Performing Classification on Edge Nodes using Edge-Centric Distributed Deep Learning and Data Reduction

Vasu Sharma
Department of Computer Science
University of Colorado Boulder
Boulder, USA
vasu.sharma@colorado.edu

Manan Khasgiwale
Department of Computer Science
University of Colorado Boulder
Boulder, USA
manan.khasgiwale@colorado.edu

***Abstract*—An immense number of IoT devices has led to an exponential growth of data moving across the network. Traditionally, Artificial Intelligence (especially Deep Learning, DL) tasks on the IoT data are usually run on the cloud or other centralized systems which leads to issues like network congestion and significantly higher latencies for real-time applications such as human activity recognition (HAR) or facial recognition. Thus, unleashing DL services using resources at the network edge near the data sources has emerged as a desirable solution. The latest research in this field is inclined towards performing Distributed Deep Learning on the edge nodes. In this paper, we aim to do inferencing on edge nodes via the compression and splitting of DNN model so as to fit on those resource constrained nodes and serve the real time task of HAR. One of the major obstacles encountered is that edge nodes have considerably smaller computational power compared to their cloud counterparts which act as a hindrance to perform heavy DL tasks.**

Keywords—Edge Computing, Deep Learning

I. Introduction

The number of IoT devices increased 31% year-over-year to 8.4 billion in the year 2017 and it is estimated that there will be 30 billion devices by 2020. The global market value of IoT is projected to reach $7.1 trillion by 2020. The exponential increase in data generated by IoT devices has made a positive impact by introducing novel and convenient use cases for the end-users. IoT devices have an extensive set of applications in consumer, commercial, industrial, and infrastructure spaces. IoT supports smart systems such as smart cities, smart health care, smart transportation, and smart energy; however, the realization of these smart systems relies on the ability to analyze the massive amount of generated data. The increase of IoT devices at the edge of the network is producing a massive amount of data to be computed at data centers, pushing network bandwidth requirements to the limit.

Traditionally IoT data is sent to the cloud for performing any type of analysis. Cloud Computing provides reliability, computation power, and scalability to fulfill simple as well as complex use cases. Despite the improvements in network technology, data centers cannot guarantee acceptable transfer rates and response times, which could be a critical requirement for many applications. This is due to the remote cloud servers which could be located thousands of miles away from the end devices leading to higher latencies, network congestion, and higher running costs of remote servers. Real-time applications like self-driving cars, voice recognition, speech to text, or human activity recognition (HAR) require prompt responses to have a functional user experience.

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the location where it is needed, to improve response times and save bandwidth. Edge Computing aims to move the computation away from data centers towards the edge of the network, exploiting smart objects, mobile phones, or network gateways to perform tasks and provide services on behalf of the cloud. By moving services to the edge, it is possible to provide content caching, service delivery, storage, and IoT management resulting in better response times and transfer rates. At the same time, distributing the logic in different network nodes introduces new issues and challenges.

DL lies at the heart of most AI applications and involves teaching a computer to learn data representations with different levels of abstraction. DL has been successful in many domains including various vision tasks, natural language processing, and speech recognition. Most well-known applications of AI rely on Cloud Computing because of the superior and scalable compute power. But owing to the limitations of the cloud with real-time applications, we can start to move tasks to the edge and get away with computing the least expensive tasks. Here, we can try to slowly build up to more complex tasks by optimizing over redundant computations. With devices being more computationally better over the years we have reached a point where we can offload some DL/AI compassion logic distributed over multiple edges to simulate a powerful cloud-like behavior.

This study investigates using edge nodes for performing DL tasks by using data reduction techniques, model slicing, and efficient sharing of gradients among edge nodes. The evaluation of the data is done on the HAR dataset which comprises the data from sensors such as gyroscopes, accelerometers mounted on different parts of the human body.

It aims to enhance the latencies of real-time applications by experimenting with distributed DL on the edge infrastructure. Data reduction is performed on the edge along with the DL tasks. This avoids the network congestion caused by traditional techniques like running heavy DL models on the cloud and brings the computation closer to the end devices. In this study, we aim to use preprocessing techniques like sliding window, data reduction techniques such as Autoencoders(AE), and inference techniques on edge nodes.

We are proposing to move the Deep Learning tasks from remote cloud servers to the distributed edge nodes and along with that use the data reduction techniques. We plan to achieve lower latencies for real-time applications like HAR as well as lower the memory footprint for running the Deep Learning tasks on edge nodes.

