Learning Distributed Communication and Computation in the IoT

Prince Abudu[a,*], Andrew Markham[a]

[a]*Department of Computer Science, University of Oxford, 15 Parks Road, Oxford, OX13QD, United Kingdom*

Abstract

In distributed, cooperative Internet of Things (IoT) settings, sensing devices must communicate in a resource-aware fashion to achieve a diverse set of tasks, (i.e., event detection, image classification). In such settings, we continue to see a shift from reliance on cloud-centric to edge-centric architectures for data processing, inference and actuation. Distributed edge inference techniques address real-time, connectivity, network bandwidth and latency challenges in spatially distributed IoT applications. Achieving efficient, resource-aware communication in such systems is a long-standing challenge. Many current approaches require complex, hand-engineered communication protocols. In this paper, we present a novel scalable, data-driven and communication-efficient Convolutional Recurrent Neural Network (C-RNN) framework for distributed tasks. We provide empirical and systematic analyses of model convergence, node scalability and communication-cost based on dynamic network graphs. Further to this, we show that our framework is able to solve distributed image classification tasks via automatically learned communication.

Keywords: Machine Learning, Internet of Things, Distributed Communication, Distributed Inference

Internet of Things (IoT) applications are becoming increasingly ubiquitous and pervading more and more of our physical world, as in the case of smart parking [1], smart industry [2], smart traffic [3], smart homes [4], smart health [5], and emergency response systems [6]. In such applications, embedded sensing devices are typically resource constrained and need to meet operational requirements whilst respecting limitations on battery energy and processing capability. Current IoT architectures are skewed in favour of cloud-centric analytics, with the widespread sensors being 'dumb' and relatively limited in capability. In part, this is a natural consequence of the need for collaborative decision making, e.g., that sensor readings cannot be considered in isolation but must be analysed as a collective spatio-temporal series.

Although this centralized architecture takes advantage of virtually unlimited cloud computing capabilities, issues arise due to latency (data has to be sent up to the cloud, analysed and then returned before actions can be taken); bandwidth (raw data is typically sent to the cloud, consuming limited network bandwidth in resource constrained, e.g., LORA or 6LowPAN networks

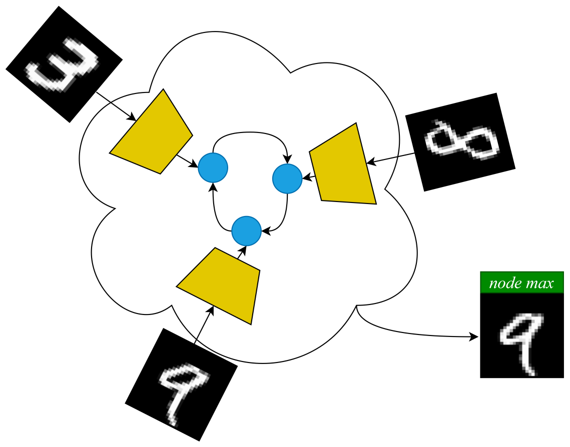

Figure 1: High level illustration for distributed MNIST classification. Multiple network nodes observe unique MNIST handwritten digits and cooperatively solve a distributed task - finding the *max* in the this case. In this work we show that nodes can communicate through their latent states to achieve this goal.

⋆

*Corresponding author:

Email address: prince.abudu@cs.ox.ac.uk (Prince Abudu

[7]); and in battery energy (nodes have to send data to the cloud, regardless of whether it is useful or not).

Recently, edge-centric computing paradigms [8] have emerged as a promising approach to address these challenges. In such settings, sensing devices are endowed with increased computing capabilities, thereby enabling inference at the edge of the network where data is collected. In distributed, cooperative scene monitoring, multiple sensing devices, (e.g., cameras) must observe part of the environment and video data from all devices must be fused to reach a collective insight. Transmitting individual sensing device data to a central node for processing is not scalable owing to bandwidth constraints. Moreover, the approach is susceptible to overall network failure in the event of a failing central node. These limitations motivate distributed, cooperative approaches in which sensing devices can perform local computations on observed visual data and cooperate with neighboring devices to perform image classification, recognition or detection tasks. To achieve these tasks, devices must communicate efficiently in a manner that respects constraints on their battery energy and bandwidth budgets. Handcrafted protocols and algorithms can be used to achieve these goals, but are tightly coupled to the particular problem and wireless network topology.

Deep Networks have achieved notable successes in domains characterized by massive amounts of data owing to their versatility, inference and predictive capacity [9]. In IoT applications, Deep Networks, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have achieved state-of-the-art performance on computer vision tasks, (i.e., image classification [10] and action recognition [11]) and time-series prediction tasks, (i.e., event detection [12] and target tracking [13]).

In this paper, we propose a novel data-driven Neural Network-based approach to achieving scalable, distributed communication at inference time. We consider a hybrid scheme which combines CNNs and RNNs. Recently, it has been shown that hybrid architectures in which CNNs are employed for feature extraction and RNNs for temporal modelling and prediction achieve better results when compared to other architectures on many tasks [11, 14, 15, 16].

As a motivating example, we extensively explore the problem of observing visual data to solve distributed tasks collectively, as illustrated in Figure 1. Our approach does not require any specialized communication protocols and automatically learns to minimize communication and handle link failure, whilst achieving high accuracies on defined objectives.

We structure our paper as follows. Section 1 presents related work, discussing distributed image classification and event detection, multi-agent communication, distributed machine learning and hybrid CNN-RNN architectures. Section II introduces our problem setting motivated by IoT scenarios. Section III provides an overview of our proposed scalable, data-driven, communication-efficient Neural Network-based approach and architecture. Section IV presents experiments that empirically evaluate node scalability, model convergence, computational and communication costs, comparison with other methods and architectural considerations/ablation on defined tasks. Finally, Section V concludes the paper and provides a discussion of future work.

1. Related work

1.1. Cooperative Deep Networks in IoT settings

1.1.1. Distributed Image Classification

Devices in IoT settings, (i.e., cameras, mobile robots) enable computer vision applications powered by Deep Networks. Recent works have proposed Deep Network-based algorithms for distributed, cooperative IoT settings in which visual data from multiple cameras is analyzed in real-time for various tasks namely, mobile object recognition [17], military coalition networks [18] and image classification [19, 20]. [20] propose an agent-based cooperative learning algorithm in which feature summary vectors are passed to neighbouring agents to achieve a global image classification task. Their work assumes unlimited communication capacity and does not optimize for communication cost. In [17], an edge-mediated, collaborative learning framework for mobile object recognition is proposed. [18] focus on exploiting the hierarchical nature of learned representations from Deep Networks to adaptively distribute computation over edge devices in coalition networks. [19] propose a Deep Network architecture instantiated over distributed computing hierarchies, consisting of the Cloud, edge devices and various sensors. Although these works are of immense promise, they explore distributed image classification in scenarios supported by edge infrastructure. In this paper we propose a more general approach, that is data-centric and does not rely on edge-mediation or Cloud support for distributed visual inference. Our approach is low-cost, can be instantiated for inference on sensing nodes and is communication-efficient. Instead of typically reasoning about how to offload computation to less resource constrained devices, our method seeks to utilize existing resources on sensing nodes to optimize its communication cost while achieving desirable accuracy on distributed, cooperative tasks.

