almost all datasets [139].

On-camera filtering on AI-enabled cameras: there are different architectures to perform the re-identification task with multiple cameras. Cameras without AI-capability have

[1] A type of DNN that uses the same weights to compute output with different inputs.

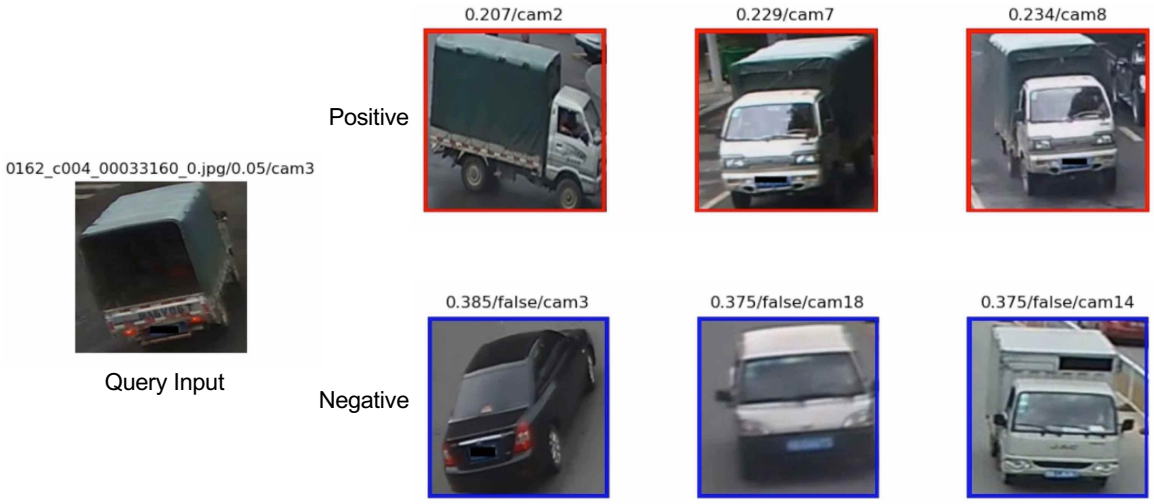

Figure 4.2: A sample query input (shown on the left) and positive/negative responses from different cameras (on the right).

to upload raw video into the cloud for further processing, as shown on the left in Figure 4.3. However, this method is costly in terms of network and computation. Instead of sending the entire video to the cloud, only cropped-out images of the relevant object will be sent to the cloud, as shown on the right in Figure 4.3. On-camera filtering resulting in fewer/smaller images send over the network, lowering the cost of network and cloud computation. Different strategies including early discard, just-in-time learning, compensate for false negatives and context-awareness are used to reduce bandwidth requirement for drones [123]. FilterFoward [31] enables on-camera filtering for different task objectives by reusing part of the computation with a series of "microclassifier", which are essentially lightweight filters. Reducto [76] leverages previously unused resources on commodity cameras for efficient filtering with cheap vision features.

The current on-camera filtering work focus on optimizing efficiency from each camera individually; none of them sharing knowledge between cameras directly. Spatula [58] leverage spatial correlation between cameras, but the analysis is done at the central server, which still stresses the network and computing resources.

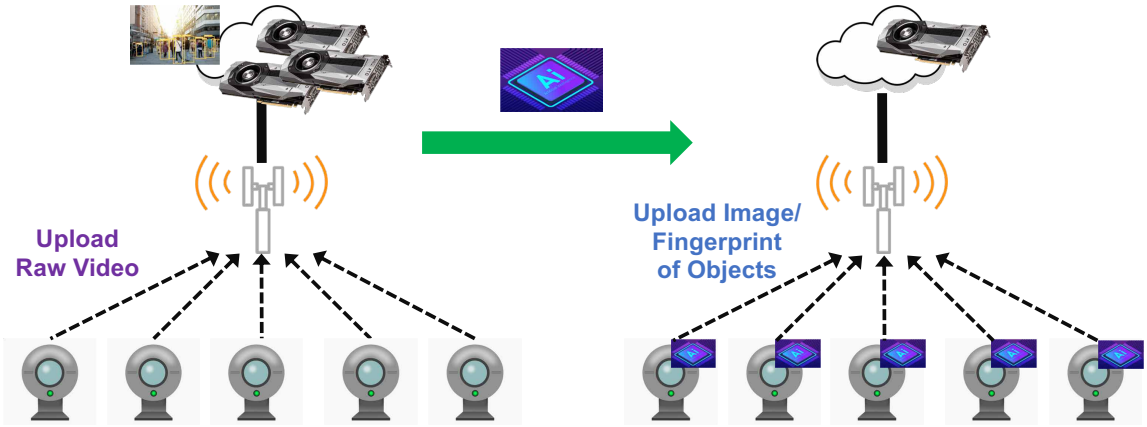

Figure 4.3: Comparison of upload raw video vs. on-camera filtering that upload image / fingerprint of target object.

4.2 Design of YOUO

Reduce spatial redundancy via cross-camera collaboration: spatial redundancy means that the same object may be captured multiple times by different cameras as more and more cameras are being deployed [58]. In YOUO, we enable cross-camera collaboration by sharing fingerprints across cameras to reduce the spatial redundancy, as shown in Figure 4.4.

The workflow of YOUO is shown in Figure 4.5. YOUO is driven by two events: the camera event, which is common in non-collaborative camera analytic systems, and the wireless event that receives fingerprints of the object detected by other cameras. Instead of upload everything to the cloud, YOUO compares the distance between newly detected objects against the list of shared fingerprints. If the distance between the newly detected object and the shared fingerprint list is smaller than a threshold, YOUO recognizes this object as a duplicated detection and does not notify the cloud or other cameras. YOUO upload fingerprints of the newly detected object into the cloud only if the distance is large than the threshold thus reducing duplicated uploads due to spatial redundancy.

Although YOUO does reduce duplicated uploads into the cloud thus reducing uplink traffic, it requires propagating fingerprints among other cameras, causing increased downlink traffic. A simple analysis on the network cost for different paradigms in Table 4.1 with N, M, k are the number of target objects, number of cameras, and average occurrence

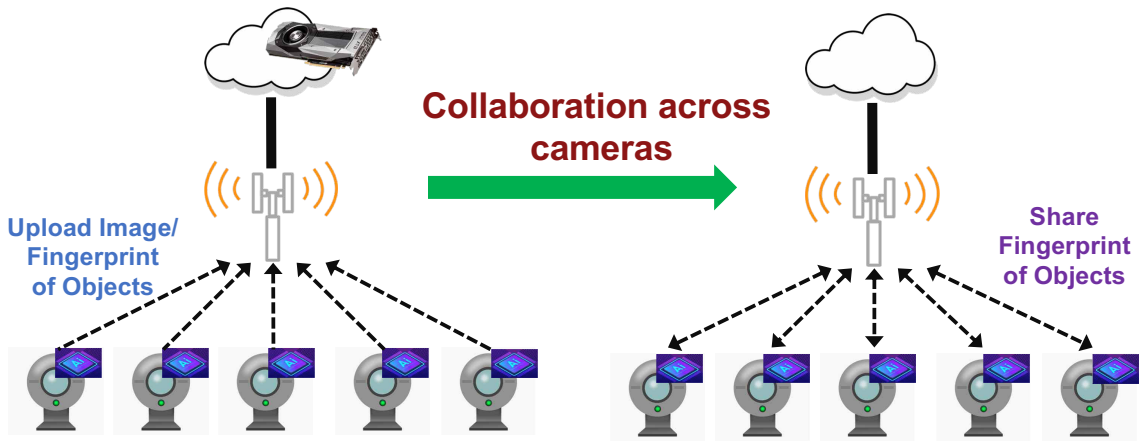

Figure 4.4: Collaboration across cameras by sharing fingerprints of objects to reduce spatial redundancy.