II. Literature Review

Edge computing improves the performance of real-time applications by eradicating the back and forth transmission of data across large network distances because it performs computation close to the data sources. This review explores the training and inferencing of DL models across resource-constrained devices.

Wang et al. [1] presented a survey on mobile edge networks focusing on computing-related issues, edge offloading, and communication techniques for edge-based computing. Wang et al. [1] identified real-time analytics as one of the future directions of edge computing. Recent work in this area has been focused in that direction. For example, A.M. Ghosh et al. [2] proposed embedding of intelligence in the edge with DL. They investigate the latencies of the real-time application by performing data reduction at the edge nodes and transferring reduced data to the remote cloud servers to perform intensive DL tasks. This article explored merging edge and cloud computing for ML with IoT data to reduce network traffic and latencies. Our study aims to put edge computing as the primary infrastructure for driving DL tasks specifically in the case of HAR [3].

X. Wang et al. [4] presented a comprehensive survey on the convergence of edge computing and deep learning. The survey focuses on a broad set of topics and techniques such as optimization of DL models on edge, distributing the training of these models by the use of gradient sharing, and DL inferencing at the edge. It also focuses on the methods used for virtualizing the edge via VMs, containers, and network slicing. K. Bhardwaj [5] et al., in their research on “Toward Lighter containers for the edge” focus on efficient containerization by splitting containerized applications into two parts:- application container and bloat-causing execution environment. Through this, they can achieve significant reductions of the resource pressure at the edge, thus presenting a path toward greater efficiency and scalability for edge computing. We can take insights from this study to offload DL tasks from the cloud to the edge and further optimize it by applying data reduction techniques. Significant research has also been done on efficiently training DL models on large datasets at the edge. C. Hardy [6] et al., in their research on “Feasibility of distributed deep learning via adaptive compression” has proposed a novel solution to reduce the ingress traffic at the parameter server using compressed updates via AdaComp. The techniques mentioned in this research can be used to train DL models at the edge.

Efficient inference techniques for large DNN models have been the recent direction of research in edge computing. Z.Zhou et al. [7], have proposed a framework named Edgent, a framework that leverages edge computing for DNN collaborative inference through device-edge synergy. They exploit two major design ideas:- 1) DNN partitioning that adaptively partitions computation between device and edge for purpose of coordinating the powerful cloud resource and the proximal edge resource for real-time DNN inference (2) DNN right-sizing that further reduces computing latency via early exiting inference at an appropriate intermediate DNN layer. We can utilize techniques and research specified in this study for efficient inference of DL models on the edge nodes.

Another complementary study on DNN inference is done by S. Han et al.[8], where they proposed a technique called Deep Compression which can reduce the storage requirements by 35x and increase energy efficiency by 7x for DNN models. They follow a three-step approach: pruning of low information nodes; quantization of similar weights by weight sharing and finally applying Huffman coding. This gives significant insights on techniques for deploying DNN on the edge node where resources are constrained.

K. Bhardwaj et al. [9] research on DNN slicing answers an open problem, given a model and a set of edge resources, how to quickly determine the best way to partition the model and create deployment-ready model partitions needed to seamlessly run the model pipeline across the edge and the cloud?. This research targets some of the techniques which could be utilized to successfully partition a DNN model among edge nodes.

III. Design Description

As part of the current design, we are focusing on performing efficient inference using the edge devices. The model training is being performed on the cloud on 70% of UCI HAR [10] training dataset while 30% of the data is being reserved as the test set to be used for inference evaluation on edge nodes. The data is sampled at a constant rate of 50Hz and the sensor signals are preprocessed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). We have split the design of this problem into four subparts :-

A. Selecting a DNN model best suited for HAR

Neural networks which are suitable for time series classification require that data be prepared in a specific manner in order to fit a model. To do this, a sliding window technique is used where one window contains one to a few seconds of observation data. To utilize the time series nature of the data, one of the models that would perform well is a LSTM model which would use its input, forget and output gate to work with the hidden and the context layers. A CNN model would be able to get a better representation of the time series data due to the correlation between the nearby signals in the data. Combining the recurrence provided by LSTM and feature representation provided by CNN would result in a DNN model which would be better suited to represent HAR data. In the first scenario, we put a CNN and LSTM sequentially by first passing the data through a CNN to get a feature representation and then utilizing the recurrence of the LSTM model. Further extending on this scenario, we perform convolutions of the CNN as part of the LSTM model. The convolution is performed at each time step as part of the input thus moving away from the sequential model architecture. The accuracy and latency measurements are carried out on both scenarios in order to find a better fit for the task of HAR.

