Neural Networks Meet Physical Networks: Distributed Inference Between Edge Devices and the Cloud

Sandeep P. Chinchali*
Stanford University
csandeep@stanford.edu

Eyal Cidon*
Stanford University
ecidon@stanford.edu

Evgenya Pergament*
Stanford University
evgenyap@stanford.edu

Tianshu Chu
Uhana, Inc.
tchu@uhana.io

Sachin Katti
Stanford University
skatti@cs.stanford.edu

ABSTRACT

We believe that most future video uploaded over the network will be consumed by machines for sensing tasks such as automated surveillance and mapping rather than for human consumption. Today's systems typically collect raw data from distributed sensors, such as drones, with the computer vision logic implemented in the cloud using deep neural networks (DNNs). They use standard video encoding techniques, send it over the network, and then decompress it at the cloud before using the vision DNN. In other words, data encoding and distribution is decoupled from the sensing goal. This is bandwidth inefficient because video encoding schemes, such as MPEG4, might send data tailored for human perception but irrelevant for the overall sensing goal.

We argue that data collection and distribution mechanisms should be co-designed with the eventual sensing objective. Specifically, we propose a distributed DNN architecture that learns end-to-end how to represent the raw sensor data and send it over the network such that it meets the eventual sensing task's needs. Such a design naturally adapts to varying network bandwidths between the sensors and the cloud, as well as automatically sends task-appropriate data features.

1 INTRODUCTION

Today, we have a host of distributed networked sensors, such as drones, security cameras, and Internet of Things (IoT) devices, that perform sensing tasks such as surveillance, target tracking, and mapping from high-dimensional streaming inputs, such as video. These distributed devices typically aggregate their information at a centralized datacenter for sensing tasks such as compute-intensive mapping, computer vision tasks, or querying frequently-updated databases such as Google Images. The sensing itself is increasingly performed using computationally intensive neural networks (often pre-trained) that can be scalably run in cloud environments.

To build such systems, these distributed sensors must encode and send their large input streams over a *wireless, bandwidth-constrained* network, and decode them before input into pre-trained machine learning models since they require raw data (e.g vision models operate on raw pixels). A natural approach would send compressed video over the network using standard schemes and then decompress it at the cloud before using a pre-trained vision DNN such as ResNet [7] or GoogleNet [25]. Such state-of-the-art deep architectures are often large (> 500 MB), memory and power hungry, as well as frequently-updated, precluding placing them on edge devices.

However, current encoding techniques are largely optimized for human perception, rather than machine sensing. For example, video compression schemes such as MPEG4 are optimized to ensure that the decoded video is perceived by the human eye as a good reproduction of the actual scene. But a vision model is interested in specific features of the video rather than the general scene which might contain quite a bit of irrelevant information. For example, tracking an endangered animal from drone wildlife footage should only send moving animals of interest even though the background scene might vary significantly. Since we anticipate more future *automated* sensing tasks, humans might only see a video to debug anomalies. Though the use of DNNs for content-aware video delivery has been recently addressed in the systems community [30], the work focuses on video streaming for human perception, not automated machine sensing.

As the diversity of edge devices and sensing tasks scales, we argue that data collection and distribution should be co-designed with the eventual sensing objective. We address a general problem for distributed, networked sensors

*Equal contribution

HotNets-XVII, November 15–16, 2018, Redmond, WA, USA

ACM ISBN 978-1-4503-6120-0/18/11...$15.00
https://doi.org/10.1145/3286062.3286070

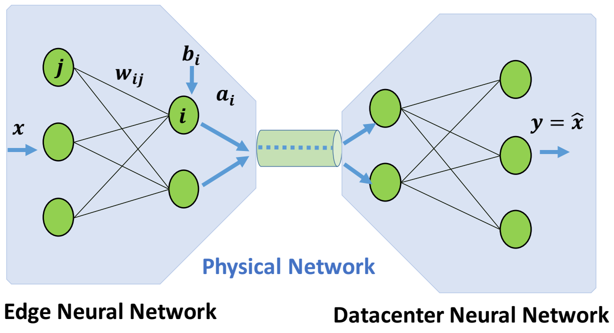

Figure 1: DNN split between an edge device and datacenter across a bandwidth-constrained link.

we call *Bandwidth and Task-Aware Encoding*. Notably, it extends to non-video based examples where standard compression schemes do not yet exist. Examples include vector time-series data from industrial plant IoT sensors or even Mars Rovers prioritizing compressed images to send from space to earth.