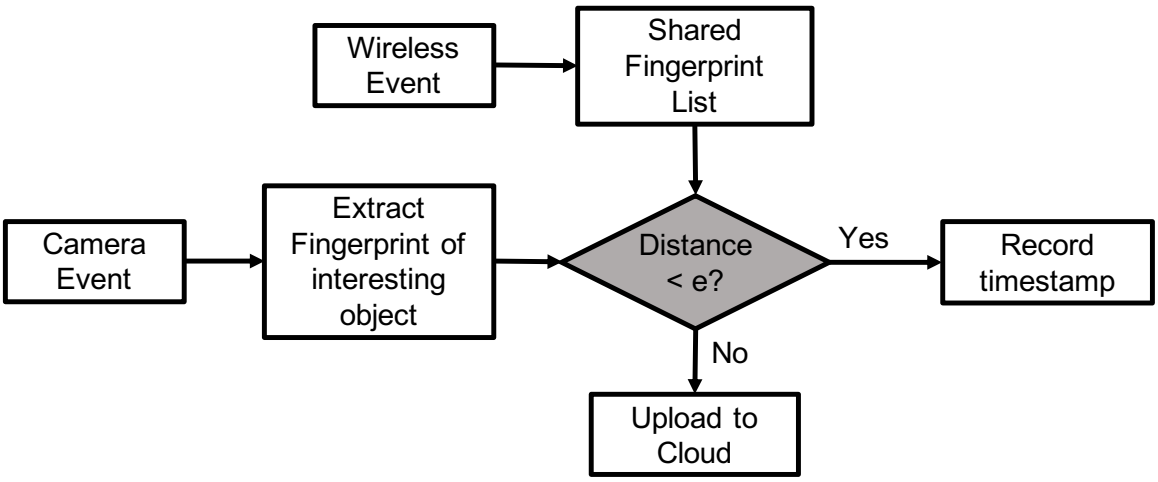

Figure 4.5: Workflow of YOUO driven by both the wireless event and camera event, while conventional non-collaborative methods are driven by camera event only, thus cannot filter out spatial redundancy.

Table 4.1: Comparing network cost for different paradigms of cross-camera analytic system

Cost	**Paradigm**		
	No Collaboration across Cameras		**Share Fingerprints (YOUO)**
	Upload images	**Upload Fingerprints**	
Uplink	$N * k * S_{image}$	$N * k * S_{fingerprint}$	$N * S_{fingerprint}$
Downlink	0	0	$N * (M-1) * S_{fingerprint}$
Total	$N * k * S_{image}$	$N * k * S_{fingerprint}$	$N * M * S_{fingerprint}$

* Assuming N target objects, M cameras, each object being captured k times on average. S_{image} and $S_{fingerprint}$ are the average file size of image and fingerprint respectively.

count per object. Notice that we assume a perfect fingerprint that each object could be identified correctly for simplicity. We also calculate payload only, ignoring protocol overhead like acknowledge and packet headers.

Uploading fingerprints generated on the camera rather than images does not change the number of uplink actions, but could reduce the network traffic as fingerprints generally much smaller than images (1kB vs. $\sim$ 10kB each). However, task accuracy may suffer as the camera may not be able to run as powerful DNN as the cloud does. We compare the performance between upload images and fingerprints in section 4.6.

From Table 4.1 we can find that YOUO uploads only once per object, reducing uplink cost by k times. However, there is a significant cost of sharing the fingerprint information to $M-1$ other cameras. As a result, the total network cost is $N * M * S_{fingerprint}$ vs. $N * k * S_{fingerprints}$ for paradigm without collaboration. Since k is smaller than M as objects appear in some of the cameras, YOUO actually costs more in terms of network bandwidth.

Fortunately, we can reduce the downlink cost by leveraging the broadcast insight of the wireless medium. In addition to point-to-point communication from the base station to each camera via unicast (as shown in Figure 4.6), it is also possible to send *the same* information from the base station to all cameras via broadcast, as well as send to a targeted group of cameras via multicast.

As a result, we can reduce downlink transmission from $N * (M-1) * S_{fingerprint}$ to $N * S_{fingerprint}$ since we can broadcast fingerprint information to the entire network at once,

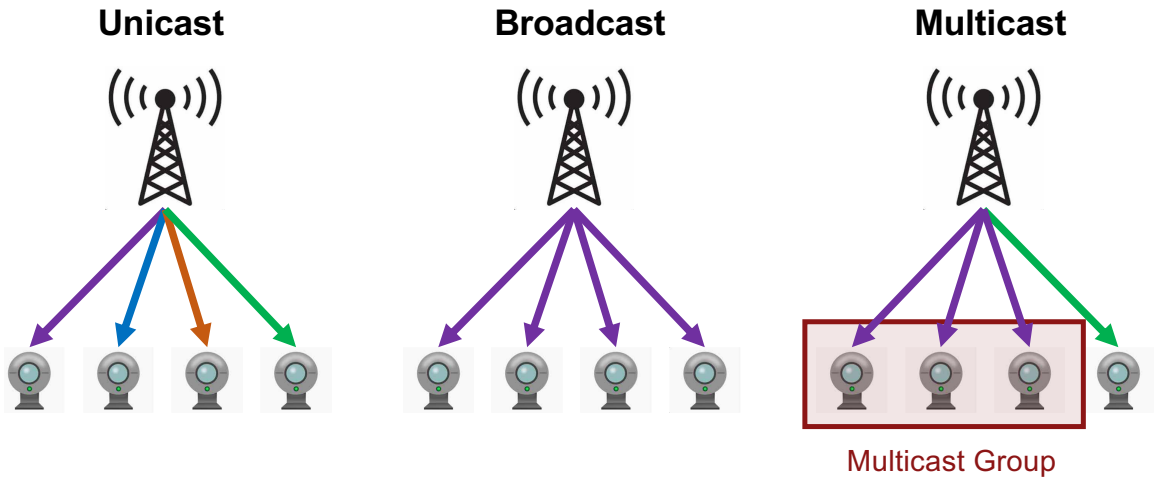

Figure 4.6: Illustration of unicast vs. broadcast vs. multicast in cellular network.

Table 4.2: Comparing network cost for different paradigms of cross-camera analytic system with Multicast support

Cost	**Paradigm**		
	No Collaboration across Cameras		**Share Fingerprints via Multicast (YOUO)**
	Upload images	**Upload Fingerprints**	
Uplink	$N * k * S_{image}$	$N * k * S_{fingerprint}$	$N * S_{fingerprint}$
Downlink	0	0	$N * S_{fingerprint}$
Total	$N * k * S_{image}$	$N * k * S_{fingerprint}$	$N * 2 * S_{fingerprint}$

* Assuming N target objects, M cameras, each object being captured k times on average. S_{image} and $S_{fingerprint}$ are the average file size of image and fingerprint respectively.

as shown in Table 4.2. The total cost $N * 2 * S_{fingerprint}$ is independent of the number of cameras, meaning that it could be scaled to a large camera network.

4.3 Implementation of YOUO

In this section, we describe the implementation details of YOUO, including the dataset and how we train the DNN for re-identification.

