IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 31, 2021, accepted February 8, 2021, date of publication February 12, 2021, date of current version February 24, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3059251

A Novel Approach of IoT Stream Sampling and Model Update on the IoT Edge Device for Class Incremental Learning in an Edge-Cloud System

SWARAJ DUBE, WONG YEE WAN, AND HERMAWAN NUGROHO, (Senior Member, IEEE)
Department of Electrical and Electronic Engineering, University of Nottingham Malaysia, Semenyih 43500, Malaysia

Corresponding author: Swaraj Dube (kecy3dsm@nottingham.edu.my)

This work was supported in part by the University of Nottingham Malaysia Campus, and in part by the Fundamental Research Grant Scheme (FRGS) by the Ministry of Higher Education, Malaysia under Grant FRGS/1/2018/ICT02/UNIM/02/4.

ABSTRACT With the exponential rise of the number of IoT devices, the amount of data being produced is massive. Thus, it is unfeasible to send all the raw data directly to the cloud for processing, especially for data that is high dimensional. Training deep learning models incrementally evolves the model over time and eliminates the need to statically training the models with all the data. However, the integration of class incremental learning and the Internet of Things (IoT) is a new concept and is not yet mature. In the context of IoT and deep learning, the transmission cost of data in the edge-cloud architecture is a challenge. We demonstrate a novel sample selection method that discards certain training images on the IoT edge device that reduces transmission cost and still maintains class incremental learning performance. It can be unfeasible to transmit all parameters of a trained model back to the IoT edge device. Therefore, we propose an algorithm to find only the useful parameters of a trained model in an efficient way to reduce the transmission cost from the cloud to the edge devices. Results show that our proposed methods can effectively perform class-incremental learning in an edge-cloud setting.

INDEX TERMS Incremental learning, convolutional neural network, IoT edge device, cloud, data sampling.

I. INTRODUCTION

In the computer vision domain, deep learning has shown a great amount of success and in some tasks even surpassing the level of human accuracy. A lot of this success has been obtained in an offline setting whereby all the data is already present on a machine before training starts and also deep learning models are trained on big datasets just once and then they are deployed. In the real-world, however, not all the data can be present with us beforehand. Even if all the data is available, it is challenging to train deep learning models as it requires powerful hardware to train such models and it is also time-consuming to train on a huge amount of data altogether. Another problem with training a deep learning model offline is that once it is trained and deployed, the model will not learn any more parameters in the future. However, deep learning models should be able to learn in a continuous environment whereby new data arrives over time. The learning of a new task is dependent on the previous task [1].

One of the main challenges of incremental learning is a phenomenon known as catastrophic forgetting [2], [3] whereby representations of the old classes are lost when the model is fine-tuned on new data. The most basic way to mitigate this problem is to use a combination of both the old and new data when new data arrives and use this combined dataset to train the model. However, if such process is performed on the edge-cloud architecture, this indicates that all of the data from each class will need to be sent to the cloud for incremental training. This leads to a more expensive transmission cost from the IoT edge device to the cloud [4], [5].

While there have been several applications of deep learning in the Internet of Things (IoT) [6]–[13] but for incremental learning, the majority of the research is being carried out on a centralized computer [14], [15], i.e., incremental learning has not been explored in the context of IoT where a model has to be partitioned in an edge-cloud architecture [16]. There

The associate editor coordinating the review of this manuscript and approving it for publication was Jerry Chun-Wei Lin.

is a clear for incremental learning to be integrated with IoT because in the real-world, smart devices that are collecting raw data can be geographically spread and new data belonging to new tasks can also be collected over time. In practice, a large number of IoT devices are highly dependent on cloud assistance for deep neural network training [18]. However, in a real IoT environment where there are a huge number of devices that are collecting data at a high rate, it is infeasible to make the devices send all the raw data to the cloud because bandwidth costs would be very high. This would result in a sheer amount of load the cloud would have to handle. Furthermore, the IoT edge device is the first device that receives new raw data. Therefore, it would be a much more feasible solution to deploy certain parts of a deep learning model to the IoT edge device in an effort to assist the cloud in the entire continuous learning process instead of relying completely on the cloud [19]. When a given deep learning model is trained in a distributed manner between two or more devices, this will present certain challenges such as deciding how many and which specific layers of a model must be run on the edge and the cloud (this is known as offloading [20]–[23]), dealing with the transmission load between the IoT edge device and the cloud [24] and evaluating whether it is important for the IoT edge device to transmit all of the data to the cloud for model training or whether some of the data be discarded [25]. These challenges have not yet been addressed in a distributed incremental learning scenario which is what this paper attempts to do. This paper largely extends the work of [16] whereby data sampling is performed at the IoT edge device, however, in [16], the new data samples do not belong to novel classes. Our data sampling algorithm is capable of sampling data from completely novel classes without any need for hyperparameters by automatically selecting the number of samples needed per incremental training round which is the core novelty of our data sampling algorithm.

Nomenclature

c	Class index
L^c	Set of losses of all samples sorted in ascending order belonging to class c
n_c	Number of samples in class c just before transmission from IoT edge device to cloud
W'	Newly initialized weights of the SoftMax layer on the IoT edge device
b'	Newly initialized biases of the SoftMax layer on the IoT edge device
x_i^c	Feature map output of image i belonging to class c
y_i^c	One-hot label corresponding to x_i^c
l_i^c	Cross entropy loss corresponding to x_i^c
$\varphi(\cdot)$	SoftMax function
μ_L^c	Median value of set L^c
σ_L^c	Deviation that denotes by how much the values of L^c deviate from μ_L^c
$\mathbb{I}(\cdot)$	Indicator function
L_{cutoff}^c	Cut-off loss of set L^c
V^c	Set containing loss values smaller than L_{cutoff}^c
τ^c	Total number of samples to be discarded from n_c
t_{mb_0}	Time taken to forward propagate the first mini batch of images on IoT edge device
t_{mb_n}	Time taken to forward propagate a mini batch of images on IoT edge device after first mini batch
lab_{str}	Label string format
FM_n	Feature map tensor at index n of feature maps of a mini batch
D_n	Depth of feature map tensor of FM_n
W_n	Width of feature map tensor of FM_n
H_n	Height of feature map tensor of FM_n
S_n	Size of FM_n
t	Incremental training round t
$\emptyset(x_t)$	Output of feature extractor of sample x belonging to incremental training round t
E_t	Set of all features at every incremental training round up until round t
OS	Operating System
RAM	Random Access Memory
C_{old}^l	Parameters of layer l of the classifier the on cloud before training
C_{updated}^l	Parameters of layer l of the classifier on the cloud after training
C_{diff}^l	Differences of values of parameters between C_{updated}^l and C_{old}^l
$\text{sorted}C_{\text{diff}}^l$	Sorted values of C_{diff}^l in ascending order
$\text{Accept}_{\text{loss}}$	Acceptable training accuracy loss
A_{ori}	Training accuracy of the trained classifier on the cloud
q	Number of quantiles
T^l	List of thresholds of layer l of the classifier on the cloud
$C_{\text{temporary}}^l$	Parameters of layer l of the temporary model
$\text{Clist}_{\text{temporary}}^l$	Parameters of layer l of the temporary classifier in a list holding every temporary model
A_{temp}	Training accuracy of $C_{\text{temporary}}$
W_{best}^l	Useful weights of layer l of the trained classifier on the cloud
W_{best}	Set of useful weights of each layer of trained classifier on the cloud
I_{best}^l	Indices corresponding to W_{best}^l
I_{best}	Set of indices corresponding to W_{best}
M_{ec}	String containing feature maps and labels transmitted from the IoT edge device to the cloud
M_{ce}	String containing W_{best} and I_{best} transmitted from cloud to the IoT edge device
NS	No Sampling
RS	Random Sampling

ES	Entropy Sampling [17]
MTS	Median Test Sampling
WRSTS	Wilcoxon Rank Sum Test Sampling
LCS	Least Confidence Sampling [17]
DDC	Data Discard Counting

Class incremental learning can have many applications. For example, in social media whereby incremental learning can be used to learn new contents, behaviors etc.

Instead of a novel AI learning algorithm, the novelty of this paper is about reducing the transmission load between the IoT edge device and the cloud without affecting the incremental learning performance regardless of the model architecture or the learning hyperparameters. The two main contributions of this paper are as follows:

- A novel data sampling technique to filter certain training samples from novel classes on the IoT edge device for reducing the transmission cost from the edge to the cloud with a very small effect on the incremental learning performance.
- An improved version of a novel algorithm [16] for sending only the useful parameters of a classifier after training back to the IoT edge device instead of sending back all the parameters of the classifier.

The rest of this paper is organised as follows: we first discuss a review of incremental learning approaches in *Related work and Motivation*. We then explain about our proposed system architecture and the working mechanisms in the *Methodology* section. *Experimental setting* explains all the learning settings used, hyperparameters, and the learning procedures. In the *Results and Discussion* section, we show and discuss our findings, and this is followed by the *Conclusion* section and the *Future Work* section.

II. RELATED WORK AND MOTIVATION

In the incremental learning context, there are three main approaches, namely regularisation, rehearsal, and dual memory system approaches. In the regularisation approach, the loss function is designed in a way to retain representations of the old classes, i.e., by not changing the values of the important parameters much of a given model. In the rehearsal approach, the focus is on using a mixture of both the old and new data in appropriate proportions. By using the old data, the old knowledge is retained.

The incremental classifier and representation learning (iCaRL) [15] techniques both use rehearsal and regularisation approaches. This approach uses exemplars of the old classes together with the new training data for learning and to retain the old knowledge. Since it is not feasible to continue storing exemplars of all the old classes every time a task is learnt, the number of exemplars per class is decreased by selecting a limited number of exemplars that is closest to their respective class mean. The downside of this approach is that the greater the number of classes that are observed, the lesser the number of exemplars will be available for each old task, and also, [15] stores raw images as exemplars which tends to have high memory demands and may not be feasible for either IoT edge devices or the cloud. Other researchers have developed an incremental classifier learning with a generative adversarial network (GAN) (ICwGAN) [26]. In [26], a similar approach to iCaRL [15] is used except that instead of using real images as exemplars, GANs are used to generate images that replicate the original exemplars. Although this approach solves the issue of privacy, the drawback is the computational overhead of training and running a GAN. In addition, the system needs to store the original images as well as a GAN to generate images and such a process is memory demanding. The REMIND approach in a neural network that can be used to prevent catastrophic forgetting [27] is another Convolutional Neural Network (CNN)-based model that tackles the problem of catastrophic forgetting using rehearsal. Unlike the prior work where raw images of the previous classes are stored, these methods store quantized tensors for rehearsal, which is less memory demanding. As the model learns incrementally, at the end of each incremental training step, instead of storing raw images, the images are passed through the convolutional layers and these output features are quantized and stored in the memory as exemplars. However, this work does not address how the model can be distributed between the cloud and IoT edge devices. Other researchers developed an Autoencoder-Based Incremental Class Learning without Retraining on Old Data method [28] which uses an autoencoder as a classifier instead of the traditional SoftMax classification layer. The main reason for this is that when using a SoftMax layer, neurons must be added to the layer whenever a new class is encountered. By using an autoencoder, this issue can be avoided. For classification, the mean of the feature maps of the respective classes is stored as code vectors thus reducing memory and computation cost. However, this work requires base training. Base training is conducted when the model is trained on a few initial classes, and only after this stage, the incremental learning begins. Synaptic intelligence [29] or Memory aware synapses [30] loss methods are added to reduce the effect of forgetting.

Federated learning [31], [32] is a new upcoming area of research in decentralized training in an edge-cloud setting i.e., there is a common shared model that is trained on millions of IoT devices using the local data that is present on the device itself. The updated model is then sent to the cloud for global model aggregation. This work greatly reduces communication with the cloud and ensures data privacy by training models on IoT edge devices. This method only transmits the encrypted updated model to the cloud instead of sending the data to the cloud. However, there are several drawbacks if this work is applied in the incremental learning context. First, the training time taken on the edge devices to learn patterns from high dimensional data such as images is very long. It is reported in [33] that for a simple dataset such as the Canadian Institute For Advanced Research (CIFAR)-10 [34], it takes 8 hours 41 minutes to train a MobileNets [35] model for one epoch on a Raspberry Pi. Moreover, Raspberry Pi is

considered a high-end IoT device but if the same model is split between the Raspberry Pi and the cloud i.e., some layers running on the IoT edge device and the remaining layers on the cloud, then the training time for one epoch is be reduced to 2.5 hours. Therefore, when dealing with high dimensional data such as images, local training of the entire model may not be feasible. Such data requires sheer processing power for training which can only be fulfilled by powerful hardware such as the graphics processing unit (GPU)s which generally reside on the cloud.