1.1.2. Distributed Event Detection

Several works [21, 12, 22] have shown that Deep Networks are well-suited for event detection and processing tasks in emerging IoT settings. This is of key importance because these models can fuse and process raw temporal data from distributed sensors over long periods of time to provide accurate inference [12]. In [22], Long Short Term Memory (LSTM) networks are proposed for predictive maintenance on distributed turbofan machines. Their model is able to capture faulty conditions based on the history of the systems being monitored from large sets of sensor data. Work by [12] characterizes *complex events*, often made up of a series of *primitive events* and employ a Deep Network approach that is coupled with a state-based event detector. [21] propose RNN-based architecture in which sensor nodes communicate to solve distributed event detection tasks. Although these works offer an exploration of model hyper-parameters as they impact model performance, they fail to provide discussions on the communication overhead incurred in distributively detecting events from multiple sensor nodes.

1.2. Communication in Deep Networks

1.2.1. Learning Multi-agent Communication

Communication has been considered across multiple domains, i.e., Cooperative IoT settings, (e.g., Wireless Sensor Networks) [23] and Multi-agent Systems [24]. In such settings, most works employ predefined communication protocols instead of automatically learned communication. Research by [25] considers agents with limited capability operating in partially observable environments. In their approach, cooperative agents individually controlled by deep feedforward networks can learn task-specific communication via back-propagation. Further works by [26] use Reinforcement Learning and centralized learning approaches with decentralized execution. In these models, end-to-end back-propagation of error signals across agents enables learning automated communication using Deep Networks. Although these approaches are elegant in design, their scalability is limited owing to the need for specialized connectivity structures and more computational power to compute multiple agent actions at each time-step at inference time. In principle, our model instantiates memory-efficient Neural Networks and learns to minimize communication.

1.2.2. Distributed Machine Learning

In the field of Machine Learning, efficiently training and optimizing Deep Networks is an issue of research importance. This has been a consequence of the growing need to train and instantiate these algorithms in scenarios plagued with constrained computational budgets, (i.e., mobile devices). To alleviate this constraint while aiding applicability to large-scale data scenarios, (i.e., IoT applications), Distributed Learning [27] has been widely explored. In distributed, cooperative IoT settings, Deep Networks employed on sensing devices for inference tasks are often trained in a decentralized fashion. High latency and low-bandwidth limitations in such settings introduces communication bottlenecks on training nodes. Such bottlenecks have inspired significant research efforts that address communication-efficiency in Distributed Learning [28, 29]. Although these methods are instrumental to low cost Distributed Learning, they purely focus on providing model training methods for optimizing gradient-based updates rather than task-aware distributed inference.

1.3. Hybrid CNN-RNN Architectures

Deep Network architectures continue to evolve to suit a wide range of tasks in IoT settings. [30, 31] experiment with standalone CNN architectures on image sensor data considering resource constraints. [32] propose an RNN-based architecture that extracts features from raw sensor data and performs recognition tasks. These standalone architectures have proved to work in problem domains where feature extraction from image and/or sensor data is necessary before making predictions. Nonetheless, hybrid architectures employing both CNNs and RNNs have reported further performance gains in the same domains, (i.e., recognition [15, 33, 11] and image classification tasks [16, 34].) Work by [33] shows that a hybrid CNN-RNN architecture for gesture recognition outperforms pure CNN architectures. Further work by [11] compares a pure RNN approach to a CNN-RNN approach to action recognition in 3-D videos. Their work reports a 13 % increase in accuracy in the hybrid approach compared to the pure RNN approach. [16, 34] propose an end-to-end trainable unified CNN-RNN framework for image classification. Motivated by these works, we propose a hybrid CNN and RNN architecture to achieve distributed image classification via automatically learned communication. In our approach, we take advantage of the CNN's feature extraction abilities and feed features into the RNN which learns communication distributively. As in [16], our hybrid approach is end-to-end trainable.

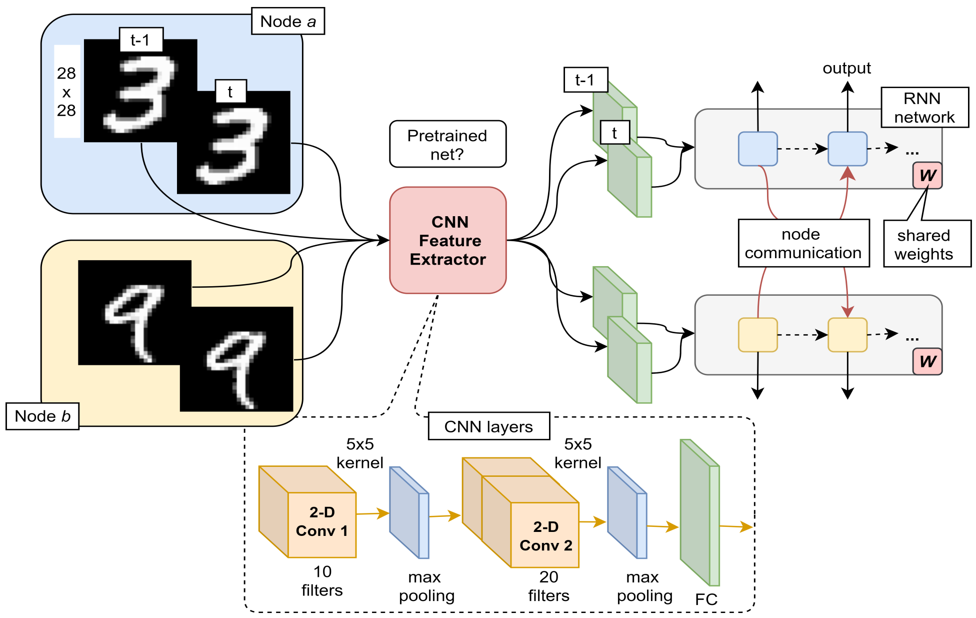

Figure 2: Overview of communicating C-RNN architecture, showing two nodes interacting over two sampling time-steps. At any given time-step *t*, each node observes visual data (in our case a single MNIST handwritten digit) then passes it through a CNN. The resulting image features are passed as input into a communicating RNN that shares a subset of its latent states with its neighbour. Each node evolves its own latent states and an output, but has identical weights. The temporal component built in the RNN is used to evolve the latent states via recurrent computation. A sequence of communication steps are performed whilst holding the input at a given sampling time-step. Consequently, the sampling time-step is slower than the communication time-step.

2. Problem Setting

We consider a scenario in which N nodes observe visual data and must communicate with neighboring nodes over potentially lossy wireless links to solve a given image classification task collectively. Our scenario is motivated by surveillance and computer vision based applications in the IoT. In such applications, widely distributed cameras are deployed and insights must be derived with minimal processing effort owing to resource constraints on the devices. To aid scalability whilst avoiding bandwidth constraints associated with performing analytics on a centralized Cloud infrastructure, each camera must perform local inference on the observed data and communicate with other devices in the network to achieve a global inference task. In our approach, we explore the problem of whether each node in the network can observe a randomized handwritten MNIST digit, perform feature extraction, then pass the features as input to an RNN architecture that can communicate a subset of latent states to neighbouring nodes to solve various tasks. In this problem setting, tasks include detecting which node is observing the *max* MNIST digit based on automatically learned communication. Further to this, we also consider a distributed *parity* problem in which the network must produce a signal value based on the *parity* of the max of the MNIST digits being observed by all nodes in the network.