Dataset: We evaluate the performance of YOUO on the VeRi dataset[78, 79, 80] that stands for **Ve**hicle **R**e-**i**dentification. It consists of 20 cameras on the main streets of a

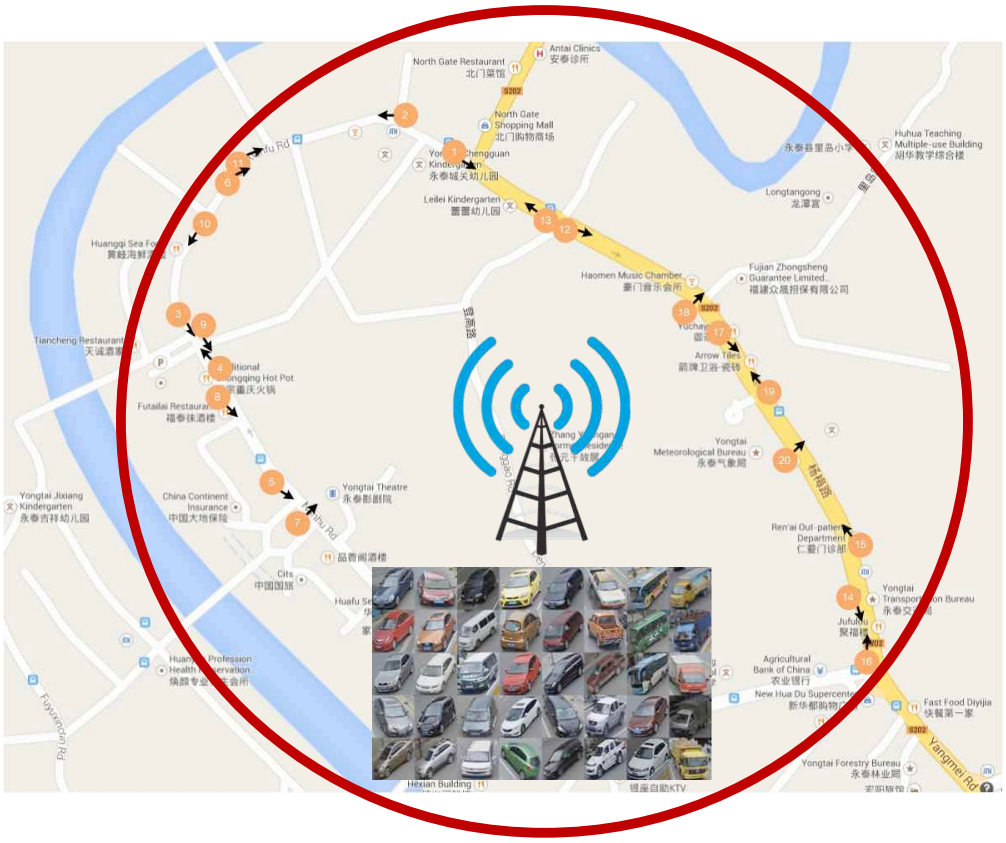

Figure 4.7: Illustration of locations and sample images of the VeRi dataset. Each orange dot represents one camera, and arrows depict orientation of them. The red circle has a radius of 700 meter showing the imaginary coverage of a cellular base station. Some sample images are shown below the base station.

small town. The location of each camera and sample images are shown in Figure 4.7. All the cameras could be covered by a base station with less than 700 meter service radius, depicted as the red circle of Figure 4.7. The dataset consists of 37,746 train images, 11,579 test images, and 1,678 query images. To our best knowledge, it is the largest open vehicle re-identification dataset in terms of the number of images and cameras, making it ideal to simulate large-scale camera analytic systems.

In Figure 4.8 we show statistics on how many different cameras capture an object. Each object is captured at least twice with a mean value of 8.3 on the test dataset and a significant number of objects are captured by almost all cameras.

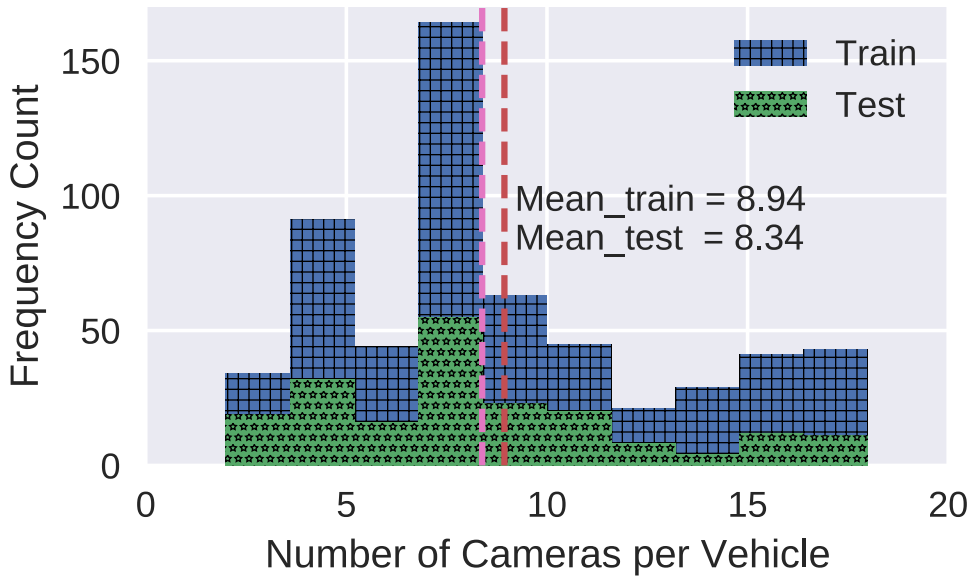

Figure 4.8: Statistics of camera occurrences per vehicle of the VeRi dataset.

Table 4.3: Comparison the size of ResNet18 and ResNet50

Item	**ResNet18**	**ResNet50**
Total Parameters(Million)	11.69	25.56
Input Size (MB)	0.57	0.57
Forward Pass Size (MB)	62.79	286.56
Parameters Size (MB)	44.59	97.49
Estimated Total Size (MB)	107.96	384.62

Re-identification model: our DNN model for re-identification is build with the FastReID [49] toolbox and PyTorch [96]. We picks two representative backbone DNN base on ResNet [48]: including ResNet18 for on-camera processing and ResNet50 for in cloud computation. A comparison between the parameter size of ResNet18 and ResNet50 is shown in Table 4.3. ResNet18 is much smaller than ResNet50, making it idea for deployment at the edge.

We train the DNN model optimizing for triplet loss with cosine similarity as the distance metric. The DNN takes three images as input when calculating triplet loss: an anchor image, a positive image that shows the same target object but from a different view-angle

or capture time, and a negative image that comes from a different object, as shown in Figure 4.9. The DNN output embeddings for each input image with the same network weight and trying to minimize the following loss function:

$$\mathcal{L}(A, P, N) = \max\left(D(f(A) - f(P)) - D(f(A) - f(N)) + \alpha, 0\right) \tag{4.1}$$

Where $f(\cdot)$ are embedding for Anchor/Positive/Negative images, α is the margin between positive and negative pairs, and $D(\cdot)$ is the distance metric function.

We use cosine similarity as the distance metric which means the cosine of the angle between embedding vectors V_1 and V_2:

$$D(V_1, V_2) = \cos(\theta) = \frac{V_1 \cdot V_2}{\|V_1\| \, \|V_2\|} \tag{4.2}$$

Different from Euclidean distance, Cosine similarity is insensitive to the magnitude of the embedding vector, leading to better performance for high dimension, sparse vectors. We illustrate the result of optimizing triplet loss with cosine similarity in Figure 4.10. The learning process tries to put the positive sample closer to the anchor while pushing the negative sample away from the anchor. After learning, we use an angular threshold to classify positive and negative samples.

Inputs to the DNN are images resized to 256*256, and the outputs are 512-D feature vectors. We use an SGD optimizer with an initial learning rate of 0.01 and a decay factor of 0.0005. We train each model for 100 epochs and the total training time is about 10 hours on four Nvidia T4 GPUs.

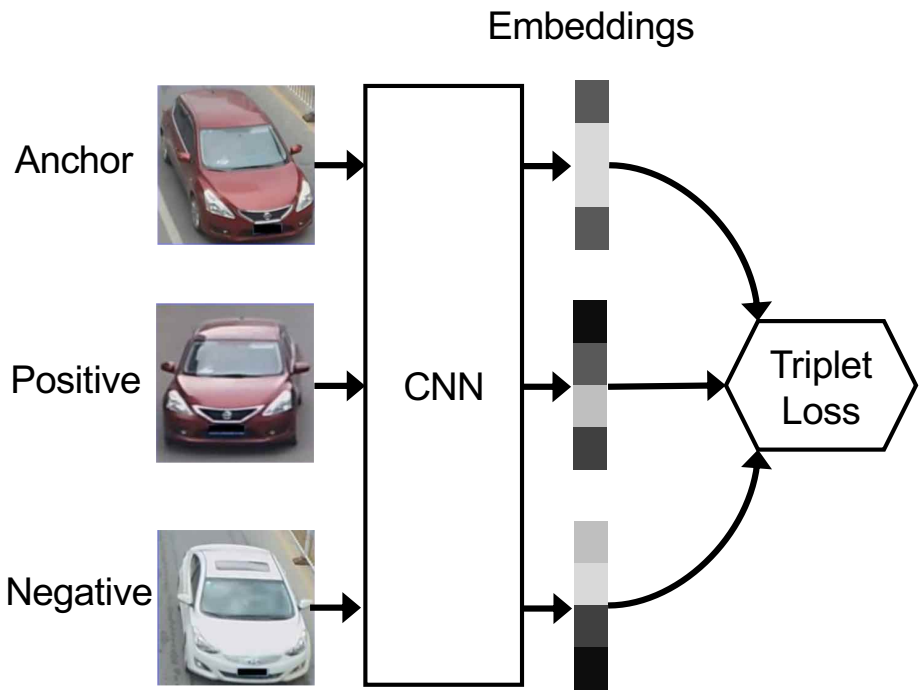

Figure 4.9: Triple loss tries to minimize the distance between anchor and positive samples (same car) while trying to maximize the distance between anchor and negative samples (different car).