In essence, we ask:

> *How should distributed sensors represent and compress data sent over bandwidth-constrained links to best achieve a downstream sensing task at a datacenter?*

1.1 Key design requirements

Bandwidth and task-aware encoding requires a re-think of neural network training algorithms with physical network bandwidth as a first class citizen in the design process. We envision a distributed NN architecture, where shallow NNs on edge devices dynamically compress useful data and a large DNN at a datacenter decompresses it and provides active feedback to edge encoders on data relevance. Our problem requirements violate today's typical design assumptions:

1. *Machine Perception*: Content necessary for a machine to make a decision could be very different from that for humans, which might not ever even see the inputs.

2. *Resource-constrained network links*: As the number of edge devices scales, they must only send features relevant for a sensing task to meet network bandwidth constraints. Further, dynamic link throughputs might cause information to not reach a datacenter DNN within the time it needs to be processed.

3. *Modularity with Pre-trained Models*: We do not want to reinvent vision or sensing DNNs, where models like GoogleNet [25] can be periodically updated with extensive compute resources in the cloud. Rather, we want to leverage advances in sensing DNNs and deliver relevant inputs over a physical network to match their standardized interfaces.

1.2 Proposed Contributions

Our proposed contributions span both systems and machine learning/AI research.

System design: Architecture for bandwidth and task-aware encoding (Section 3).

Bandwidth and Task-Aware Training Algorithm:
Core techniques in current NN design include *dropout* [23] and *attention* [4, 29]. In dropout, connections in a NN are randomly dropped during training to reduce overfitting and increase robustness. In large images or a text corpus, *attention* focuses a DNN on sub-components of a sensory input stream and slowly pans across them to help in training.

We propose novel AI research that requires a *bandwidth and task-aware* as opposed to random dropout policy as a consequence of stochastic network links whose properties we can model or predict.

Paper Organization: We first introduce a motivating example and relevant background in Section 2. Then, we introduce our system design in Section 3 and evaluate it on a simplified drone target-tracking example in Section 4. We conclude with future research avenues in Section 6.

2 MOTIVATION AND BACKGROUND

2.1 Drone Target Tracking Problem

We introduce a recurring example for this paper in Figure 3, where three drones with limited fields of view must track a human (abstractly a red pixel) by using a pretrained image classifier like GoogleNet [25] at a datacenter. The drones each have a large video input that is too large to be sent over a wireless network with stochastic bandwidth. Hence, they must prioritize which changing features of the image to send in a bandwidth-efficient manner while exploiting the fact that the drones have partially overlapping fields of view.

2.2 Neural Networks for Vision and Control

DNNs [14], shown in Figure 1, learn to model datasets $\mathcal{D} = \{x_l, y_l\}$ of inputs x_l and corresponding desired outputs y_l. Training a DNN amounts to finding an optimal set of parameters θ, namely network weights and biases, to minimize a loss function penalizing mismatch between true output y_l with approximation $\hat{y}_l = f(x_l)$, given by $\mathcal{L}(y_l, \hat{y}_l)$.

Convolutional neural networks (CNNs) have been widely successful in computer vision, such as identifying objects $y_l = f_{\text{classify}}(x_l)$ from high-dimensional pixel input x_l. We also propose using autoencoder NNs, which learn how to best encode and decode information to minimize reconstruction error [3, 28]. In a typical autoencoder (Figure 1), a high-dimensional input x_l is passed to a neural network with a small number of neurons in a secondary "hidden layer" which must learn a sparse, latent representation to reconstruct the desired output so $y_l \approx \hat{x}_l = f_{\text{autoencoder}}(x_l)$.