3. Communicating C-RNN

In this section, we provide an overview of our proposed scalable, data-driven, communication-efficient Convolutional Recurrent Neural Network model and architecture. Further to this, we discuss how each node observes visual data then achieves feature extraction, communication, training and inference.

3.1. Network Node Architecture

Our proposed communicating C-RNN architecture is designed to enable communication amongst multiple distributed nodes observing visual data and collectively solving distributed objectives, (i.e., *max and max-parity*). The communicating C-RNN architecture can be extended to edge devices, (i.e., cameras and sensors) cooperatively operating in a smart environment, (i.e., parking occupancy detection, surveillance). To achieve their intended tasks, distributed nodes must collect data from their environment, communicate and perform inference locally. Figure 2 provides an overview of our architecture, showing latent state communication and inference for the nodes in the network.

Our communicating C-RNN architecture can be extended to an arbitrary number of nodes, N. We represent the topology with an undirected network adjacency graph G = (V, ξ) where $V = \{1,..., N\}$ are the nodes in the network and $\xi \subset V$ x V represents the presence or absence of a link between any two nodes *(i, j)* $\in \xi$. Based on the network graph G, each given node has a set of neighbours with which it can directly communicate. Node connections can be task-specific and user defined. Nonetheless, communication is learned by the network. The communication paradigm for our architecture is discussed in detail later in this section.

3.2. Feature Extraction

Our communicating C-RNN architecture models distributed nodes observing visual MNIST handwritten digits and collectively solving distributed tasks based on each node's unique observations. To achieve this under resource constraints, we design a distributed CNN-based feature extractor that encodes the image into a reduced set of features. At each sampling time-step t and for each node, the feature extractor takes as input a 28 x 28 image in gray-scale then passes it into two 2-D ReLu-activated convolutional layers with max pooling for downsampling and dropout [35]. The first convolutional layer filters the image with 10 kernels of size 5 x 5. The second convolutional layer has 20 kernels of size 5 x 5. The resulting features are then passed into a fully connected layer with f_{in} units. The number of units f_{in} for the fully connected layer corresponds to the number of features that are input to communicating RNN framework. The weights of the feature extractor are initialized from a pre-trained MNIST handwritten digits classifier. The feature extractors across all the nodes in the network are identical in architecture and have the same number of weights.

3.3. Node Communication

In our model, multiple nodes connected via an arbitrary graph must communicate to distributively solve various tasks. To achieve this, we design a novel communication handler that takes as input, at any given communication time-step t, the latent state space $\mathbf{s} = \{s_1, ..., s_n\}$ of all nodes in the network, where s_n is the latent state of the n^{th} node in the network. The communication handler ψ performs a mapping operation $\hat{\mathbf{s}} = \psi(\mathbf{s}, G)$ in which $\hat{\mathbf{s}} = \{\hat{s}_1, ..., \hat{s}_n\}$ is the original latent state updated with node communication as specified by the network graph, G. Note that this allows nodes to have arbitrary numbers of neighbours in the graph. Figure 3 shows in more detail how the mapping, $\hat{\mathbf{s}} = \psi(\mathbf{s})$ is achieved.

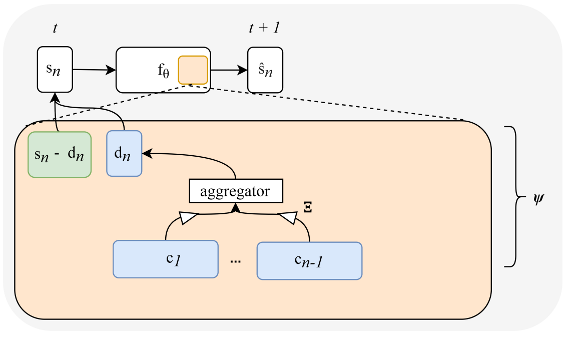

Figure 3: Overview of the communication handler ψ in the overall communicating C-RNN architecture for the n^{th} node in the network. At communication time-step t, the node evolves a latent state through the RNN model f_θ. The latent state s_n is then taken by ψ as input to achieve communication. In the case above, a gating function Ξ, is applied to allow nodes to choose to communicate or not at each time-step t. The remaining nodes $1 \dots n-1$ pass their communication c_1 and c_{n-1} as specified by the adjacency graph G to the n^{th} node via an aggregation/pooling function. The aggregated communication to this node d_n, is concatenated to the n^{th} node's latent states s_n to produce updated states $\hat{s}_n$ at communication time-step t+*1*.

At each communication time-step t, the latent state s_n (of an arbitrary dimension z, corresponding to the number of neurons in the RNN) of the n^{th} node is passed to the communication handler ψ which aggregates communicated data from nodes which are connected via the graph G. A subset of the latent state for the n^{th} node $c_n \subset s_n$ is used to communicate partial state to neighbours. Rather than communicating the entire latent state of dimension z, we instead send a smaller state subset of size $v < z$ for reasons of bandwidth efficiency. The network thus needs to learn what state representation to send.

To further enhance efficiency, we explore whether nodes can learn to decide when to communicate or not. We do so through a simple gating function Ξ which controls if data is actually transmitted over the wireless channel or not. As a simple example of a gating function, using a RELU activation and only transmitting information if the output is non-zero will limit communication to strictly positive values, with negative or zero values not being sent. Without loss of generality, it is clear that this could be extended to arbitrary functions. Thus, this describes how nodes send data to one another.

Through the adjacency graph G and modulated by each node's gated output, a node must collect communicated states from its neighbours. As the network graph is arbitrary and the number of nodes is not predetermined, it is clear that the number of communicated states is variable. To handle this, an aggregation or pooling function is used to aggregate the variable input into a single vector d, also of size v. This aggregation function can be learned or pre-specified. In our implementation, we use $max()$ as an aggregation function. The vector d, which represents the aggregated communicated and gated states from peers is then concatenated with the latent state s. Note that each node runs its own aggregation, based on received communicated states and is hence fully distributed. However, for maximizing training efficiency, the communication handler ψ is built inside a homologous and unidirectional RNN and performs the mapping $\hat{\mathbf{s}} = \psi(\mathbf{s})$ simultaneously for all nodes at each communication time-step t within the communicating RNN model.

3.4. Training and Inference

As in [19, 21], the nodes in our network distributively observe unique sequences of data and must collectively perform an objective task. Let x_n be the n^{th} node's sequence of input features. At each sampling time-step t, input features, $\mathbf{x} = \{x_1, ..., x_n\}$ are passed from distributed feature extractors attached to each node. At each given sampling time-step t, each node evolves its individual latent states with a series of updates being performed iteratively as:

$$\hat{h}_n^t = f_\theta(x_n, \hat{s}_n^t) \tag{3}$$

where $\hat{h}_n^t$ is the model updated latent state of the n^{th} node in the network. The communication handler ψ performs the mapping $\hat{s}_n^t = \psi(s_n^t)$, resulting in the latent states, $\hat{s}_n^t$ with node communication. The latent states, $\hat{s}_n^t$ are then passed into the model f_θ together with the input features x_n. The model f_θ takes the form of a fully connected, light-weight RNN with model parameters and biases θ. The model's parameters are tied amongst all the nodes in the network and optimized using Adam optimizer [36]. Using tied weights significantly reduces the number of network parameters and enables our model to train faster. This also ensures that all nodes in the network are identical in operation, i.e., they all run the same network. The layers in f_θ are activated using a non-linearity, ReLu. At each sampling time-step t, each node computes an output $\hat{y}_n^t$ as given by:

$$\hat{y}_n^t = \sigma(\hat{h}_n^t) \tag{4}$$

where σ is a non-linearity. We evaluate our model with joint optimization considering both model accuracy and communication cost. For the network's accuracy loss, we use Mean Squared Error loss (MSE) as in Equation (5):

$$L_{acc}(\hat{y}, y; \theta) = \frac{1}{z}\sum_{i=1}^{z}(\hat{y}_i - y_i)^2 \tag{5}$$

where $\hat{y}$ is the overall network prediction and y is the network target. For joint optimization, we implement a combined loss $L_{combined}$ that enables our network to learn both the objective tasks and optimize for node communication:

$$L_{comm}(\hat{a}, a; \theta) = \frac{1}{z}\sum_{i=1}^{z}(\hat{a}_i - a_i)^2 \tag{6}$$

$$L_{combined} = w\ (L_{acc} + L_{comm}) \tag{7}$$

In Equation (6), $\hat{a}$ is the number of messages communicated across the nodes at inference time and a is the target number of communicated messages for the network based on a communication budget. In Equation (7), the combined loss, $L_{combined}$ for each sample passed to the network during training with a weight w is given. Our distributed communicating C-RNN model is end-to-end trainable. A single network is learned across all nodes, although each node evolves its own latent states based on unique observations.

For inference, a single target y_t is defined for each input sample, x_t at any given sampling time-step t. The nodes in the network communicate based on the connections defined in the network graph G. At the end of the inference pipeline, any node in the network must have a unified view of the environment to perform inference on the objective tasks based on the communicate information from other nodes in the network. In [19] a local aggregator is used to determine if a combined summary of information from all the end devices is sufficient for the classification task. Our approach does not employ any central coordination for the nodes thereby enabling any node in the network to have sufficient information for inference at the end of the inference pipeline. This guarantees a seamless distributed operation based on automatically learned communication. It is worthwhile to note that model inference is dependent on the node connections in the network and must be modelled based on the objective task.

4. Experiments

In this section, we evaluate the efficacy of our communicating C-RNN model quantitatively using models trained on the popular MNIST handwritten digits dataset [37]. In our experiments, we quantitatively assess model convergence, node scalability and communication cost on the *max and max-parity* tasks. Our experimental setting is composed of feature extractors for the MNIST digits and unrolled communicating RNNs implemented in PyTorch.

4.1. Experimental Setup

To assess whether our proposed communicating C-RNN model can learn to communicate, we consider a setup where we have a varying number of network nodes, N between 2 and 40. In our model, parameters of the feature extractors on each node are initialized with weights from a pre-trained MNIST digits classifiers. The classifiers are pre-trained over 4 epochs and use a training batch size set to 64. The classifiers have parameters optimised using SGD [38] with a learning rate of 0.01 and a momentum of 0.5. The size of FC units is varied based on the the number of input features f_{in}, passed to the communicating RNN model. The communicating RNN model is trained with f_{in} = 10 and 25. Our setup uses [36] for optimization with a learning rate l_r = 0.005. The communicating RNN model has 4 hidden layers, the first two with 50 units and the other 2 with 8 units. Table 1 highlights hyper-parameter choices.

Description	Quality/Quantity
Platform	PyTorch
Pre-trained CNN layers	$l_r = 0.01$, $momentum = 0.5$
CNN training	batch size = 64, epochs = 4
Input features	$f_{in} = 10, 25$
RNN layers	$l_r = 0.005$, *act* = Linear, Relu
Weights	Shared
Optimizer	*Adam*
Training method	SDG
Loss	L2
Model parameters	$\alpha = 1284, 2169$
Number of nodes	$N = 2$ to 40

Table 1: Model hyper-parameters

The network parameters (which vary by f_{in}) are initialized with samples from a uniform distribution. Depending on the size of f_{in}, the number of parameters in the RNN α is either 1284 (f_{in} = 10) or 2169 (f_{in} = 25), making it amenable to deployment on low-end microcontrollers with only a few kbytes of memory. Each node is trained on randomly shuffled training samples $M_{train} = \{m_1, ..., m_i\}$ from the MNIST dataset, where m_i is the i^{th} training sample. We evaluate our model in terms of Root Mean Squared Error (RMSE). Further to this, we employ an error index similar to the measure used in [22] defining our model's mean percentage error. This % error augments RMSE by estimating the impact of the error made on each sample at inference time as RMSE only provides a generic error estimation based on mean 'distance' between model predictions and targets. The error rate is calculated as:

$$Error\ \% = \frac{1}{z}\sum_{i=0}^{z}\frac{|\hat{y}_i - y_i|}{y_i} \tag{8}$$

where y_i, $\hat{y}_i$ and z are same as values defined in Equation (5).

Using RMSE and error percentages, we are able to assess our model's predictive performance at inference time continuously by learning how much the model 'misses' the targets on both the *max* and the *max-parity* tasks.

4.1.1. Distributed Max and Max-parity tasks

First, we consider a *max* task setup in which nodes must learn the maximum MNIST digit being observed across all nodes. Further to this, we consider a parity problem (*max-parity* task) in which distributed nodes must first cooperatively work out the maximum MNIST digit observed, then determine parity of that digit. All the nodes in the network show convergence (with M_{train} = 1000, M_{test} = 500) on the overall objective and achieve low error percentages on test data.

		max		*max-parity*	
N	α / f_{in}	Error %	RMSE	Error %	RMSE
2	1284 / 10	1.03	0.195	2.24	0.454
	2169 / 25	0.87	0.190	1.46	0.421
5	1284 / 10	0.18	0.150	1.79	0.461
	2169 / 25	0.17	0.156	2.40	0.488
8	1284 / 10	0.11	0.113	1.83	0.456
	2169 / 25	0.09	0.104	1.74	0.462

Table 2: Summary of model performance results for N = 2, 5, 8, with communication channel size v = 2 and α = 1284, 2169 for f_{in} = 10, 25 respectively.

Table 2 provides a summary of results showing model performance on the distributed node *max* and *max-parity* tasks as the number of nodes (N) and number of image features f_{in} are varied.

Our model achieves the lowest error percentages when N = *8* and RMSE improves with increasing nodes.

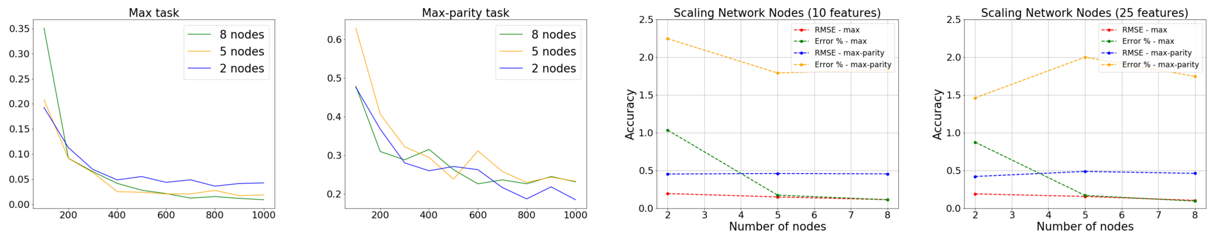

Figure 4: Training losses, RMSE and error percentages for *max* and *max-parity* tasks for $M_{train} = 1000$.

