Deep Learning Model Reuse and Composition in Knowledge Centric Networking

Dolly Sapra
University Of Amsterdam
Amsterdam, Netherlands
d.sapra@uva.nl

Andy D. Pimentel
University Of Amsterdam
Amsterdam, Netherlands
a.d.pimentel@uva.nl

***Abstract*—Machine learning has inadvertently pioneered the transition of big data into big knowledge. Machine learning models absorb and incorporate knowledge from large scale data through training and can be regarded as a representation of the knowledge learnt. There are multitude of use cases where this acquired knowledge can be used to enhance future applications or speed up the training of new models. Yet, the efficient sharing, exploitation and reusability of this knowledge remains a challenge. In this paper we propose a framework for deep learning models that facilitates the reuse of model architectures, transfer coefficients between models for knowledge composition and updates, and apply compression and pruning techniques for efficient storage and communication. We discuss the framework and its application in the context of Knowledge Centric Networking (KCN) and demonstrate the framework potential through various experiments, i.e. when knowledge has to be updated to accommodate new (raw) data or to reduce complexity.**

***Index Terms*—Knowledge Centric Networking, Deep Models, Knowledge Reuse**

I. Introduction

In the age of the Internet of Things (IoT), where we have a complex network of connected devices, sensors and computing units, there is a large amount of data churning up every minute. As these networks grow with more devices and users, the rapid growth of data can overwhelm the underlying communication channels and resources. The data can be highly redundant and obtaining data might not be the end objective in such networks. Converting data to knowledge allows utilization in beneficent ways for different tasks, such as visual monitoring in smart homes, remote assistance in medical care or analyzing environmental data for agricultural use. Machine learning is increasingly being deployed to convert raw data into meaningful knowledge. This conversion is majorly performed on a central server or knowledge creator node with high computation capabilities, which can create bottlenecks with high volumes of data communication. To overcome these issues, Knowledge Centric Networking (KCN) was conceptualized in [1], which proposes a paradigm shift, in a network, from data centric communication to knowledge centric communication. KCN emphasizes three key aspects about knowledge: creation, composition and distribution.

Deep learning models or Neural Networks (NNs) are a popular knowledge modality for big data, storing the knowledge in the form of a brain-like architecture and thousands to millions of coefficients, which are trained from large amount of data.

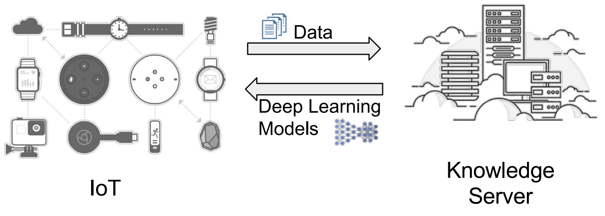

Fig. 1. Traditional IoT network with centralized knowledge server for intelligence. Deep learning models are trained centrally using data from devices and then distributed.

They can be viewed as assimilating and extracting knowledge by storing data statistics and domain specific characteristics through training. Figure 1 illustrates how IoT networks and deep learning models are deployed and used in the traditional sense. There is a central knowledge server that creates deep learning models from device data, analyzes new incoming data and decides when to update the knowledge model. This knowledge server frequently distributes appropriate deep learning models to all devices, in order to promote the use of better and more intelligent applications. KCN visualizes the knowledge creation at the edge wherever possible, and these deep learning models serve as the basic communication paradigm. New dynamic data is continuously sensed and devices are added to the network all the time. Sharing the assimilated knowledge models, building newer knowledge models on top of existing ones, and distributing decision making capabilities are aspired as this would make the network more resilient and adaptable to changes over the course of its active lifetime.

In such a distributed intelligent network, there is a need for frequent knowledge updates. However, we are not aware of any work on standardizing the knowledge update process. In a domain specific task, the high correlation of deep learning models can be exploited by different devices to reuse and combine each other's knowledge to create a better application. Standardizing the knowledge update process with the aim of reusing this high correlation, motivates the creation of our framework. Our framework coordinates efficient communication, exchange and update of deep learning models, while being adaptable to accommodate new data generated and insights learnt.

We differ from domain adaptation [2], knowledge distillation [3], and similar teacher-student algorithms [4], which attempt to create new models for similar tasks in different domains or constraints. In essence, these techniques can be used with our framework to simplify the creation of student models and streamlining their distribution through the network.

In short, our framework facilitates creation of a new network and modification of existing ones, allows combination of multiple models to compose a new model, replaces part(s) of a model with other sub-model(s), isolates model layers that can be individually transferred while supporting packaging and compression for distribution. There are multitude of ways in which deep learning models can be modified, both weights and architecture of the neural network can be updated, e.g. add/prune layers, add residual connections, change layer activation. All of these tasks are unrelated to the training of models or knowledge creation directly, even so, these tasks are needed to keep the network's knowledge contemporary and maintain communication brevity with frequent updates in the context of KCN.

The contributions of this paper are fourfold:

1) A framework for deep learning models to facilitate knowledge exchange, reuse and frequent updates.
2) Discussion of various knowledge update techniques from existing literature and formulating them in the context of Knowledge Centric Networking.
3) A model partition and isolation paradigm to distribute only the relevant slices of the deep learning model.
4) Validating the framework with experiments to update knowledge models by varying data information over time as well as model complexity.

The rest of the paper is organized as follows. We introduce preliminary concepts and background of our work in Section II. We then present various knowledge update techniques and how they are incorporated in our framework in Section III and results of our evaluations and validations in Section IV. Finally, we discuss related works in Section V and conclude the paper in Section VI.

II. Background

In this section, we discuss the core ideas and terminologies which form the foundation of our current work. We discuss how IoT and deep learning are used together. Then, we introduce KCN and discuss its communication model. Finally we talk about interoperability among different devices in an IoT network.

A. IoT and Deep Learning

The Internet of Things (IoT) is the network of devices that embeds technology to sense the external environment and perform some predefined tasks. These devices are capable of communicating with the physical world, virtual (computing and network environment) objects as well as with each other. They may be similar in terms of data collected or the task they are performing. For example, in a video surveillance environment, most devices will be cameras with facial recognition capabilities using similar knowledge models. In some systems, different devices are collecting different types of data but sharing the same goal. For instance, in a health monitoring system with on-body sensors, each sensor is measuring a different function like heartbeat rate or body movement but they need to combine their data and knowledge in a way to achieve a health overview.

As these IoT devices may generate a lot of data, global Machine to Machine (M2M) IP traffic will grow more than seven-fold in 5 years, from 3.7EB per month in 2017 to more than 25EB by 2022, as projected by [5]. These enormous amounts of data generated cannot be analyzed by humans, therefore the need for artificial intelligence models such as neural networks to be utilized in data analysis and knowledge extraction. In neural networks, the input data pass through multiple layers in sequence, wherein each layer performs matrix multiplications on the data. The final layer of this neural network is a predictor or a classifier for the expected output. An example of a neural network is illustrated in Figure 2. When the model contains many layers, the neural network is known as a deep neural network (DNN) or deep learning model. Convolutional Neural Networks (CNNs), e.g. Alexnet [6], Resnet [7], are a special case of DNNs where matrix multiplications include convolutional filter operations designed for image and video analysis. There are other types of DNNs designed for variety of tasks; an overview of different types of deep learning models can be found in [8].

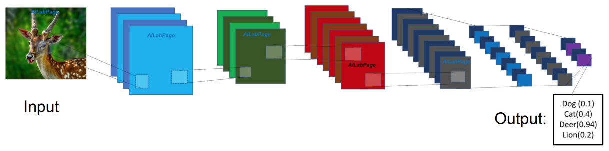

Fig. 2. Deep neural network example for image classification

There are a large number of parameters in the matrix multiplications, called model weights or coefficients, which are estimated during training on data using stochastic gradient descent [9]. This results in many computations being performed and some devices do not have enough hardware resources to cope with all the incoming data in a reasonable amount of time. However, once computed the size of total coefficients together is a tiny percentage of all the data generated and analyzed over a period of time. There are other design choices, called *hyperparameters*, to specify the deep learning model architecture (like parameters of each layer, and the number of layers). This model architecture and coefficients together form the knowledge modality that we consider as the source of intelligence in IoT systems.

B. Knowledge Centric Networking (KCN)

In traditional IoT networks with cloud computing support, all devices transfer the data to the cloud and delegate heavy computational tasks to the central server. Data then converge and are progressively used for model training and fine tuning. This approach implicates heavy data transmission and storage

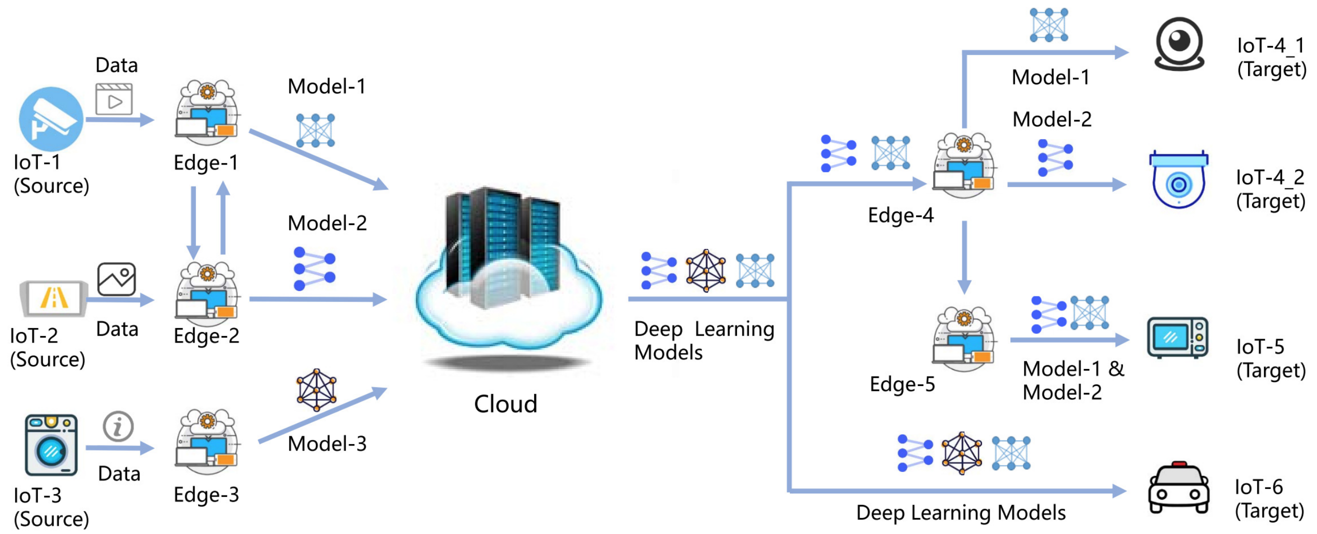

Fig. 3. KCN paradigm on IoT network with edge computing. Deep learning models can be created on edge and transmitted from edge to edge, edge to cloud as well as edge to IoT device [10].

requirements on the network. The huge amount of data threatening to throttle the network has prompted various research ideas, e.g. data compression and quantization techniques [11], [12]. Despite very efficient data compression techniques, the network still can be overwhelmed to meet the demand for high efficiency with highly redundant data transmissions. A novel concept of KCN was envisioned and proposed in [1], which emphasises the communication of knowledge models instead of raw data. IoT devices continuously sensing the environment and generating huge amounts of temporal data, can use additional edge devices, which are closer to them or maybe embedded along with sensors in the IoT device, for knowledge extraction. KCN is based on the Edge computing paradigm [13], which promotes more computation on edge devices and less on the cloud, thereby reducing costs for data bandwidth and storage. Figure 3 illustrates this concept: deep learning models can be created and updated at the edge and transmitted to other IoT devices, edge devices or cloud. Besides improving latency and scalability of the system, KCN also reduces exposure to privacy and security attacks by removing the private or sensitive data moving around in the connected network.

This network allows different granularity and hierarchy of knowledge creation, re-composition, update and exchange. Edge nodes can extract local knowledge from the sensed data and then upload to the cloud. The cloud server collects models from different devices and can perform composite knowledge updates. It can then further distribute the updated models to interactive and decision making devices, which might be considered as front-ends of the system. Edge nodes can also request generated models at another edge node directly to perform its own task efficiently. This results in frequent knowledge communication in the network as well as recurring updates to knowledge models stored at different devices.

C. *Interoperability*

In any environment with multiple products and systems, it is desired that all interfaces are well understood and are capable of comprehending each other. There are many powerful deep learning languages, toolsets and frameworks and it is possible to have an environment where all of these are present in at least one of the devices. Further optimizing deep learning models for specific hardware is difficult and since each one (cloud/edge, CPU/GPU, etc.) has different capabilities and characteristics, the problem becomes extremely hard and complex. Models from a variety of frameworks need to run on a variety of platforms. It is very time consuming to optimize all the different combinations of frameworks and hardware. A solution to train in any framework but being able to communicate anywhere on the cloud or edge is needed. Keeping this in mind, our framework is built using the Open Neural Network eXchange framework (ONNX) [14]. ONNX is well suited for this task as it encapsulates architecture and coefficients in a single modality and is widely seen as a solution to the interoperability problem concerning different deep learning frameworks. Most of them already allow exporting or converting the model to the ONNX format, such as PyTorch [15], Caffe2 [16], Apache MXNet [17], Microsoft Cognitive Toolkit [18]. Converted models to ONNX can run on a variety of platforms and devices directly using ONNX Runtime [19]. In the context of KCN, by standardizing the exchange format, we simplify the design and implementation of dynamically evolving deep learning models.