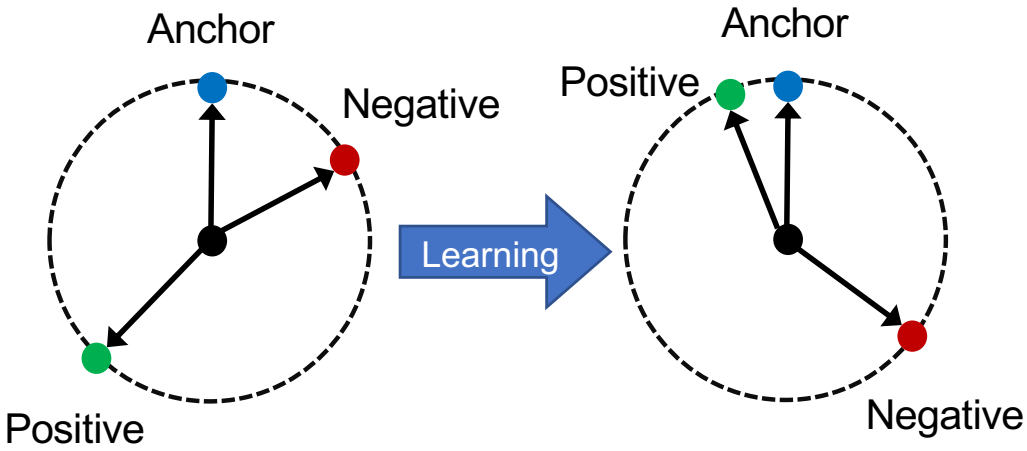

Figure 4.10: Illustration of optimizing cosine similarity for triplets. After the learning process, the angle between anchor and positive is much smaller while the angle between anchor and negative is much larger.

4.4 YOUO benchmark results

We present trace-driven benchmark results of YOUO in this section, include comparing the accuracy of local and cloud models and performance benchmark with the re-identification and object counting tasks. We also test the scalability of YOUO with a simulated dataset that contains more cameras.

4.4.1 Comparison of local and cloud models

We benchmark the accuracy of trained local and cloud DNN model with two common tasks:

- **Binary classification**: we test the DNN's ability to distinguishing between positive and negative samples by randomly sampling two images from the test dataset and calculate the cosine distance between them. The classifier outputs whether two images are from the same vehicle or not by comparing the distance with a threshold. We vary the threshold to achieve different trade-offs between True Positive Rate and False Positive Rate. We use a metric called AUC (Area Under Curve) to evaluate how good the model is. A good classifier will have an AUC very close to 1, which means a very high True Positive Rate while keeping the False Positive Rate low. The result of the local (ResNet18) and cloud (ResNet50) DNN model is shown in Figure 4.11. We plot the x-axis in log scale to magnify the difference. The AUC scores are 0.992 v.s. 0.995, meaning that ResNet18 is almost as good as ResNet50.

- **Re-identification task**: we run the re-identification task on the query dataset. For each query input, the model tries to find k closest images from the test dataset. We define a query as true if any of the k images shows the same vehicle as the query input. We show the top-k (1 to 9) query accuracy of the local (ResNet18) and cloud (ResNet50) in Figure 4.12. We again magnify the difference by setting the range of the y-axis as [0.955, 0.990]. The result indicates that the query accuracy between the local and cloud mode is only 0.5% to 1.5% across the range.

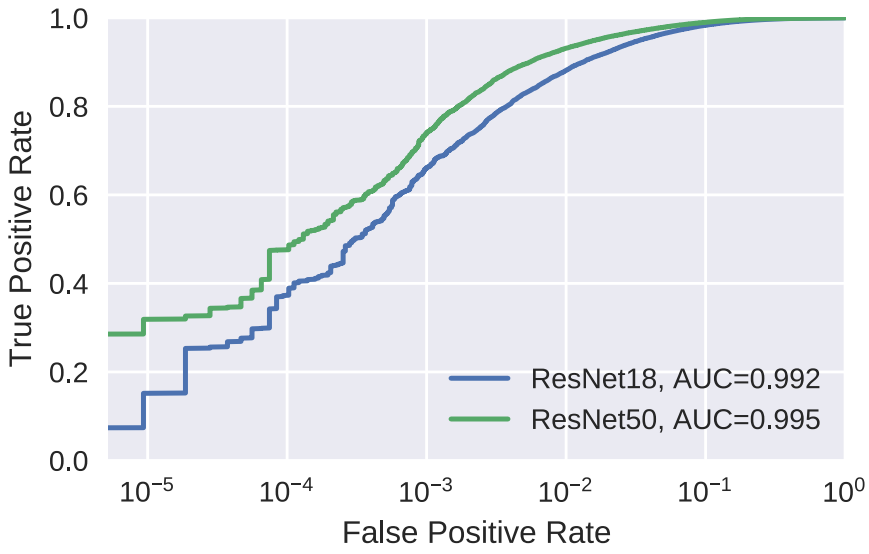

Figure 4.11: AUC metric for local (ResNet18) and cloud (ResNet50) model in the binary classification task. Notice that we magnify the difference by plotting the x-axis in log scale. Both model has similar AUC score meaning the performance gap between them is very small.

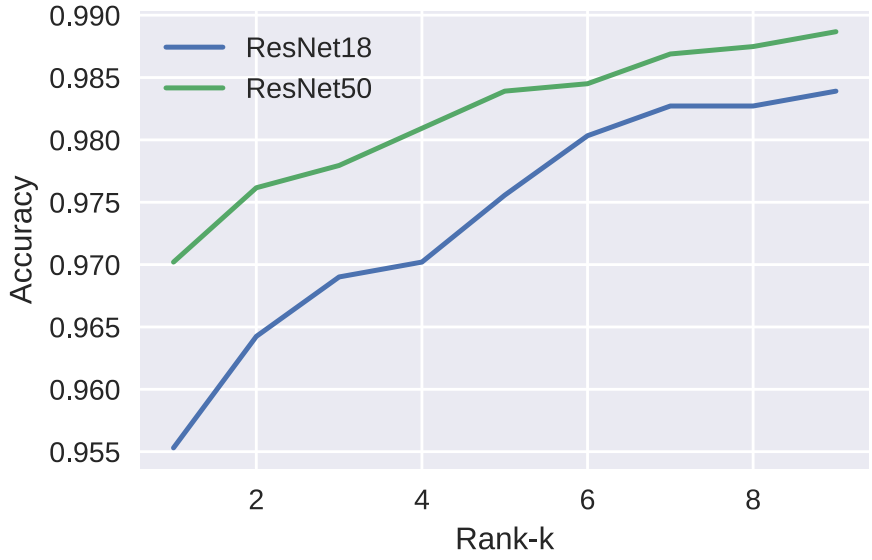

Figure 4.12: Rank-k re-identification accuracy for local (ResNet18) and Cloud (ResNet50) model. The gap between them is only 0.5% to 1.5%.