2.2.1 DNNs for Control.

DNNs can serve as the decision making logic to solve dynamic control problems, which are often represented as Markov Decision Processes (MDPs) [1]. A discrete-time MDP, where time is indexed by t, is represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, R, \gamma)$ with states $s^t \in \mathcal{S}$ and actions $a^t \in \mathcal{A}$. Probabilistic state-transition dynamics are given by $\mathbb{P} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \rightarrow [0, 1]$, and reward function $R : \mathcal{S}\times\mathcal{A} \rightarrow \mathbb{R}$ rewards good control actions with scalar reward $r^t = R(s^t, a^t)$.

Solving an MDP amounts to finding an optimal control policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected gamma-discounted reward over a planning horizon T, given by $\mathbb{E} \sum_{t=0}^{T-1} \gamma^t R(s^t, a^t)$. Deep reinforcement learning (deep RL) [19, 22, 24, 26] approximates the solution to an MDP by representing a control policy $a^t = \pi(s^t)$ with a DNN whose parameters θ are optimized through simulations.

2.3 Relevant Work

We differ from recent work on using DNNs for content-aware video streaming [30] since we address automated machine sensing tasks, use a control theoretic approach to explicitly provide feedback on data relevance for a sensing goal, and incorporate pre-trained, third-party vision DNNs at the cloud. Though a growing body of research has addressed DNN model compression, the works typically assume compressed models run on a single physical device [5, 6, 9, 15].

3 SYSTEM DESIGN

The main goals of our design are as follows:

Sensing Task-Awareness: To compress information, edge devices must dynamically send important, fast-changing features needed to maximize an end-to-end goal with a pre-trained model.

Network Bandwidth Awareness: Since the sensory input stream of edge devices is possibly too large to send over a physical network, edge devices must dynamically compress relevant information and account for link unreliability.

Modularity: The architecture should allow for complex pre-trained decision making models, such as GoogleNet [25], from an external developer to be used. It should accomodate cloud software updates without pushing changes to edge devices as long as the standard API input to the pre-trained model remains similar.

Edge Device Compute Efficiency: Edge encoder NNs must be shallow enough to be memory, power, and compute efficient.

3.1 System Architecture

To address the above goals, we propose a general system architecture illustrated in Figure 2. We envision the approach to be broadly applicable, where edge devices could be drones with video stream inputs or IoT sensors with timeseries inputs. The overall prediction goal could be search and rescue for drones or distributed map-making for robotics. The key system components are:

1. Distributed Edge Encoders, Fig 2-1:
Each of N edge devices i receives a large sensory input stream x_i^t with novel and possibly redundant information from previous timesteps or other edge devices. Edge encoder i must prioritize important features and compress them efficiently into a message z_i^t to send over a physical network.

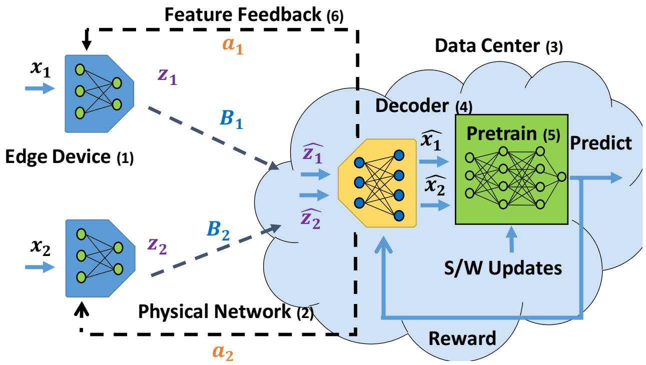

Figure 2: System architecture with distributed edge encoders and datacenter post-processing.

2. Physical Network Links, Fig 2-2:
Each edge device sends its message z_i^t over a physical network, such as cellular or WiFi, with a time-variant throughput B_i^t. In practical control applications, message z_i^t must be received at the DC within a time deadline or horizon H to be useful for decision making. The key challenge is that message z_i^t may not be received at the DC within deadline H due to low network throughput B_i^t. Hence, critical data could be effectively "dropped out" from downstream decision making at the datacenter. The vector of all physical link bandwidths is denoted by $\hat{\mathbf{B}}^t = [B_1^t, \ldots, B_N^t]$.

The following components reside at a datacenter (DC), shown in Fig. 2-3.