A common technique to accelerate training of deep neural networks without degrading accuracy is by discarding data samples that have very low loss values after a number of training epochs where the loss values of such samples do not decrease further [36]–[38]. This is because once the loss values of certain samples do not decrease, it means the model already understands such samples very well, and thus training is accelerated by focusing on samples that have high loss values that are yet to be understood by the model. However, such approaches can only perform the data sampling after model training begins, whereas we aim to perform data sampling before training starts. Another way to perform data sampling is by eliminating redundant images from a given dataset [39]. The downside of this approach is the slow computational speed because every image has to be compared with every image in the dataset to find out all the dissimilarities. When training support vector machines (SVM) on large-scale datasets, pre-selecting support vectors is a solution to accelerate SVM training [40], techniques include using genetic algorithms [41], clustering to select scattered samples because samples that are densely clustered are deemed redundant [42], enclosing samples in a convex hull and selecting boundary points [43], [44]. However, our work focuses on pre-selecting data for reducing the number of samples being transmitted to the cloud and accelerating neural network training.

Active learning [17] also has several query strategies that are used to select samples based on a given criteria, for example, samples with least confidence, highest loss, highest expected model change, etc. However, these methods do not mention how many such samples should be selected from a given data distribution in a way such that the selected data samples can still yield nearly the same learning performance if trained on a given model as compared to using all of the data distribution.

The FitCNN [16] approach proposes a cloud-assisted framework to run deep learning models on IoT devices. The method proposes two main strategies as follows: firstly, a data sampling algorithm i.e., to reduce data transmission to the cloud during incremental learning and secondly, to select useful weights of the new model trained on the cloud and to update the old model on the IoT edge device only based on these useful weights. To reduce the amount of data transmission to the cloud, a CNN runs inference on the IoT edge device and only sends the samples to the cloud for further learning [45]–[47] if the confidence of the samples is less than a certain threshold value. To keep the CNN model on the IoT devices up to date, after carrying out model training on the cloud, a weight difference is computed between the trained model and the old model. The weight difference is then used to select which updated parameters of the model should be sent back to the IoT edge device.

In general, FitCNN [16] is the most closely related work to ours except that [16] is a single task incremental learning system i.e. the model learns examples of the same class in an incremental learning manner. Our method is a multitask incremental learning system i.e., learning completely new classes incrementally. That is why it is of paramount importance to have efficient data streaming techniques in place for the multi-task incremental learning scenarios. Next, the parameters of the trained model on the cloud must also be transmitted back to the IoT edge device effectively. FitCNN [16] has already achieved this by sending back only the important parameters. We propose to improve this algorithm by finding the important parameters much faster.

III. METHODOLOGY

Fig. 1 shows our system flow chart. The design is capable of running on multiple IoT edge devices. The blocks highlighted in Fig.1 are our contributions. To explain the methodology from a high-level perspective, an ImageNet [48] pre-trained feature extractor is run on the IoT edge device along with a classifier. The classifier on the IoT edge device, however, is used only for inference. The training of the classifier takes place on the cloud. The weights associated with the newly added neurons in the SoftMax layer of the classifier are randomly initialized on both the IoT edge device and the cloud for the new classes to be learnt. The learning rate of the feature extractor is set to 0 i.e., the CNN feature extractor layers are frozen. This indicates that the feature extractor parameters will not be modified any further and implies that once backpropagation takes place on the cloud, there is no need to transmit the gradients back to the IoT edge device.

At every incremental training round, n number of classes are trained together at a time. Data sampling takes place per class therefore, all samples in each class are forward propagated through the model to obtain the loss values of all the samples. The samples with very low loss values with respect to the losses of training samples are counted on the IoT edge device based on which data sampling takes place. After forward propagating the selected samples through the feature extractor of the CNN, the output feature maps of the selected samples are converted from tensor to a string format and stored on the IoT edge device. As the tensor converted strings are stored on the IoT edge device, RAM consumption increases which can slow down the processing on the IoT edge device, and as soon as a slowdown is detected, the tensor strings are transmitted to the cloud via the Transmission Control Protocol (TCP/IP). Otherwise, transmission takes place once data sampling has been performed for a given class. The cloud listens for the incoming data streams and converts the strings into tensor feature maps and

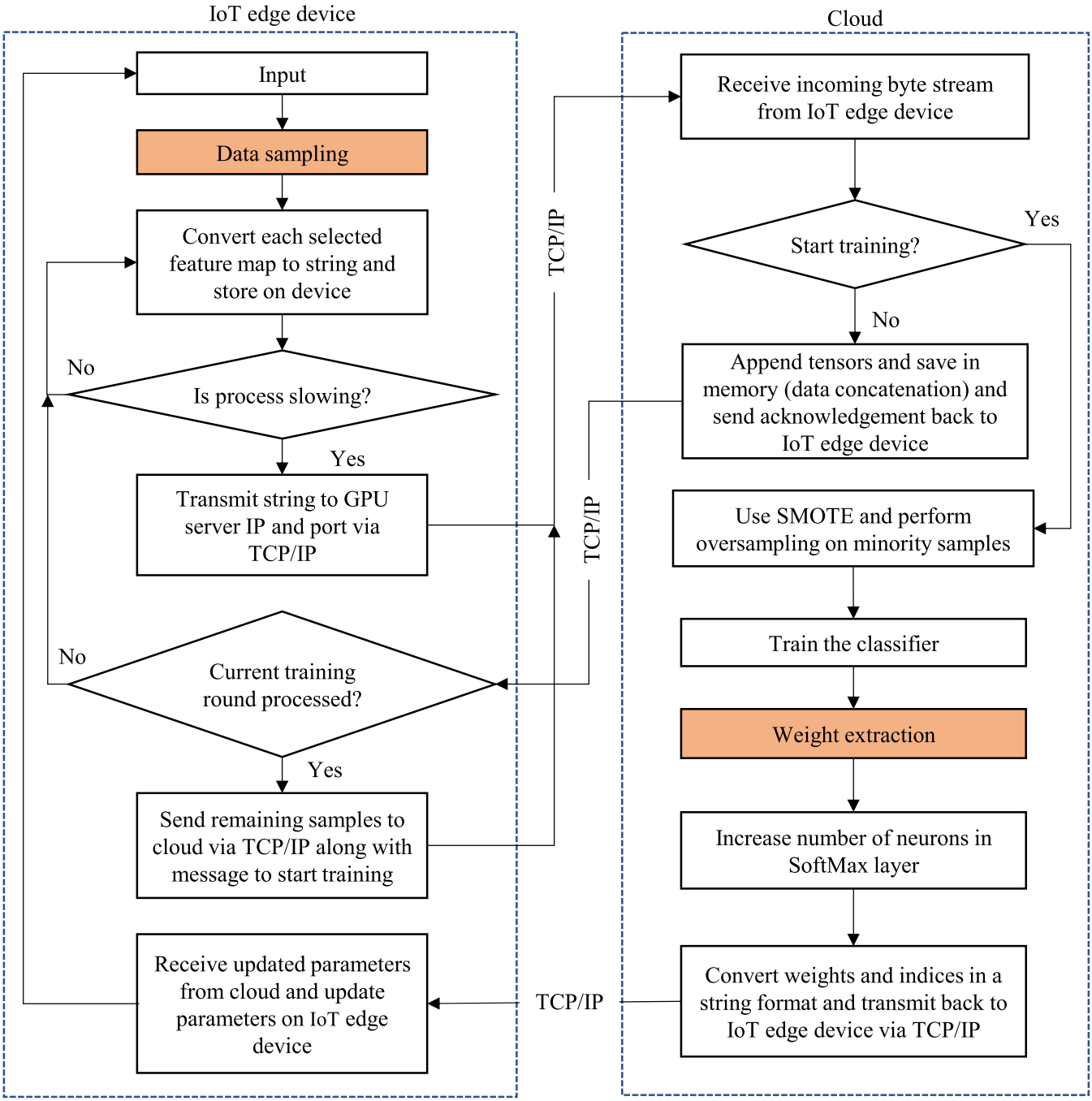

FIGURE 1. Our proposed system flow chart. Incremental learning system partitioned between an IoT edge device and the cloud.

saves them. The training on the cloud only starts once all the classes belonging to an incremental training round have arrived. As new classes are sent to the cloud for training, if an imbalanced dataset is detected then the minority classes are oversampled using the Synthetic Minority Over-sampling Technique (SMOTE) [49]. The features of all the previous classes are used as exemplars for incremental learning. So, when learning new tasks, all the features of the previous classes are used together with the features of the current samples belonging to the new classes to train the classifier on the cloud.

Once all of the data of a particular incremental training round has been sent to the cloud, all the tensor feature maps on the cloud are used to train the classifier. During the classifier training on the cloud, the only part of the classifier that changes dynamically is the SoftMax classification layer on the cloud i.e., new neurons are added to the SoftMax layer based on the number of new tasks to be learnt after every

incremental training round. Once the classifier is trained, weight extraction is performed whereby only the useful weights of the trained model are chosen to be transmitted back to the IoT edge device. Together with the useful weights, the indices of the useful weights are also transmitted to the IoT edge device. These indices denote the exact connections of the classifier at the IoT edge device that must be updated with the useful weights received from the cloud.

A. DATA SAMPLING FROM NOVEL CLASSES ON AN IOT EDGE DEVICE

The objective of data sampling is to discard certain samples on the IoT edge device and to reduce the communication load, in exchange for a very small difference in the incremental learning performance. This can improve the efficiency of a deep learning model partitioned between the cloud and an IoT edge device while retaining the incremental learning performance. Although data sampling may result in slight accuracy changes, we found that a 3% difference in classification accuracy is acceptable. This is because when a given deep learning model is trained on separate occasions with the same hyperparameters and model architecture, the model will yield slightly different classification accuracies. This is due to the random initialization of the model weights. So, because of this natural stochastic property of neural network training in which slightly different classification accuracies are yielded every time a model is run, a slight difference in accuracy should also be acceptable due to data sampling on the IoT edge device. The size of the test dataset can also vary for different datasets. Consequently, incremental learning performance can also vary with respect to data sampling. This is why we choose a margin of 3% for classification accuracies obtained after data sampling as compared to no data sampling.