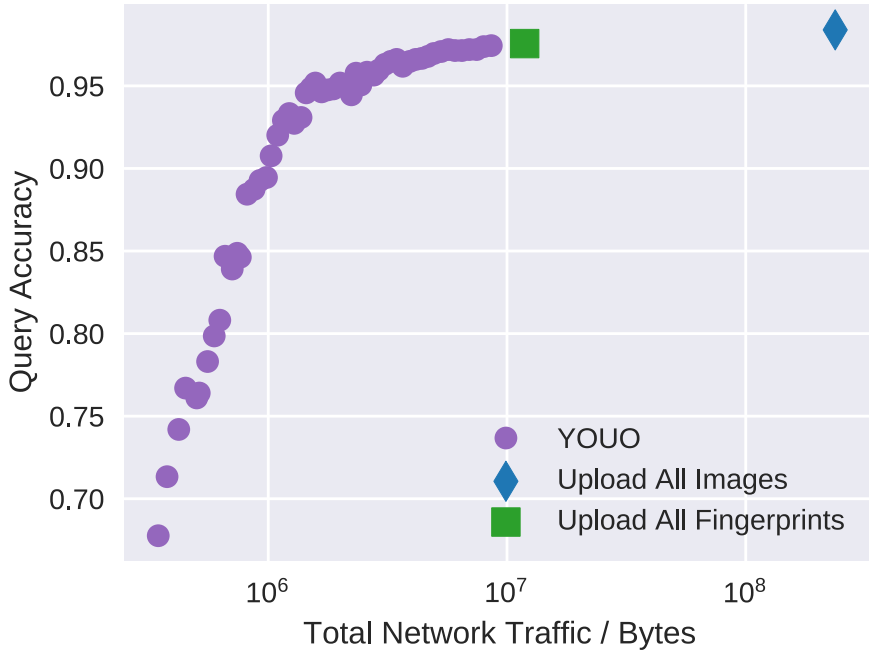

Figure 4.13: Re-identification task performance of YOUO v.s. upload all images/fingerprints. Uploading all images lead to the highest query accuracy but costs more than one order of magnitude of network bandwidth. YOUO could achieve similar performance as upload all fingerprints while providing a graceful trade-off between query accuracy and network traffic.

4.4.2 YOUO Performance Benchmark

ReID Task Benchmark: we compare the Re-identification task performance of YOUO against uploading all images/fingerprints in this section. The performance metrics are total network traffic needed to run all queries in the query dataset and top-5 query accuracy.

The result is shown in Figure 4.13. The X-axis is total network traffic in Bytes in and the Y axis is query accuracy. The difference in network traffic usage is so large that we have to plot X-axis in log scale. The result indicates that both YOUO and upload all fingerprints use more than one order of magnitude less bandwidth than upload all images while achieving similar query accuracy. YOUO provides a graceful performance trade-off for bandwidth saving by adjusting the distance threshold for uploading.

We further zoom in to analyze the network traffic and query accuracy trade-off of

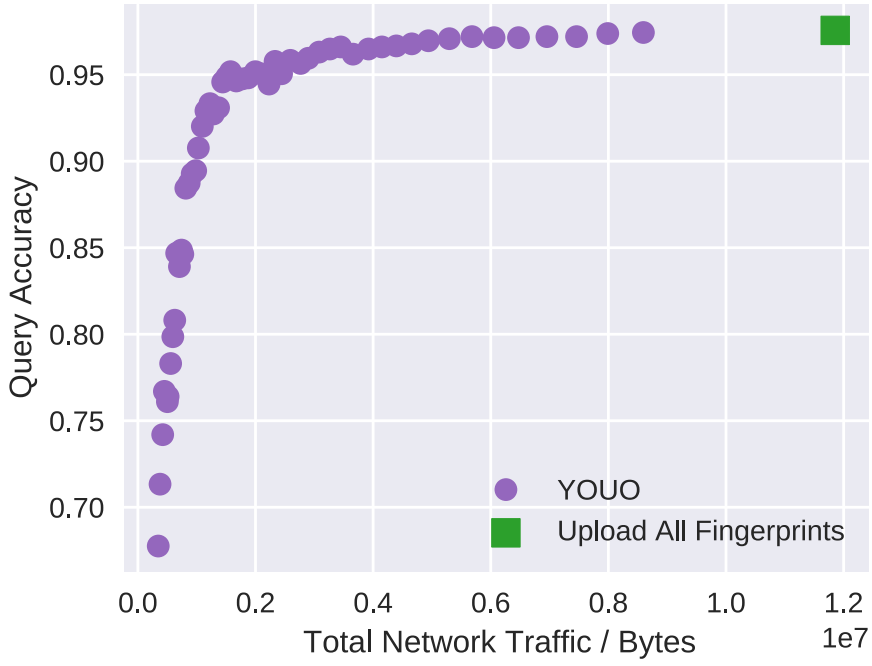

Figure 4.14: Re-identification task performance of YOUO v.s. upload all fingerprints. YOUO is more than six times as efficient in network usage with slight query accuracy loss (3%).

YOUO. This time we compare against uploading all fingerprints only with a linear x-axis, as shown in Figure 4.14. We can find YOUO is more than six times as efficient in bandwidth usage with slight query accuracy loss, indicating huge savings of network cost.

We analyze the reason for the performance gain by investigating the average number of uploads per query object for YOUO. The result is shown in Figure 4.15. The result indicates that at least 8.4 uploads are required without cross-camera collaboration, while YOUO requires only 3 to 4 uploads with similar accuracy. YOUO could still maintain a query accuracy of 0.67 when performing only one upload per object, which is a very challenging task as the camera may capture objects from different view angles.

Object Counting Task Benchmark: we also benchmark the performance of YOUO on the object counting task that tries to count how many unique objects on the query dataset. We use the same metric definition as the binary classification task in section 4.4.1.

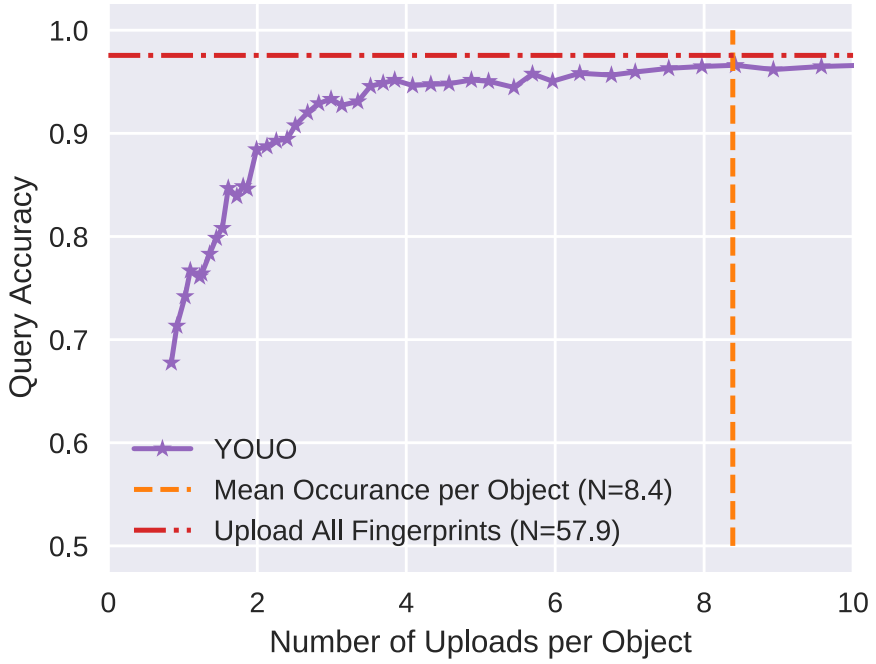

Figure 4.15: Number of uploads per object for YOUO with different upload distance threshold. The horizontal dot-dash line shows accuracy for uploading all fingerprints and vertical dash line shows the mean occurrence per object. YOUO requires only a few uploads per object while providing very similar accuracy.

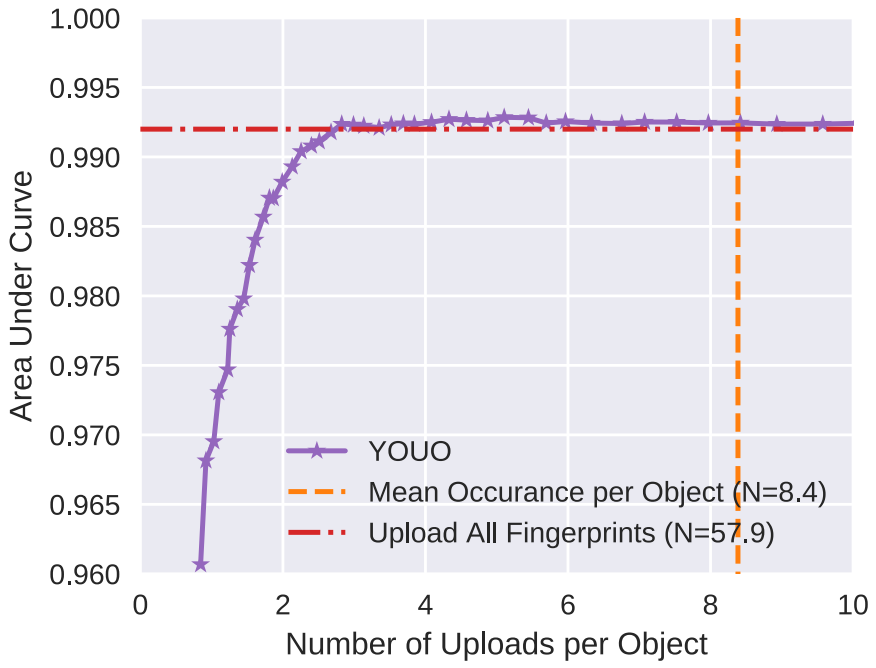

Figure 4.16: Object counting benchmark for YOUO. The horizontal dot-dash line shows accuracy for uploading all fingerprints and vertical dash line shows the mean occurrence per object. YOUO requires only a few uploads per object while providing similar (or even better) accuracy in this task.

YOUO also performs well in counting how many different objects, as shown in Figure 4.16. On the x-axis is the number of uploads per object and y-axis is the area under curve of the receiver operating characteristics plot that balances true positive rate and false-positive rate. YOUO achieves an area under curve of 0.96 in the receiver operating characteristic plot while uploading only once, which is more than 8 times as efficient as the optimal case without collaboration for the counting task.

Both benchmark results show YOUO is much more efficient than non-collaborative methods, confirming the analysis in Table 4.2.

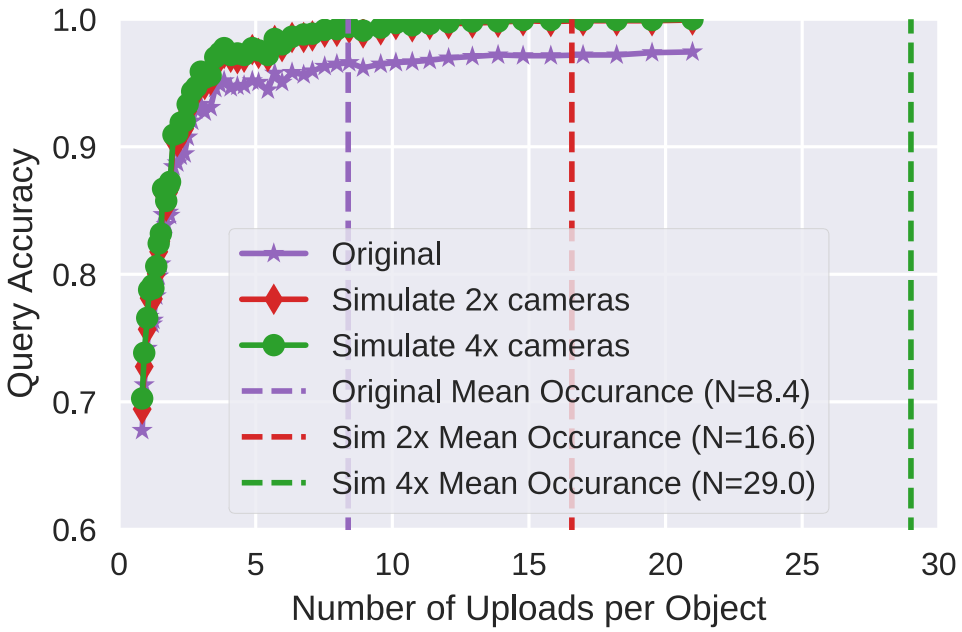

Figure 4.17: Re-identification task performance of YOUO when simulating more Cameras. Dashed vertical lines shows the mean occurrence per object for the original, 2x and 4x split dataset. The number of uploads per object of YOUO does not increase as more and more cameras capture the same object.

4.4.3 Scalability of YOUO

Notice that the network cost of YOUO depends on the number of objects only, rather than the number of occurrences or number of cameras, which means YOUO could work efficiently in a large-scale camera network as the same object may be captured by a huge number of cameras.

In this section, we simulate more cameras by partition images from the same camera into two and four virtual cameras respectively. We perform the re-identification task on this synthesized dataset as in the previous section. The result is shown in Figure 4.17. The experiment result indicates that the number of uploads is mostly the same for YOUO when there are more cameras. The results confirm the scalability of YOUO, especially for dense camera deployment.

4.5 Discussion

Limitations of YOUO: YOUO is built on top of wireless cameras that each of them is capable of running a small DNN, which may be a small fraction of all existing cameras. However, we believe that more and more wireless smart cameras will be deployed as wireless connections simplify the installation process, and running a DNN on camera could avoid flooding the network and storage with redundant information.

In addition, YOUO is a trace-driven simulation on a dataset with 20 cameras. We believe a larger dataset or a real-world deployment could make YOUO more convincing:

- **Larger dataset**: the VeRi dataset contains over 50,000 images captured by 20 cameras covering an 1.0 km^2 area. While in the real world, the camera density could be as high as 657 cameras per square km with more than 1 million cameras per city[2]. Cross-camera collaboration and on-camera filtering are crucial to reducing the network and computation load of the overall system, and we expect a much larger performance gap between YOUO and baseline methods on such datasets.

- **Real-world deployment of YOUO**: in this paper, we approximate the cost of multicast to unicast, and ignoring various overheads associated with transmission/retransmission; it is also unknown how does YOUO work under various light conditions and weather situations. A real-world deployment, even with only a few cameras, could help to benchmark the performance of YOUO in the wild.

Future work: we believe there is great potential for better collaboration across-cameras analytics, including the adoption of Device-to-Device (D2D) communication and leveraging spatio-temporal correlations:

- **Device-to-Device (D2D)**: both LTE and 5G support direct communication between two nearby cellular network users without traversing the cellular base station [18, 20]. The transmission power of D2D is usually lower to minimize interference. It is also possible to equip cameras with short-range radios like WiFi. D2D communication enables sharing of information (or even computation) between neighboring cameras

[2]https://www.visualcapitalist.com/mapped-the-top-surveillance-cities-worldwide/

at a low cost of the cellular network. Comparing the performance and cost of sharing information locally v.s. cellular-wide or a combination of them could lead to a more efficient design of camera analytic system.

- **Leveraging spatio-temporal correlations**: cameras could be correlated in multiple ways including spatially and temporally. Spatial correction means geographical association between cameras and temporal correlation represents the association between cameras over time. It is possible to build a connectivity graph model to predict spatio-temporal profiles and reduce computation and network load of the camera analytic system [58].

4.6 Conclusion

We present YOUO, a cross-camera analytic system that leverages the broadcast insight of cellular links. YOUO broadcast DNN feature information to the other cameras in the network for efficient collaboration across cameras. YOUO is 3x as efficient in bandwidth when compared with optimal single-camera filtering with minimal accuracy loss while achieving 70%+ accuracy when uploading only once per object (the absolute minimum) for the entire camera network. Experiment on simulated dataset shows YOUO is a scalable architecture for large-scale camera analytic systems.