This shows that our model can scale while retaining high inference accuracy on given tasks. It is not surprising that the *max-parity* task is much harder to learn compared to the *max* task. Efficiently learning *parity* in distributed systems is typically difficult. The network nodes learn to integrate information over multiple time-steps ($T = 2, 5, 8$) via recurrent computation. For both the *(max and max parity)* tasks, the error percentages are slightly higher with $f_{in} = 10$, but this does not drastically affect model performance. We also compare RMSE on training data and RMSE on test data. A significantly lesser train RMSE compared to test RMSE means a model is over-fitting, whilst a greater means a model under-fits. Our model neither under-fits nor over-fits. Our results illustrate our model's ability to instantiate latent state communication with a communicating RNN model to solve distributed objectives scalably. Figure 4 shows training losses, RMSE and error percentages for $N = 2, 5, 8$.

4.2. Model Generalization

The success of any Neural Network model heavily depends on its ability to generalise. This remains a difficult challenge for most Neural Network architectures. To show the effectiveness of our model on adapting to unseen network structures, (i.e., networks with varying number of nodes) we conduct experiments in which training is done on $N = 2, 3, 5$ network nodes and inference is done on $N = 6, 8, 12, 16, 20$ and 40 nodes.

The results shown in Table 3 are comparable to cases when only data from a single network structure is used for training. At training time we vary N for batches of training samples $M_{train} = 400$ as shown in Figure 5.

Results on the test data show that our model generalises well and does not encounter any performance degradation when tested on 20 and 40 nodes. Our model is robust to a dynamic number of nodes, N during inference and generalizes well to learn the dependencies on the defined tasks in spite of varying network structures. This is of key importance for IoT systems which can face dynamic changes in the number of nodes due to node loss or replacement.

N	Error %	RMSE	Avg. Test Loss
6	0.15	0.137	0.018
8	0.14	0.141	0.019
12	0.11	0.121	0.013
16	0.12	0.134	0.014
20	0.11	0.120	0.012
40	0.11	0.121	0.011

Table 3: Model performance results for inference on $N = 6, 8, 10, 12, 16, 20, 40$ with communication channel size $v = 2$ and $f_{in} = 25$ for the *max* task.

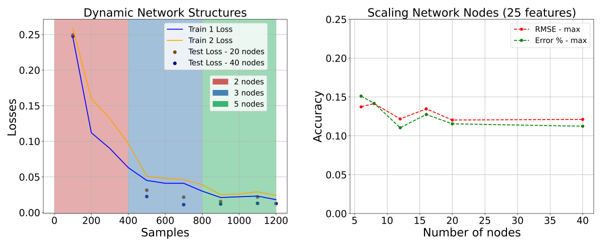

Figure 5: Left: Train and test losses on the *max* task. The network uses dynamic network structures with N= 2,3,5 for $M_{train} = 400$ for each network structure at training time. Right: Model performance for $N = 6, 8, 12, 16, 20, 40$.

4.3. Communication-efficiency

In addition to demonstrating that our model can learn to cooperatively solve distributed tasks via latent state communication, we quantitatively assess communication-efficiency at inference time. In resource constrained systems, the total number of messages sent to solve a problem must be minimized to conserve node energy and limit network bandwidth. In our experimental setup, we consider the total number of messages

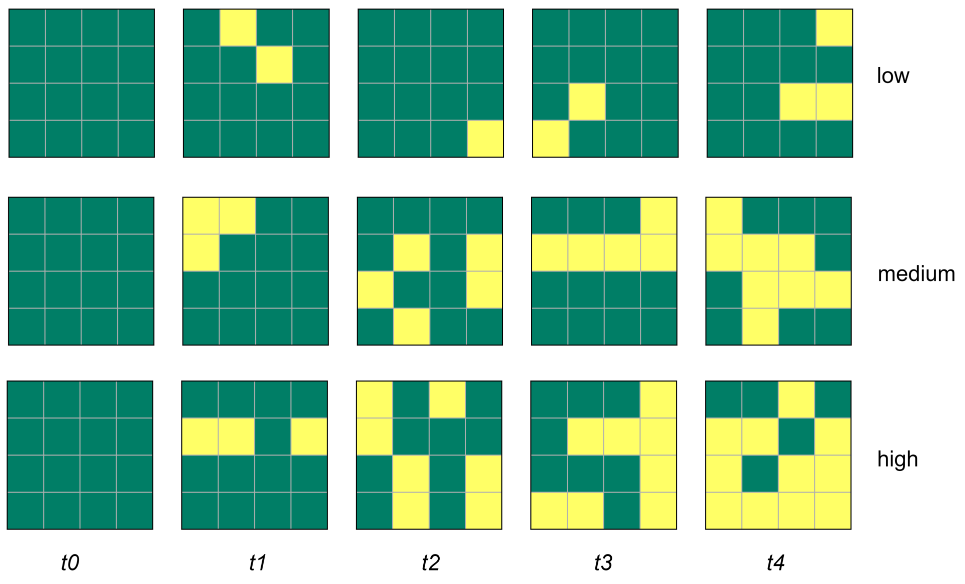

Figure 6: Communication patterns for $N = 16$ nodes across 5 sampling time-steps with low (factor: 0.25), medium (factor: 0.5) and high (factor: 0.8) penalization on communication cost optimization. Nodes in ON state (communicating nodes) are indicated by the green color whilst nodes in OFF state (non-communicating nodes) are indicated by the yellow color. We can see that our model optimizes for communication cost (more or less aggressively) depending on the impacting factor on the communication cost function.

communicated t_{comm}, during inference. The communicated messages t_{comm} are determined purely by the model which optimizes a communication loss function L_{comm} by penalizing the total number of messages sent. Figure 6 shows communication patterns in a 16-node network when we vary the penalization on communication cost optimisation. Our model is able to automatically adapt to the impact on penalization.

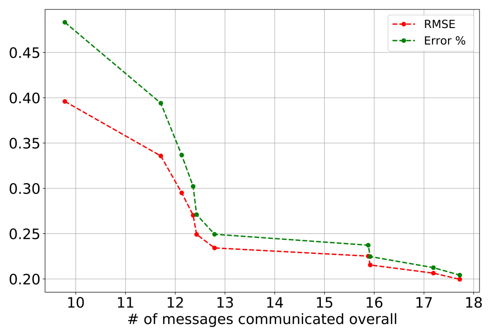

Figure 7: Model performance with varying number of messages communicated t_{comm} for $N = 5$ on the *max* task. This demonstrates the tradeoff between task accuracy and network resources. An operator could choose to focus on efficient communication at the cost of a small reduction in accuracy. Network nodes can only communicate for a total maximum of $t_{comm} = 25$ for each forward push.

As an architectural consideration, we set the size of the node output communication channel $v = 2$. Our results show that our model jointly optimizes for communication cost whilst maintaining high predictive accuracy at inference time. In Figure 7, we show the trade-off between model performance and total number of communicated messages in t_{comm} for $N = 5$.

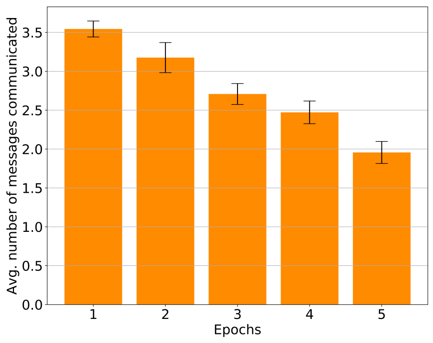

Figure 8: Average number of messages communicated t_{comm} for $N = 5$ as training progresses on the *max* task. Note how the network learns to slowly reduce the amount of communication needed.