III. Framework Design

In this section, we present our framework and its features, while discussing the design choices that we contemplate to work best in different scenarios that can occur in knowledge-based IoT networks. There is a plethora of algorithms and techniques available in the literature that modify a neural network in different possible ways. We draw some techniques from this pool for their suitability to KCN. We not only describe different forms of model modifications in this section but also envision how and where they can be used for efficient knowledge exchange. We specify which techniques are supported by our framework and we try to point out their applicability and possible use case settings. The novelty of this framework lies in the detailed study and consolidation of deep learning modification and communication techniques applicable to a progressive learning and frequent update paradigm. We recognize that some of our work is about picturing various scenarios in the context of KCN and building a framework around it. We believe that formulating knowledge composition, update and exchange methodologies will cater to the dynamic IoT network to be better serviced for a longer period of time and it is a step forward towards moving the KCN paradigm from a vision to reality.

A. Coefficients Update

The simplest form of knowledge update occurs in a neural network by training the existing model with more data, which is of similar nature as presented before. The model architecture remains exactly the same, yet all coefficients get updated to reflect knowledge from new data. Our framework implicitly handles the coefficients update by updating the model file with new weight values after training. Training with new data is done at the edge and it is expected that this operation is recurrent in nature on most, if not all, edge devices. With the knowledge model being the important storage and communication entity, the data is expected to be discarded after training. The frequency of training is driven by storage capacities at the edge devices and training can be triggered with sufficient accumulated data. Depending on data type and format, there may be a data cleanup and pre-processing pipeline in place before the training operation. The updated model is then available to be distributed or exchanged through the network as needed.

B. Architecture Update

Designing a neural network is not trivial, and not all architectures are equal in terms of their capacities and knowledge representation capabilities. Deeper networks, with a higher number of layers, allow a more complex and non-linear function to be learnt from data. However, apart from needing huge amounts of training data to be able to meaningfully learn, they require a large memory and powerful computation units, usually lacking in edge devices. There exists a performance-resource trade off in embedded and low power systems, which is usually dissimilar for different types of devices. While exchanging deep learning models between different nodes, there is a chance that at the receiving device, the model is too large or demands more computation resources than available, setting off a need to reduce the model complexity. In some other cases, the initial data available is not enough to train a large model, so a small model may be built to kick start the knowledge extraction process which also prevents over-fitting on the small data-set at the same time. A larger model is built later on to fully utilize all data that has been sensed, which can be fabricated by expanding the current model capacity instead of training from scratch. This progressive expansion is also expected to reduce the upfront overhead of computational costs that training a large model entails.

In all feasible and available scenarios for architecture update, it is assumed vital for a model to be updated in a function preserving fashion, so that the model does not leak its knowledge. All algorithms implemented in the framework have been chosen to consider either the function preservation or minimal loss possible. Some of them were motivated by the genetic operators in evolutionary neural architecture search [20], where function preservation is crucial. The small loss of performance with a major update is expected to be gained back by training more and more as new data keep arriving around the clock. Our framework handles architecture updates by considering each layer as a named node and storing its coefficients as a separate but connected block. Each node can be isolated and its parameters, coefficients and formats updated individually. It automatically checks for data format changes to be done in subsequent layers of the modified node to keep the model valid and consistent. The named nodes are important and should be unique to be able to be used as identifiers for all update and modification operations.

1) Increasing the model capacity: Out of all possible ways to increase the model's capacity, our framework currently handles two ways to increase the capacity of the model: increase the number of layers of the neural network or increase the number of units or neurons in each or some of the layers. Our framework emulates Net2Net [21] for function preserving expansion of the network. There are two operations available in Net2Net: Net2Deeper and Net2Wider, to increase neural network depth and layer size respectively. Any convolutional or fully connected layer is replaced by two layers of the same type and size, with one layer having the same coefficients as the original layer and the new layer's coefficient matrix as initialized to an identity matrix. To increase the number of units in each layer, the coefficient matrix is expanded to the required size and then random layer units are selected and duplicated in the expanded coefficient matrix. Any increase in number of units causes the output of the layer to increase by the same amount, which means that there is a parallel increase in the number of inputs to the subsequent layer. This causes the coefficient matrix to expand as well, which is then appended with randomly initialized values. These new layers and units remain free to train further to take on any value later. Hence, the effect of increasing layers and units in this way is only to provide a good initialization to the newly created snippets in the knowledge. With further training on the data, the coefficients gets updated to preserve any new knowledge and slowly diverge from their initial values.

2) Decreasing the model capacity: Contrary to the capacity increasing operations, it is not beneficial to delete layers or sub-units of the model in an ad-hoc manner. Each layer of the model holds information or features that subsequent layers use to build their own sub-knowledge. Various pruning techniques have been proposed in the last decades to reduce model complexity, redundancy and over-fitting. In recent works, [22] and [23] suggested to remove all connections whose weight was lower than a threshold and retrain the rest of the network to fine-tune the model again. This approach leads to a reduction in model size to the tune of $9\times - 13\times$ without any loss of accuracy but leads to lightly populated coefficient matrices. This reduces memory footprint of the model but fails to reduce the computation cost because of irregular sparsity in the pruned network. To overcome this issue, our framework emulates the *independent pruning* technique from [24]. This approach prunes the layer size by removing the least important units of a layer. The relative importance of a unit in each layer is calculated by the sum of its absolute weights. This value gives an expectation of the magnitude of the output of each unit, also called a feature map. Feature maps with smaller weights tend to produce outputs with weak activations when compared to the other units in that layer. Based on the target reduction size, the units with the smallest sum values and their corresponding feature maps are removed from the model. To regain the accuracy that was lost by pruning, a prune-then-retrain strategy needs to be adopted. In our framework, we prune filters of multiple layers at once followed by further training, though it is also possible to iteratively prune small slices of the model and retrain the network repeatedly. The iterative process may yield better results, but it requires more data and training epochs to reach original performance.

Another way in which our framework decreases the network capacity is by performing quantization on the coefficient values. Quantization refers to the process of reducing the number of bits that represent a number. In the context of deep learning for research and deployment, the predominant numerical format used has been 32-bits floating point. Our framework currently converts 32-bits floating point numbers to 16-bits, thereby halving the memory requirements for the model on devices. As shown by many research works, replacement of high precision numbers by lower-precision numerical formats can be done without incurring significant loss in accuracy [25], [26], thus preserving most of the knowledge encapsulated in the converted model.