3. Centralized Decoder, Fig 2-4:
The centralized decoder at the DC must reconstruct the large sensory inputs x_i^t from each edge device from compressed features z_i^t, some of which may be replaced with stale information if z_i^t is dropped out in deadline H due to low throughput. Reconstructed estimates $\hat{x}_i^t$ should leverage overlapping views from edge devices to account for dropped features z_i^t.

4. Pre-trained Decision Making Model, Fig 2-5:
This model can be supplied and get *periodic cloud software (S/W) updates* from an external service, such as image-classification neural networks like GoogleNet[25] and ResNet[7] or Cartographer[8] for map-making in robotics. Our approach is general, where the pre-trained decison maker need not be a neural network and simply outputs a prediction $y = f_{\text{pretrain}}(\bar{\mathbf{x}}^t)$ from standardized input $\bar{\mathbf{x}}^t = [x_1^t, \ldots, x_N^t])$ specified in its API.

5. Dynamic Feature Selection Agent, Fig 2-4,6:
The decoder is a controller, such as a data-driven reinforcement learning (RL) agent, which selects an action a_i^t per edge device which represents the prioritized features to send over

the network. The features a_i^t, described subsequently, are chosen to maximize the prediction accuracy of the pre-trained model with *decoded* inputs while being *network bandwidth efficient*.

The decoder control policy π_{decode} selects an action vector for all drones $\bar{\mathrm{a}}^t = [a_1^t, \ldots, a_N^t]$ based on current compressed inputs $\bar{\mathrm{z}}^t$, the past *decoded* estimate $\hat{\mathrm{x}}^{t-1}$, and estimate of physical link bandwidths:

$$\bar{\mathrm{a}}^t = \pi_{\text{decode}}(\hat{\mathrm{x}}^{t-1}, \bar{\mathrm{z}}^t, \hat{\mathbf{B}}^t) \tag{1}$$

Reconstructed outputs from the decoder are given by

$$\hat{\mathrm{x}}^t = f_{\text{decode}}(\hat{\mathrm{x}}^{t-1}, \bar{\mathrm{z}}^t, \hat{\mathbf{B}}^t) \tag{2}$$

Decoder policy π_{decode} is optimized to maximize prediction accuracy so its reward r^t contrasts the true output $y^t = f_{\text{pretrain}}(\bar{\mathrm{x}}^t)$ from the pretrained model with that from *reconstructed* inputs $\hat{y}^t = f_{\text{pretrain}}(\hat{\mathrm{x}}^t)$. To do so, it uses a standard *loss function* $\mathcal{L}(y^t, \hat{y}^t)$, such as root mean square error (RMSE, $|y - \hat{y}|_2^2$) or classification losses. Overall reward R^t weights prediction accuracy through the loss function, a bandwidth-usage penalty r_{BW}^t, and possibly control reward r_{ctrl}^t, such as minimizing edge device battery usage.

$$r^t = -\underbrace{\mathcal{L}(y^t, \hat{y}^t)}_{\text{accuracy}} + \beta_{\text{BW}} \underbrace{r_{\text{BW}}^t}_{\text{BW usage}} + \beta_{\text{ctrl}} \underbrace{r_{\text{ctrl}}^t}_{\text{ctrl goal}} \tag{3}$$

6. Feedback from DC to Edge Devices: The principal goal of the feedback action $\bar{\mathrm{a}}^t$ from the DC to edge devices is to dynamically prioritize the sensory input stream x_i^t from edge device i based on the DC's *global view* of dynamic bandwidth fluctuations, rapidly changing, but important input content, and overall accuracy. Hence, though edge device i can only see its own input x_i^t, action a_i^t allows it be both *bandwidth-aware* and *task-aware* from the DC's centralized view, where all $\bar{\mathrm{a}}^t$ are computed jointly.

Actions a_i^t are general, allowing them to change both physical and synthetic parameters such as:

- Camera Angle, Zoom and Tilt (Physical)
- Filters or Key Hyperparameters in Pre-trained Encoding/Convolutional Neural Networks
- Combining elements of a vector timeseries, such as from an IoT sensor

Formally, edge device i's encoder creates a compressed code z_i^t based on feedback a_i^t and a possible estimate of its own bandwidth $\hat{B}_i$.

$$z_i^t = f_{\text{encode}}(x_i^t, a_i^t, \hat{B}_i^t) \tag{4}$$

We anticipate action feedback from the DC to edge devices to use minimal network bandwidth, which can be accounted for in reward (3) since they will be concise commands compared to large sensory data inputs. Further, most networks are uplink constrained (edge to DC) rather than downlink (action feedback)[20].