By taking the total number of messages communicated across all nodes, t_{comm} and dividing it by the maximum limit (the denominator) of the number of messages that can be communicated during inference, we learn that only 51% - 56% communication is necessary to reach a good accuracy on the *max* task. This intuitively makes sense, as nodes observing small digits can stop participating in the process as it is clear that they do not satisfy the *max* operator. In Figure 8, we show how the network learns to communicate less with increasing number of training epochs. This shows that nodes first learn how to achieve good accuracy and then can reduce communication cost through subsequent optimization.

4.4. Computational Complexity

Intelligent spatially distributed IoT applications must execute a resource-aware operation that effectively utilises limited computational resources available on embedded IoT devices. Although Deep Networks, (i.e., CNNs) are powerful for inference tasks in many IoT settings, they typically take up a significant amount of memory. In our experiment, we quantify the computational overhead on both the feature extractors and the communicating RNN model.

Op	Params, α (K)	FLOPs (K)/ %
Conv2d-1	0.26	149.76 / 30.28
MaxPool	-	4.32 / 0.87
ReLu	-	2.88 / 0.58
Dropout/Conv2d-2	5.02	320 / 64.7
MaxPool	-	0.96 / 0.19
ReLu	-	0.64 / 0.13
Linear-1	16.05	16.00 / 3.24
Totals	21.33	494.56 / 100

Table 4: Layer by layer computational overhead of a feature extractor applied on a single node at inference time. Model parameters, α are given and FLOPs are estimated for extracting features for a single image passed as a 4-D tensor on a single node. Feature extraction kernels in the convolutional layers take up more computational resources as compared to the fully connected layers.

Table 4 shows the number of parameters and FLOPs[1] (FLoating-point OPerations) per layer in the models. We can clearly see that most of the model's computation (30.28 % in the first convolutional layer and 64.7 % in the second convolutional layer) is incurred at the individual nodes' feature extraction kernels during convolution operations. Although the linear layer only accounts for 3 % of the computation, it involves 3x more parameters when compared to the convolutional layers. In total, the feature extractor takes approximately 0.16 MB of memory for inference on a single input image considering the forward/backward pass and parameters.

[1]We consider a FLOP as half a Multiply-Add operation. Thus, a single Multiply-Add is 2 FLOPs

	f_{in} = 10		f_{in} = 25	
Op	α (K)	FLOPs (K)	α (K)	FLOPs (K)
Linear-1	0.75	0.700	1.5	1.450
ReLu	-	0.050	-	0.050
Linear-2	0.408	0.400	0.408	0.400
ReLu	-	0.008	-	0.008
Linear-3	0.126	0.112	0.261	0.232
ReLu	-	0.014	-	0.029
Totals	1.284	1.284	2.169	2.169

Table 5: Layer by layer computational overhead for the communicating RNN model. Model parameters, α are given for f_{in} = 10, 25 respectively and FLOPs are estimated for a single forward push at timestep t during inference on a single node. Using ReLu for layer activation saves on computation cost as other activation functions, i.e., Sigmoid are more computationally costly as they use more FLOPs.

Table 5 shows the computational overhead for our communicating RNN model. FLOPs and model parameters scale with the dimensionality of the input.

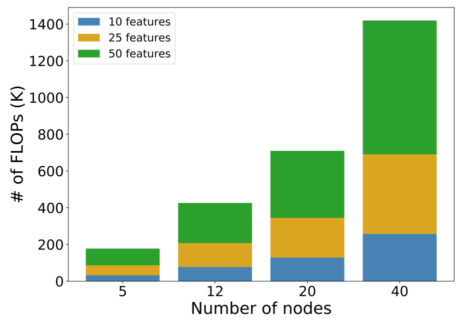

Figure 9: Total (sum of all nodes) computational load with increasing $N = 5, 12, 20, 40$ at inference time on the *max* task. Model parameters scale with increasing f_{in}. FLOPs are estimated as a product of N, t and number of FLOPs per forward push, where t is the number of timesteps for which we hold the model input.

On a 8-node system, we realize a 5 % performance difference on the *max task* when comparing f_{in} = 10 and f_{in} = 25, although the model performs 51 % more FLOPs when f_{in} = 25. Nodes in our model learn to distributively and cooperatively solve tasks via latent-state communication. To achieve this, the model performs a series of latent state updates across all nodes in the network at inference time.

In Figure 9, we show the computational overhead across N = 5, 12, 20, 40. It is worth noting how doubling the nodes in the network results in 100 % increase in FLOPs (the computational overhead scales linearly). This is due to the model performing more latent state updates with additional nodes in the network. Our results show that the computational overhead of the latent state updates is negligible with respect to the overall model.

f_{in}	N	Params (K)	FLOPS (K)	Error %
10	12	22.617	571.60	0.13
	40	22.617	751.36	0.15
25	12	23.499	624.70	0.11
	40	23.499	928.36	0.11
50	12	24.974	713.20	0.10
	40	24.974	1,223.36	0.11

Table 6: Summary of computational overhead and model performance for the unified C-RNN model on the *max* task with N = 12,40 and α = 1284, 2169, 3644 for f_{in} = 10, 25, 50 respectively. FLOPs are given for inference on a single image.

As can be seen from Table 6, our unified model with CNN-based feature extractors integrated with the communicating RNN model at each single node is suited for embedded, resource constrained devices. This is considering our model size (< 25K parameters) and the number of FLOPs needed for inference in a N node network on a single image. Further studies are necessary to explore opportunities for optimisation on feature extractors through quantization and model compression.

4.5. Network Robustness

In distributed multi-node settings, network robustness is of paramount importance. We model lossy-links in which network nodes may fail to communicate their latent states due to obstructions or multi-path etc.

Link failure	0 %	20 %	40 %	60 %
Error %	0.170	0.176	0.188	0.194
t_{comm}	7.0	10.0	13.0	13.0

Table 7: Model performance and average t_{comm} with link failure.

Table 7 shows a summary of our results with various percentages of link failure with $N = 5$ and $v = 2$. Our communicating C-RNN model's performance degrades by only 3.4 %, 13.6 % and 10.05 %, respectively for 20%, 40% and 60% node failure on both *max* and *max-parity* tasks. Moreover, our model automatically learns to increase the total number of messages, t_{comm} sent amongst nodes with increasing link failure. This shows that our model is robust to failing nodes.

4.6. Comparison with other methods

When considering distributed inference on local, resource-constrained devices, communication cost is an important concern. We evaluate our model's computational footprint by comparing it with 4 baseline methods.

Energy-Unaware Distributed C-RNN Baseline In this case, we setup an energy unaware variation of our communicating C-RNN model. This model does not optimise for communication cost as in the case with our communicating C-RNN model. As in the communicating C-RNN model, each node observes a unique 28 x 28 MNIST digits in gray-scale, extracts 25 features and feeds features in to the communicating RNN, which distributely and cooperatively solve the objective functions (*max and max-parity* tasks) locally.

Centralised Baseline: In this model, each node observes a unique 28 x 28 MNIST digit in gray-scale and transmits it to a central compute point that computes the objective function.