Here, we propose our Data Discard Counting (DDC) algorithm. Let c be the class index of the current class being processed, n_c be the number of samples in each class thus the number of losses of all samples in class c is also n_c. Consider a set L^c which contains all the losses for all the samples in the current training class, such that $L^c \in \{l_0^c, l_1^c, l_2^c, \ldots l_{n_c}^c\}$ where L^c is sorted in ascending order. L^c is computed using the cross-entropy loss function and μ_L^c is the median of the losses of n_c samples. In our method, we use a pre-trained feature extractor with a single layer neural network whose weights are completely randomly initialized for data sampling. The formulas used to discard training samples at the IoT edge device are described one step at a time below.

$$l_i^c = -\sum\nolimits_j (y_i^c)_j \cdot \log_e(\varphi(W' x_i^c + b'))_j \tag{1}$$

In (1), the cross-entropy values of incoming samples are calculated as the first step of our DDC algorithm where x_i^c is the feature map i of class c coming into the SoftMax layer, y_i^c is the one-hot label corresponding to x_i^c. W' and b' are the weights and biases of the new randomly initialized n neurons in the SoftMax layer on the IoT edge device where n is the new number of classes to be learnt. $\varphi(\cdot)$ is the SoftMax function. j is the index of the newly added neurons (in the SoftMax layer). l_i^c is the loss of the sample i of set L^c.

$$\sigma_L^c = \sqrt{\frac{\sum_{i=1}^{n_c} (l_i^c - \mu_L^c)^2}{n_c}} \tag{2}$$

In (2) we propose the term σ_L^c that denotes by how much the metrics l_i^c differ from the median (μ_L^c) of the distribution of the set L^c of class c. We use the median value instead of the mean in order to protect our DDC algorithm from skewed entropy values in L^c. For example, a skewed value can be an extremely high entropy value in L^c that would largely affect the mean value of L^c. This could greatly increase the value of L_{cutoff}^c in (3) which would lead to an extremely high data sampling rate and thus affecting the learning performance of the model on the cloud. However, the median of L^c does not get affected by such skewed values. As a result, the data sampling rate remains stable as per (3) and (4). This is the second step of our DDC algorithm.

$$L_{\text{cutoff}}^c = \begin{cases} (l_i^c \cdot \mathbb{I}\left(\left|\mu_L^c - l_i^c\right| < \sigma_L^c\right)), & \mathbb{I}(*) = 1, i > 0 \\ l_0^c, & \mathbb{I}(*) = 0, i = 0 \end{cases} \tag{3}$$

In (3), $\mathbb{I}(\cdot)$ is the indicator function whose output is 1 if the condition inside the indicator function is true or else the output is 0. The term L_{cutoff}^c is the cut-off loss which also tells us about the number of samples with low loss values with respect to the training distribution. Determining the value of L_{cutoff}^c is the third step of our DDC algorithm. The total number of samples to be discarded from class c (τ^c) is expressed in (4) which is the final step of our DDC algorithm. In (4), let V^c be a set containing all loss values smaller than L_{cutoff}^c.

$$\tau^c = \sum\nolimits_{l \in V^c} 1 \tag{4}$$

In (3), if the distance between the low loss values in L^c and μ_L^c is smaller than σ_L^c, then we count all such samples that satisfy this condition. This process repeats iteratively until a sample is encountered where the distance between its loss and the median is greater than the standard deviation which means the largest loss value in L^c that satisfies (3) is considered to be the cut off loss (L_{cutoff}^c). L_{cutoff}^c is then used to count the number of samples to be discarded i.e., the number of values in L^c that have values less than L_{cutoff}^c is denoted by τ^c i.e., the number of samples to be discarded.

For each class, the loss distribution L^c is sorted in ascending order. σ_L^c denotes how far apart the values in L^c differ from μ_L^c in general. However, in such distributions, there can be values in L^c whose distance from μ_L^c is less than σ_L^c and there can also be values in L^c whose distance from μ_L^c is greater than σ_L^c due to the variations in the images per class which also means varying entropies of images per class as well. Images with high loss values can be beneficial for neural network training, because a high loss value implies a lot of weights will have to be fine-tuned thus improving the generality of the model as compared to images with low loss

values. Images with already low loss values cannot have their values greatly reduced as compared to images with high loss values. Hence, we must count the low loss values in L^c that vary slightly from μ_L^c with respect to σ_L^c which is why (3) has been designed. By doing so, we can determine the number of samples to be discarded from a class just before transmission to the cloud.

B. TRANSMISSION OF DATA TO THE CLOUD

Initially, the IoT edge device has no data. However, during training, when new class data arrives, such data is forward propagated through a CNN feature extractor. The output feature map is then converted to a string format and saved in a buffer along with its respective label. The same process is applied to the other incoming mini batches of images. All the output feature maps of images are concatenated to a buffer which stores all the tensor converted strings. This buffer (M_{ec}) is what is transmitted to the cloud via the TCP/IP protocol. The format of M_{ec} is shown in (7). The description of ''act'' is as follows: "a!" (for data concatenation), "t!" (for training), "d!" (for process termination).

Along with the feature maps, the associated labels must also be transmitted to the cloud. Since each feature map has one unique label (denoted by ''lab'' in (5)), let N be the total number of feature maps and thus the label format is shown in (5).

$$\text{lab}_{\text{str}} = \text{lab}_n, \text{lab}_{n+1}, \ldots \text{lab}_{n+N} \tag{5}$$

A single feature map will have a depth of one and the same width (W_n) and height (H_n) as the overall feature map of an image. The string format of a single feature map can be expressed as shown in (6).

$$\text{FM}_n = \text{val}_{n,i}, \text{val}_{n,i+1}, \text{val}_{n,i+2}, \ldots \text{val}_{n,S_n} \tag{6}$$

All values are comma-separated when the feature maps and the labels are converted to a string and each feature map string is separated by the character 'k'. The overall message format is written as shown in (7).

$$M_{ec} = D_n, W_n, H_n, <\text{lab}_{\text{str}}>, \text{FM}_n, k, ..\text{FM}_N, <\text{act}>! \tag{7}$$

On the cloud, the incoming stream is accepted, and the reading operation continues until the end of message character '!' is detected. Once the end of message character is detected, the action character (''act'') is obtained, and based on that action, an appropriate process is carried out. For example, if the action character is $'a'$ then tensor concatenation is carried out, and if the action character is $'t'$ then training is carried out, and if the action character is $'d'$ then the program on the cloud stops executing. At the end of every incremental training round on the cloud, only the useful weights of the trained classifier are sent back to the IoT edge device. Upon receiving this message, the IoT edge device then proceeds with processing the next batch in the dataset i.e., converting the images and labels to strings. This process continues until all the samples in the dataset are processed.

In (7), the end of message character (''act'') is appended at the end of the tensor converted string because the TCP/IP protocol is not a message-based protocol but a stream-based protocol. It means that there is no guarantee that all the bytes that are transmitted will arrive at the recipient. So, the recipient must keep listening for the incoming string from the IoT edge device and only stop reading when the end of message character '!' is encountered. The character '!' is always appended at the end of the message, therefore if this character is encountered, the recipient device knows that all the transmitted bytes from the sender have been received.

Synchronization between the IoT edge device and the cloud is ensured by the following steps: firstly, when the IoT edge device transmits feature maps of the images to the cloud, the IoT edge device waits for a reply from the cloud. Secondly, if the reply is not received from the cloud, the IoT edge device will not carry out any other processes. Thirdly, the cloud will keep listening for incoming data and will only proceed once the end of message character is read ('!').

The amount of memory available in the IoT edge device hardware is limited and this is a key factor to consider in distributed processing scenarios. Since we store the CNN feature extractor output of the input images at the IoT edge device, this may lead to RAM scarcity which causes a slowdown in the processing speed. This is because in any modern operating system (OS), when a program requires more RAM, the OS will allocate the required memory to that program. However, when RAM starts to run out, the OS will move some of the program's memory to disk. In other words, the OS now needs to move the data more frequently between disk and RAM, resulting in a slower response time. For a given deep learning model, if it is required to store more feature maps on the IoT edge device, more memory will be consumed with respect to the number of samples, resulting in a slowdown of the IoT edge device processing speed. To account for this scenario, a very simple algorithm is formulated in (8).

$$\text{transmit} = \begin{cases} \text{yes}, & (t_{\text{mb}_0} - t_{\text{mb}_n}) > 1 \\ \text{no}, & \text{otherwise} \end{cases} \tag{8}$$

In (8), the time taken to forward propagate the first mini batch is recorded (denoted by t_{mb_0}). If the difference between the time taken to process subsequent mini batches (denoted by t_{mb_n}) and t_{mb_0} exceeds one second, it indicates that the processing speed of the IoT edge device is slowing down. In such a scenario, all feature maps, stored on the IoT edge device thus far, are sent to the cloud and subsequently, the memory on the IoT edge device will be cleared.

C. INCREMENTAL LEARNING ON THE CLOUD

In our work, only the SoftMax classification layer grows in size. The number of neurons added to the SoftMax layer at every incremental training round is equal to the number of new tasks to be learnt at every training round. Once the feature extractor outputs are sent from the IoT edge device to the cloud, these features are saved on the cloud and as the training

process continues, these features are used as exemplars. The exemplar set is constructed following (9) below.

$$E_t = \emptyset\,(X_t) \cup \emptyset\,(X_{t-1}) \cup \emptyset\,(X_{t-2}) \cup \ldots \emptyset\,(X_0) \tag{9}$$

In (9), let X be all the samples of an incremental training round and $\emptyset(\cdot)$ be the output of the CNN feature extractor and t be the incremental training round. E_t is defined as the exemplar set at incremental training round t and represents the new set of samples being fed into the classifier. The new batch of feature maps and their respective labels are randomly shuffled after every epoch to maintain the classification accuracy. Next, the combined feature maps (E_t) are passed through the fully connected layers. During the backpropagation process, only the fully connected layers are updated (we freeze the feature extractor layers). This process of training is known as joint training and is one of the most common techniques to alleviate catastrophic forgetting.

D. IMPROVED WEIGHT EXTRACTION ALGORITHM

After training the classifier on the cloud, the updated parameters of the classifier need to be transmitted back to the IoT edge device to keep the model on the IoT edge device up to date. The work in [16] shows that it is not mandatory for all the weights of a trained model on the cloud to be transmitted back to the IoT edge device as there are a number of parameters in a model that do not change after the training i.e. the difference between the weight value of a connection in the trained classifier and the pre-trained classifier is the same. The juicer strategy (proposed in FitCNN [16]) performs useful weight extraction layer by layer. For each layer, the FitCNN [16] juicer algorithm takes the difference between the weights of a layer of the trained and the pre-trained model and divides the weight difference distribution into 30 quantiles to obtain a threshold list. For each threshold value, if the absolute weight difference between a classifier's post-trained weight and pre-trained weight is greater than the threshold value, then for that particular connection, the post trained weight value will be considered useful, or else the pre-trained weight value will be considered useful. A temporary model is then formed that has the same architecture as the trained model which has a mixture of useful and non-useful weights. The training accuracy is then computed on this temporary model. For every threshold value, the constructed temporary model is different in terms of the number of the useful parameters. Therefore, for every threshold value, the training accuracy, the number of useful weights, and the indices of the useful weights of the temporary model are stored. For a given threshold, as soon as the difference between the training accuracy of the temporary model and the trained model is greater than the predefined acceptable training accuracy loss, the useful weights and indices associated with the previous threshold value are considered useful weights for the layer of the model being processed. Next, the layer of the trained model being processed is frozen, and the next layer of this model undergoes the same weight extraction process. This is how the useful weights of a trained model are extracted in FitCNN [16].

However, in the case of class incremental learning, the number of negative weight differences can increase with respect to the training rounds due to catastrophic forgetting. Fig. 2 shows the frequency distribution of the weight differences where the x-axis represents the value of the weight difference and the y value represents the frequency i.e., the number of times a specific weight difference occurs. For example, the points highlighted in the red circles in Fig. 2 show the number of weight differences whose values are zero. This means the value of such weights before and after training remain the same. So, it is not necessary to send such weights back to the IoT edge device.

As shown in Fig. 2b, if the FitCNN [16] juicer strategy is used in our problem, the threshold list will contain more negative threshold values which will simply yield the same number of useful weights. This is because according to the FitCNN [16] juicer algorithm, if the absolute value of a weight difference is greater than a threshold, the corresponding weight of the post trained model will be considered useful. However, we find that the absolute value of any weight difference value will always have positive values. Hence, the condition in line 15 of Algorithm 1 will never be satisfied if the threshold is negative thus all the weights of the trained model will be considered useful. Choosing negative threshold values, therefore, yields a lot of redundancy and is also time-consuming, especially for class incremental learning. In class incremental learning, the size of the training dataset can be very large as new tasks arrive which is why there is a clear need to extract useful weights using as few thresholds as possible from the weight difference distribution list.

We propose to sort the weight difference distribution of each layer in ascending order and use only non-negative threshold values (line 6 of Algorithm 1) in order to speed up the process of finding the minimum useful weights of a trained classifier with negligible training accuracy loss. Algorithm 1 shows our improved version of the original FitCNN [16] juicer algorithm. In Algorithm 1, all the weight matrices are converted into 1-D vectors to make it easier to work with indexing.