B. DNN model compression

For the inference of the DNN model, the most intensive task is the calculation in each layer. Not every node contributes equally to the inference, also, some contributing the least. Network pruning is used to trim the neural network to reduce the cost(time) of inference. This also helps save space needed to store the complete model as preventing the model from overfitting. A method proposed by Han et al, uses weight quantization and huffman encoding,in addition to pruning, to reduce the side further with no accuracy loss. There are some solutions out there such as PocketFlow by Tencent and NNI by Microsoft. These focus optimally pruning the nodes which contribute the least. While we work to have the model compatible to one of the frameworks, we are using randomized pruning provided by Keras library in which the model is written. This has helped to reduce the model size and time for inference while not impacting accuracy. We'll be moving to Tensorflow implementation as it provides a much more lower level API control over the parameters to the model.

C. DNN model splitting framework

One of the core limitations one faces is the limited capability of edge devices be it memory, processor speed or battery power. Hence to overcome this once we have a compressed model, we can split it across edge devices as per the model and device requirement. Here, the solution provided by Bharadwaj et al, Couper can help in slicing the model and deploying the slices across the edge. It splits the model based on having similar compute time for each slice.

D. Data dimensionality reduction

The input to the model is referred to as the dimensionality. If we reduce this number, then we'll have fewer calculations down the model and can speed up inference. An autoencoder(AE) is a neural network capable of finding structure within data in order to develop a compressed or reduced input. It is generally used to reconstruct the input back from the compressed output. But since we'll encode the AE on the same node as the inference start node, we'll pass the output from AE as an input to our model.

Our test-bed are personal laptops which are used as edge nodes for DL computations and the same network is used for any hyper-parameter transfers from one node to a different edge node in case of DNN splitting.

IV. Results and Evaluation

We are using the UCI HAR dataset[10] and classifying human activities using DL inference on edge nodes. We are going to utilize some of the techniques mentioned in the literature review as well as apply data reduction using non-linear transformation models like Autoencoders to reduce the data and the latency at the inference layer where the resources are constrained.

Firstly, we train the model on the cloud/machine and compare inference results on both edge and cloud.

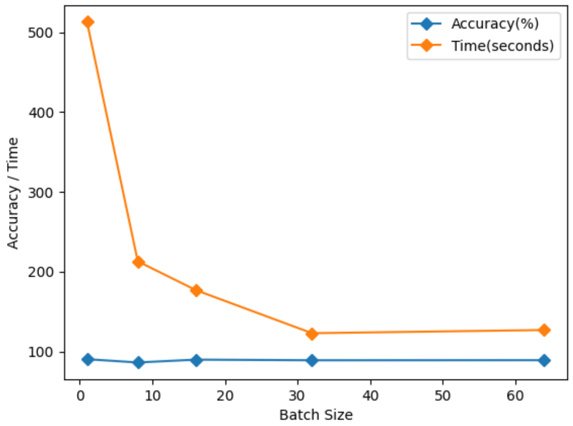

Fig. 1. ConvLSTM graph for model time/accuracy vs batch size

A. DNN model selection for HAR

So according to the design description mentioned above, the first task was selecting a DNN model that would be better suited for the task of HAR. We went ahead with trying LSTM and ConvLSTM for this task. The model evaluation is summarized in Table 1. As is evident in the table, the ConvLSTM model leads to much better performance but due to the lack of pruning and quantization techniques for complex layers like ConvLSTM, we go ahead with the LSTM model and apply compression techniques on that particular model. With LSTM as the final model, we explored multiple batch sizes for tuning the accuracy and time on the test set. As we can see from Fig1, saw that batch size 64 leads to a lot less training time with minimal reduction in accuracy.

Model name	Accuracy(%)	Model size
ConvLSTM	91.562	3.5 MB
LSTM	90.499	1.6 MB
LSTM pruned	90.498	1.1 MB
LSTM pruned and post quantized	90.432	69 KB

Table 1. Evaluation of models on HAR

B. Compression techniques on DNN model

We can infer from Table 1 that the pruned and quantized LSTM offers the best performance / ratio. Using these compression techniques we achieve around 20x model compression with minimal hits in the accuracy. The compression techniques used pruning and quantization of weights which involve reduction in the precision of model weights. So, we go ahead with a pruned and quantized LSTM model as the model for our edge devices while we evaluate its performance with it's uncompressed counterpart on the cloud.

