Starfish: Resilient Image Compression for AIoT Cameras

Pan Hu
panhu@stanford.edu
Stanford University

Junha Im
junhaim@stanford.edu
Stanford University, Samsung Electronics

Zain Asgar
zasgar@stanford.edu
Stanford University

Sachin Katti
skatti@stanford.edu
Stanford University

ABSTRACT

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 optimized for 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 a $3 AIoT accelerators but also outperforms JPEG, a well-established baseline, by up to 3×, in terms of bandwidth efficiency and up to 2.5× as efficient in energy consumption. It also features graceful and gradual performance degradation in the presence of packet loss. The application-level simulation indicates that Starfish could deliver 3.7× images while providing better image quality.

CCS CONCEPTS

• **Computer systems organization → Neural networks**; • **Networks → Error detection and error correction**; **Cyber-physical networks**.

KEYWORDS

artificial intelligence, internet of things, compression, resilient

ACM Reference Format:
Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. 2018. Starfish: Resilient Image Compression for AIoT Cameras. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/1122445.1122456

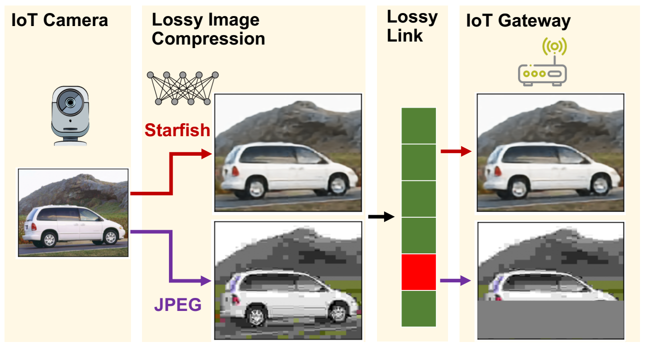


Figure 1: Illustration of uploading image from IoT camera to Gateway with Starfish and conventional JPEG-based methods. DNN-based Starfish is resilient to packet loss and results in better image quality.

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 [51]. Cities and construction sites use cameras for intelligent transportation and monitoring to increase safety and security [2, 10, 44]. Oil refineries and farms use cameras to perform predictive maintenance and irrigation [49, 66]. The camera, being one of the most information-rich and versatile sensors maximizes it's 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 LP-WAN(Low Power Wide Area Network) 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.

Studies show that reducing the packet error rate by 20% reduces network capacity by 5.2 times [75] in LoRaWAN.

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; 3) 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, as shown in Figure 1. 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. DNN does not need the header in which data loss could be fatal. 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. 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 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 a $3 AIoT accelerator, 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 simplifying wireless protocol design, improves the network throughput and battery life of IoT cameras.
- To our best knowledge, Starfish is the first DNN-based compression framework that runs efficiently on low-cost AIoT devices. We generate DNN configuration with NAS automatically, making it future-proof and generalizable to a diverse set of DNN architectures and AIoT hardware.
- Benchmarks on a large-scale image dataset and IoT links suggest that Starfish is up to 3~4× as efficient in compression size, and up to 2.5× as efficient in terms of time and energy efficiency for lossless traffic, due to the task-awareness and content-awareness of Starfish. Simulation of 100 to 1000 nodes suggests Starfish could deliver more than 3.7× images with better image quality in lossy traffic scenarios.

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 [27], 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 [15]. This scarcity in network capacity 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 [54, 62, 65]. Also, the study [46] shows that frequent retransmission is unrealistic due to the limited capacity of acknowledgments at the gateway, increased energy

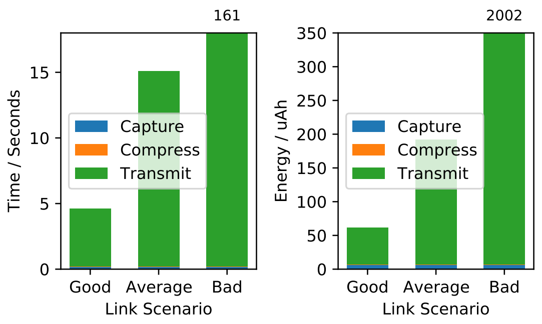
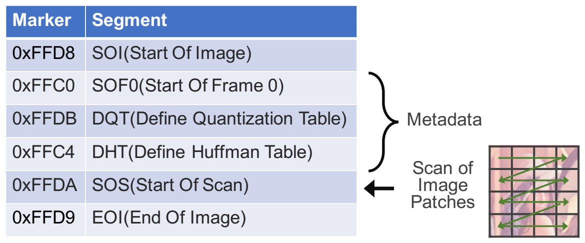


Figure 2: Time and energy consumption breakdown for image capture, compression, and transmission over LoRaWAN. The time and energy spent on compression are barely visible because JPEG compression is hardware-accelerated and takes 16ms. Transmission over LoRaWAN dominates time/energy consumption in all scenarios and exceeds the range for both charts when the link condition is poor.