3) Replacing Layers: Our framework allows for replacing a layer block in a model with a layer block from another model, which was trained on similar data but on another device. For this replacement to work, the input and output format of the switching block have to be same in both source and target models while the size of the block can be different. Figure 4 illustrates the layer replacing approach in our framework. If model complexity reduction is desired, a block of n layers is replaced by a block of layers of size $<n$ from the source model, given the layers are at the same cluster position as defined in [27], so that the input and output formats of the layer blocks are identical . The inverse action is also possible to increase model capacity. This scenario in KCN is plausible when multiple knowledge models in the IoT network are

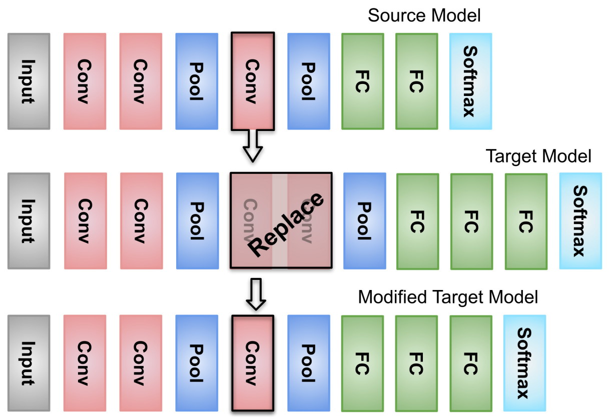

Fig. 4. Example of layer replacement. A layer from the source model replaces a similar layer section in the target model to achieve a new model containing fewer layers.

trained on same data but resulted in different deep learning models based on each device's own optimization for model creation and composition over the course of time. Accuracy lost by performing this action can be regained by retraining with more data, though sometimes the new model might never be as proficient as the old one.

The model complexity of neural networks can also be reduced by replacing a fully connected layer of a convolutional neural network with the Global Average Pooling layer. The network in Network architecture [28] and GoogLenet [29] achieve state-of-the-art results on several benchmarks by adopting this idea. Within our framework, replacing layers is achieved by creating a new model using the relevant named nodes from source models, followed by an associated coefficients copy operation. Coefficients for nodes which cannot be found in the source model are then randomly initialized, these are essentially now the new layers of our model.

C. *Activation function update*

Our framework allows changes to other parameters such as layer activations, drop-out units and normalization techniques as well. Activation functions are used for introducing non-linearity into the neural network model so that the network can progressively learn more effective feature representations. Rectified Linear Units(ReLU) [30]–[32] are the most popular activations as deep networks with ReLUs are more easily optimized than networks with *sigmoid* or *tanh* units [33]. Figure 5 shows some popular activation functions.

Changing an activation function in the neural network has a huge impact on the model behavior and changing them for an already trained models is not a good idea. However, changing the activation function from related functions is desirable in some cases, such as using Leaky-ReLU will avoid the *dead ReLU problem* which happens when the ReLU activation always have values under 0, which completely blocks further learning. Concatenated RelU [34] and Parametric Rectified Linear Unit (PReLU) [35] are proposed to reduce redundancy

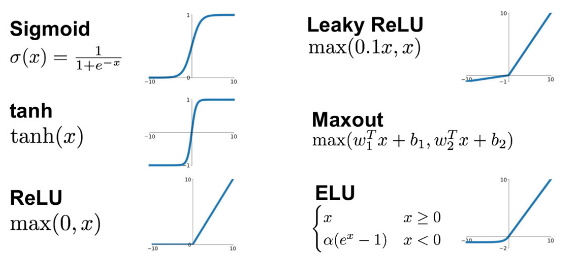

Fig. 5. Popular activation functions for neural networks.

and better generalize the traditional ReLU. In principle, changing the activation function is carried out in our framework by changing node parameters. However, the more disparate and dissimilar activation functions are, the more knowledge are lost in the process.

D. Multi-source knowledge fusion

Our framework facilitates fusion of multiple deep learning models to assimilate knowledge from different sources into a larger model encapsulating *the bigger picture*. By using the same layer node names as source nodes, the fused model represents replicas of the parts of different models joined together within the new model. After the new model is created, it looks for associated source layer coefficients to be copied for any computation needed on it.

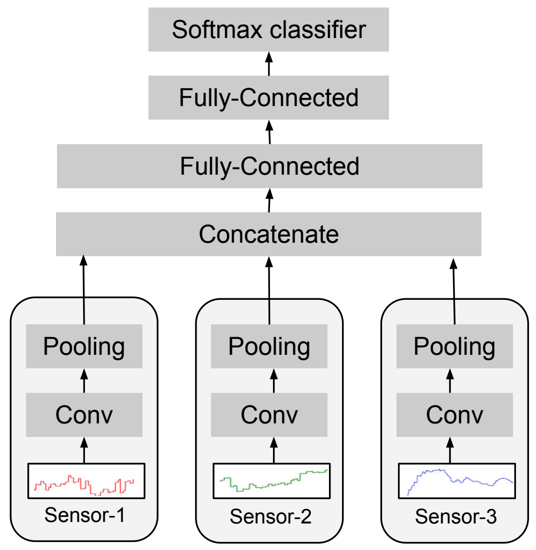

Fig. 6. Knowledge fusion model with different sensors in Human Activity Recognition (HAR). Each time series data from sensors is locally processed and fused together at a central node to extract inter-sensor relations to detect human activity [36].

This feature is useful in IoT networks with temporal sensing such as with Human Activity Recognition (HAR) [36]–[38], which is a vital step in an application like skill assessment and smart home assistant. Each sensor has its own knowledge extraction module to analyze and obtain the local salient features from the data. This knowledge is pooled centrally

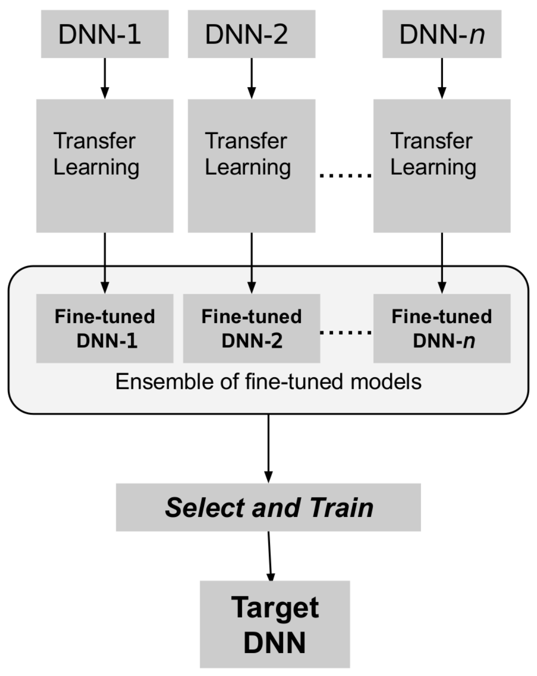

Fig. 7. Knowledge fusion model with multi-source transfer learning using an ensemble of deep models from different sensors/devices. Knowledge is transferred from each source database to a target model. A selection process combines models into an ensemble that is used to train a single randomly initialized neural network.

and processed further to extract inter-sensor dependencies and detect associated human activity in the given time period. Figure 6 illustrates the concept of multi-source knowledge fusion with deep learning models for time series input sensors.