Chapter 5

Conclusion

This dissertation has provided two examples, STARFISH and YOUO, to demonstrate how integrating characteristics of the underlying wireless link could benefit the design of camera analytic systems. In STARFISH we leverage the lossy link characteristic, builds a resilient image compression framework designed for LPWAN that processes all the information loss in the application layer, thus simplifying wireless protocol design and improves task performance. In YOUO we leverage the broadcast characteristic that enables efficient collaboration across cameras, resulting in a scalable architecture for large-scale camera analytic systems.

Future Research Directions: we just scratched the surface of co-designing AI systems and the wireless link. We envision the following wireless characteristics should also be considered for a better design:

- **Dynamic:** wireless links are inherently dynamic due to noise, fading, interference, and mobility. How to ensure stable AI system performance, especially those with delay constraints, under the dynamic link? Can we cache prior knowledge at throughput peaks and use prior knowledge to improve performance during the low bandwidth period?

- **High bandwidth:** some wireless links, such as mmWave and light-based communication are capable to provide high bandwidth at a low cost. How does the AI system

leverage the abundant bandwidth? Would sharing computation and intermediate results of DNN over such links lead to better system design?

- **Heterogeneity**: more and more devices are equipped with more than one kind of wireless link such as WiFi, 5G, Bluetooth, or even satellite. How does the link heterogeneity affect the design of AI systems, for example, distributed federated learning?

It is thrilling to live in the era of 5G connections that provides unprecedented speed, along with thousands of communication satellite orbiting the earth. On the other hand, AI is also evolving fast that surpasses human performance on various tasks. We thus encourage researchers to explore exciting opportunities at the crossroad of wireless and AI, for camera analytic systems and beyond.

Bibliography

[1] Automated customer analytics for retail stores. https://www.crowdvision.com/solutions-retail/. Accessed: 2021-05-01.

[2] Better portable graphics. https://bellard.org/bpg/.

[3] Building a new future: Transforming australia's construction industry with digital technologies.

[4] Cartoon set image dataset. https://google.github.io/cartoonset/download.html.

[5] dahuffman. https://github.com/soxofaan/dahuffman.

[6] jpeg2png: Silky smooth jpeg decoding - no more artifacts! https://github.com/victorvde/jpeg2png. Accessed: 2021-05-01.

[7] Maxipy. https://maixpy.sipeed.com/en/get$_started/maixpyide.html$.

[8] Multi-class weather dataset. https://www.kaggle.com/vijaygiitk/multiclass-weather-dataset.

[9] Nncase compiler. https://github.com/kendryte/nncase.

[10] Perceptual jpeg encoder. https://github.com/google/guetzli.

[11] Pillow: Python imaging library (fork). https://https://github.com/python-pillow/Pillow. Accessed: 2021-05-01.

[12] Size of progressive jpeg. https://superuser.com/questions/379404/what-is-the-difference-between-progressive-and-optimized-jpegs-in-photostop.

[13] Tensorflow lite micro magic wand example. https://github.com/tensorflow/tensorflow/tree/master/ tensorflow/lite/micro/examples/magic_wand.

[14] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[15] Siripuram Aditya and Sachin Katti. Flexcast: Graceful wireless video streaming. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, pages 277–288, 2011.

[16] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 221–231, 2019.

[17] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *computer*, 50(10):58–67, 2017.

[18] Rafay Iqbal Ansari, Chrysostomos Chrysostomou, Syed Ali Hassan, Mohsen Guizani, Shahid Mumtaz, Jonathan Rodriguez, and Joel JPC Rodrigues. 5g d2d networks: Techniques, challenges, and future prospects. *IEEE Systems Journal*, 12(4):3970–3984, 2017.

[19] Giuseppe Araniti, Massimo Condoluci, Pasquale Scopelliti, Antonella Molinaro, and Antonio Iera. Multicasting over emerging 5g networks: Challenges and perspectives. *Ieee network*, 31(2):80–89, 2017.

[20] Arash Asadi, Qing Wang, and Vincenzo Mancuso. A survey on device-to-device communication in cellular networks. *IEEE Communications Surveys & Tutorials*, 16(4):1801–1819, 2014.

[21] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[22] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[23] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018.

[24] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems*, 3, 2021.

[25] Jim Bankoski, Paul Wilkins, and Yaowu Xu. Technical overview of vp8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*, pages 1–6. IEEE, 2011.

[26] Carsten Bockelmann, Nuno Pratas, Hosein Nikopour, Kelvin Au, Tommy Svensson, Cedomir Stefanovic, Petar Popovski, and Armin Dekorsy. Massive machine-type communications in 5g: Physical and mac-layer solutions. *IEEE Communications Magazine*, 54(9):59–65, 2016.

[27] Martin C Bor, Utz Roedig, Thiemo Voigt, and Juan M Alonso. Do lora low-power wide-area networks scale? In *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 59–67, 2016.

[28] Ozgun Y Bursalioglu, Giuseppe Caire, and Dariush Divsalar. Joint source-channel coding for deep-space image transmission using rateless codes. *IEEE Transactions on Communications*, 61(8):3448–3461, 2013.

[29] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[30] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[31] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya R Dulloor. Scaling video analytics on constrained edge nodes. *arXiv preprint arXiv:1905.13536*, 2019.

[32] Tong Chen, Haojie Liu, Qiu Shen, Tao Yue, Xun Cao, and Zhan Ma. Deepcoder: A deep neural network based video compression. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2017.

[33] Sandeep P Chinchali, Eyal Cidon, Evgenya Pergament, Tianshu Chu, and Sachin Katti. Neural networks meet physical networks: Distributed inference between edge devices and the cloud. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2018.

[34] François Chollet et al. Keras. https://keras.io, 2015.

[35] Jaegul Choo and Shixia Liu. Visual analytics for explainable deep learning. *IEEE computer graphics and applications*, 38(4):84–92, 2018.

[36] Lee D Davisson. Rate-distortion theory and application. *Proceedings of the IEEE*, 60(7):800–808, 1972.

[37] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[38] Sharaf E Elnahas, Kou-Hu Tzou, Jerome R Cox, Rexford L Hill, and R Gilbert Jost. Progressive coding and transmission of digital diagnostic pictures. *IEEE transactions on medical imaging*, 5(2):73–83, 1986.

[39] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.

[40] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[41] John Emmons, Sadjad Fouladi, Ganesh Ananthanarayanan, Shivaram Venkataraman, Silvio Savarese, and Keith Winstein. Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 27–32, 2019.

[42] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Pattern Recognition Workshop*, 2004.

[43] Branden Ghena, Joshua Adkins, Longfei Shangguan, Kyle Jamieson, Philip Levis, and Prabal Dutta. Challenge: Unlicensed lpwans are not yet the path to ubiquitous connectivity. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–12, 2019.

[44] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, pages 1–31, 2021.

[45] Douglas Gray and Hai Tao. Viewpoint invariant pedestrian recognition with an ensemble of localized features. In *European conference on computer vision*, pages 262–275. Springer, 2008.

[46] Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7033–7042, 2019.

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

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[49] Lingxiao He, Xingyu Liao, Wu Liu, Xinchen Liu, Peng Cheng, and Tao Mei. Fastreid: A pytorch toolbox for general instance re-identification. *arXiv preprint arXiv:2006.02631*, 2020.

[50] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

[51] 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. *arXiv preprint arXiv:1704.04861*, 2017.

[52] Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. Starfish: resilient image compression for aiot cameras. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 395–408, 2020.

[53] Pan Hu, Rakesh Misra, and Sachin Katti. Dejavu: Enhancing videoconferencing with prior knowledge. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 63–68, 2019.

[54] Pan Hu, Pengyu Zhang, and Deepak Ganesan. Laissez-faire: Fully asymmetric backscatter communication. *ACM SIGCOMM computer communication review*, 45(4):255–267, 2015.

[55] Pan Hu, Pengyu Zhang, Mohammad Rostami, and Deepak Ganesan. Braidio: An integrated active-passive radio for mobile devices with asymmetric energy budgets. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 384–397, 2016.

[56] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision

weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[57] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[58] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. Spatula: Efficient cross-camera video analytics on large camera networks. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 110–124. IEEE, 2020.