C. *Model evaluation on Cloud vs Edge server*

The cloud machine configurations and the edge server configurations were as follows :-
Cloud server machine deployment region : us-central 1a
Cloud server machine specs : c2-standard-8 (4v CPU, 16 GB)

Edge server machine deployment region : local machine docker container
Edge server machine specs : i5 8th gen (quad core), 8GB

The time readings were taken over 2900 samples of the test split from the HAR dataset. We can infer from Table 2, that even with running compressed model version of LSTM on a powerful cloud machine, the average round-trip latency overpowers the model invoke time which is lesser in the case of cloud for a compressed model. In the case of an edge server, the compressed model performs much better in the terms of latency times.

Environment	Model type	Average inference time (ms)	Average invoke time (ms)	Average round-trip latency time (ms)
Edge server	Compressed LSTM	5.01	4.45	0.56
Cloud server	Uncompressed LSTM	124.12	43.37	80.75
Cloud server	Compressed LSTM	40.26	4.04	36.22

Table 2. Evaluation of models on Environment

V. Future work

In the future we suggest that the following techniques may be used to further reduce the latencies involved for inference on the edge server :

- Data dimensionality reduction : We can explore autoencoders for dimensionality reduction to reduce the current model inference latencies on the edge..
- DNN model splitting : We can explore splitting the model based on the runtime of the layers involved in the neural network by using a framework like Couper [9] which identifies the candidate split points of the input DNN model and finds the most efficient representation. The limitation with this technique is that these frameworks have not been open sourced.
- Weight Clustering : It reduces the number of unique weight values in a model. It first groups the weights of each layer into N clusters, then shares the cluster's centroid value for all the weights belonging to the cluster. This would significantly reduce the model size resulting in easy deployment. Currently LSTM or ConvLSTM layers don't support weight clustering.

VI. Conclusion

In the task of determining human activities based on the sensor data, we found that the edge server performed better compared to the Cloud server for the LSTM model. This is due to the fact that the uncompressed LSTM model had fewer trainable weight layers. Our findings are not generalizable for all use cases because the original DNN model, even after compression, could be too large for the edge server to handle without model splitting.

References

[1] Wang et al., “A survey on mobile edge networks: Convergence of computing, caching and communications” IEEE Access, vol. 5, pp. 6757–6779, 2017.

[2] A. M. Ghosh and K. Grolinger, "Edge-Cloud Computing for the Internet of Things Data Analytics: Embedding Intelligence in the Edge With Deep Learning," in IEEE Transactions on Industrial Informatics, vol. 17, no. 3, pp. 2191-2200, March 2021, DOI: 10.1109/TII.2020.3008711.

[3] H. F. Nweke, Y. W. Teh, M. A. A.-G., and U. R. Alo, “Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges,” Expert Syst. Appl., vol. 105, pp. 233–261, 2018.

[4] Wang, X., Han, Y., Leung, V. C. M., Niyato, D., Yan, X., & Chen, X. (2020). Convergence of edge computing and deep learning: A comprehensive survey. IEEE Communications Surveys & Tutorials, 22(2), 869–904.

[5] Park, M., Bhardwaj, K., & Gavrilovska, A. (n.d.). Toward lighter containers for the edge. Usenix.Org. Retrieved March 2, 2021, from https://www.usenix.org/system/files/hotedge20_paper_park.pdf

[6] Hardy, C., Le Merrer, E., & Sericola, B. (2017). Distributed deep learning on edge-devices: Feasibility via adaptive compression. 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA).

[7] E. Li, L. Zeng, Z. Zhou and X. Chen, "Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing," in IEEE Transactions on Wireless Communications, vol. 19, no. 1, pp. 447-457, Jan. 2020, DOI: 10.1109/TWC.2019.2946140.

[8] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. Retried from https://arxiv.org/pdf/1510.00149.pdf.

[9] Hsu, K.-J., Bhardwaj, K., & Gavrilovska, A. (2019). Couper: DNN model slicing for visual analytics containers at the edge. Proceedings of the 4th ACM/IEEE Symposium on Edge Computing.

[10] https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones