Data Generator-based Continual Learning System for Edge Devices.

Journal:	*Transactions on Services Computing*
Manuscript ID	TSCSI-2021-03-0144
Manuscript Types:	SI- Edge AI-as-a-Service:
Keywords:	I.5.1.d Neural nets < I.5.1 Models < I.5 Pattern Recognition < I Computing Methodologies, H.4.1.g Workflow management < H.4.1 Office Automation < H.4 Information Technology and Systems Applications < H Information Tech

Data Generator based Continual Learning System for Edge Devices.

Brijesh Vora, Suri Bhasker Sri Harsha, Kalidas Yeturu

Abstract—

Neural networks suffer from Catastrophic forgetting problem when deployed in a continual learning scenario. Pseudo rehearal is a technique where a generator is used to synthetically generate training data of the previous task to retrain the neural network to prevent forgetting. Edge devices usually have severe computational and memory constraints which limits the deployment of pseudo rehearsal schemes directly on them. In this work, we demonstrate a continual learning system that deploys the generator on a server and regularly updates the neural networks deployed on the edge whenever required.

Index Terms—Continual Learning, Workflow systems, Catastrophic forgetting, Neural Networks.

1 Introduction

Recent advancements in deep neural networks have demonstrated near-human abilities while solving tasks requiring complex cognitive capabilities [1]. It is clear that wider adoption of deep neural networks will prove to be beneficial in terms of obtained applications and economics. However, current neural networks require significant computational resources and are usually deployed on the cloud. With the rising computational power of edge devices, deployment of neural networks on them will allow for wider adoption of this technology. In this work, we propose a system to pseudo-rehearse a neural network to manage the *catastrophic forgetting problem* that might arise in networks deployed at edge. We propose an architecture where the neural network deployed at the edge is only responsible for inferencing while the synthetic data generation is done on cloud with higher hardware capabilities.

1.1 Catastrophic forgetting problem

In continual learning scenario, where neural networks are expected to learn new batches of data sequentially, neural networks suffer from *catastrophic forgetting problem*, where they forget previously learnt information after being trained on a new batch of data [2]. When trained on a new batch of data from a significantly different data distribution, the weights of the neural network tend to change resulting in distortion of the learned decision boundary. This distortion of the boundary leads to forgetting of the previously learnt information. The problem has been well studied and many solutions have been proposed to address this issue [3].

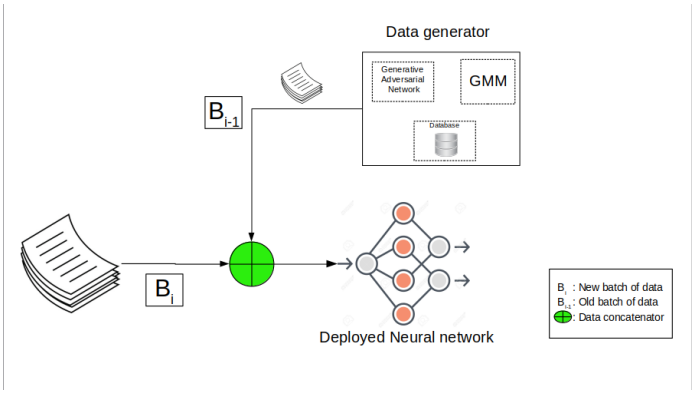
1.2 Rehearsal and Pseudo rehearsal

Rehearsal and *Pseudo-rehearsal* are two concepts proposed by [4], where the neural network is retrained on the original or synthetic version of previous data while learning a new batch of data. Generative replay [5] and Genetic rehearsal [6] are examples of techniques using the concept of *pseudo rehearsal* where synthetic data of the previous task is generated using Generative Adversarial Networks (GAN) [7] or Genetic Algorithms.

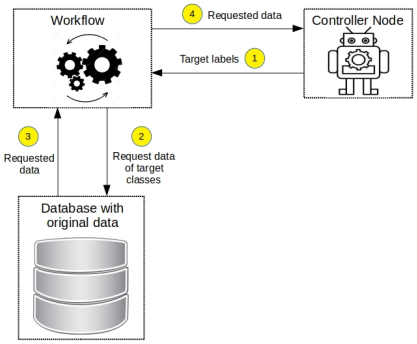
1.3 Edge AI

Currently, deep learning networks are deployed on the cloud servers because of their computational requirements [8]. Any edge device requiring the service of the model sends a *service-request* to the cloud, which in-turn responds with the requested service. However, such kind of architectures cannot be deployed in scenarios where there is a strict constraint on the latency. For example, in applications like autonomous cars, smart surveillance [9] or smart manufacturing, the model is required to take quick decisions to avoid damage to any involved parties. When the model is deployed on the cloud, the delay in transmission of data between the edge device and the cloud service could result in sub-optimal performance of the system. To avoid such latencies, a better solution would be to utilize the increasing computational powers of the edge devices and deploy the machine learning models on the edge. Increasing adoption of AI Hardware accelerators like Qualcomm Snapdragon 8 series [10], Google TPUs [11], Intel Xeon D-2100 [12] or NVIDIA Turing GPUs [13] by edge devices has resulted in increase of capabilities to host deep neural networks on them. Popular deep learning packages like Tensorflow [14] released libraries that aim to deploy fully trained neural network on edge devices. Despite witnessing an increase, edge devices still have smaller computational and memory resources compared to server hardware deployed at the cloud. A smarter architecture would use a smart combination of cloud and edge hardwares to optimize performance as well as reduce latency. Neural networks that are deployed to edge devices can expected to only perform inferences while carrying out the training steps on the cloud with hardware accelerators like GPU or TPU.

- *All the authors were with the Computer Science and Engineering Department, Indian Institute of Technology Tirupati, Andhra Pradesh, India. E-mail: {CS17B032, CS18S506, ykalidas}@iittp.ac.in .*



(a)



(b)

Fig. 1: Figure 1a shows the concept of *pseudo rehearsal*. The synthetic data of previous batch is generated using a data generator using Gaussian mixture models or Generative Adversarial Networks and interleaved with the newly arrived training data. Figure 1b shows the interactions between the Controller and the database containing the original data.

In this work, we specifically address the *continual learning* scenario where the deployed model is expected to learn new batches of data from different data distributions, without forgetting previously learnt information. As suggested in the concept of *pseudo rehearsal*, hosting a separate *generator* network to synthetically generate data on demand might reduce or prevent forgetting of previous information. However, despite of increase in computational capabilities of edge devices, hosting an additional *generator* network on the edge device might be a sub-optimal solution as it might consume additional power, resulting in decreased work time of the edge device.

In this work, we demonstrate a scalable, modular and platform agnostic software workflow that provides effective management of synthetic data generation by connecting the generators on the servers with neural networks deployed at the edge.

1.4 Our Contributions

- Proposed a novel model management scheme for neural networks deployed on edge in a continual learning scenario.
- Introduced an easy-to-scale, flexible, production grade workflow system software for managing neural networks on edge.

2 Methods

2.1 Remote Data Generation (RDG) architecture

We propose a scalable and flexible system to implement synthetic data generation as a service in this work. The architecture is called **Remote Data Generation (RDG)** architecture and it consists of a *Data service*, *Prediction service*, *Training service*, and *Controller service*. The interactions between the above mentioned services is orchestrated by the *workflow system*. Two types of nodes are possible in this workflow: User-Interface (UI) nodes such as the *controller*, where the parameters of the model are configured, and *computational nodes* such as *Prediction service*, *Training service*, *Data service*. The role of each service is as follows.

2.1.1 *Prediction-service*

The prediction services' primary function is to generate predictions for given input data using the deployed neural network. In case of a federated learning setting, the prediction service could be running on the edge device. And in case of a centralized setting, the prediction service also could be running on the cloud using a copy of the deployed neural network to improve latency times. The prediction service plays an integral role when generating synthetic data using Genetic Algorithms. The synthetic data is generated by constant interactions between the data service and the prediction service.

2.1.2 *Data service*

Data Service's primary function is to generate synthetic data or provide original data for retraining neural networks deployed on edge devices. It takes labels or class as input and produces respective data as output. The data service could consist of a data generator like GAN, GMM or Genetic Algorithms or could be a database consisting of original data directly.

2.1.3 *Training-service*

Training-service is designed to retrain the model using the synthetic-data generated by Data service. It updates the model, and that updated version can be deployed to the edge devices. One significant advantage of implementing a separate microservice for training the neural networks is that, while other services can be deployed on cheaper hardware with low computational capabilities, the training service can be deployed on specialized hardware that are optimized for training processes.

2.1.4 *Workflow system*

The workflow system is responsible for controlling the flow of information between various microservices in the system. The system continuously polls for any messages in the Workflow Database. As soon as any new message arrives, it routes the incoming message from one node to another node after evaluating it. The system can modify the messages in between which allows for content-based routing in the

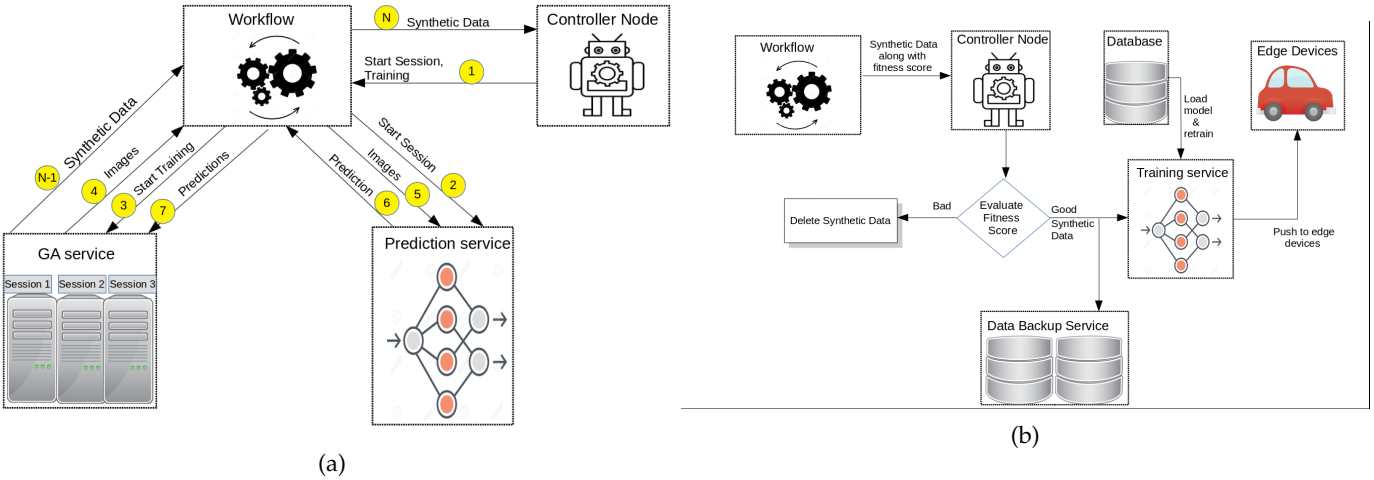


Fig. 2: Block diagram showing the Genetic Rehearsal implementation of the proposed system. Figure 2a shows the interaction between controller and the database via the workflow system when generating synthetic data using Genetic Algorithms. Figure 2b shows the retraining process of the system after synthetic data has been generated.

architecture. Also, the routing logic for the workflow system can be dynamically imported.

To demonstrate the flexibility offered by the proposed *Remote Data Generation* architecture, we implemented various data generators like Generative Adversarial Networks (GAN), Genetic Algorithm and Gaussian Mixture model as the *Data service*. In addition, we also implemented an *original data* as-a-service architecture, where the original data is stored in the cloud and a subset of it is requested for the purpose of *pseudo-rehearsal*.

2.2 Original data as-a-service

[4] proposed the concept of *Rehearsal*, where the neural network is trained on original data of the previous task to prevent catastrophic forgetting. However, storing of entire previous data requires allocation of considerable memory resources which is not feasible for edge devices. Therefore, by deploying the original data on the cloud *as-a-service* and then requesting the subset of it according to the need of the deployed model might be an optimal solution. The proposed architecture can be used in both federated learning

[15] setting where each deployed model is personalized according to the user, or a centralised setting where a central controller periodically pushes one uniform model to all the edge devices. Figure 1b depicts a centralised setting where a human controller initiates the request for data. The controllers request is routed to the workflow, which inturn raises a request with the original data service. The data service returns the requested data to the workflow system which routes it to the controller. The controller is then presented with a choice whether to re-train the model on it or not. If the controller chooses to retrain the model, the synthetic data is passes to the *training service* which loads the model and retrains it. This retrained network can be considered as the upgraded model which is then pushed to edge devices for deployment. In contrast, in a federated learning setting, the retrain service is present on the edge device itself, where the deployed model is retrained and then deployed.

2.3 Genetic Algorithm as a Data Generator

This is an micro-service style implementation of the system proposed by [6], where synthetic data is generated by a series of communications between Genetic Algorithms and the deployed neural network. To generate the synthetic data for a target class, the Genetic Algorithm begins with a random set of images which are then given to the deployed network for prediction. The softmax confidence of the network on these images for the target class is considered as their fitness scores. The fittest 24% individuals are sent to the next generation, where a series of mutation and crossover operation are followed to populate the next generation. This is repeated until the organisms of a given generation reach a certain fitness threshold.

To implement this technique, we implemented a Genetic Algorithm as the generator service, which we refer to as *GA service* here on. The GA service generates synthetic images and sends them to the prediction service via the workflow. The prediction service predicts the score using the deployed model and returns it to the workflow system. GA service uses the predicted scores to generate a new batch of images. The controller specifies the *number of generations* up to which the service should generate images. Each generation will generate images where the fitness of population is greater than previous generation's.

Figure 2a shows the steps involved in generating synthetic data from a time and event perspective. GA service and prediction services are computational nodes, and the controller is a user-interface (UI) node. A step-wise list of interactions between the services is provided below:

- STEP 1: At time step T_1, the *controller* node requests to begin the training via workflow and the GA service generates the first batch of images.
- STEP 2: At T_2, images or synthetic samples that are generated by the GA service are sent to Prediction Service via the workflow.
- STEP 3: At T_3, the Prediction service receives the images and then returns the prediction scores for all the images.

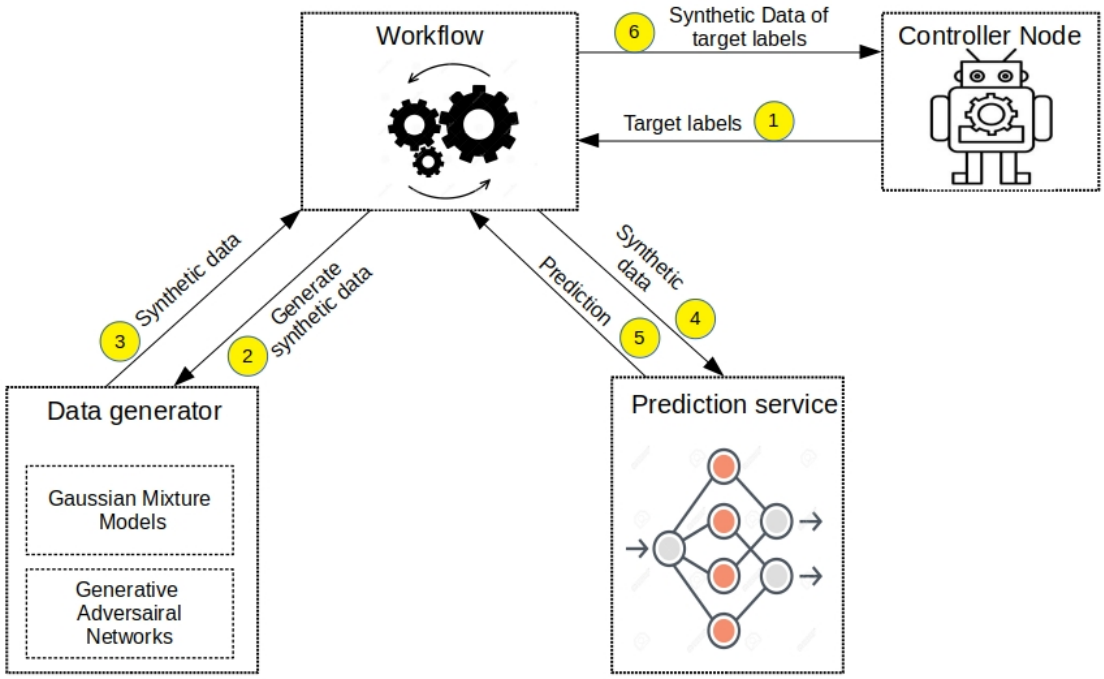


Fig. 3: The block diagram shows the interactions between various services when either Gaussian Mixture models or Generative Adversarial Networks are used as Data services. Unlike in the case of using Genetic Algorithm to generate synthetic data, here the number of interactions between microservices in relatively lower.

- STEP 2 and 3 are repeated until a certain fitness threshold is reached or until the number of iterations is finished.
- STEP N-2: At T_{N-2}, the score for each synthetic sample is sent to the GA service.
- STEP N-1: At T_{N-1}, the GA Service realising that the fitness threshold has been reached will send the final synthetic data to the controller via the workflow.
- STEP N: At time step T_N, the workflow routes the synthetic data to the controller and message is sent to GA service to stop the training.

All the request which happen between nodes/services and workflow happen as post requests and data flows as JSON objects. An independent session is opened between the GA service and prediction service to increase the throughput of the system by allowing simultaneous executions for multiple systems.

Figure 2b shows the steps involved after this synthetic data is received by the controller from the workflow. The controller has two jobs to do: (1)To start the session; (2) To evaluate the quality of the synthetic data generated. After receiving the synthetic data, the controller is presented with an option to whether or not retrain the model on this new synthetic data. On being satisfied with the quality of the generated data, the controller can push the synthetic data along with the command to "retrain" the model to the workflow system. The workflow system then routes the synthetic data to the *Training service*.

A copy of the synthetic data sent is also sent to the *data backup service* that stores the synthetic data for future use. If the generated images' fitness score is not high enough, then the controller instructs the workflow (by setting the message "delete_data") to delete the synthetic data. Training service will retrain the model based on the synthetic data and upgrade the model. Now this upgraded model can be sent to edge devices for deployment.

2.4 GAN as a Data Generator

[5] suggested use of Generative Adversarial Networks (GANs) as *generators* to generate synthetic data. In this technique, instead of storing the original data, a Generative Adversarial Network is trained until it can synthetically recreate the original data. This fully trained GAN is stored in a database and the original data is then discarded. This technique greatly saves space as storing a GAN consumes much lesser space compared to storing entire original data. In the beginning of the process, the controller sends the request to workflow with the required *target labels* as shown in Figure 3. The *workflow system* then sends the request to *data service* which in this case has a GAN in it. The request is send as a JSON object which has message field with various flags and variables that describe the desired properties of the synthetic data. Data service then generates the synthetic images upon request and sends back to the workflow system. Since the data generated by GAN is unlabelled, the workflow system sends the synthetic data to the prediction service where it predicts the labels of the generated samples. These predictions are sent back to

the workflow. The workflow system finally filters out the samples belonging to the target classes based on the newly generated labels and sends these samples to the controller for retraining.

2.5 GMM as Data Generator

In a Gaussian Mixture model, the dataset is assumed as a collection of n Gaussians. A Gaussian Mixture Model can also be used as a data generator for pseudo rehearsal. Just like using GAN as a generator, using of GMM as a generator follows a similar process. The controller first initiates a request to generate samples of the target classes with the workflow. The workflow sends a command to the GMM which is in the data service, to start generating synthetic data. The data service responds back to the workflow system with the synthetic data. As the synthetic data is unlabelled, the workflow systems sends the data to the prediction service which labels the data. The workflow system uses these labels to filter out the samples belonging to the target classes and sends them to the controller.

2.6 Implementation Details

The proposed system was implemented completely in Python language, however, it has to be noted that the user is free to implement any of web-services using a different as the proposed system is platform agnostic. *Flask* library [16] was used to build and deploy all the web-services while *Requests* library was used to implement the HTTP POST and GET requests. *Numpy* [17] was used to represent and generate synthetic data throughout the system. Representing the data using Numpy arrays allows us to use the system in non-image applications as well. MongoDB was used to implement all the databases in the system. The reason for selecting MongoDB over SQL is that JSON objects were used to transmit data between different services in our workflow. As SQL has static *table-oriented* design where the colums of the table have to be predefined, inserting JSON objects with varying sizes is not possible. As MongoDB is the NO-SQL database with little restrictions over the structure of the database, JSON objects with varying sizes can be inserted into MongoDB with ease. This flexibility offered by MongoDB influenced our decision to select it over an SQL based database. *Pymongo* library was used to connect the webservices with the Mongo Database.

All the neural networks were developed in Keras [18] with Tensorflow [19] running in the back-end. The Gaussian Mixture Model was implemented in Sklearn [20]. The system was tested on MNIST Digits [21] and MNIST Fashion [22] datasets which were available as a standard dataset in Keras.

The experimentation was carried out on a network of three systems with Intel Core i7 Quadcore. The three systems had 7.7GB, 7.7GB and 16GB of available RAM space. The systems were connected using a Wi-Fi (802.11n) router with a link speed of 150Mbps between each system and the router. All the micro-services were launched on different systems.

3 Experiments and Results

Inorder to test the proposed architecture, we created a continual learning scenario, where the data generator is expected to synthetically generate requested data for MNIST Digits dataset. Genetic Rehearsal, Generative Replay and Gaussian Mixture models were used as data generators. Variables like transmission latency, generation latency and amount of data being transferred between micro-services were systematically measured for all the above proposed schemes. In addition, the original data as-a-service model was also implemented and all the above mentioned parameters were obtained for them.

When Genetic Algorithm was implemented as data service, the transmission time between the data service and workflow was less than 2 seconds. The interaction between workflow system and prediction service again took less than 2 seconds. The size of each generation was 8, so a total of 8 images of size 28x28 were exchanged between the Data service and prediction service during synthetic data evolution. This meant that approximately 3KB of data was exchanged per interaction. It has to be noted that we used a standard Wi-fi router to connect all the systems in our experimental setup. The latency of the system is influenced by the *link speed* between the router and the system. Usage of LAN based network to connect the systems will greatly improve the latency times. For MNIST digits, approximately 5 generations of evolution is required to generate data for one class. As 30 cultures are usually used, a total of 150 interactions happen between the data service, workflow system and the prediction service. This means that it takes approximately 12.5 minutes to generate synthetic data for one single class.

In scenarios like original data as-a-service and GMM/GAN as data generators, the number of interactions between the modules are significantly less. There is only one iteration of interaction between the data service, workflow system and prediction service which brings the total communication latency time to less than 5 seconds. However, since the synthetic data that is generated is transported from data service to the workflow and then to the prediction service, the transmission time is influenced by the transmission capacities of the network on which the user implements the proposed system.

4 Discussions

In this work, we demonstrated a scalable, modular and platform agnostic workflow system for maintaining models deployed on edge devices in a continual learning setting. The proposed system could easily be adopted to a federated learning setting where there are personalized machine learning models deployed on the edge devices or a centralized setting where one network is periodically updated and pushed to large number of edge devices by just altering the deployment of the prediction services. Protecting these workflow systems from cyber-security threats is essential for the well being of all the involved parties. In our future works, we would like to implement end-to-end encryption of communication between all the webservices. We would also like to test the proposed architecture by actually deploying the neural networks on edge device like Raspeberry

Pi and measuring the real world latencies and energy consumption of the proposed system.

5 Conclusion

In this work we demonstrated a scalable continual learning workflow system to maintain neural networks deployed on edge devices. Various kinds of data generation schemes were implemented as a *service* to generate synthetic data for pseudo rehearsal.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[3] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[4] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[5] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," *arXiv preprint arXiv:1705.08690*, 2017.

[6] B. S. H. Suri and K. Yeturu, "Pseudo rehearsal using non photorealistic images," *arXiv preprint arXiv:2004.13414*, 2020.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[8] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[9] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, 2018.

[10] Qualcomm, "Snapdragon 8 series mobile platforms," https://www.qualcomm.com/products/snapdragon-8-series-mobile-platforms.

[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[12] Intel, "Advanced intelligence for high density edge solutions," https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon/d-2100-brief.html.

[13] NVIDIA, "Graphics cards with Turing GPU architectures," https://www.nvidia.com/en-us/geforce/turing/.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[15] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[16] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[18] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

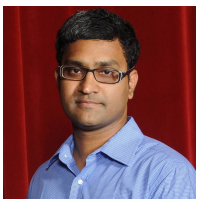
[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[21] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[22] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

Brijesh Vora is currently pursuing his B.Tech in Computer Science and Engineering from Indian Institute of Technology Tirupati, India. His research interests include machine learning and workflow software systems.

Suri Bhasker Sri Harsha is currently pursuing his MS by research under Dr. Y. Kalidas at Indian Institute of Technology Tirupati in Computer Science Department. He completed his B.Tech from Geethanjali College of Engineering and Technology, Hyderabad. He is currently working on Continual Learning problem in Neural networks.



Dr. Yeturu Kalidas is an Assistant Professor at Indian Institute of Technology Tirupati in Computer Science Department. He completed his Ph.D. from Indian Institute of Sciences, Bangalore. His research interests include Machine learning algorithms and applications, Big data technologies, Bioinformatics and Internet of things.