Semi-centralised Baseline: Each node observes a unique 28 x 28 MNIST digit in gray-scale, extracts 25 features from each image before transmitting them to a central point that computes the objective function.

Tree-based Baseline: This approach considers a classic Wireless Sensor Network arrangement as in [39]. We model a tree structure with N = 12, 40. At inference time, the nodes observe a unique 28 x 28 MNIST digit in gray-scale, extracts 25 features and communicates features via links to neighbouring nodes to an aggregating node. The aggregating node then passes features to a central point for computation of the function from features across all nodes.

We evaluate performance based on the communication and compute footprints of the baseline models compared to our communicating C-RNN model. The communication footprint measures the actual size of the messages being transmitted/communicated whilst the compute footprint gives a measure of the distance from the very edge of the network to a central compute point. In our case we model the distance from nodes at the edge of the network to a central compute point as 4 units. All baselines use the same architecture for feature extraction as given in section 3.2. In turn, the memory requirement (1.8 MB for a 12-node system and 6 MB for a 40-node system) for feature extraction on a single image is similar across all baselines.

Baseline	N	Comms	Compute	Error %
Energy-Unaware C-RNN	12	240	12.0	0.09
	40	800	40.0	0.10
Semi-centralised	12	1200	48.0	0.08
	40	4000	160.0	0.09
Centralised	12	9408	48.0	0.09
	40	31360	160.0	0.09
Tree-based	12	3400	23.0	0.10
	40	8100.16	160	0.11
Communicating C-RNN	12	137.6	12.0	0.11
	40	440	40.0	0.11

Table 8: Communication (bytes) and compute (units) footprints for the different baselines considered in our experiment on the *max* task with N = 12, 40 for f_{in} = 25. The communication and compute footprints are given for inference on a single image across all nodes in the network.

From results in Table 8., we can clearly see that the communication and compute footprints for the centralised baselines scales quadratically when compared to our communicating C-RNN model. Transmitting images to a central compute point is 8x more expensive when compared to transmitting features from individual nodes. Although the communication and compute footprints of the centralised approaches seem hefty, the models achieve slightly higher accuracy on the objective task. Optimising for communication cost, as in our communicating C-RNN model reduces the communication footprint, on average, by 1.7x when compared to the energy-unaware variation of the same model. For the two models, there is a marginal difference (10 %) in model performance. Our communicating C-RNN model uses 25x less communication and 1.9x less compute resources when compared to the Tree-based baseline in a 12-node system. These results demonstrate that our communicating C-RNN achieves an encouraging balance on communication and compute resource utilisation whilst automatically learning to optimise communication to solve distributed tasks locally. Moreover, our approach automatically discovers complex communication patterns without compromising on performance.

4.7. Architectural Considerations/Ablation

Our communicating C-RNN model presents opportunities for further optimization across the architectural considerations. We manually vary the size of the node output communication channel using v = 2, 10. The node output communication channel size, v corresponds to the number of latent states each node can communicate. With v = 10, the model achieves an error percentage 6 % lower than that of v = 2. This is because most of the nodes' latent state information is retained for computation within the model f_θ's layers. We also consider varying the number of features passed as input f_{in} from the feature extractors into the communicating C-RNN network. Our results show that there is only 5 - 10 % increment in the model's performance from a setup with f_{in} = 10 to a setup with f_{in} = 25. The model's parameters scale with increasing f_{in} and v values considering that these parameters directly affect the size of the latent states of our model. Our results show that we can achieve remarkable model performance even with small values of both f_{in} and v.

5. Conclusion

In this paper, we have presented a novel, scalable, data-driven and communication-efficient Convolutional Recurrent Neural Network (C-RNN) framework for distributed settings. Our model enables seamless latent state communication without the need for complex predetermined, specialized communication protocols. We showed through an array of experiments that our model learns to communicate scalably and is suited for distributed IoT scenarios with visual data. Our model achieves high accuracy with low communication cost on defined target distributed objectives while being robust to node failure. Realizing optimal communication in resource constrained environments while meeting required accuracies on target objectives is a key challenge. Our work makes strides toward achieving this using intelligent Machine Learning algorithms that can sit on distributed, cooperating, resource-constrained end devices in IoT settings. The concept of employing latent state communication across multiple agents has been explored in [21, 25], nonetheless, to the best of our knowledge, this is the first time a communication-efficient, yet scalable neural network architecture is proposed for distributed inference on visual data whilst considering latent state communication cost. Our communicating C-RNN architecture presents a wide array of research directions on experimenting with more sophisticated multiple node connection architectures.

With increasing number of nodes in a network, node connections can easily become complex resulting in a computationally intensive system. To solve this, future work will entail coupling our model with quantized neural network architectures [31] for resource constrained devices. Moreover, it will be interesting to extend our model to more complex scenarios in which heterogeneous nodes are observing multimodal data at inference time.

References

[1] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Vairo, Car parking occupancy detection using smart camera networks and Deep Learning, in: 2016 IEEE Symposium on Computers and Communication (ISCC), IEEE, 2016, pp. 1212–1217. doi:10.1109/ISCC.2016.7543901.
URL http://ieeexplore.ieee.org/document/7543901/

[2] L. Li, K. Ota, M. Dong, Deep Learning for Smart Industry: Efficient Manufacture Inspection System With Fog Computing, IEEE Transactions on Industrial Informatics 14 (10) (2018) 4665–4673. doi:10.1109/TII.2018.2842821.
URL https://ieeexplore.ieee.org/document/8370640/

[3] J. Barthélemy, N. Verstaevel, H. Forehead, P. Perez, Edge-Computing Video Analytics for Real-Time Traffic Monitoring in a Smart City, Sensors 19 (9) (2019) 2048. doi:10.3390/s19092048.
URL https://www.mdpi.com/1424-8220/19/9/2048

[4] M. Manic, K. Amarasinghe, J. J. Rodriguez-Andina, C. Rieger, Intelligent Buildings of the Future: Cyberaware, Deep Learning Powered, and Human Interacting, IEEE Industrial Electronics Magazine 10 (4) (2016) 32–49. doi:10.1109/MIE.2016.2615575.
URL http://ieeexplore.ieee.org/document/7792825/

[5] L. R. Nair, S. D. Shetty, S. D. Shetty, Applying spark based machine learning model on streaming big data for health status prediction, Computers & Electrical Engineering 65 (2018) 393–399. doi:10.1016/J.COMPELECENG.2017.03.009.
URL https://www.sciencedirect.com/science/article/pii/S0045790617305359

[6] B. K. Dar, M. A. Shah, H. Shahid, F. Fizzah, Z. Amjad, An Architecture for Fog Computing Enabled Emergency Response and Disaster Management System (ERDMS), in: 2018 24th International Conference on Automation and Computing (ICAC), IEEE, 2018, pp. 1–6. doi:10.23919/IConAC.2018.8749064.
URL https://ieeexplore.ieee.org/document/8749064/

[7] H. A. Al-Kashoash, A. H. Kemp, Comparison of 6LoWPAN and LPWAN for the Internet of Things, Australian Journal of Electrical and Electronics Engineering 13 (4) (2016) 268–274. doi:10.1080/1448837X.2017.1409920.

[8] C. Li, Y. Xue, J. Wang, W. Zhang, T. Li, Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management, ACM Comput. Surv 51. doi:10.1145/3154815.
URL https://doi.org/10.1145/3154815

[9] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, Tech. rep., MIT Press (2015).