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 [34], thus drastically increase the energy efficiency of LoRaWAN nodes and total link capacity, as shown in Table 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 airtime and energy consumption by half, thus double the total link capacity.

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 2. 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 for LPWAN. Although many lossy image codecs have been proposed like JPEG [68], JPEG2000 [55], WebP [14] and BPG [1], IoT devices tend to support JPEG only due to its simplicity and low resource requirement. Similarly, JPEG-compatible encoders like Guetzli [6] 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 [71], as well as the fact that they send at a very low frame rate so the gain from inter-frame prediction may be limited.



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

JPEG compression consists of three steps: perform DCT (Discrete Cosine Transform) on patches of the input image, quantize DCT coefficients according to the quantization table determined from compression quality parameter, 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. 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:

- **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**: standard JPEG encoder cannot 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.

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

Scenario	Bitrate (bps)		Payload / Packet Size (Bytes)		Air-time (seconds) / Image[2]		TX Energy (uAh) / Image	
	Baseline	Lossy	Baseline	Lossy	Baseline	Lossy	Baseline	Lossy
Good	5,469	10,938[3]	242 / 255	242 / 255	4.40	2.20	55.61	27.76
Average	1,758	3,125	115 / 128	242 / 255	14.89	7.78	188.25	97.16
Bad	245	448	51 / 64	51 / 64	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] Assuming an image size of 2.5kB.
[3] Bandwidth=250kHz

Table 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[8]/PyTorch[52, 53]	TensorFlow Lite[8]	TensorFlow Lite Micro[8]

- **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 [29]. 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 [77] and gesture recognition with accelerometer [7], 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 4.

We compare the specs of different AI platforms to better understand the performance of AI accelerators for IoT as shown in Table 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 §3.1.

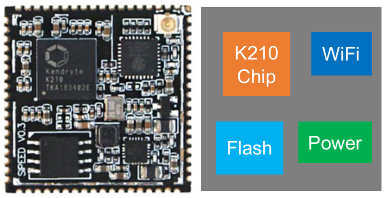


Figure 4: An \$8 K210 Kendryte AI accelerator module. The corresponding system layout is shown on the right. It includes K210 chip (with RISC-V CPU and AI inference accelerator), an ESP8266 WiFi module, 16MB Flash storage and power management chip.

3 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.

3.1 Design of Compression DNN

To perform DNN-based image compression, we typically use a a DNN encoder **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 **Q** that maps floating point values into low-precision integers $\mathbb{Z}^N$. They are further compressed with an entropy encoder **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 **D** to output recovered image $\mathbb{I}'^{(H,W,C)}$. The entire process is shown in Equation 1:

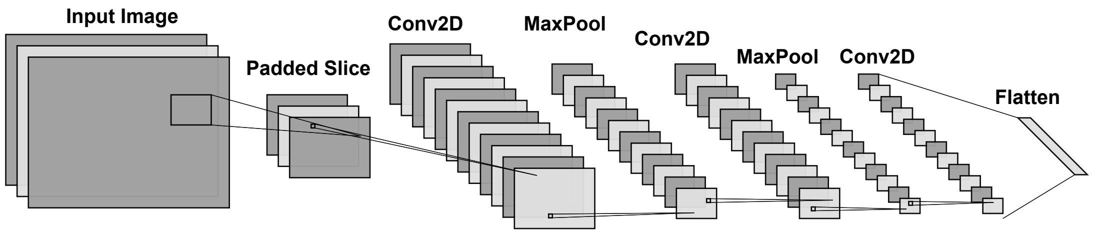


Figure 5: Architecture of encoder DNN. Images are optionally sliced into patches, then zero-padded as quality degrades on edges of decoded images. Padded patches are processed multiple stages of 2D convolution (stride=1 with 3*3 kernel) followed by 2*2 max-pooling layer to reduce spatial resolution, then flattened to generate binary representation.