[59] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[60] Junchen Jiang, Yuhao Zhou, Ganesh Ananthanarayanan, Yuanchao Shu, and Andrew A Chien. Networked cameras are the new big data clusters. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 1–7, 2019.

[61] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[62] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[63] Giannis Kazdaridis, Stratos Keranidis, Polychronis Symeonidis, Panagiotis Tzimotoudis, Ioannis Zographopoulos, Panagiotis Skrimponis, and Thanasis Korakis. Evaluation of lora performance in a city-wide testbed: Experimentation insights and findings. In *Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, pages 29–36, 2019.

[64] Aaron Kiely and Matthew Klimesh. The icer progressive wavelet image compressor. *IPN Progress Report*, 42(155):1–46, 2003.

[65] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. 2016.

[66] Victoria Kostina, Yury Polyanskiy, and Sergio Verd. Joint source-channel coding with feedback. *IEEE Transactions on Information Theory*, 63(6):3502–3515, 2017.

[67] Victoria Kostina and Sergio Verdú. Lossy joint source-channel coding in the finite blocklength regime. *IEEE Transactions on Information Theory*, 59(5):2545–2575, 2013.

[68] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[70] Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[71] Jooyoung Lee, Seunghyun Cho, and Seung-Kwon Beack. Context-adaptive entropy model for end-to-end optimized image compression. *arXiv preprint arXiv:1809.10452*, 2018.

[72] Wei-Cheng Lee, David Alexandre, Chih-Peng Chang, Wen-Hsiao Peng, Cheng-Yen Yang, and Hsueh-Ming Hang. Learned image compression with residual coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[73] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[74] Mu Li, Wangmeng Zuo, Shuhang Gu, Debin Zhao, and David Zhang. Learning convolutional networks for content-weighted image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3214–3223, 2018.

[75] Wei Li, Rui Zhao, Tong Xiao, and Xiaogang Wang. Deepreid: Deep filter pairing neural network for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 152–159, 2014.

[76] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 359–376, 2020.

[77] Xiaochen Liu, Pradipta Ghosh, Oytun Ulutan, BS Manjunath, Kevin Chan, and Ramesh Govindan. Caesar: cross-camera complex activity recognition. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 232–244, 2019.

[78] Xinchen Liu, Wu Liu, Huadong Ma, and Huiyuan Fu. Large-scale vehicle re-identification in urban surveillance videos. In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2016.

[79] Xinchen Liu, Wu Liu, Tao Mei, and Huadong Ma. A deep learning-based approach to progressive vehicle re-identification for urban surveillance. In *European conference on computer vision*, pages 869–884. Springer, 2016.

[80] Xinchen Liu, Wu Liu, Tao Mei, and Huadong Ma. Provid: Progressive and multimodal vehicle reidentification for large-scale urban surveillance. *IEEE Transactions on Multimedia*, 20(3):645–658, 2017.

[81] Zihao Liu, Tao Liu, Wujie Wen, Lei Jiang, Jie Xu, Yanzhi Wang, and Gang Quan. Deepn-jpeg: a deep neural network favorable jpeg-based image compression framework. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[82] Franz Loewenherz, Victor Bahl, and Yinhai Wang. Video analytics towards vision zero. *Institute of Transportation Engineers. ITE Journal*, 87(3):25, 2017.

[83] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

[84] Michael Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 271–271. IEEE Computer Society, 2002.

[85] Paul J Marcelis, Vijay S Rao, and R Venkatesha Prasad. Dare: Data recovery through application layer coding for lorawan. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 97–108. IEEE, 2017.

[86] Michael W Marcellin, Michael J Gormish, Ali Bilgin, and Martin P Boliek. An overview of jpeg-2000. In *Proceedings DCC 2000. Data Compression Conference*, pages 523–541. IEEE, 2000.

[87] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 21–26, 2019.

[88] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. *ICT express*, 5(1):1–7, 2019.

[89] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4394–4402, 2018.

[90] Alexander E Mohr, Eve A Riskin, and Richard E Ladner. Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction. *IEEE journal on selected areas in communications*, 18(6):819–828, 2000.

[91] Robert Moriarty, Kathy O'Connell, Nicolaas Smit, Andy Noronha, and Joel Barbier. A new reality for oil and gas, 2015.

[92] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[93] Shadi A Noghabi, Landon Cox, Sharad Agarwal, and Ganesh Ananthanarayanan. The emerging landscape of edge computing. *GetMobile: Mobile Computing and Communications*, 23(4):11–20, 2020.

[94] Ken'ichiro Ogura, Akio Miyazaki, and Yoshihiko Akaiwa. An error resilient still image transmission system for mobile radio communication. In *1999 IEEE 49th Vehicular Technology Conference (Cat. No. 99CH36363)*, volume 3, pages 2004–2008. IEEE, 1999.

[95] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[96] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[97] Juha Petäjäjärvi, Konstantin Mikhaylov, Rumana Yasmin, Matti Hämäläinen, and Jari Iinatti. Evaluation of lora lpwan technology for indoor remote health and wellbeing monitoring. *International Journal of Wireless Information Networks*, 24(2):153–165, 2017.

[98] Majid Rabbani. Jpeg2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging*, 11(2):286, 2002.

[99] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[100] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[101] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[102] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[103] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[104] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2922–2930. JMLR. org, 2017.

[105] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G Anderson, and Lubomir Bourdev. Learned video compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3454–3463, 2019.

[106] Frederic Runge, Danny Stoll, Stefan Falkner, and Frank Hutter. Learning to design rna. *arXiv preprint arXiv:1812.11951*, 2018.

[107] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[108] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[109] P Greg Sherwood and Kenneth Zeger. Progressive image coding on noisy channels. In *Proceedings DCC'97. Data Compression Conference*, pages 72–81. IEEE, 1997.

[110] Amin Shokrollahi. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.

[111] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *arXiv preprint arXiv:1406.2199*, 2014.

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

[113] David Slepian and Jack Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on information Theory*, 19(4):471–480, 1973.

[114] Jothi Prasanna Shanmuga Sundaram, Wan Du, and Zhiwei Zhao. A survey on lora networking: Research problems, current solutions, and open issues. *IEEE Communications Surveys & Tutorials*, 22(1):371–388, 2019.

[115] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[116] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[117] The TensorFlow Team. Flowers, jan 2019.

[118] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314, 2017.

[119] Shuai Tong, Jiliang Wang, and Yunhao Liu. Combating packet collisions using non-stationary signal scaling in lpwans. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 234–246, 2020.

[120] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. Farmbeats: An iot platform for data-driven agriculture. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 515–529, 2017.

[121] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[122] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[123] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao-Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 159–173. IEEE, 2018.

[124] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[125] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[126] Pete Warden and Daniel Situnayake. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. " O'Reilly Media, Inc.", 2019.

[127] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[128] Xianjin Xia, Yuanqing Zheng, and Tao Gu. Ftrack: Parallel decoding for lora transmissions. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 192–204, 2019.

[129] Ning Xie, Gabrielle Ras, Marcel van Gerven, and Derek Doran. Explainable deep learning: A field guide for the uninitiated. *arXiv preprint arXiv:2004.14545*, 2020.

[130] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[131] Xiufeng Xie and Kyu-Han Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[132] Zixiang Xiong, Angelos D Liveris, and Samuel Cheng. Distributed source coding for sensor networks. *IEEE signal processing magazine*, 21(5):80–94, 2004.

[133] Yuanlu Xu, Bingpeng Ma, Rui Huang, and Liang Lin. Person search in a scene by jointly modeling people commonness and person uniqueness. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 937–940, 2014.

[134] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Deep metric learning for person re-identification. In *2014 22nd International Conference on Pattern Recognition*, pages 34–39. IEEE, 2014.

[135] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[136] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

[137] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128*, 2017.

[138] Rui Zhao, Wanli Ouyang, and Xiaogang Wang. Unsupervised salience learning for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3586–3593, 2013.

[139] Liang Zheng, Yi Yang, and Alexander G Hauptmann. Person re-identification: Past, present and future. *arXiv preprint arXiv:1610.02984*, 2016.

[140] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.