[10] T. Chavdarova, F. Fleuret, Deep Multi-camera People Detection, Proceedings - 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017 2017-December (2017) 848–853.
URL http://arxiv.org/abs/1702.04593

[11] R. Zhao, H. Ali, P. van der Smagt, Two-Stream RNN/CNN for Action Recognition in 3D Videos, IEEE International Conference on Intelligent Robots and Systems 2017-September (2017) 4260–4267. doi:10.1109/IROS.2017.8206288.
URL http://arxiv.org/abs/1703.09783http://arxiv.org/abs/1703.09783

[12] T. Xing, M. Roig Vilamala, L. Garcia, F. Cerutti, L. Kaplan, A. Preece, M. Srivastava, DeepCEP: Deep Complex Event Processing Using Distributed Multimodal Information, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 87–92. doi:10.1109/SMARTCOMP.2019.00034.
URL https://ieeexplore.ieee.org/document/8784054/

[13] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, K. Schindler, Online Multi-Target Tracking Using Recurrent Neural Networks, Tech. rep., Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17) (2017).
URL www.aaai.org

[14] K. Dutta, P. Krishnan, M. Mathew, C. V. Jawahar, Improving CNN-RNN hybrid networks for handwriting recognition, in: Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR, Vol. 2018-August, Institute of Electrical and Electronics Engineers Inc., 2018, pp. 80–85. doi:10.1109/ICFHR-2018.2018.00023.

[15] B. Shi, X. Bai, C. Yao, An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (11) (2017) 2298–2304. doi:10.1109/TPAMI.2016.2646371.

[16] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, W. Xu, CNN-RNN: A Unified Framework for Multi-label Image Classification, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2016-December, IEEE Computer Society, 2016, pp. 2285–2294. doi:10.1109/CVPR.2016.251.

[17] D. Li, T. Salonidis, N. V. Desai, M. C. Chuah, DeepCham: Collaborative Edge-Mediated Adaptive Deep Learning for Mobile Object Recognition, in: 2016 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2016, pp. 64–76. doi:10.1109/SEC.2016.38.
URL http://ieeexplore.ieee.org/document/7774674/

[18] D. Roy, G. Srinivasan, P. Panda, R. Tomsett, N. Desai, R. Ganti, K. Roy, Neural Networks at the Edge, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 45–50. doi:10.1109/SMARTCOMP.2019.00027.
URL https://ieeexplore.ieee.org/document/8784077/

[19] S. Teerapittayanon, B. McDanel, H. Kung, Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2017, pp. 328–339. doi:10.1109/ICDCS.2017.226.
URL http://ieeexplore.ieee.org/document/7979979/

[20] N. Nordlund, H. Kwon, L. Tassiulas, Cooperative Learning for Multi-perspective Image Classification, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 75–80. doi:10.1109/SMARTCOMP.2019.00032.
URL https://ieeexplore.ieee.org/document/8784012/

[21] P. Abudu, A. Markham, Distributed Communicating Neural Network Architecture for Smart Environments, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 233–240. doi:10.1109/

SMARTCOMP.2019.00058.
URL https://ieeexplore.ieee.org/document/8784048/

[22] D. Bruneo, F. De Vita, On the Use of LSTM Networks for Predictive Maintenance in Smart Industries, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 241–248. doi:10.1109/SMARTCOMP.2019.00059.
URL https://ieeexplore.ieee.org/document/8784003/

[23] T. Luo, S. G. Nagarajan, Distributed Anomaly Detection Using Autoencoder Neural Networks in WSN for IoT, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018, pp. 1–6. doi:10.1109/ICC.2018.8422402.
URL https://ieeexplore.ieee.org/document/8422402/

[24] J. N. Foerster, Y. M. Assael, N. de Freitas, S. Whiteson, Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks, arXiv.
URL http://arxiv.org/abs/1602.02672

[25] S. Sukhbaatar, A. Szlam, R. Fergus, Learning Multiagent Communication with Backpropagation, arXiv.
URL https://arxiv.org/pdf/1605.07736v1.pdf

[26] J. N. Foerster, Y. M. Assael, N. de Freitas, S. Whiteson, Learning to Communicate with Deep Multi-Agent Reinforcement Learning, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, p. 2145–2153.
URL http://arxiv.org/abs/1605.06676

[27] F. J. Provost, Scaling Up: Distributed Machine Learning with Cooperation., in: AAAI'96: Proceedings of the thirteenth national conference on Artificial intelligence, 1996, pp. 74–79.
URL http://www.aaai.org/Library/AAAI/1996/aaai96-011.php

[28] N. Agarwal, A. T. Suresh, F. Yu, S. Kumar, H. B. Mcmahan, cpSGD: Communication-efficient and differentially-private distributed SGD, Advances in Neural Information Processing Systems 2018-December (2018) 7564–7575.
URL http://arxiv.org/abs/1805.10559

[29] H. Wang, S. Sievert, Z. Charles, S. Liu, S. Wright, D. Papailiopoulos, ATOMO: Communication-efficient Learning via Atomic Sparsification, Advances in Neural Information Processing Systems 2018-December (2018) 9850–9861.
URL http://arxiv.org/abs/1806.04090

[30] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, F. Kawsar, An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices, in: Proceedings of the 2015 International Workshop on Internet of Things towards Applications - IoT-App '15, ACM Press, New York, New York, USA, 2015, pp. 7–12. doi:10.1145/2820975.2820980.
URL http://dl.acm.org/citation.cfm?doid=2820975.2820980

[31] J. Wu, C. Leng, Y. Wang, Q. Hu, J. Cheng, Quantized convolutional neural networks for mobile devices, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2016-December, IEEE Computer Society, 2016, pp. 4820–4828. doi:10.1109/CVPR.2016.521.

[32] S. W. Pienaar, R. Malekian, Human Activity Recognition Using LSTM-RNN Deep Neural Network Architecture, 2019 IEEE 2nd Wireless Africa Conference, WAC 2019 - Proceedings.
URL http://arxiv.org/abs/1905.00599

[33] Y. Hu, Y. Wong, W. Wei, Y. Du, M. Kankanhalli, W. Geng, A novel attention-based hybrid CNN-RNN architecture for sEMG-based gesture recognition, PLOS ONE 13 (10) (2018) e0206049. doi:10.1371/journal.pone.0206049.
URL https://dx.plos.org/10.1371/journal.pone.0206049

[34] Q. Yin, R. Zhang, X. Shao, CNN and RNN mixed model for image classification, MATEC Web of Conferences 277 (2019) 02001. doi:10.1051/MATECCONF/201927702001.

[35] G. E. Hinton, Improving neural networks by preventing co-adaptation of feature detectors, CoRR abs/1207.0580.
URL http://arxiv.org/abs/1207.0580

[36] D. P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, CoRR abs/1412.6980.
URL http://arxiv.org/abs/1412.6980

[37] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2323. doi:10.1109/5.726791.

[38] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536. doi:10.1038/323533a0.
URL http://www.nature.com/doifinder/10.1038/323533a0

[39] O. Durmaz Incel, A. Ghosh, B. Krishnamachari, K. Chintalapudi, Fast data collection in tree-based wireless sensor networks, IEEE Transactions on Mobile Computing 11 (1) (2012) 86–99. doi:10.1109/TMC.2011.22.