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

Design and implement DNN on AIoT accelerators is not as straight-forward as on general purpose GPU. As we've shown in Table 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[1]. 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 5. 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 [50] activation for most layers in encoder and decoder DNN, except: 1) `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 [13].

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 2:

[1]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.

Table 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

$$\begin{aligned} N_{activation} &= h \times w \times C_{out} \\ N_{weights} &= k \times k \times C_{in} \times C_{out} + C_{out} \end{aligned} \qquad (2)$$

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× as original configuration.

3.1.1 *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. 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[2], so the cost of searching through all possible combinations of 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.

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 6, 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.

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

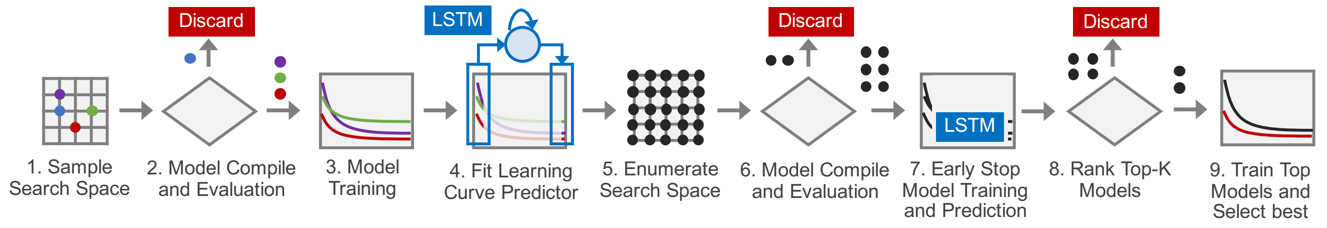

Discard
LSTM
Discard
LSTM
Discard
1. Sample Search Space
2. Model Compile and Evaluation
3. Model Training
4. Fit Learning Curve Predictor
5. Enumerate Search Space
6. Model Compile and Evaluation
7. Early Stop Model Training and Prediction
8. Rank Top-K Models
9. Train Top Models and Select best

Figure 6: Workflow of hardware-aware NAS with learning curve extrapolation.

Table 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%

(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 7. A more detailed mini-benchmark is shown in Table 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× 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.

3.1.2 *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

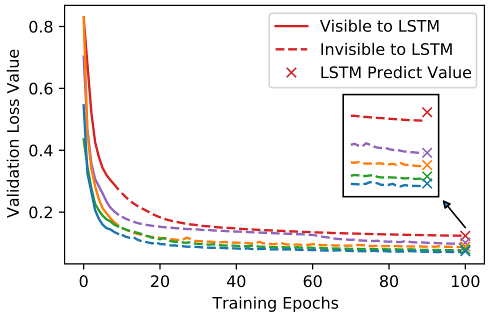


Figure 7: Predict final loss value with LSTM-based learning curve predictor. Different colors indicate different DNN configuration. The predictor takes first loss value of first 10 epochs as input. Loss values after that shown as dashed line are invisible to the predictor. A zoom-in of the LSTM predicted values shows we can predict final loss value accurately by only look at first 10 epochs.

to be content-aware could be done 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 [69]/MS-SSIM [70]. We choose MS-SSIM as representative quality estimator as it performs the best [70] among them.
- Perceptual quality estimator: recent studies suggest perceptual loss [33] that calculate metrics using DNN features to be "unreasonably effective" [76] 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 [37] and VGG [61]. In this paper we use the output of last layer before Softmax of VGG16 pretrained on Imagenet [22] 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

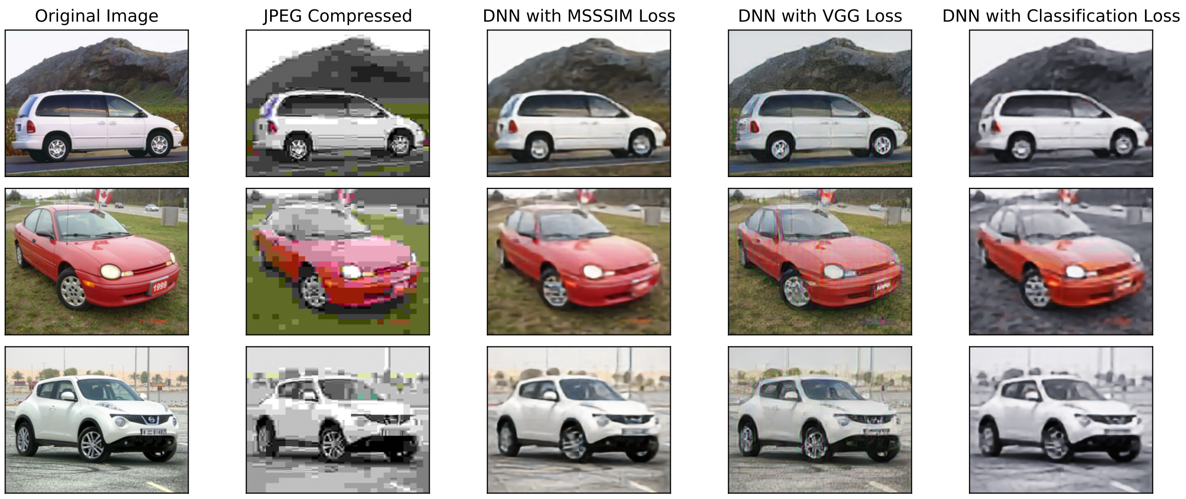


Figure 8: Visual quality comparison with different loss functions. All compressed images have a similar size of about 2.5kB. JPEG has a lot of block artifacts while DNN optimized for objective MS-SSIM loss tends to produce a more smoothed image; DNN optimized for perceptual quality with VGG Loss preserves 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 5.1.

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

$$\mathcal{L}_{cross\_entropy} = -\sum_{i=1}^{C} d_i log(d_i') \tag{3}$$

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 8. 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 5.

3.1.3 *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 [28] 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 [3, 31] coding to further compress the data. Another version without Huffman coding, which is more suitable for lossy networks is described in section 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 [12, 30] during training to simulate the effect of quantization. We can vary the scale of noise according to the quantization step and range of activations.

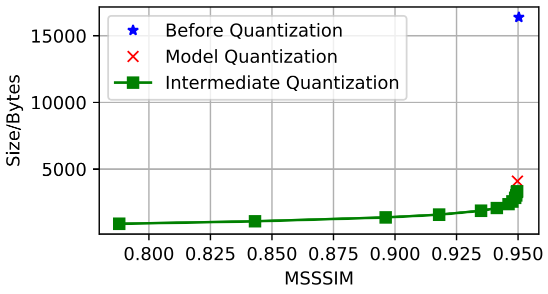


Figure 9: Average image MS-SSIM vs. file size for model/intermediate quantization. There is minor SSIM loss when quantize model with TFLite but results in a 75% smaller file size. Quantization of intermediate representation allows flexible size vs. MS-SSIM trade-off. It is possible to significantly decrease the file size without significant loss in accuracy or compression with intermediate quantization.

In Figure 9 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 10. There is no strong visible distortion (as those in JPEG compressed images) even after aggressive quantization.

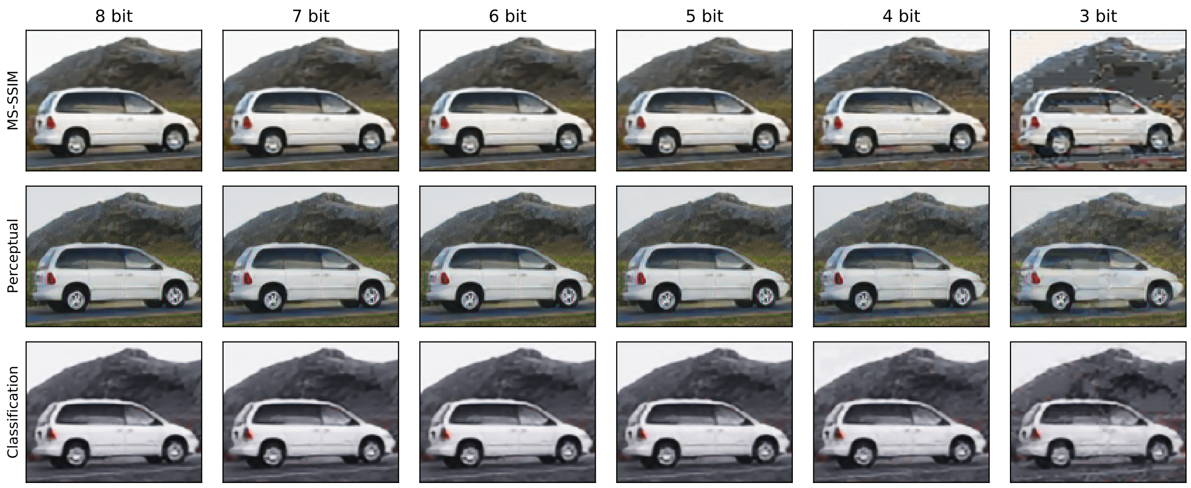


Figure 10: Visual quality comparison showing the effect of intermediate representation quantization with different precision. The first row are result of DNN trained with MS-SSIM loss, and second/third row are trained with VGG and classification loss.