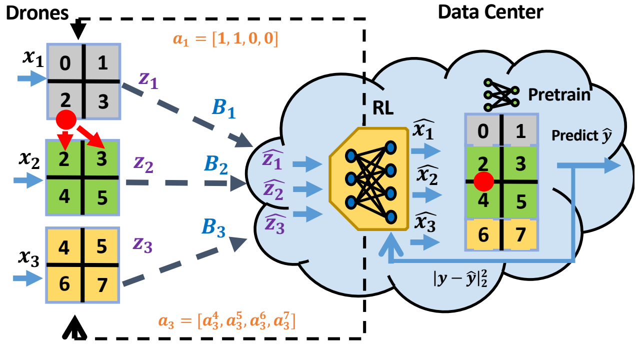

Figure 3: Drone target-tracking example where three drones prioritize regions 0-7 to track a red pixel.

3.2 Mathematical Formulation

We consider a fixed planning horizon of T discrete timesteps. Given sensory inputs and bandwidths from N drones as well as accurate labels y from the pre-trained model given by $\{\bar{\mathrm{x}}^t, y^t, \hat{\mathbf{B}}^t\}_{t=1}^T$, the control problem is

$$\text{maximize}_{\bar{\mathrm{a}}^0, \ldots, \bar{\mathrm{a}}^{T-1}} \quad \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}(r^t) \tag{5a}$$

$$\text{subject to} \quad Eqs.\ (1),\ (2),\ (3),\ (4). \tag{5b}$$

where $\bar{\mathrm{a}}^t$ is the global action for all drones chosen according to control policy π_{decode} (Eq.(1)).

Eq. (5) is too complex to be solved analytically for complex problems. Fortunately, we can formulate it as an MDP and solve it recursively by reinforcement learning (RL) with a data-driven simulator, where each episode lasts planning horizon T. RL finds a control policy $\pi_{\text{decode}} : \mathcal{S} \to \mathcal{A}$ that maximizes the return $R(\pi_{\text{decode}}) = \sum_{t=1}^{T} \gamma^{t-1} r_t$, with discount factor $\gamma \in [0, 1)$.

4 EVALUATION

We now provide an initial evaluation of the Drone Target Tracking Problem from Section 2.1 and Figure 3.

Sensing Task: Three drones with limited fields of view must track a target (abstractly a red pixel) by using a pre-trained image classifier at a DC. Drone i sees a video input x_i^t as a series of changing images that is too large to be sent over its wireless datacenter link of time-variant throughput B_i^t. By pooling their images at the DC, the drones must prioritize regions likely to have the target, which requires learning the transitions of both the target and network bandwidth *without* prior knowledge. In this preliminary work, we assume the drones act as fixed sensors and only the target moves.

Drone Image Content: A drone's image x_i^t consists of 16 pixels, divided into four regions with four pixels with an intensity value randomly distributed in $[0, 1]$. Notably, the target is a *single* red pixel of intensity 20 within a region, which makes it hard, but important, to track. Figure 3 shows image overlap, where top drone 1 sees regions 0-3, middle drone 2 regions 2-5, and bottom drone 3 regions 4-7. The DC maintains an estimate of the full image $\hat{x}_i^t \in \mathbb{R}^{32}$ for 8 distinct, four-pixel regions. Notably, the drones can only update a few elements of $\hat{x}_i^t$ due to bandwidth constraints

Network Bandwidth Constraints: Each drone may only be able to send parts of its image and hence only update a portion of global estimate $\hat{x}_i^t$ in the DC. Thus, an RL agent must learn where the target might appear in a bandwidth and task-aware manner. Bandwidth vector $\mathbf{B}^t = [B_0^t, B_1^t, B_2^t] \in \mathbb{R}^3$ represents the link quality for each of 3 drones. Since drone i can send upto its full input x_i^t of 16 pixels, we model $\hat{B}_i^t \in [0, 16]$, where 16 indicates it can send the whole image and 0 indicates it cannot send any pixel.

RL solution approach: We train a deep RL agent, which resides at the datacenter and decodes inputs with learned policy π_{decode}. The state of the RL agent is $s_t = [\hat{x}_i^{t-1}, \hat{B}_i^t] \in \mathbb{R}^{35}$, which consists of three measured bandwidths and the previous image estimate $\hat{x}_i^t \in \mathbb{R}^{32}$ in the datacenter in step $t-1$. Based on state s^t, RL chooses $\bar{\mathbf{a}}^t = \pi_{\text{decode}}(s^t)$, where $\bar{\mathbf{a}}^t = [a_{0,0}^t, \ldots, a_{2,7}^t] \in \mathbb{R}^{12}$. Action $a_{i,j}^t \in [0, 1]$ indicates the *fraction* of input drone i should send from region j (Figure 3), leading to a dimension of 12 for 3 drones and 4 regions.

Drone i attempts to send $\tilde{B}_i^t = 4 \times \sum_j a_{i,j}^t$ pixels across all its regions j based on its actions $a_{i,j}^t$. If drone i tries to send more pixels than available given its bandwidth, given by $\tilde{B}_i^t > B_i^t$, data is *randomly* dropped to meet constraint B_i^t. Hence, the RL agent must cleverly send few, but important pixels to the DC and exploit region overlaps if another drone has a better connection.

Task Reward: We define the task reward as

$$r^t = -\underbrace{\mathcal{L}(y^t, \hat{y}^t)}_{\text{tracking accuracy}} + \underbrace{\begin{cases} \beta_{\text{BW}} \cdot (\tilde{B}_i^t - B_i^t), & \text{if } \tilde{B}_i^t > B_i^t \\ 0, & \text{otherwise} \end{cases}}_{\text{bandwidth penalty}} . \quad (6)$$

The pretrained NN maps an image $\bar{\mathbf{x}}^t \in \mathbb{R}^{32}$ to a target region $y^t \in [0, 7]$. Loss function $-\mathcal{L}(y^t, \hat{y}^t)$ is positive if the red pixel was correctly localized and, to penalize drastically missing the target, a negative value corresponding to the Manhattan distance between centers of wrongly predicted region $\hat{y}^t$ and correct region y^t. The *bandwidth penalty* in Eq.6 penalizes the RL agent from sending more pixels $\tilde{B}_i^t$ than the link capacity B_i^t.

4.1 Results

We evaluate RL as follows:

(1) *Performance:* Reward relative to an offline optimal solution (Fig. 4).
(2) *Task Awareness:* Correctly tracking the target pixel (Fig. 5).
(3) *Bandwidth Awareness:* Intelligently allocating limited bandwidth (Fig. 6).

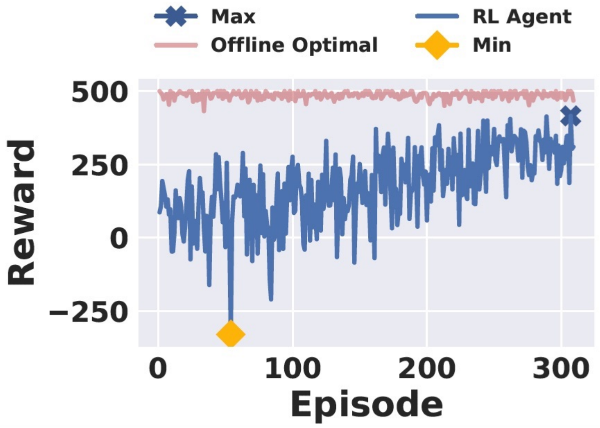

Figure 4: RL agent's learning curve.

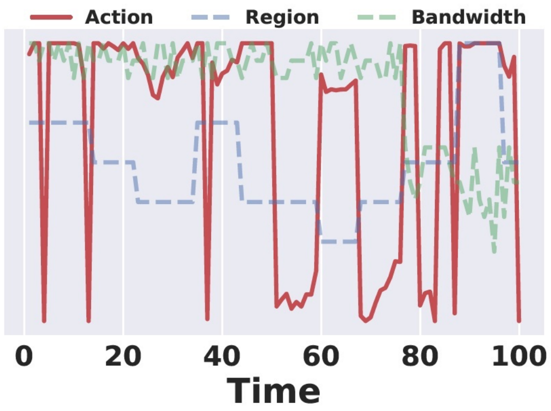

Figure 5: Action and bandwidth at region of target.