For each layer in the model, the weight differences are computed between the weights of the layers before and after training which are then sorted in ascending order. Just like the FitCNN [16] juicer algorithm, we divide the weight difference distribution into 30 quantiles. However, we store the weight difference value at every quantile as a threshold only if the weight difference value is non-negative. Next, a temporary model with the same architecture as the classifier on the cloud is created which has the same parameters as the newly trained model but if the weight difference of a parameter of the trained model at a specific index is less than the threshold, the weight of the pre-trained model will be inserted at the given index into the temporary model. The accuracy of the temporary model is computed using the data used to train the model at the respective incremental training round. If the

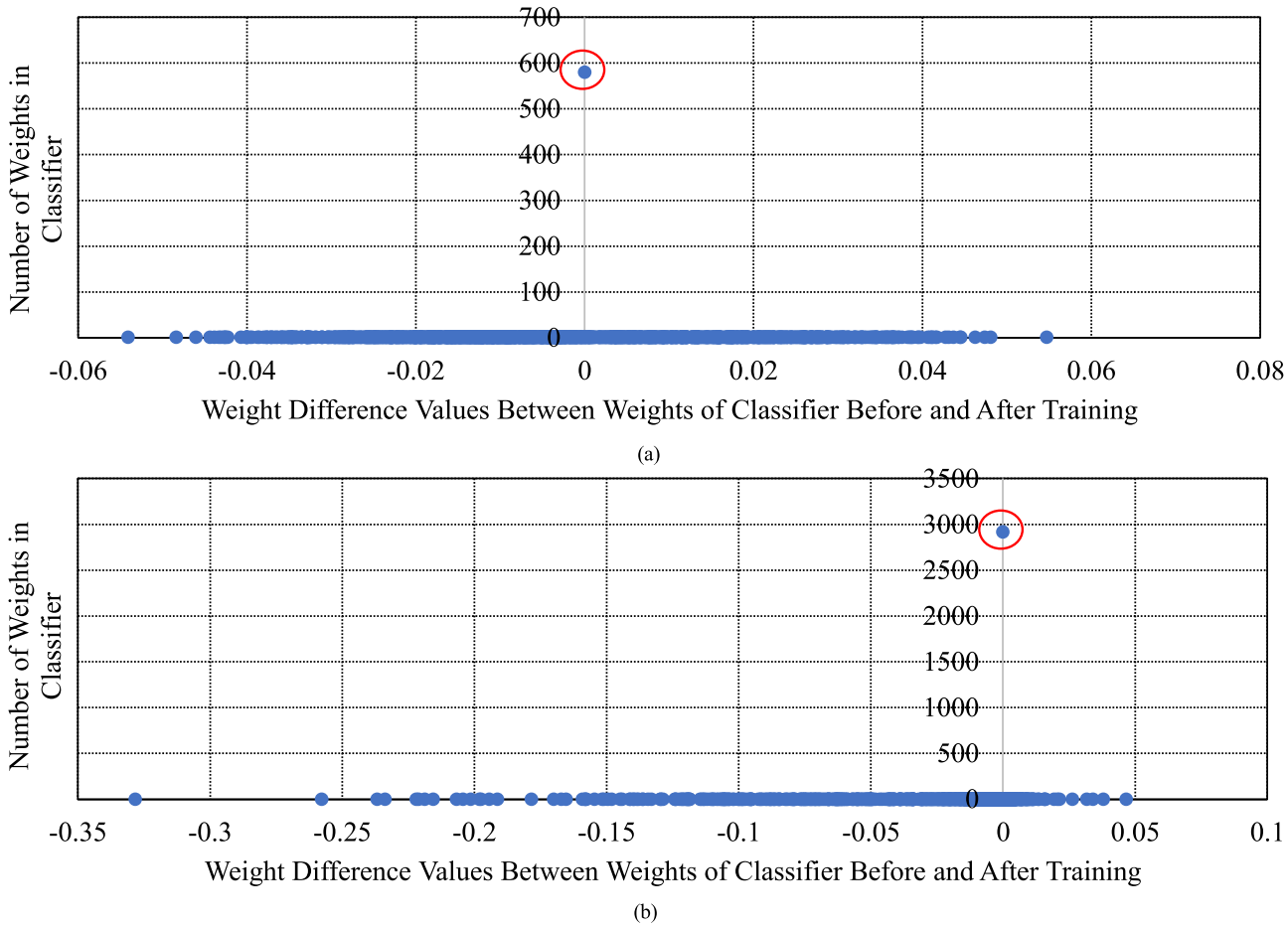

FIGURE 2. Distribution of the weight difference between the classifier on the cloud before and after training when learning (a) the first 10 classes (b) last 10 classes of the CIFAR-100 [34] dataset with rehearsal (training all samples accumulates so far) using the SqueezeNet [50] model.

difference between the accuracy of the temporary model and the newly trained model is within A_{ori}, then the process of finding the useful weights continues using a larger weight difference threshold value i.e., the next threshold value in our threshold list. The larger the threshold value, the higher the chances of the condition in line 15 being satisfied thus the number of useful weights of a layer decreases, and as a result, reduces the communication costs when transmitting only the useful weights of the trained model back to the IoT edge device.

Once the difference between the accuracy of the temporary model and the original trained model is greater than A_{ori}, then the weights of the latest temporary model are considered to be the final set of useful weights and the iterative process will stop. Our algorithm reduces the number of iterations needed to find the useful weights as compared to the FitCNN [16] juicer algorithm by simply using positive threshold values of the weight difference distribution. At the same time, we ensure that the accuracy of the temporary model is similar to the accuracy of the original trained model. Once the useful weights are determined for a specific layer in the temporary model, the parameters of that layer are frozen in order to find the useful weights in the next layer of the temporary model that undergoes the same weight extraction process.

Line 11 in Algorithm 1 is another improvement that we propose. We first use the maximum value in the threshold list to find the useful weights and check if the accuracy of the temporary model is close to the accuracy of the trained model. We do this because, if this is the case, then there will be no need to iterate through the entire threshold list to find the useful weights which further reduces the number of iterations needed to find the useful weights.

Apart from the useful weights, it is also mandatory to send the indices of the weights of the trained model back to the IoT edge device. This is to let the IoT edge device know which specific connections of a layer of the model must be updated. Indices are chosen depending on the number of useful weights. If the useful weights are greater than the number of non-useful weights, then obviously the number of indices that indicate the position of the useful weights will be larger. However, to make the communication more efficient, if the number of useful weights is greater than the number of non-useful weights then the indices of the non-useful weights will be sent back to the IoT edge device. The message format

Algorithm 1 Our Proposed Improved Juicer Algorithm

Input: C_{updated}, C_{old}, $\text{Accept}_{\text{loss}}$, A_{ori}, q
Output: W_{best} and I_{best}

for l in L **do**
  $C^l_{\text{diff}} = C^l_{\text{updated}} - C^l_{\text{old}}$
  $\text{sorted}C^l_{\text{diff}} \leftarrow \text{sort}\ (C^l_{\text{diff}})$
  $\text{Divider} \leftarrow \text{lengthofsorted}C^l_{\text{diff}}/q$
  for w_{diff} in $\text{sorted}C^l_{\text{diff}}$ **do**
    if $|w_{\text{diff}}| > 0$ **then**
      $[T^l] \leftarrow w_{\text{diff}}$ at every divider
    end if
  end for
  $\max_t \leftarrow \text{Max}\ (T^l)$
  $T^l \leftarrow$ Remove $\max_t$ from the last index of T^l and insert it at the first index of T^l
  $C_{\text{temporary}} \leftarrow C_{\text{updated}}$
  for threshold in T^l **do**
    for w_{diff} in C^l_{diff} **do**
      if $|w_{diff}| <$ threshold **then**
        $C^l_{\text{temporary}} \leftarrow C^l_{\text{old}}$ at respective index
      end if
    end for
    $[\text{Clist}_{\text{temporary}}] \leftarrow C_{\text{temporary}}$
    $A_{\text{temp}} \leftarrow$ Get $C_{\text{temporary}}$ accuracy on training data
    if $(A_{\text{ori}} - A_{\text{temp}}) \geq \text{Accept}_{\text{loss}}$ **then**
      $W^l_{\text{best}}, I^l_{\text{best}} \leftarrow \text{Clist}^l_{\text{temporary}}$ at previous index of list, indices of W^l_{best}
      break
    else if (threshold == $\max_t$) **then**
      $W^l_{\text{best}}, I^l_{\text{best}} \leftarrow \text{Clist}^l_{\text{temporary}}$ at current index of list, indices of W^l_{best}
      break
    else
      $W^l_{\text{best}}, I^l_{\text{best}} \leftarrow \text{Clist}^l_{\text{temporary}}$ at current index of list, indices of W^l_{best}
    end if
  end for
  $[W_{\text{best}}] \leftarrow W^l_{\text{best}}$
  $[I_{\text{best}}] \leftarrow I^l_{\text{best}}$
end for

of sending the weights and indices back to the IoT edge device is expressed in (10) below.

$$M_{\text{ce}} = < W_{\text{best}} >, i, < I_{\text{best}} >, b, < B_{\text{trained}} >, < \text{act} > ! \quad (10)$$

W_{best} is converted into a 1-D vector and as shown in (10), this flattened weight vector is converted into a comma-separated string. The character $'i'$ denotes that the next sequence of comma-separated strings is the indices. The character $'i'$ denotes the next sequence of comma-separated strings are the biases. The term "act" is either the character $'y'$ or $'n'$. Character $'y'$ indicates that the indices present in the string are exactly where the weights need to be assigned in the weight matrix on the IoT edge device and the character $'n'$ indicates that the indices present in the string are where the updates are not needed.

IV. EXPERIMENTAL SETTING

In this section, we conduct experiments to evaluate the performance of our DDC algorithm in the context of the incremental learning system shown in Fig. 1. The datasets used in the experiment are the CIFAR-100 [34] dataset and the Caltech-UCSD Birds (CUB)-200 [51] dataset. CIFAR-100 [34] dataset contains RGB images of size $32 \times 32 \times 3$ pixels and has a total of 100 classes with 500 training images per class and 100 testing images per class. For the CUB-200 [51] dataset, there are a total of 200 classes where the number of training and testing images per class is different. This is an imbalanced dataset and the spatial dimensions of the images in this dataset are also different i.e., not all the samples have the same width and height.

Before the incremental training begins, we randomly initialize our SoftMax layer with n neurons where n is the number of classes to be learnt per incremental training round.

For the communication between the IoT edge device and the cloud, the features are encoded into a utf-8 format and then transmitted via the TCP/IP protocol. Since the aim of this work is not representation learning, we use pre-trained CNN models as feature extractors on the IoT edge device and a single SoftMax classification layer on the cloud. The number of neurons in the SoftMax layer increase by n after every incremental training round.

For CIFAR-100 [34], this dataset contains samples from 100 classes that are shuffled. Therefore, before beginning incremental training, all of the samples are grouped together with respect to the label. We use the pre-trained convolutional layers of both SqueezeNet [50] and ShuffleNet V2 [52] as the feature extractors. We use the Adam Optimizer [53] together with the categorical cross-entropy loss function. All the layers except the SoftMax classification layer use the ReLU activation function, the fully connected layers use dropout with a probability of 0.25, and a batch size of 128. However, for SqueezeNet [50], 70 epochs of training are used per incremental training round with a learning rate of 0.0001 and for ShuffleNet V2 [52], 25 epochs of training are used per incremental training round with a learning rate of 0.001.

The CUB-200 [51] dataset is organized in sub-directories whereby each sub-directory represents a single class, therefore we automatically assign an integer label for each sub-directory. The pre-trained convolutional layers of both ShuffleNet V2 [52] and SqueezeNet [50] are used as feature extractors with a single SoftMax classification layer. We use the Adam Optimizer [53] together with the categorical cross-entropy loss function. All the layers except the SoftMax classification layer use the ReLU activation function, the fully connected layers use dropout with a probability of 0.25, and a batch size of 128. For the CUB-200 [51] dataset, the images are resized to $90 \times 90 \times 3$ pixels. However, for SqueezeNet [50], 25 epochs of training are used per incremental training round with a learning rate of 0.0015 whereas, for ShuffleNet V2 [52], 30 epochs of training are used per incremental training round with a learning rate of 0.002.

All the accuracies reported in the experiments are the top-5 accuracies (rounded off to 2 decimal places) on the test dataset and averaged over executing the respective experiment three times. We choose SqueezeNet [50] and ShuffleNet V2 [52] feature extractors because these models are lightweight and are specially designed for embedded devices with resource constraints. The PyTorch [54] library is used for developing and testing the experiments. The input images are normalized by converting the RGB images from a range of 0 to 255 to become 0 to 1. A laptop with an i7 processor and 8 Gigabytes RAM is used as an edge device. The Google cloud platform is used as the cloud with the Nvidia Tesla K80 GPU.