In the context of KCN, this kind of model will have difficulties to start knowledge extraction locally because all the sensor data needs to be trained together to extract knowledge about both independent and inter-dependent features for a meaningful activity recognition. All sensors might be served by same edge node to collect data and start training the model. Once sufficient performance is reached, the model parts might be isolated and distributed to relevant sensors. The sensors can then convert their local raw data into knowledge using their own sub-model and send their knowledge to the same edge node again to be fused with knowledge from other nodes. Our framework allows layer isolation and model fusion to seamlessly execute this possible workflow.

Another example of knowledge fusion, called multi-source transfer learning [39], [40] is currently popular in medical image analysis. It is based on an ensemble of models that are each created using single source transfer learning from a variety of domains with similar data characteristics. In single source transfer learning a number of consecutive layers are transferred from a chosen pre-trained model (teacher) to initialize its counterpart target model (student). The rest of the student model is created anew and randomly initialized. The layers obtained from the teacher are usually frozen and the student model is trained on the target data-set to be fine-tuned for the intended domain. In our framework, single source transfer learning is realized by duplicating the source model, freezing the coefficients of appropriate layers and then replacing the rest of the layers with new randomly initialized replacement layers. After re-training and fine tuning for the target task, the new models can be saved at the central server.

An ensemble of all of the student models is then used to train another model which is essentially now being generated by training from a knowledge ensemble instead of raw data. Figure 7 shows the multi-source transfer learning methodology. In KCN, this paradigm is very useful when a new device is added to the network, allowing it to gain knowledge from already available intelligence in the network and does not need to wait for a lot of data to be collected before being fully utilized and deployed in its intended task. Our framework does not perform this training but facilitates the creation of ensembles of relevant student models. Using the ONNX file format, any available or desired deep learning framework can be used to train from the model ensemble.

E. *Isolation and compression*

As previously mentioned, our framework is capable of isolating layers and their coefficients for all types of modifications done on the models. Compression and packaging of raw coefficients data are vital for efficient transmission and exchange throughout the system, especially when distributing only a part of the model. We implemented basic support for general purpose compression and decompression using the popular ZLIB compression library [41]. But, other compression libraries can also be added to extend our framework.

IV. Validations

In this section, we validate the framework and its viability to act as a knowledge modification and update environment. We present the framework setup followed by two scenarios, a smart camera network and a multi-sensor network, simulating the KCN environment. We discuss different use cases for the framework within these scenarios.

We implemented the framework using Java 8 and the Protocol Buffers (protobuf) library [42] to build the ONNX components. Our framework reads ONNX files and then alters them as per the requested use case and writes the updated ONNX files onto the storage system again. For further training and validation, we imported the ONNX files into the Python based Caffe2 framework (from the Pytorch library) [15]. Our framework itself is reasonably lightweight and runs without any GPU support, though we utilized a GeForce RTX 2080 [43] GPU to train all the deep models.

A. *Smart camera network for object recognition*

We simulated a KCN environment for a smart camera network for object recognition as outlined in various other frameworks as well [44]–[46] (see Figure 8). In the simulated environment, each camera is represented by a unique convolutional model called a *cam-model*. In other words, each *cam-model* in the environment appears for a virtual camera and

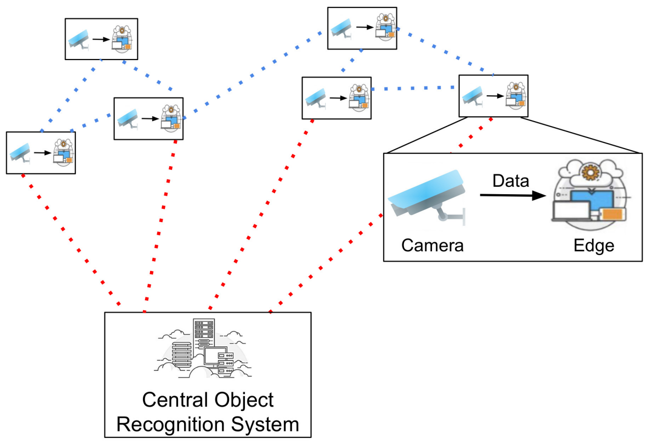

Fig. 8. Smart camera network with a central knowledge server.

defines its object recognition capabilities. In the experimental setup, we trained *cam-models* on CIFAR-10 [47], which is a popular object recognition dataset. CIFAR-10 consists of 32×32 pixels RGB images classified into 10 categories and is further divided into two sets of 50000 training and 10000 test images, to train and validate the model respectively.

A *cam-model* is an assembly of multiple convolutional layers interspersed with two maxpool layers for input size reduction, followed by fully connected layers. The convolutional layers are also termed as feature extractor and its main role is to convert raw image pixel data into meaningful features of the input image. To correctly classify the image, the feature information is then passed through a series of fully connected layers, also called as a classifier. Figure 9 illustrates a basic neural network structure used in the current setup. Each *cam-model* has its own distinct topology where the number of convolutional and fully connected layers as well as layer parameters such as the number of units and kernel sizes are randomly sampled taking the pre-defined constraints into account. This is to reflect that in a real smart camera network, cameras added over a long period of time will have different local compute and storage capabilities and might have been initialized with a distinctive deep learning model.

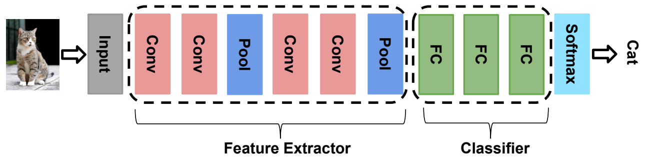

Fig. 9. Deep neural network depicting convolutional feature extractor and fully connected classifier.