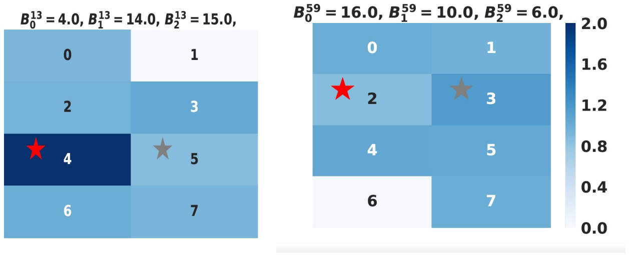

Figure 6: RL action $\bar{\mathbf{a}}^t$ at t = 13 (left) and t = 59 (right).

Convergence close to optimal solution: Figure 4 shows the per-episode total reward as the RL agent learns, where each episode consists of $T = 100$ timesteps. The target starts at a random initial pixel at the start of each episode and then moves according to a Markov process through the image.

We compare RL's per-episode total reward with that of an offline optimal solution, which has *perfect* knowledge of the current and future image, red pixel location and bandwidth. Though the offline optimal solution is infeasible in practice, the RL agent still reaches upto 84% of its reward in evaluation traces, showing it can adapt to the sensing task and bandwidth purely from data-driven learning.

RL correctly achieves task: We now deep-dive into a specific episode in order to show how RL effectively tracks the target while sending minimal data. Figure 5 shows how the pixel target moves between regions (dashed blue) overlaid with the RL action and bandwidth for this specific, moving region. The key takeway is that RL learns when the target (dashed blue) switches regions and quickly responds with an action $a^t_{i,j}$ close to 1 to send the changed location. If the region is overlapped by two drones we plot the action and bandwidth of the drone with the most bandwidth available at that time since that drone will likely send such an important region.

Figure 5 shows a strong correlation between the action taken on the red pixel region and times at which the region was changed, such as during $t = 43$ and $t = 76$. RL learns when the red pixel moves and that it should send an update at that time. At $t = 43$, when the target just moved, RL sends with an action close to 1, but then stops sending the target around $t = 50$ when it is not moving and has localized it well. However, it correctly reacts at $t = 59$ when the target moves again and sends the pixel with an action close to 1. Notably, RL on average sends 46% less pixels than if all 3 drones sent all their images, while still correctly localizing the red pixel 94 out of 100 times.

RL is bandwidth-aware: As a further testament to how RL smartly allocates scarce bandwidth for the sensing task, we show the spatial distribution of RL's actions in the heatmaps of Figure 6 when the target just moved regions at $t = 13$ and $t = 59$. The heatmap scale represents the action taken on that region, where the maximum action is 2 since in overlapping regions both drones can send the entire region. RL can only see the previous location of the target, indicated by a gray star, and must send relevant sub-regions to localize the target at its new location indicated by a red star. We annotate available bandwidth for each drone at time t in the title.

Both heatmaps show how the agent prioritizes the region it last saw the target (gray star) and predicts adjacent regions the target will likely move to. In fact at $t = 13$ the agent correctly predicts the red pixel will move to region 4 and reinforces that region by sending it in two drones. Additionally, RL respects bandwidth limits, for example it does not send region 1 in $t = 13$ and region 6 at $t = 59$ due to low bandwidths and no possibility of the red target moving there.

5 RELATED WORK AND DISCUSSION

Our proposed system must be robust to real world uncertainty. To be robust to changing network dynamics, we propose training with a diverse set of emulated throughput traces, an approach that has been proven successful in the context of mobile video streaming [18]. To adapt to improved versions of cloud DNNs or model drift, we need to analyze if edge encoders can adapt with online cloud feedback.

Further, feedback from the cloud to edge devices can be achieved by a general set of control theoretic methods such as Model Predictive Control (MPC) [2] other than Deep RL. Though Deep RL is expressive, it often has a large training time and is uninterpretable, so simpler controllers can be built with domain knowledge.

Related Work: Prior work splits layers of large DNNs between edge devices and the cloud, often running convolutional layers on the edge, compressing intermediate results using JPEG or variants [16], and computing fully connected layers in the cloud [12, 13]. A suitable split is determined by profiling per-layer energy consumption and latency [12], and hence requires analysis of rapidly-evolving CNN internals.

Another relevant work pools inputs between spatially distributed edge DNNs and the cloud, but crucially uses a special *BranchyNet* architecture to avoid computing later layers if the result from initial layers is accurate enough [27]. A large complementary line of work focuses on building compressed DNNs for mobile devices by techniques including weight quantization, pruning connections [5, 6], or more efficient convolutional units [9].

In video analytics, [10] addresses how to share common NN layers between different applications by specializing only higher DNN layers using multi-task learning, allowing one to optimize where (edge or cloud) to compute different layers. [11] addresses how to dynamically configure and re-tune video parameters to curb resource utilization, which is similar to our motivation of creating a bandwidth-aware encoding scheme. Finally, theoretical studies such as [17, 21] develop efficient schemes to estimate parameters from distributed sensor networks over bandwidth-limited links, but do not address sensing tasks with modern CNNs.

In contrast, our central thesis is that rapid progress of CNNs often cause internal architectures to change, leading us to instead focus on a bandwidth and sensing task appropriate encoding scheme to best deliver inputs to pre-trained DNNs.

6 FUTURE RESEARCH DIRECTIONS

Our initial evaluation can be extended in several ways.

Expressive Action Space: A key design problem is to choose a concise but powerful set of feedback actions to edge devices to compress data in a task-aware manner rather than simply drop sub-regions. The action set should be general to capture compression, encoding, and even object detection.

Control and Active Sensing: Drones can be instructed where to move to gather novel information for the sensing goal. We can also address control problems, where poor bandwidth conditions and task-awareness can adjust control risk.

Conclusion: As distributed networked sensors and their automated sensing tasks diversify, we argue that co-designing data encoding and transmission with an end-to-end goal is increasingly imperative. Hence, we propose a novel distributed DNN architecture that treats the sensing goal and limited network bandwidth as first-class citizens in the design process.

REFERENCES

[1] R. Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

[2] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[3] C. Doersch. Tutorial on variational autoencoders. Available at: https://arxiv.org/abs/1606.05908, 2016.

[4] K. Fukushima. Neural network model for selective attention in visual pattern recognition and associative recall. *Applied Optics*, 26(23):4985–4992, 1987.

[5] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[6] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1271–1278. IEEE, 2016.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 29–42, Boston, MA, 2018. USENIX Association.

[11] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266. ACM, 2018.

[12] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices*, 52(4):615–629, 2017.

[13] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. *CoRR*, abs/1802.03835, 2018.

[14] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[15] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[16] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan. Deepn-jpeg: a deep neural network favorable jpeg-based image compression framework. In *DAC*, pages 18:1–18:6. ACM, 2018.

[17] Z.-Q. Luo. Universal decentralized estimation in a bandwidth constrained sensor network. *IEEE Transactions on information theory*, 51(6):2210–2219, 2005.

[18] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[20] J. Oueis and E. C. Strinati. Uplink traffic in future mobile networks: Pulling the alarm. In *International Conference on Cognitive Radio Oriented Wireless Networks*, pages 583–593. Springer, 2016.

[21] A. Ribeiro and G. B. Giannakis. Bandwidth-constrained distributed estimation for wireless sensor networks-part i: Gaussian case. *IEEE transactions on signal processing*, 54(3):1131–1143, 2006.

[22] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In T. Jebara and E. P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395. JMLR Workshop and Conference Proceedings, 2014.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[24] R. Sutton and A. Barto. Reinforcement learning: an introduction. *Neural Networks, IEEE Transactions on*, 9(5):1054–1054, 1998.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[26] C. Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

[27] S. Teerapittayanon, B. McDanel, and H. T. Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, pages 328–339, 2017.

[28] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010.

[29] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[30] H. Yeo, S. Do, and D. Han. How will deep learning change internet video delivery? In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 57–64. ACM, 2017.