V. RESULTS AND DISCUSSIONS

In this section, we show the results of our DDC and improved weight extraction on the cloud algorithms, analyze, and discuss the results. For our DDC algorithm on the IoT edge device, we compare our method with three baseline methods: NS, MTS, and WRSTS. We compare our weight extraction algorithm with the novel FitCNN [16] weight extraction algorithm. For evaluating our DDC algorithm and the improved weight extraction algorithm in the class incremental learning scenario, we test our proposed algorithms by incrementally learning a different number of classes at a time. This is to observe the performance of our algorithms under different learning settings. As learning a different number of classes at a time also means learning a different number of samples at a time, we would like to observe the performance of our DDC algorithm when it learns a different number of samples at a time. This can give a good insight as to whether the data sampling algorithms can maintain the classification accuracy of the model when learning a different number of data samples per incremental training round. It is important to test our proposed juicer algorithm under these settings where a different number of classes are learnt incrementally at a time because the learning process of a deep learning model varies with respect to the number of samples. We can also observe how many useful weights the improved juicer algorithm can extract when learning a different number of classes at a time.

It is very important to note that though we carry out data sampling using various methods such as RS, ES [17], LCS [17], MTS, WRSTS. We apply our DDC algorithm to RS, ES [17], and LCS [17] and we use NS, MTS, and WRSTS as the baselines. NS is the upper boundary since it involves no data sampling at all. The reason why MTS and WRSTS are also treated as baselines is because they can automatically detect a statistical difference between a selected data distribution with respect to the overall data distribution thus naturally becoming data sampling algorithms. MTS and WRSTS are non-parametric statistical significance tests that deduce whether there is a significant difference between any two given data distributions. MTS does this by comparing the medians of the data distributions and WRSTS does this by comparing the rank sums of the given data distributions. This is the reason these statistical tests are also chosen as baselines. We apply MTS and WRSTS on the loss distribution of samples obtained in (1) and transmit the samples with high loss values such that statistically, the distribution of the high loss samples is minimum in terms of the number of samples and such that it also represents the overall loss distribution. For statistical significance, we set the p-value for MTS and WRSTS at 5%. When applying our DDC algorithm to LCS [17], we simply replace the loss values in our DDC algorithm with the SoftMax probabilities of a sample at its given label index, τ^c samples with the highest SoftMax probabilities are discarded. When applying our DDC algorithm to ES [17], τ^c samples with the lowest entropies are discarded.

A. EVALUATION OF CIFAR-100

The reason for choosing this dataset is because of the large number of classes and a relatively large number of images per class. A large number of classes implies more incremental training rounds which provides a better testing ground

TABLE 1. Incrementally learning 10 classes at a time from CIFAR-100 [34] using SqueezeNet [50] and ShuffleNet V2 [52].

	Total number of classes trained									
	10	20	30	40	50	60	70	80	90	100
Classification accuracies when using SqueezeNet [50] feature extractor (%)										
NS	95.40	83.70	78.43	71.45	66.60	64.85	62.56	60.56	58.73	56.72
MTS	94.60	83.45	78.03	71.45	66.56	64.08	62.71	60.24	58.33	56.67
WRSTS	95.40	83.25	78.13	71.33	66.80	64.72	63.09	59.85	58.09	56.60
RS (DDC)	94.20	82.65	77.13	70.75	66.10	64.20	62.11	60.28	57.62	56.30
ES [17] (DDC)	95.70	82.60	76.93	69.50	66.02	63.23	61.04	59.34	57.29	55.28
LCS [17] (DDC)	94.60	81.30	75.80	69.63	65.16	62.93	60.96	59.09	56.74	55.24
Accuracy when using ShuffleNet V2 [52] feature extractor (%)										
NS	94.70	85.20	80.17	75.70	71.86	69.50	68.14	66.35	64.73	62.78
MTS	94.40	84.30	79.97	75.10	71.42	68.53	67.53	65.50	63.94	62.41
WRSTS	94.70	84.85	80.30	75.58	71.76	69.07	67.70	65.76	64.10	62.33
RS (DDC)	94.50	84.65	79.87	74.60	71.20	69.07	67.43	65.36	64.03	62.55
ES [17] (DDC)	94.60	83.85	79.13	73.70	70.40	67.70	66.49	64.60	63.19	61.61
LCS [17] (DDC)	93.90	83.05	78.70	74.10	70.54	67.97	66.76	65.06	63.24	61.56

TABLE 2. Incrementally learning 20 classes at a time from CIFAR-100 [34] using SqueezeNet [50] and ShuffleNet V2 [52].

	Total number of classes trained				
	20	40	60	80	100
Classification accuracies when using SqueezeNet [50] feature extractor (%)					
NS	83.80	71.67	65.23	60.99	56.94
MTS	83.05	71.35	64.22	60.32	56.64
WRSTS	83.45	71.65	64.63	60.79	56.63
RS (DDC)	83.25	71.25	64.47	60.45	56.39
ES [17] (DDC)	83.50	71.15	63.33	59.04	55.26
LCS [17] (DDC)	81.20	68.98	62.82	58.43	54.74
Classification accuracies when using ShuffleNet V2 [52] feature extractor (%)					
NS	84.60	74.98	69.03	65.85	63.06
MTS	83.85	74.20	68.38	65.30	62.32
WRSTS	84.35	74.65	68.73	65.48	62.65
RS (DDC)	84.15	74.08	68.47	65.09	62.27
ES [17] (DDC)	83.55	73.38	67.25	64.30	61.30
LCS [17] (DDC)	83.45	73.40	67.18	64.13	61.40

for our DDC algorithm to see whether incremental learning performance can be retained at every incremental training round.

To test our DDC algorithm, we train our model by learning a different number of classes incrementally i.e. training 10 and 20 classes incrementally on CIFAR-100 [34]. By applying our DDC algorithm to RS, ES [17], and LCS [17], it can be seen from Table 1 and Table 2 that for each incremental training round, the classification accuracies obtained is less than 3% irrespective of the data sampling method with respect to the accuracies obtained without any data sampling. This shows that our DDC algorithm can be successfully integrated with various data sampling techniques, resulting in the transmission of less samples to the cloud and still retain the model performance with respect to no data sampling. Fig. 3 shows the other performances of incremental learning under various data sampling techniques such as the number of samples transmitted to the cloud, the training time on the cloud, the number of useful weights extracted from the classifier, and the number of iterations needed to find the useful weights.

From Fig. 3, all data sampling methods result in a smaller number of samples being transmitted to the cloud and a faster training time on the cloud as compared to NS. The WRSTS method appears to be sending more samples to the cloud as compared to other data sampling techniques indicating that this method is able to quickly detect a statistical significance difference between the overall loss distribution and the distribution of the losses whose associated samples are to be transmitted to the cloud. Such quick detection of statistical significance difference is undesirable in such cases because we can clearly see that using other data sampling methods results in fewer samples being transmitted to the cloud as compared to WRSTS while the model performance is similar to that of the accuracies obtained using NS.

It can be noted that learning 10 classes at a time takes more time than learning 20 classes at a time on the cloud because the more the number of classes to be learnt at a time, the more the number of rehearsals needed for incremental learning thus the overall training time on the cloud increases. As Fig. 3c and Fig. 3d show, the cloud training time is faster after data sampling is performed at the IoT edge

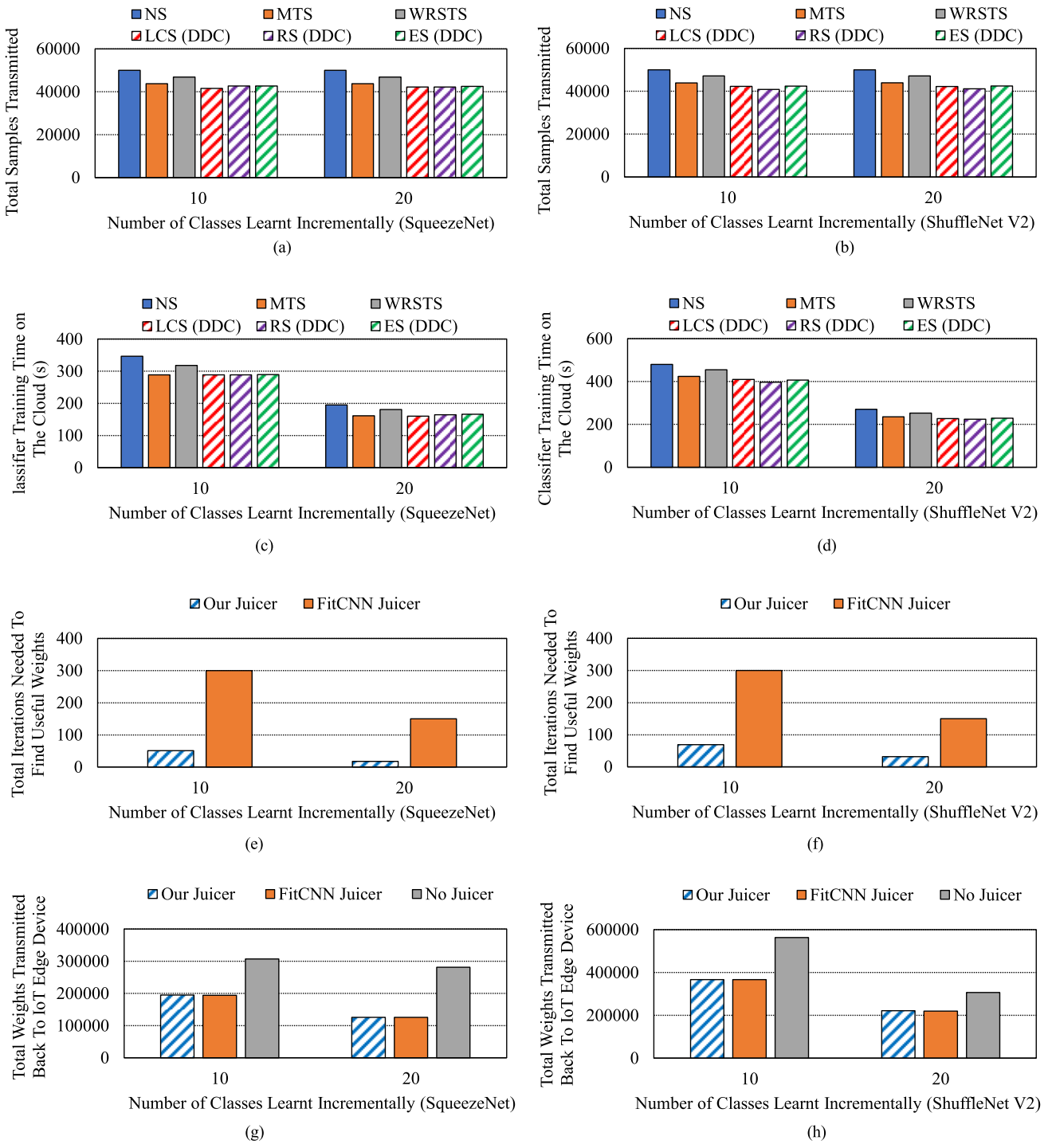

FIGURE 3. Total samples transmitted to the cloud (a) using SqueezeNet [50] (b) using ShuffleNet V2 [52]. The cloud training time (c) using features from SqueezeNet [50] (d) using features from ShuffleNet V2 [52]. Total iterations needed to find useful parameters of the classifier (e) using classifier associated with SqueezeNet [50] (f) using classifier associated with ShuffleNet V2 [52]. Total useful parameters received by the IoT edge device from (g) using classifier associated with SqueezeNet [50] (h) using classifier associated with ShuffleNet V2 [52]. Evaluation of CIFAR-100 [34] dataset on SqueezeNet [50] and ShuffleNet V2 [52] using the following data sampling techniques: NS, RS (DDC), ES [17] (DDC), LCS [17] (DDC), MTS, WRSTS.

device because not all samples have been transmitted to the cloud. The classifier residing on the cloud is trained on lesser data samples when data sampling is applied at the IoT edge device.

For the improved juicer algorithm that we propose, the number of useful weights we extract from the classifier is the same as that of FitCNN [16]. However, the main improvement is in the number of iterations needed to find the useful

TABLE 3. Incrementally learning 40 classes at a time from CUB-200 [51] using SqueezeNet [50] and ShuffleNet V2 [52].

	Total number of classes trained				
	40	80	120	160	200
Classification accuracies when using SqueezeNet [50] feature extractor (%)					
NS	58.49	43.85	36.68	31.53	27.77
MTS	51.15	37.81	29.44	23.59	20.90
WRSTS	56.84	42.53	33.59	29.90	25.60
RS (DDC)	57.39	42.71	36.53	29.00	27.53
ES [17] (DDC)	57.85	43.01	35.19	30.03	28.30
LCS [17] (DDC)	57.66	42.88	34.87	30.49	27.29
Classification accuracies when using ShuffleNet V2 [52] feature extractor (%)					
NS	71.99	59.94	49.26	41.98	37.15
MTS	63.73	49.85	39.16	32.85	28.46
WRSTS	69.15	55.58	43.95	37.45	33.26
RS (DDC)	71.63	57.68	47.33	40.29	35.30
ES [17] (DDC)	69.77	57.21	46.75	39.42	35.15
LCS [17] (DDC)	69.97	56.59	46.75	39.18	34.82

weights. The original FitCNN [16] juicer algorithm requires 30 iterations after every training round i.e. using 30 threshold values to find the best set of weights that represent the trained model which can be sent to the IoT edge device. In all the cases, we reduce the number of iterations required to find the best set of weights by at least 75% which greatly reduces the computational cost on the cloud required to find the useful set of weights. Since the classification accuracies at every incremental training round are less than 3% after applying the juicer algorithm, this suggests that not all the parameters learnt after training are useful.

B. EVALUATION OF CUB-200

We choose to test our proposed algorithms on this dataset because it has a much smaller number of samples per class as compared to CIFAR-100 [34] but with twice as many total classes as CIFAR-100 [34]. Performing data sampling is very challenging if the number of samples per class is small, for example, discarding 2 samples out of 100 samples reduces the sample size by only 2% but discarding 2 samples out of 10 samples reduces the sample size by 20% which is why it is very critical to test our DDC algorithm on small scale datasets. We test our algorithms under two settings i.e., learning 40 and 50 classes at a time. This is done to observe the performance of our algorithms when learning a large number of classes at a time.

For data sampling, it is important to note the size of image features, for example, when using the SqueezeNet [50] feature extractor, its output has a dimension of $3 \times 3 \times 512$ when the size of the input image is $90 \times 90 \times 3$. However, the size of the output feature map of ShuffleNet V2 [52] is $1024 \times 3 \times 3$, when the input image size is $90 \times 90 \times 3$. The ShuffleNet V2 [52] feature extractor output size is 2 times greater than the SqueezeNet [50] feature extractor output size in this case. The reason why we emphasize this is because even if the data sampling rate is very low at the IoT edge device, the transmission cost reduced can still be high given that the size of the feature maps obtained is very large either due to the CNN feature extractor architectural design and/or a higher image resolution. This shows the importance of data sampling at the IoT edge device for a large output feature map size and/or dealing with very high dimensional data.

To transmit data over TCP/IP protocol, the data must be converted to a byte stream. Moreover, since deep learning systems work with high precision floating-point numbers, each value in the feature map can contain many decimal places. For example, if the value at a given index of a single feature map is 0.462134632, then this value is worth 11 bytes. Therefore, the size of the data to be sent to the cloud increases in proportion to the precision of floating-point values. Similarly, when a high dimensional feature map is converted to a byte stream, this will need a huge amount of data to be transmitted. Hence, it becomes more important to reduce communication costs in an edge-to-cloud IoT context.

It can be seen from Table 3 and Table 4 that using MTS and WRSTS results in a severe model performance degradation in terms of the classification accuracies at every incremental training round. This implies that when using such non-parametric data sampling techniques, a lot of images per class are discarded at the IoT edge device as evident from Fig. 4a and Fig. 4b, leading to very few samples being transmitted to the cloud which also results in a smaller cloud training time as compared to other data sampling algorithms. When using the MTS method, by the time this test detects a significant difference between the means of the selected samples with respect to the total number of samples per class, a lot of samples have already been discarded. The same applies for WRSTS in which case a lot of samples already get discarded by the time the test finds a significant difference between the filtered samples and the overall dataset. This is very well because of the small number of images per class. Given the fact that the classification accuracies obtained when using the MTS method are extremely low, it means that WRSTS is faster at detecting statistical differences between the entropies of samples to be transmitted as compared to the entropies of all the samples. However, both of these non-parametric data

TABLE 4. Incrementally learning 50 classes at a time from CUB-200 [51] using SqueezeNet [50] and ShuffleNet V2 [52].

Total number of classes trained				
	50	100	150	200
Classification accuracies when using SqueezeNet [50] feature extractor (%)				
NS	53.42	42.81	34.15	27.48
MTS	47.08	33.98	27.21	21.36
WRSTS	52.05	38.95	29.73	25.47
RS (DDC)	55.72	40.24	32.01	27.53
ES [17] (DDC)	54.64	41.23	31. 28	26.96
LCS [17] (DDC)	51.48	37.93	29.06	27.89
Classification accuracies when using ShuffleNet V2 [52] feature extractor (%)				
NS	68.97	56.60	44.52	38.43
MTS	61.12	46.48	34.94	28.67
WRSTS	65.51	51.45	40.21	34.25
RS (DDC)	67.39	54.12	42.27	36.79
ES [17] (DDC)	67.10	53.12	41.65	36.03
LCS [17] (DDC)	66.16	53.10	42.00	35.75

sampling methods fail to retain classification accuracies with respect to NS. This shows that a very high data sampling rate at the IoT edge device can lead to a very fast training time but at the expense of severe model performance degradation.

On the contrary, when our DDC algorithm is extended to RS, ES [17], and LCS [17] data sampling methods, the training time of the classifier on the cloud is nearly the same because the number of samples transmitted to the cloud is also very similar for each of these methods as evident from Fig. 4a, Fig. 4b, Fig. 4c, and Fig. 4d. Table 3 and Table 4 show that the accuracy difference between NS and DDC is within 3% at every incremental training round with the exception of LCS [17] (DDC). When using LCS [17] (DDC), at certain incremental training rounds, the classification accuracies are less than 3%. LCS [17] uses SoftMax probabilities for data sampling, however, the results show that SoftMax probabilities of novel samples cannot be used as a basis for data sampling as it leads to a larger amount of catastrophic forgetting with respect to NS as compared to RS (DDC) and ES [17] (DDC).

For the improved juicer algorithm that we propose, we can again observe from Fig. 4e, Fig. 4f, Fig. 4g, and Fig. 4h that our juicer algorithm is able to find the same number of useful weights as FitCNN [16] juicer algorithm but using a lot fewer iterations. Our algorithm is able to reduce the number of iterations needed to find the useful weights of the trained classifier on the cloud by up to 71%.

C. OVERALL DISCUSSION

Firstly, because of the random initialization of weights in deep learning models, there is no guarantee that the classification accuracies that are obtained after incremental learning will be the same if the training is repeated despite having no changes in the input dataset and the hyperparameters. This is the reason why the classification accuracies, number of samples discarded at the IoT edge, cloud training time also vary slightly whenever the same experiments are run. To observe objectively, all the experiments are carried out three times.

For data sampling, we can immediately conclude that out of all the data sampling algorithms we evaluated, WRSTS and MTS are not suitable data sampling methods because they lead to a huge amount of catastrophic forgetting when evaluated on the CUB-200 [51] dataset. This shows that these non-parametric tests are not able to detect a statistical significance difference fast enough between the entropies of the samples to be transmitted to the cloud with respect to the entropies of all the samples. This is why a lot of samples per class are discarded at the IoT edge device which affects the incremental learning process. Furthermore, when evaluating the CIFAR-100 [34] dataset, using WRSTS leads to the worst data sampling performance as a lot of samples are transmitted to the cloud as compared to other data sampling methods.

In order to find the best data sampling algorithm out of RS (DDC), ES [17] (DDC), and LCS [17] (DDC), we compute how much the classification accuracies obtained with DDC deviate from the classification accuracies obtained with NS. At every incremental training round, we compute the standard deviation of the classification accuracies obtained with NS and DDC after which we average all of the standard deviations. Table 5 shows the results.

It can be noted that the majority of the least amount of standard deviations in the classification accuracies are obtained when using RS (DDC). Therefore, we can conclude that RS (DDC) is the best data sampling algorithm. In LCS [17] (DDC), τ^c samples with the highest SoftMax probabilities are discarded, in ES [17] (DDC), τ^c samples with the lowest entropies are discarded and in RS (DDC), τ^c samples are discarded randomly. The reason why RS (DDC) performs better than ES [17] (DDC) and LCS [17] (DDC) is that after every incremental training round, when new neurons are added to the SoftMax layer in our classifier on the cloud, the weights associated with these new neurons are randomly initialized. Furthermore, due to the stochastic nature of neural networks, there is no guarantee that the initial entropies of all samples

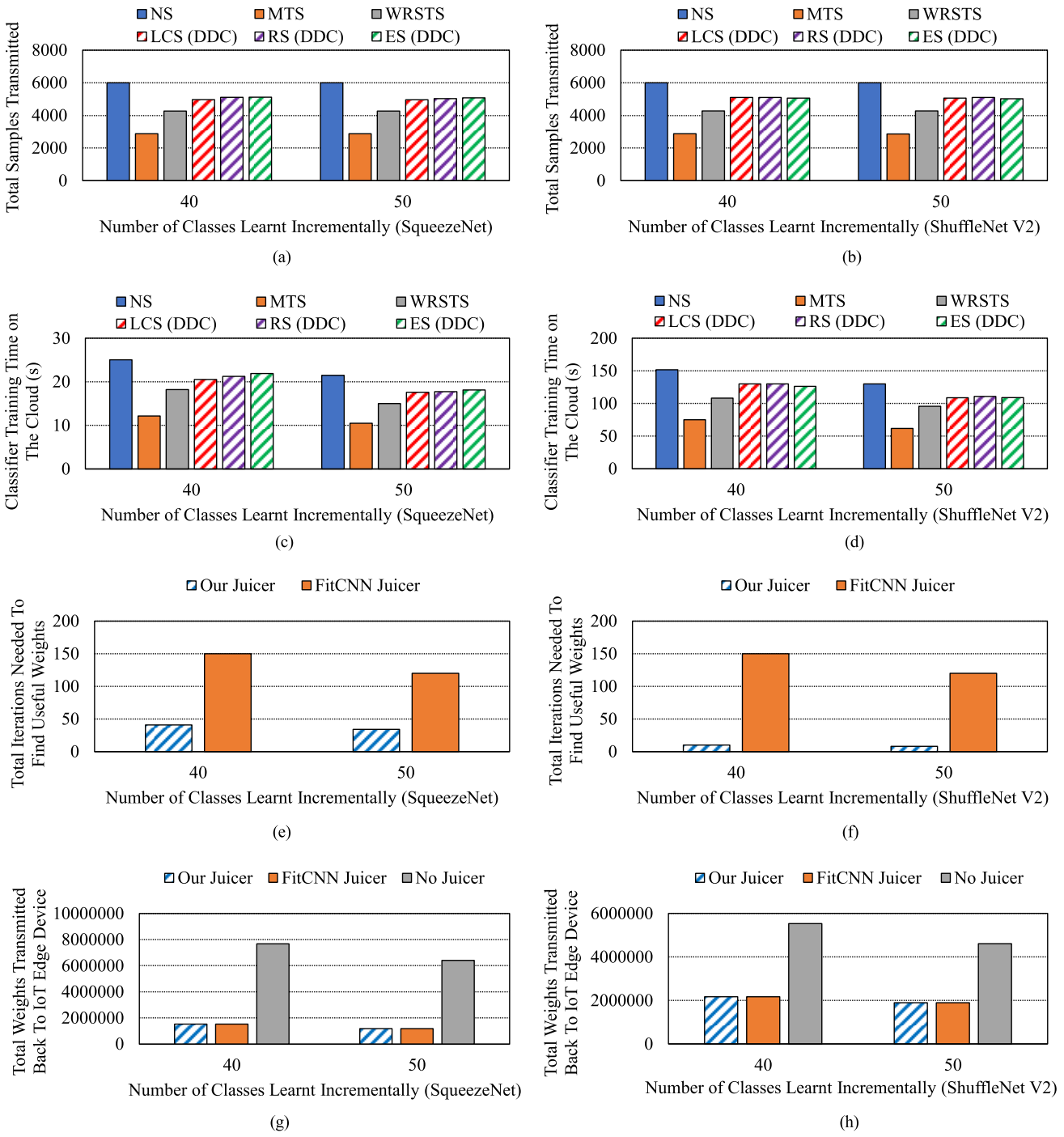

FIGURE 4. Total samples transmitted to the cloud (a) using SqueezeNet [50] (b) using ShuffleNet V2 [52]. The cloud training time (c) using features from SqueezeNet [50] (d) using features from ShuffleNet V2 [52]. Total iterations needed to find useful parameters of the classifier (e) using classifier associated with SqueezeNet [50] (f) using classifier associated with ShuffleNet V2 [52]. Total useful parameters received by the IoT edge device from (g) using classifier associated with SqueezeNet [50] (h) using classifier associated with ShuffleNet V2 [52]. Evaluation of CUB-200 [51] dataset on SqueezeNet [50] and ShuffleNet V2 [52] using the following data sampling techniques: NS, RS (DDC), ES [17] (DDC), LCS [17] (DDC), MTS, WRSTS.

will reduce in magnitude after training. A number of samples can end up with higher entropy values as compared to their respective initial loss values thus affecting the performance of the model as evident from Table 5. The same concept applies to SoftMax probabilities thus the low accuracies which lead to higher standard deviations from NS as evident from Table 5. In order to prevent such samples to affect the model training on the cloud, novel samples must be discarded randomly because this reduces the probability of letting such samples affect the incremental training process.

TABLE 5. Standard deviation of the classification accuracies obtained via data sampling (DDC) and without data sampling.

Dataset	Model	Classes Learnt at a time	RS (DDC) (%)	ES [17] (DDC) (%)	LCS [17] (DDC) (%)
CIFAR-100 [34]	SqueezeNet [50]	10	0.54	0.90	1.24
	ShuffleNet V2 [52]	10	0.42	0.98	1.01
	SqueezeNet [50]	20	0.40	0.90	1.76
	ShuffleNet V2 [52]	20	0.49	1.09	1.13
CUB-200 [51]	SqueezeNet [50]	40	0.73	0.71	0.73
	ShuffleNet V2 [52]	40	1.15	1.70	1.84
	SqueezeNet [50]	50	1.25	1.88	2.18
	ShuffleNet V2 [52]	50	1.41	1.88	2.03

TABLE 6. Effectiveness of RS (DDC) in maintaining classification accuracies while reducing data sampling rate and training time on the cloud.

Dataset	Model	Classes learnt at a time	Standard deviation of classification accuracies (%)	Reduction in the samples transmitted (%)	Reduction in training time on the cloud (%)
CIFAR-100 [34]	SqueezeNet [50]	10	0.54	15.83	13.52
	ShuffleNet V2 [52]	10	0.42	18.42	17.33
	SqueezeNet [50]	20	0.40	14.53	10.76
	ShuffleNet V2 [52]	20	0.49	18.10	19.21
CUB-200 [51]	SqueezeNet [50]	40	0.73	14.83	11.36
	ShuffleNet V2 [52]	40	1.15	14.59	12.42
	SqueezeNet [50]	50	1.25	14.72	14.22
	ShuffleNet V2 [52]	50	1.41	14.79	14.83

The central part of our DDC algorithm is the formulation shown in (3). If we observe and analyze (3) more closely, we are trying to find the maximum entropy value (L^c_{cutoff}) in L^c such that all the entropies smaller than L^c_{cutoff} differ from the median of L^c by a magnitude that is even smaller than the general deviation of all entropies in L^c from the median of L^c. This theoretically means that the total number of samples with entropy values that satisfy (3) are samples with entropies that hardly differ from the median of L^c and such samples do not largely help neural networks generalize the data distribution they belong to [37,38]. The results obtained in Table 5 and Table 6 support our theory.

Even though we claim RS (DDC) is the best data sampling algorithm out of all the data sampling algorithms we used, we must still test the effectiveness of RS (DDC). An effective data sampling algorithm is able to retain a model's learning performance while improving the model training time and reducing the number of samples being transmitted to the cloud. Table 6 shows the standard deviation of the classification accuracies obtained with RS (DDC) from NS, the reduction in the number of samples transmitted to the cloud, and the reduction in the training time on the cloud.

It can be seen that when evaluating CIFAR-100 [34], the standard deviation of the classification accuracies after applying DDC is within 1%, the reduction in the number of samples being transmitted to the cloud is greater than 14% and the reduction in the classifier training time on the cloud is greater than 10%. When evaluating CUB-200 [51], the standard deviation of the classification accuracies after applying DDC is within 1.5%, the reduction in the number of samples being transmitted to the cloud is greater than 14% and the reduction in the classifier training time on the cloud is greater than 11%. This proves that after applying RS (DDC), the changes in the classification accuracies are very small as compared to NS whereas the reduction in the classifier training time on the cloud and the transmission cost is huge.

The objective of the juicer algorithm is to send only the most useful weights of the trained model on the cloud back to the IoT edge device. The juicer algorithm requires only one hyperparameter and that is the acceptable accuracy loss($\text{Accept}_{\text{loss}}$). The goal is to minimize the number of useful weights of a trained model on the cloud that must be transmitted back to the IoT edge device. However, we must also ensure that the training accuracy of the model on the cloud is not affected due to weight extraction which is why $\text{Accept}_{\text{loss}}$ is a required hyperparameter. $\text{Accept}_{\text{loss}}$ is set to 0.25 in the FitCNN [16] juicer algorithm and also in our juicer algorithm (Algorithm 1). As mentioned earlier, the number of useful weights that our improved juicer algorithm is able to extract is the same as the FitCNN [16] juicer algorithm. However, we reduce the number of iterations needed to find the useful weights of the classifier after every incremental

training round. Our improved juicer algorithm can loosely be compared to the early stopping strategy used in neural networks. In early stopping, model training is stopped as soon as the validation error starts increasing, similarly, our improved juicer algorithm will stop the iterative process of using different threshold values to find the useful weights as soon as the accuracy of the temporary model falls outside the acceptable accuracy limit. Hence, our approach uses much fewer iterations to find the useful weights of the trained model on the cloud.

Results show that our weight extractor algorithm greatly speeds up the process of finding the useful parameters of a trained model as compared to the weight extractor algorithm presented in [16]. We achieve this speedup by modifying the original FitCNN [16] algorithm by sorting the weight differences between the trained model and the temporary model in ascending order and using only positive weight differences as thresholds for determining the useful parameters.

VI. CONCLUSION

In this paper, we present two main contributions. Firstly, by performing data sampling on an IoT edge device in a class incremental learning scenario and secondly, by efficiently finding specific useful weights in a trained model on the cloud to be sent back to the IoT edge device. Both the contributions reduce the communication costs between the IoT edge device and the cloud.

After an extensive set of experiments, we show that our proposed DDC algorithm is able to perform data sampling at the IoT edge device and is able to retain the learning performance whereby the classification accuracy that we obtain at every incremental training round is within 3% compared to the baseline method (no data sampling) at every incremental training round irrespective of the dataset, CNN, fully connected layers based classifier, hyperparameters, and the number of classes being learnt incrementally. From the results obtained, we conclude that applying our DDC algorithm to RS is the most consistent method compared to all the data sampling algorithms we used for our experiments. RS (DDC) always reduces the transmission cost from the IoT edge device to the cloud and leads to a faster training time on the cloud while maintaining the class incremental learning performance.

We also propose an algorithm for extracting only the useful weights of a trained model to be sent back to the IoT edge device in an effort to reduce the transmission cost. Our weight extraction algorithm is able to extract the same number of weights as FitCNN [16] but our work improves the efficiency of the original juicer algorithm [16] and manages to reduce the workload on the cloud in terms of the total iterations needed for finding the useful weights by at least 75% when evaluating CIFAR-100 [34] on both SqueezeNet [50] and ShuffleNet V2 [52] and by at least 71% when evaluating CUB-200 [51] on both SqueezeNet [50] and ShuffleNet V2 [52].

VII. FUTURE WORK

There are a number of advancements that can be made as an extension to this work such as having multiple IoT devices in which case the cloud needs to sync the overall learning mechanism across several devices. Coming up with a novel cost function that can incrementally place importance on certain classes with respect to the imbalance can be another useful idea because this will eliminate the need for oversampling of underrepresented classes. Another main advancement that can be made is to design an algorithm for IoT edge device such that they can automatically assign labels for the new incoming classes.

We would like to stress that the number of samples we discard at the IoT edge device is not the optimal number of samples that can be discarded before training, we believe that to reap the full benefits of data sampling, a certain number of samples can be discarded at the IoT edge device in an effort to reduce both the transmission cost and the cloud training cost. However, certain samples out of the transmitted samples must be sent and discarded on the cloud during training.

Although our DDC algorithm is able to largely retain incremental learning classification accuracies while greatly reducing data transmission costs and training time on the cloud, we understand that some researchers would prefer control over the trade-off between the classification accuracies and the data transmission rate depending on their applications. In order to do so, our DDC algorithm can easily be re-used by multiplying the term in (4) with a hyperparameter ω as shown in (11) below.

$$\tau^c = \left\lfloor \omega \cdot \left(\sum\nolimits_{l \in V^c} 1 \right) \right\rfloor \tag{11}$$

The hyperparameter ω should be in the range 0 and 1. The term $\lfloor \cdot \rfloor$ is known as the floor function which is used to round-off a floating-point number to the greatest integer less than or equal to the floating-point number, the floor function is required because ω can represent a floating-point number. When ω is 0, no data sampling takes place, however, when the value of ω starts to increase, the magnitude of data sampling also increases with a small deviation in the classification accuracies starting to become evident. Therefore, researchers can tweak the value of ω to control the trade-off between the data sampling rate and the deviation in the incremental learning classification accuracies as per their applications.

A shortcoming of our weight extraction algorithm is that this algorithm must be run on very powerful hardware accelerators. Our weight extraction algorithm computes the training accuracy of the classifier a few times to come up with the most useful parameters. This means multiple forward passes on the classifier are required suggesting that if the hardware accelerator is not powerful enough, this process itself could be very time-consuming. This shortcoming leads to an open research question and that is how to quickly determine the useful weights of a neural network-based classifier after training or perhaps during the training process itself.

REFERENCES

[1] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Inf. Fusion*, vol. 58, pp. 52–68, Jun. 2020, doi: 10.1016/j.inffus.2019.12.004.

[2] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychol. Learn. Motivat.*, vol. 24, pp. 109–165, Jan. 1989, doi: 10.1016/S0079-7421(08)60536-8.

[3] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends Cogn. Sci.*, vol. 3, pp. 128–135, May 1999, doi: 10.1016/S1364-6613(99)01294-2.

[4] W. Fang, X. Yin, Y. An, N. Xiong, Q. Guo, and J. Li, "Optimal scheduling for data transmission between mobile devices and cloud," *Inf. Sci.*, vol. 301, pp. 169–180, Apr. 2015, doi: 10.1016/j.ins.2014.12.059.

[5] W. Fang, Y. Li, H. Zhang, N. Xiong, J. Lai, and A. V. Vasilakos, "On the throughput-energy tradeoff for data transmission between cloud and mobile devices," *Inf. Sci.*, vol. 283, pp. 79–93, Nov. 2014, doi: 10.1016/j.ins.2014.06.022.

[6] E. Baccarelli, S. Scardapane, M. Scarpiniti, A. Momenzadeh, and A. Uncini, "Optimized training and scalable implementation of conditional deep neural networks with early exits for fog-supported IoT applications," *Inf. Sci.*, vol. 521, pp. 107–143, Jun. 2020, doi: 10.1016/j.ins.2020.02.041.

[7] D. Mrozek, "Fall detection in older adults with mobile IoT devices and machine learning in the cloud and on the edge," *Inf. Sci.*, vol. 530, pp. 148–163, Oct. 2020, doi: 10.1016/j.ins.2020.05.070.

[8] Y. Zhang, H. Guo, Z. Lu, L. Zhan, and P. C. K. Hung, "Distributed gas concentration prediction with intelligent edge devices in coal mine," *Eng. Appl. Artif. Intell.*, vol. 92, Jun. 2020, Art. no. 103643, doi: 10.1016/j.engappai.2020.103643.

[9] F. F. X. Vasconcelos, R. M. Sarmento, P. P. R. Filho, and V. H. C. de Albuquerque, "Artificial intelligence techniques empowered edge-cloud architecture for brain CT image analysis," *Eng. Appl. Artif. Intell.*, vol. 91, May 2020, Art. no. 103585, doi: 10.1016/j.engappai.2020.103585.

[10] W. Xiong, Z. Lu, B. Li, Z. Wu, B. Hang, J. Wu, and X. Xuan, "A self-adaptive approach to service deployment under mobile edge computing for autonomous driving," *Eng. Appl. Artif. Intell.*, vol. 81, pp. 397–407, May 2019, doi: 10.1016/j.engappai.2019.03.006.

[11] S.-C. Huang, J.-N. Hwang, S.-Y. Kuo, A. P. D. Binotto, D. Upadhyay, and P. C. K. Hung, "Special issue on Internet of Things (IoT) for in-vehicle systems," *Eng. Appl. Artif. Intell.*, vol. 85, pp. 874–875, Oct. 2019, doi: 10.1016/j.engappai.2019.103235.

[12] G. S. Fischer, R. D. R. Righi, G. D. O. Ramos, C. A. D. Costa, and J. J. P. C. Rodrigues, "ElHealth: Using Internet of Things and data prediction for elastic management of human resources in smart hospitals," *Eng. Appl. Artif. Intell.*, vol. 87, Jan. 2020, Art. no. 103285, doi: 10.1016/j.engappai.2019.103285.

[13] L. Kang, R.-S. Chen, W. Cao, Y.-C. Chen, and Y.-X. Hu, "Mechanism analysis of non-inertial particle swarm optimization for Internet of Things in edge computing," *Eng. Appl. Artif. Intell.*, vol. 94, Sep. 2020, Art. no. 103803, doi: 10.1016/j.engappai.2020.103803.

[14] R. Kemker and C. Kanan, "FearNet: Brain-inspired model for incremental learning," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–16.

[15] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "ICaRL: Incremental classifier and representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5533–5542, doi: 10.1109/CVPR.2017.587.

[16] D. Liu, C. Yang, S. Li, X. Chen, J. Ren, R. Liu, M. Duan, Y. Tan, and L. Liang, "FitCNN: A cloud-assisted and low-cost framework for updating CNNs on IoT devices," *Future Gener. Comput. Syst.*, vol. 91, pp. 277–289, Feb. 2019, doi: 10.1016/j.future.2018.09.020.

[17] B. Settles, "Active learning literature survey," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep., 2009.

[18] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, L. Tang, "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers," *ACM SIGARCH Comput. Archit. News.*, vol. 43, no. 3, pp. 27–40, 2016, doi: 10.1145/2872887.2749472.

[19] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, doi: 10.1109/JPROC.2019.2921977.

[20] F. Lu, L. Gu, L. T. Yang, L. Shao, and H. Jin, "Mildip: An energy efficient code offloading framework in mobile cloudlets," *Inf. Sci.*, vol. 513, pp. 84–97, Mar. 2020, doi: 10.1016/j.ins.2019.10.008.

[21] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, and Y. Pan, "Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment," *Inf. Sci.*, vol. 537, pp. 116–131, Oct. 2020, doi: 10.1016/j.ins.2020.05.057.

[22] X. Xu, X. Liu, X. Yin, S. Wang, Q. Qi, and L. Qi, "Privacy-aware offloading for training tasks of generative adversarial network in edge computing," *Inf. Sci.*, vol. 532, pp. 1–15, Sep. 2020, doi: 10.1016/j.ins.2020.04.026.

[23] P. Zhang, A. Zhang, and G. Xu, "Optimized task distribution based on task requirements and time delay in edge computing environments," *Eng. Appl. Artif. Intell.*, vol. 94, Sep. 2020, Art. no. 103774, doi: 10.1016/j.engappai.2020.103774.

[24] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for Internet of Things data analysis: A survey," *Digit. Commun. Netw.*, vol. 4, no. 3, pp. 161–175, Aug. 2018, doi: 10.1016/j.dcan.2017.10.002.

[25] X. Wang, Y. Feng, Z. Ning, X. Hu, X. Kong, B. Hu, and Y. Guo, "A collective filtering based content transmission scheme in edge of vehicles," *Inf. Sci.*, vol. 506, pp. 161–173, Jan. 2020, doi: 10.1016/j.ins.2019.07.083.

[26] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, Z. Zhang, and Y. Fu, "Incremental classifier learning with generative adversarial networks," 2018, *arXiv:1802.00853*. [Online]. Available: http://arxiv.org/abs/1802.00853

[27] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan, "REMIND your neural network to prevent catastrophic forgetting," 2019, *arXiv:1910.02509*. [Online]. Available: http://arxiv.org/abs/1910.02509

[28] E. Choi, K. Lee, and K. Choi, "Autoencoder-based incremental class learning without retraining on old data," 2019, *arXiv:1907.07872*. [Online]. Available: http://arxiv.org/abs/1907.07872

[29] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, vol. 8, 2017, pp. 6072–6082.

[30] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proc. Eur. Conf. Comput. Vis.*, in Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11207. Cham, Switzerland: Springer, 2018, pp. 144–161, doi: 10.1007/978-3-030-01219-9_9.

[31] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, and T. Van Overveldt, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*. [Online]. Available: http://arxiv.org/abs/1902.01046

[32] X. Zhang, X. Zhu, J. Wang, H. Yan, H. Chen, and W. Bao, "Federated learning with adaptive communication compression under dynamic bandwidth and unreliable networks," *Inf. Sci.*, vol. 540, pp. 242–262, Nov. 2020, doi: 10.1016/j.ins.2020.05.137.

[33] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtepe, H. Kim, and S. Nepal, "End-to-end evaluation of federated learning and split learning for Internet of Things," 2020, *arXiv:2003.13376*. [Online]. Available: http://arxiv.org/abs/2003.13376

[34] A. Krizhevsky. (2009). *Learning Multiple Layers of Features From Tiny Images*. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[36] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio, "Variance reduction in SGD by distributed importance sampling," 2015, *arXiv:1511.06481*. [Online]. Available: http://arxiv.org/abs/1511.06481

[37] A. H. Jiang, D. L.-K. Wong, G. Zhou, D. G. Andersen, J. Dean, G. R. Ganger, G. Joshi, M. Kaminksy, M. Kozuch, Z. C. Lipton, and P. Pillai, "Accelerating deep learning by focusing on the biggest losers," 2019, *arXiv:1910.00762*. [Online]. Available: http://arxiv.org/abs/1910.00762

[38] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 6, 2018, pp. 3936–3949.

[39] V. Birodkar, H. Mobahi, and S. Bengio, "Semantic redundancies in image-classification datasets: The 10% you don't need," 2019, *arXiv:1901.11409*. [Online]. Available:https://arxiv.org/abs/1901.11409

[40] A. R. de Mello, M. R. Stemmer, and F. G. O. Barbosa, "Support vector candidates selection via delaunay graph and convex-hull for large and high-dimensional datasets," *Pattern Recognit. Lett.*, vol. 116, pp. 43–49, Dec. 2018, doi: 10.1016/j.patrec.2018.09.001.
[41] M. Kawulok and J. Nalepa, "Support vector machines training data selection using a genetic algorithm," in *Structural, Syntactic, and Statistical Pattern Recognition* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7626. Berlin, Germany: Springer, 2012, pp. 557–565, doi: 10.1007/978-3-642-34166-3_61.
[42] X.-J. Shen, L. Mu, Z. Li, H.-X. Wu, J.-P. Gou, and X. Chen, "Large-scale support vector machine classification with redundant data reduction," *Neurocomputing*, vol. 172, pp. 189–197, Jan. 2016, doi: 10.1016/j.neucom.2014.10.102.
[43] D. Wang, H. Qiao, B. Zhang, and M. Wang, "Online support vector machine based on convex hull vertices selection," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 4, pp. 593–609, Apr. 2013, doi: 10.1109/TNNLS.2013.2238556.
[44] A. López Chau, X. Li, and W. Yu, "Convex and concave hulls for classification with support vector machine," *Neurocomputing*, vol. 122, pp. 198–209, Dec. 2013, doi: 10.1016/j.neucom.2013.05.040.
[45] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, "An energy efficient IoT data compression approach for edge machine learning," *Future Gener. Comput. Syst.*, vol. 96, pp. 168–175, Jul. 2019, doi: 10.1016/j.future.2019.02.005.
[46] F. Xhafa, B. Kilic, and P. Krause, "Evaluation of IoT stream processing at edge computing layer for semantic data enrichment," *Future Gener. Comput. Syst.*, vol. 105, pp. 730–736, Apr. 2020, doi: 10.1016/j.future.2019.12.031.
[47] Y. Li, A.-C. Orgerie, I. Rodero, B. L. Amersho, M. Parashar, and J.-M. Menaud, "End-to-end energy models for edge cloud-based IoT platforms: Application to data stream analysis in IoT," *Future Gener. Comput. Syst.*, vol. 87, pp. 667–678, Oct. 2018, doi: 10.1016/j.future.2017.12.048.
[48] L. Fei-Fei, J. Deng, and K. Li, "ImageNet: Constructing a large-scale image database," *J. Vis.*, vol. 9, no. 8, p. 1037, 2010, doi: 10.1167/9.8.1037.
[49] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.
[50] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: http://arxiv.org/abs/1602.07360
[51] P. Welinder, S. Branson, T. Mita, C. Wah, and F. Schroff, "Caltech-UCSD birds 200," Comput. Neural Syst., California Inst. Technol., Pasadena, CA, USA, Tech. Rep. CNS-TR, 2010, pp. 1–15. [Online]. Available: http://www.vision.caltech.edu/visipedia/CUB-200-2011.html
[52] N. Ma, X. Zhang, H. T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis.* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11218. Cham, Switzerland: Springer, 2018, pp. 122–138, doi: 10.1007/978-3-030-01264-9_8.
[53] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.
[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison, "PyTorch: An imperative style, high-performance deep learning library," 2019, *arXiv:1912.01703*. [Online]. Available: http://arxiv.org/abs/1912.01703

SWARAJ DUBE received the M.Eng. degree from the Department of Electrical and Electronic Engineering, University of Nottingham Malaysia, in 2017, where he is currently pursuing the Ph.D. degree. He worked as a Research and Development Engineer with ViTrox Corporation Berhad, from 2017 to 2018. He is also a registered Graduate Engineer with the Board of Engineers Malaysia. His research interests include deep learning, edge computing, and the Internet of Things.

WONG YEE WAN received the Ph.D. degree in electrical and electronic engineering from the University of Nottingham Malaysia, in 2011. She worked as an Assistant Professor with the University of Nottingham Malaysia, from 2011 to 2019, and promoted to an Associate Professor, in 2020. She is currently working as a Senior Data Scientist with the industry. Her research interest includes applied artificial intelligence in various domains.

HERMAWAN NUGROHO (Senior Member, IEEE) received the bachelor's degree from the Bandung Institute of Technology, Indonesia, in 2005, the M.Sc. and Ph.D. degrees from the Universiti Teknologi PETRONAS (UTP), Malaysia, in 2007 and 2009, respectively, and the Ph.D. degree from Indonesia, in 2014. He worked as a Lighting Consultant, before continuing his master's degree. After finishing his M.Sc. degree, he worked as a Research Officer for several research projects under UTP and ViTrox Technologies. He currently works with the University of Nottingham Malaysia. He manages several research projects with the Centre for Intelligent Signal and Imaging Research (CISIR), UTP. He received his Professional Engineer status from Indonesia, in 2015.