All models were built with ReLU activations and were trained (and re-trained) with a learning rate of 0.0005 and batch size of 90 using the Adam optimizer. We restricted the GPU memory usage to 5GB during training to limit the size of Neural Network from becoming too large. We discuss some use cases for our framework below:

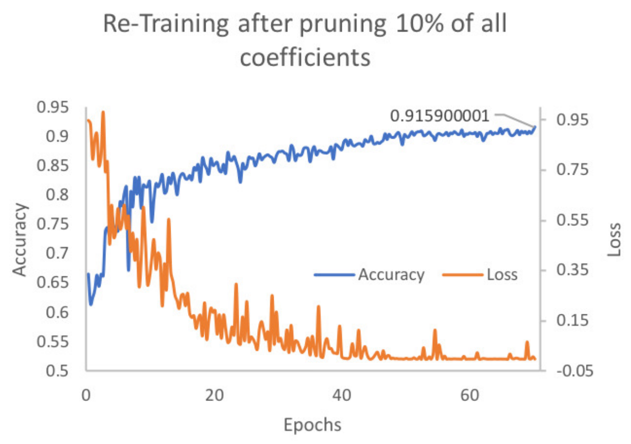

Fig. 10. Training curves for pruned model by 10%

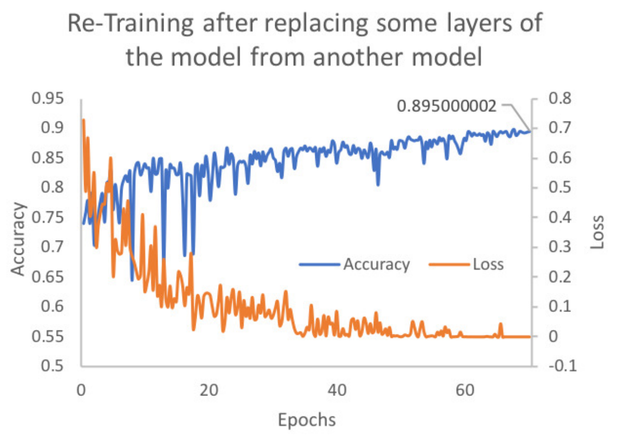

Fig. 11. Training curves for modified model with some convolutional layers replaced from another pre-trained model

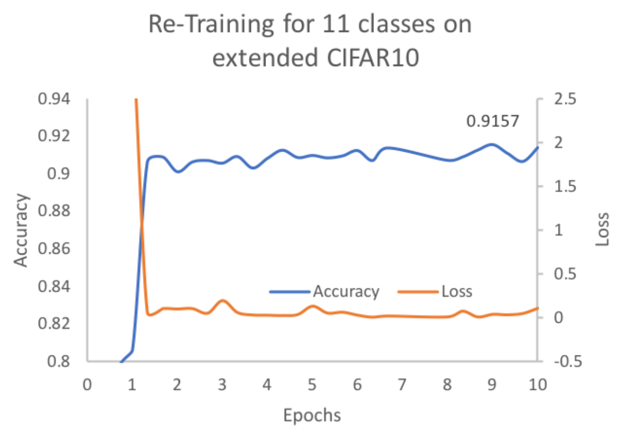

Fig. 12. Training curves for modified pre-trained model to include a new class "Flower" to the existing CIFAR-10 dataset.

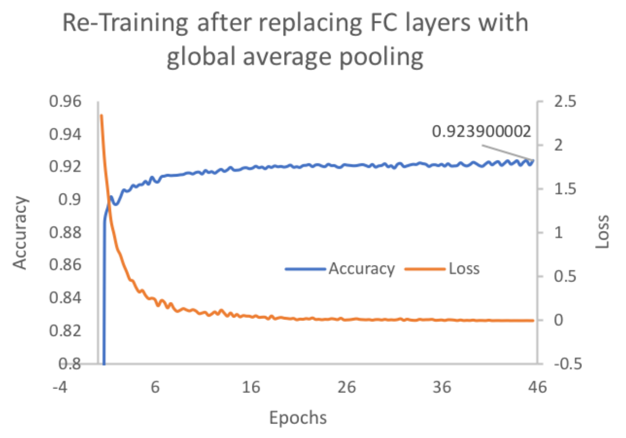

Fig. 13. Training curves for modified pre-trained model with Global Average Pooling as classifier, replacing fully connected layers.

1) Pruning a model by 10%*:* Memory is usually limited on an edge device and pruning is a very efficient technique to reduce the model's memory footprint. To demonstrate that performance is not degraded when removing redundant information, we performed this experiment on a *cam-model* comprising 13 convolutional layers and 3 fully connected layers having a total of $\approx$10 million coefficients.

We pruned the network to reduce all weights by 10%, resulting in a new model with $\approx$9 million coefficients. The resulting model uses less memory while also reducing the number of MAC (Multiply-Accumulate) operations needed in each run. The original model has 91.4% accuracy while the pruned model actually performed slightly better with 91.59% accuracy after re-training, see Figure 10. The increase in performance is caused by weak activations removal which were not contributing considerably to the model intelligence.

2) Model composition from two neural networks : For this use case, we chose a *cam-model* with 10 wide convolutional layers and 3 fully connected layers ($\approx$ 19 million coefficients) with test accuracy of 90.81% with the aim of reducing storage size of the model on the edge. We picked another *cam-model* with 13 (smaller) convolutional and 2 fully connected layers ($\approx$ 12 million coefficients) having an accuracy of 92.09%.

We chose a block of 2 convolutional layers from the latter model and used it to replace a block of 4 convolutional layers in the first one. All the blocks that were selected were operating on same input and output dimensions and were roughly in a similar phase of feature extraction. The resulting model now has 8 convolutional layers and 3 fully connected layers with $\approx$ 14 million coefficients. We re-trained it further and the new model was able to achieve 89.5% accuracy, which is much less than its parent. Figure 11 shows the related training curves. This illustrates the point that composing a model from two different models is not always a preserving function, however the benefit is still observed by lowering the model complexity through a reduction in the number of layers, amount of arithmetic computations and storage size.

3) Increasing the number of output classes: As mentioned above, there are 10 output classes for the CIFAR-10 dataset. There are possible scenarios where sensed data or the environment has evolved and there is a need to define an extra output class. It is not desirable to train from scratch, especially when original data were not saved in the system. We added a class "Flower" to an existing, pre-trained model on CIFAR-10. The number of images available for the new class were kept at half of existing class samples to reflect the fact that in a dynamic environment new output classes will not be equally represented, specially in the early stages of new data being sensed. To fine tune the model, we preserved the feature extractor and expanded the last fully connected layer in the classifier to include new output, totalling the number of outputs to 11. The new model was fine-tuned by training with an input data set containing all 11 classes, though only the last few layers were available to be updated as we froze the feature extractor.

The randomly chosen *cam-model*, consists of 15 convolutional layers and 3 fully connected layers with 92.6% accuracy (with 10 classes). After only 10 epochs training, the reached accuracy for the extended CIFAR-10 dataset is 91.57%, see Figure 12.

We did not observe an accuracy increase after 10 epochs. And we noticed that there is a loss of performance, but handling the trade-off between speed of knowledge update and best performance achievable is a complex task to fulfill. The trade-off will generally vary with different data types, model complexities and device computation capabilities. As future work, the decision choices regarding how long to train and how many layers to freeze might also be dynamically incorporated into the framework to get the best model in terms of its performance when data characteristic changes over a long period of time.

4) Replacing layers: As discussed in Section III, replacing *heavy* fully connected layers with Global Average Pooling decreases the model capacity in architecture update, and reduces the computation cost. We took the same *cam-model* as above (with 10 output classes) and replaced the fully connected layers based classifier with a Global Average Pooling layer. In the given network it resulted in $\approx$ 1.5 million fewer multiply-accumulate operations, which leads to faster inference, along with lower power consumption, computation and memory demands on the device.

We re-trained the new network until the loss became stagnant, which was at 45 epochs. Original model accuracy is 92.6%, and even though the new model has fewer coefficients, it displayed a very small performance degradation by reaching an accuracy of 92.39%, see Figure 13.

The last two graphs (Figures 12-13) are noticeably smoother than the first two because the feature extractor was frozen in these two experiments and only the classifier part of the network was actually re-trained.

B. Multi-sensor based activity recognition

This scenario demonstrates the use cases based on multi-source knowledge fusion. We performed an activity recognition task based on the PAMAP2 dataset [48], which provides data recordings from four sensors, 13 channels each from three Inertial Measurement Units (IMU) and a single channel from a heart rate monitor. All these sensors are body worn and are on distinct locations such as hand, chest and ankle, jointly forming a small network which also involves communicating sensor data to a central controller which recognizes the activity being performed. The setup in our experiment is based on the CNN-IMU architecture from [49] as shown in Figure 14.

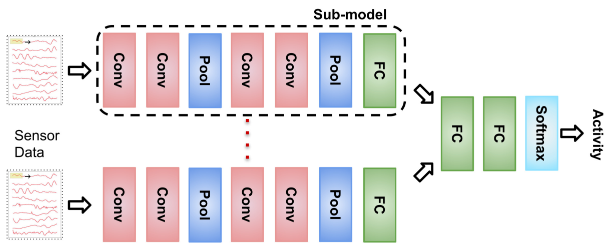

Fig. 14. Activity Recognition based on multiple sensor data. Each sensor has a convolutional sub-model, fused at the end with fully connected layers.

Each sensor has its own branch of four convolutional layers intermixed with two maxpool layers, followed by a fully connected layer. The output from these branches is concatenated and goes through fully connected layers that predict the activity. The whole network is trained together at once with the RMSProp optimizer using a batch size of 50 and learning rate of 10^{-4} after downsampling the IMUs recordings to 30 Hz and a sliding window of 3s (100 samples) and a step size of 660ms (22 samples).

1) Isolating branches: Using our framework, we isolated sub models from the branches and their respective layers into individual ONNX files. The sub model can be deployed close to the sensor itself, hence removing the need for sensors to transmit all the data to the central controller. For a 13 channel IMU, data needed to analyse each 3s window in our setup is equivalent to approximately 8kBps. By computing the sub models close to the sensor, only the output of the last layer is sent over to the central server, which is approximately 3kBps, resulting in a 60% decrease of bandwidth requirement. The bandwidth saving becomes more important when there are many sensors in the network.

V. Related Work

The concept of incorporating knowledge based communication into networking is not new. There have been numerous works proposing the idea, such as [50]–[52]. However, the focus of these works is towards network communication analysis and control, while largely ignoring the utilization of knowledge modality in all other aspects of an IoT network. In direct contrast, KCN perceives the knowledge as the chief content operating in the network, from front-end to back-end and from sensing to action. Knowledge creation, composition and distribution are the primary expected features of every device on the network.

Some works proposed in recent years investigate various different aspects of the knowledge centric paradigm to further the research on KCN infrastructure. For knowledge distribution in KCN, [10] investigated inter-model compression for compact representation of models to further reduce the data bandwidth requirements. Our works are similar in respect that we believe interoperability is the forefront of knowledge communication and exchange, but diverge very quickly in the scope they investigated. Our focus is on knowledge modification as opposed to knowledge compression for distribution. Deep learning model compression is a small part of our framework whose aim is to allow coefficients isolation and efficient packaging in scenarios where only some parts of the model need to be exchanged.

Foreseeing the problem of the knowledge-based forwarding on the knowledge router, [53] proposed a novel data structure for the knowledge routing table index. This results in faster movement of knowledge on a physical network and its efficient routing in terms of shorter latency, low memory consumption, and fast routing table update. On similar lines, [54] proposed an intelligent routing algorithm for knowledge models in KCN based vehicular ad hoc networks. These works, though dissimilar to ours, are vital to achieve KCN deployment in physical IoT networks. These works strengthen the viability of a KCN infrastructure implementation and in turn makes our framework more practical and serviceable in a KCN based system.

The concepts of lifelong learning and never-ending learning [55], [56] have also been around since a while. These ideas focus on individual models to be better learners of a variety of data types *by being able to learn how to learn*, and thus keep evolving with the evolving environment. Even though the concepts have been proposed for individual models, they can also be reformulated for the KCN paradigm where the dynamic intelligence of the network is able to improve implicitly over the course of time.

The implementation of systems that will last a long time is also possible through progressive learning methodologies for neural networks such as for multi-class classification [57], face recognition [58] and Speech Enhancement [59]. In progressive learning, the neural network starts learning from a small set of data but can expand automatically on introduction of new classes while still retaining the knowledge of previous classes. Unlike our framework, these techniques let the model to grow with each update and do not perform model reduction or consider power-memory constraints that are common with devices in the IoT network.

VI. Conclusion

In this paper, we presented a framework for deep learning models that facilitates knowledge update, composition and reuse in the scope of KCN. We emphasized that our framework can be used to allow deep learning models in a IoT networks context for dynamic knowledge modality. We envisioned multiple possible scenarios where neural networks will need to be remodeled to suit evolving intelligence of the system and discussed ways to solve some of those scenarios with an appropriate methodology. We also demonstrated with evaluations that our framework is able to update models in a variety of scenarios that are likely to occur in KCN. We showed that it is possible to use neural networks as a dynamic knowledge modality, which can be continuously modified and maintained in line with dynamic system behaviors and changing requirements.

With significant recent advances in deep learning models and deployment of more and more IoT devices, it is probable that KCN will become the key to control the *too much data* problem. There are still some open problems that need to be solved in order to see KCN being deployed in reality. Specifically, knowledge creation and modification strategies that are geared towards low-power embedded devices and real time constraints. As future work we plan to investigate these issues and extend our framework to integrate resource/performance trade-offs based model modification techniques and faster update mechanisms for real time requirements.

Acknowledgement

This project has received funding from the European Unions Horizon 2020 Research and Innovation programme under grant agreement No. 780788.

References

[1] D. Wu, Z. Li, J. Wang, Y. Zheng, M. Li, and Q. Huang, "Vision and challenges for knowledge centric networking," *IEEE Wireless Communications*, 2019.

[2] M. Ghifary, W. B. Kleijn, and M. Zhang, "Domain adaptive neural networks for object recognition," in *Pacific Rim international conference on artificial intelligence*. Springer, 2014, pp. 898–904.

[3] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[4] J. Li, M. L. Seltzer, X. Wang, R. Zhao, and Y. Gong, "Large-scale domain adaptation via teacher-student learning," *arXiv preprint arXiv:1708.05466*, 2017.

[5] V. Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, vol. 1, 2018.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[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*, 2016, pp. 770–778.

[8] F. Van Veen, "The neural network zoo," *The Asimov Institute*, 2016.

[9] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT*. Springer, 2010, pp. 177–186.

[10] Z. Chen, L.-Y. Duan, S. Wang, Y. Lou, T. Huang, D. O. Wu, and W. Gao, "Toward knowledge as a service over networks: A deep learning model communication paradigm," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1349–1363, 2019.

[11] C. Feng, M. Lin, X. Xie, and M. Zhang, "Data compression scheme for fronthaul based on vector quantization," in *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*, 2019, pp. 252–257.

[12] M. R. Khosravi and S. Samadi, "Data compression in visar sensor networks using non-linear adaptive weighting," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 1–8, 2019.

[13] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[14] (2019) Onnx: Open neural network exchange formet. [Online]. Available: https://onnx.ai/

[15] (2019) Pytorch: An open source deep learning platform. [Online]. Available: https://pytorch.org/

[16] (2019) Keras: The python deep learning library. [Online]. Available: https://keras.io/

[17] (2019) Apache mxnet: A flexible and efficient library for deep learning. [Online]. Available: https://mxnet.apache.org

[18] (2019) The microsoft cognitive toolkit. [Online]. Available: https://docs.microsoft.com/en-us/cognitive-toolkit/

[19] (2019) Onnx runtime: cross-platform, high performance scoring engine for ml models. [Online]. Available: https://github.com/microsoft/onnxruntime

[20] D. Sapra and A. D. Pimentel, "Constrained evolutionary piecemeal training to design convolutional neural networks," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2020.

[21] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," in *International Conference on Learning Representation*, 2016.

[22] 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*, 2015, pp. 1135–1143.

[23] 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.

[24] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representation*, 2017.

[25] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1131–1135.

[26] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[27] D. Sapra and A. D. Pimentel, "An evolutionary optimization algorithm for gradually saturating objective functions," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2020.

[28] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[29] 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*, 2015, pp. 1–9.

[30] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, p. 947, 2000.

[31] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *IEEE International Conference on Computer vision*. IEEE, 2009, pp. 2146–2153.

[32] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on Machine Learning*, 2010, pp. 807–814.

[33] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[34] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *International Cconference on Machine Learning*, 2016, pp. 2217–2225.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[36] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, "Convolutional neural networks for human activity recognition using mobile sensors," in *6th International Conference on Mobile Computing, Applications and Services*. IEEE, 2014, pp. 197–205.

[37] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019.

[38] J. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[39] S. Christodoulidis, M. Anthimopoulos, L. Ebner, A. Christe, and S. Mougiakakou, "Multisource transfer learning with convolutional neural networks for lung pattern analysis," *IEEE journal of biomedical and health informatics*, vol. 21, no. 1, pp. 76–84, 2016.

[40] J. Li, S. Qiu, Y.-Y. Shen, C.-L. Liu, and H. He, "Multisource transfer learning for cross-subject eeg emotion recognition," *IEEE transactions on cybernetics*, 2019.

[41] (2019) Zlib: A massively spiffy yet delicately unobtrusive compression library. [Online]. Available: https://www.zlib.net/

[42] (2019) Protocol buffers. [Online]. Available: https://developers.google.com/protocol-buffers

[43] (2019) Nvidia geforce rtx 2080 gpu. [Online]. Available: https://www.nvidia.com/en-in/geforce/graphics-cards/rtx-2080/

[44] A. Rahimpour, A. Taalimi, J. Luo, and H. Qi, "Distributed object recognition in smart camera networks," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 669–673.

[45] R. Marroquin, J. Dubois, and C. Nicolle, "Ontology for a panoptes building: Exploiting contextual information and a smart camera network," *Semantic Web*, vol. 9, no. 6, pp. 803–828, 2018.

[46] P. Chen, P. Ahammad, C. Boyer, S.-I. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan *et al.*, "Citric: A low-bandwidth wireless camera network platform," in *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*. IEEE, 2008, pp. 1–10.

[47] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[48] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *2012 16th International Symposium on Wearable Computers*. IEEE, 2012, pp. 108–109.

[49] F. Moya Rueda, R. Grzeszick, G. A. Fink, S. Feldhorst, and M. Ten Hompel, "Convolutional neural networks for human activity recognition using body-worn sensors," in *Informatics*, vol. 5, no. 2. Multidisciplinary Digital Publishing Institute, 2018, p. 26.

[50] F. C. Maruf, V. Ermagan, H. Latapie, C. Cassar, J. Evans, F. Maino, J. Walrand, and A. Cabellos, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, 2017.

[51] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrows intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[52] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE Access*, vol. 7, pp. 65 579–65 615, 2019.

[53] Q. Zhang, X. Jiang, S. Chen, J. Xie, J. Yang, and L. Xing, "An information feature extraction and rapid updating scheme for knowledge centric networking," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 94–99.

[54] T. Zhang, X. Chen, and C. Xu, "Intelligent routing algorithm based on deep belief network for multimedia service in knowledge centric vanets," in *2018 International Conference on Networking and Network Applications (NaNA)*. IEEE, 2018, pp. 1–6.

[55] D. L. Silver, Q. Yang, and L. Li, "Lifelong machine learning systems: Beyond learning algorithms," in *2013 AAAI spring symposium series*, 2013.

[56] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel *et al.*, "Never-ending learning," *Communications of the ACM*, vol. 61, no. 5, pp. 103–115, 2018.

[57] R. Venkatesan and M. J. Er, "A novel progressive learning technique for multi-class classification," *Neurocomputing*, vol. 207, pp. 310–321, 2016.

[58] L. Lin, K. Wang, D. Meng, W. Zuo, and L. Zhang, "Active self-paced learning for cost-effective and progressive face identification," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 1, pp. 7–19, 2017.

[59] T. Gao, J. Du, L.-R. Dai, and C.-H. Lee, "Densely connected progressive learning for lstm-based speech enhancement," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5054–5058.