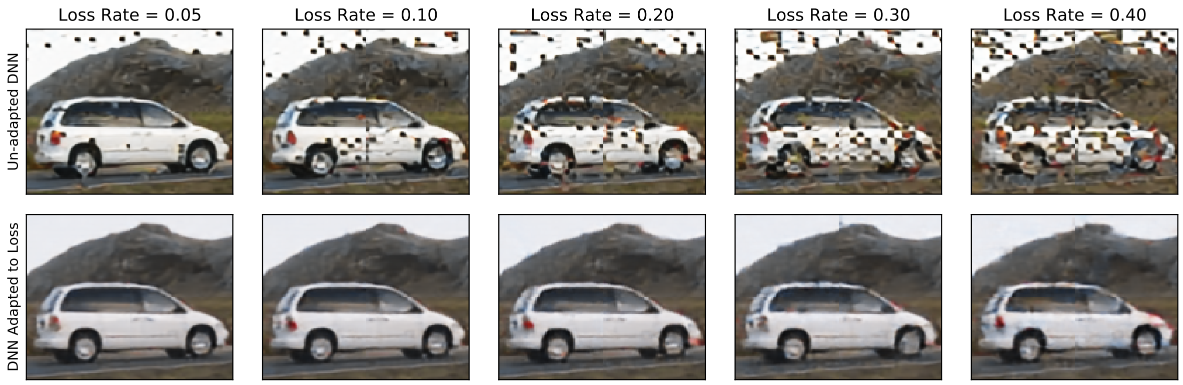


Figure 11: Visual quality comparison of DNN before (top) and after (bottom) adaption to packet loss. Both models are trained with MS-SSIM loss. DNN with adaption is trained with packet loss rate of 0.1. Adapted DNN could mask reconstruction errors due to packet loss and its image quality degrades more gracefully, even if the actual loss rate deviates from the trained rate.

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 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.

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 11. 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 to those due to quantization (as shown in Figure 10).

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 [21] 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

Figure 12: Sample images from Stanford Cars, Caltech Birds, Tensorflow Flower and Caltech101 dataset.

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 [5]. After flashing model to the AIoT platform, we develop MicroPython code in MaxiPy IDE [4] 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 reproducing. We summarize the datasets we've used in Table 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 12. 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 [38], which is a standard resolution for many DNN models pre-trained on ImageNet [22].

Table 5: Summary of Image Datasets Used

Dataset	Labeled Images	Classes
Stanford Cars [36]	8144	196
Caltech Birds 2011 [67]	11788	200
TensorFlow Flowers [63]	3670	5
Caltech 101 [26]	9144	102

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 `tf.keras.applications.ResNet50`. Classification model are pretrained on ImageNet [22] 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 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).

Table 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	7.81 uAh

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.

5.1 Compressed size vs. quality

Size-quality benchmark: similar to rate-distortion curve, in Figure 13 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 [12, 58] 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 14 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

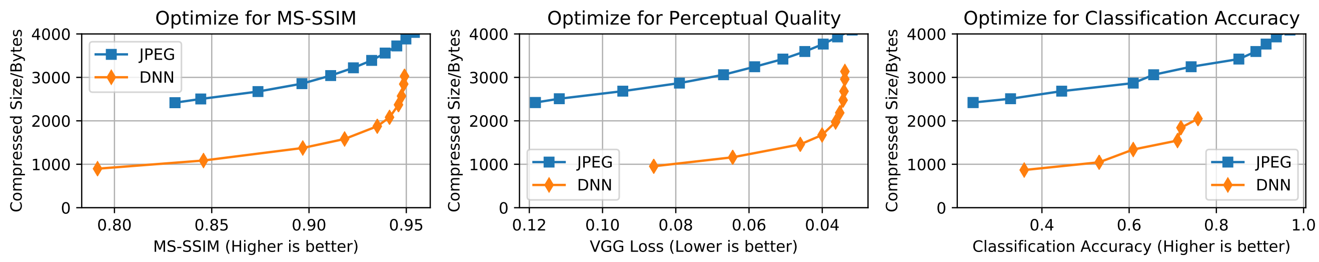


Figure 13: Size-quality benchmark for DNN in Starfish and JPEG with different optimization objectives. Compressed File Size is the smaller the better for a given quality metric. JPEG compressed files could be up to 2.7× as large as DNN with the same quality for MS-SSIM metric, and about 3× as large for perceptual quality and classification accuracy.

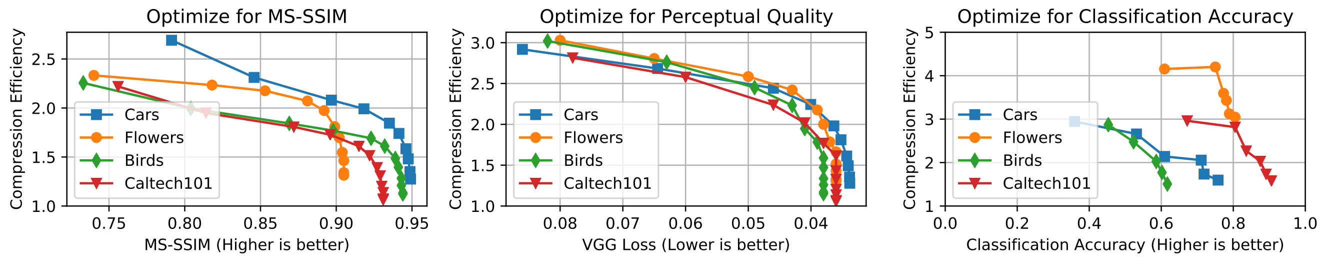


Figure 14: Compression efficiency of Starfish on different datasets with different optimization goals relative to JPEG. The compression efficiency of JPEG is defined as 1, so a compression efficiency of 2.5 means DNN uses only 40% of the bandwidth when compared with JPEG. DNN significantly outperforms JPEG on all datasets with all optimization goals.

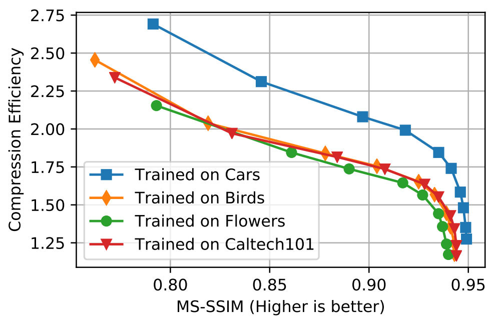


Figure 15: Content-aware benchmark by comparing models trained on different datasets and evaluate on Stanford Cars dataset. The model that train and test on the same dataset clearly outperforms models trained on a different one. However, all models still outperform JPEG by wide margins.

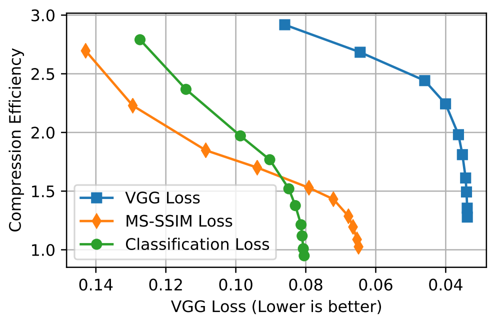


Figure 16: Task-aware benchmark by comparing models trained with VGG/MS-SSSIM/Classification loss and evaluated with perceptual quality using VGG loss. Despite the significant shift in working region, both the max and mean efficiency drops for DNN trained with MS-SSIM and classification loss, with efficiency degrades close or even worse than JPEG at small VGG loss values.

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

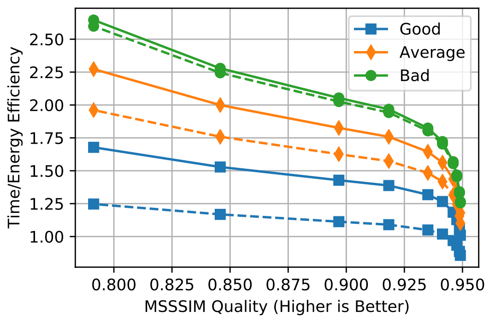


Figure 17: Time and energy efficiency of Starfish under different link conditions. Solid lines showing time efficiency and dashed lines show energy efficiency. Two times as efficient means we need to spend only half of the time/energy for the image when compared against JPEG.

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 17. Results suggest that we can get improve time/energy efficiency by up to 2.5× for the link with bad condition.

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 18. 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.

5.3 System-level benchmark

The benefit of loss tolerance for a single link shown in Table 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 [16] 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[3]. All

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

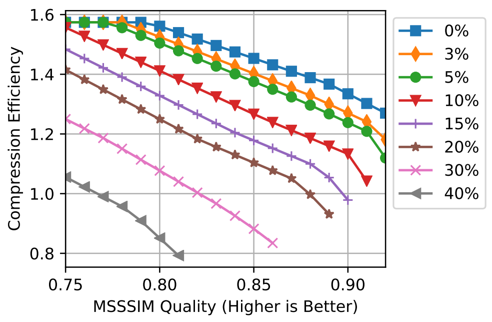


Figure 18: Compression efficiency of resilient DNN in Starfish against JPEG under different packet loss rates ranging from 3% to 40%. Overall performance gain is lower than non-resilient DNN as we have to remove Huffman coding. Result shows that Starfish degrades gracefully in the presence of data loss. Starfish outperforms JPEG even under heavy loss.

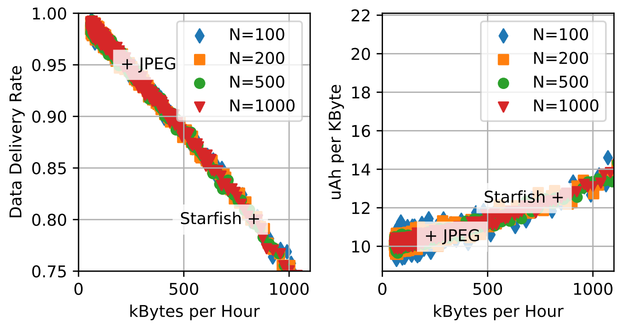


Figure 19: Simulated LoRaWAN total throughput (left) and energy efficiency (right) for a single gateway with 100 to 1000 IoT devices in the network. Data deliver rate drops sharply as total traffic increases, thus tolerating data loss brings huge throughput gain.

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 19. 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× 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.

6 RELATED WORKS AND DISCUSSION

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 [12], recurrent neural network [64], context-adaptive [39], residual coding [40], content-weighted [41], conditional probability models [48], GAN(Generative Adversarial Networks) [9, 13], and real-time compression on desktop grade GPUs [58]. 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 of 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 [43, 74], 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 [24]:

(1) *Learning curve extrapolation*[11, 35] that extrapolate learning curve after a few episodes of training.
(2) *Lower fidelity estimates*[57, 60] that training on partial or down-sampled data.
(3) *Weight inheritance*[17, 23] that warm-start new models by inheriting weights from parent model.
(4) *One-shot model*[18, 73] 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 [46] is an application layer coding that recovers data from redundancy and outperforms repetition coding by 21% percentage; NScale [65] tries to resolve concurrent transmission leveraging subtle inter-packet time offsets; FTrack[72] 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 [20, 25, 47]. However, these methods are not designed for application scenarios where human-readable images are required for additional verification or the data should be re-purposed for other AI tasks. Also distributed inference tends to put more pressure on the network side, which is not desired in LPWAN.

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 AIoT compiler and hardware supports them, which may happen in the near furture as AIoT hardware is evolving at an extremely fast pace. Another direction of research would be more aggressive quantization like XNORNet [56] or Dorefa-net [78] with customized implementation on FPGA devices to further push the limit of energy consumption.
- *Coordination across AIoT cameras* [32, 42]. 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 [19, 45, 59] 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.

7 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 two key insights: firstly, solving data loss in the application layer could replace the need for reliable transmissions, thus brings drastic performance gain and simplifies the design of network stack; secondly, codecs can benefit from being content-aware and task-aware in IoT settings that have fixed task objective and less variation in image contents. We carefully design and implement Starfish on low-cost AIoT accelerators with strict resource constraints. Experiment results indicate that Starfish achieves significant (up to 2∼4×) performance gain over JPEG based solutions in terms of compression, time, and energy efficiency. Simulation of large-scale LoRa network also demonstrates the power of loss resiliency in Starfish by sending 3.7× as many images over the network.

ACKNOWLEDGMENTS

We thank the anonymous shepherd and reviewers for their insightful comments.

REFERENCES

[1] [n.d.]. Better Portable Graphics. https://bellard.org/bpg/.

[2] [n.d.]. Building a new future: Transforming Australia's construction industry with digital technologies. https://customers.microsoft.com/en-us/story/pcl-construction-professional-services-azure

[3] [n.d.]. dahuffman. https://github.com/soxofaan/dahuffman.

[4] [n.d.]. Maxipy. https://maixpy.sipeed.com/en/get_started/maixpyide.html.

[5] [n.d.]. NNCase compiler. https://github.com/kendryte/nncase.

[6] [n.d.]. Perceptual JPEG encoder. https://github.com/google/guetzli.

[7] [n.d.]. TensorFlow Lite Micro Magic Wand example. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/magic_wand.

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

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

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

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

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

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

[14] Jim Bankoski, Paul Wilkins, and Yaowu Xu. 2011. Technical overview of VP8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*. IEEE, 1–6.

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

[16] Martin C Bor, Utz Roedig, Thiemo Voigt, and Juan M Alonso. 2016. 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*. 59–67.

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

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

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

[20] Sandeep P Chinchali, Eyal Cidon, Evgenya Pergament, Tianshu Chu, and Sachin Katti. 2018. 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*. 50–56.

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

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

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

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

[25] John Emmons, Sadjad Fouladi, Ganesh Ananthanarayanan, Shivaram Venkataraman, Silvio Savarese, and Keith Winstein. 2019. 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*. 27–32.

[26] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2004. Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. *Computer Vision and Pattern Recognition Workshop* (2004).

[27] Branden Ghena, Joshua Adkins, Longfei Shangguan, Kyle Jamieson, Philip Levis, and Prabal Dutta. 2019. Challenge: Unlicensed LPWANs Are Not Yet the Path to Ubiquitous Connectivity. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–12.

[28] Mark Horowitz. 2014. 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)*. IEEE, 10–14.

[29] Pan Hu, Rakesh Misra, and Sachin Katti. 2019. Dejavu: Enhancing Videoconferencing with Prior Knowledge. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. 63–68.

[30] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

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

[32] Junchen Jiang, Yuhao Zhou, Ganesh Ananthanarayanan, Yuanchao Shu, and Andrew A Chien. 2019. Networked Cameras Are the New Big Data Clusters. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*. 1–7.

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

[34] Giannis Kazdaridis, Stratos Keranidis, Polychronis Symeonidis, Panagiotis Tzimotoudis, Ioannis Zographopoulos, Panagiotis Skrimponis, and Thanasis Korakis. 2019. 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*. 29–36.

[35] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. 2016. Learning curve prediction with Bayesian neural networks. (2016).

[36] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3D Object Representations for Fine-Grained Categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia.

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

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

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

[40] Wei-Cheng Lee, David Alexandre, Chih-Peng Chang, Wen-Hsiao Peng, Cheng-Yen Yang, and Hsueh-Ming Hang. 2019. Learned Image Compression with Residual Coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 0–0.

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

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

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

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

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

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

[47] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. 2019. 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*. 21–26.

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

[49] Robert Moriarty, Kathy O'Connell, Nicolaas Smit, Andy Noronha, and Joel Barbier. 2015. A New Reality for Oil and Gas.

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

[51] Shadi A Noghabi, Landon Cox, Sharad Agarwal, and Ganesh Ananthanarayanan. 2020. THE EMERGING LANDSCAPE OF EDGE COMPUTING. *GetMobile: Mobile*

Computing and Communications 23, 4 (2020), 11–20.

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

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.

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

[55] Majid Rabbani. 2002. JPEG2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging* 11, 2 (2002), 286.

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

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

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

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

[60] Frederic Runge, Danny Stoll, Stefan Falkner, and Frank Hutter. 2018. Learning to design RNA. *arXiv preprint arXiv:1812.11951* (2018).

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

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

[63] The TensorFlow Team. 2019. Flowers. http://download.tensorflow.org/example_images/flower_photos.tgz

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

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

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

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

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

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

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

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

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

[73] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).

[74] Xiufeng Xie and Kyu-Han Kim. 2019. Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[75] Asif M Yousuf, Edward M Rochester, Behnam Ousat, and Majid Ghaderi. 2018. Throughput, coverage and scalability of LoRa LPWAN for internet of things. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

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

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

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