Edge Machine Learning for Resource-constrained IoT Devices

Haocheng Fang
Carnegie Mellon University
hfang@andrew.cmu.edu

Yichen Ruan
Carnegie Mellon University
yichenr@andrew.cmu.edu

Jinhang Zuo
Carnegie Mellon University
jzuo@andrew.cmu.edu

Abstract

Traditionally, deploying machine learning models, especially deep learning models, requires a significant amount of computing power. However, it is not always viable to use such powerful computing devices. For example, it is very expensive to rent a server and people may have concerns over security when sending sensible information across the net. In our project, we propose an edge computing solution to this problem, which fully utilizes the IoT devices around us to deploy machine learning tasks without sending data to the powerful server. We propose a tree-based hierarchical classification model and design a pruning algorithm for these resource-constrained IoT devices. We implement our algorithm on 8 Raspberry Pis for image classification. Results show that compared to using a complicated model to classify all the data, it achieves faster image classification through parallelization.

1 Introduction

Edge computing, also known as fog computing, aims to fully utilize the computing power of the computing devices on the edge of the network. In the context of the Industrial Internet of Things (IIoT), these "edge devices" are usually the computing infrastructures or Internet of Things (IoT) devices that exists close to the source of data. Compared to traditional cloud services, edge computing promises light weight solutions for budget-constrained and latency-sensitive computing tasks.

The deployment of machine learning algorithms on the edge computing architecture is appealing and has great practical value. For applications like real-time anomaly detection and offline identity verification, edge machine learning avoids the latency of data transmission, and often offers strong privacy protection as well as security guarantee. However, the main challenge of deploying traditional machine learning tasks on the edge computing architecture is that the IoT devices are usually resource-constrained and have far less computing power compared to traditional servers.

A straightforward idea to deploy machine learning on edge devices is using distributed machine learning algorithms. However, these existing methods typically fail to work due to the resource-constrained nature of edge IoT devices. Thusly, there are emerging demands for novel models and algorithms. In this project, we propose a tree-based classification model for multiple resource-constrained IoT devices. It utilizes prior knowledge of the dataset (e.g., hierarchy in Figure 1) to build a large classification tree with identical hierarchy to the prior knowledge and then prunes the tree node by node while minimizing accuracy loss. Pruning stops when the number of classifier in the classification tree is equal to the number of available IoT devices and we reach an optimal state where the final model can fit on the limited number of devices while having minimal accuracy loss.

There are two main advantages of this classification model. The first one is that we have very low requirements on the computing power of each node. We can combine numerous resource-constrained devices to obtain a classification model with high accuracy that is otherwise not possible on such devices. The model can also scale up and down whenever the structure of the nodes changes while maintaining maximum accuracy. The second advantage is that in our tree-based classification model,

each node only propagates the data to the lower branch with the highest probability. Therefore, the utilization of each node is lower on the bottom layers and this allows for high levels of parallelization and increased throughput when working on large batch of images.

For our experiments, we use Neural Networks(NNs) to solve image classification tasks. Therefore, instead of running a huge network that classifies the entire dataset, we can run multiple small networks that distinguish specific elements. For example, if we are to classify between thousands of species of animals, we can first use prior knowledge about the hierarchy of the animals to create nodes that classify between different species and then create different nodes that classify between various breeds under a specific species. The entire framework will look like a tree with each node being a small NN running on an IoT device. We selected Raspberry Pis as the edge computing nodes since they have similar computing power to traditional IoT devices and are easy to work and test on. For the dataset, we chose to work on the CIFAR-100 dataset for the image classification task since it has an inherent structure and the classification task is very close to real life situations. The main challenge of the project is to create an optimal model that best splits the data while being able to fit the NNs on each node.

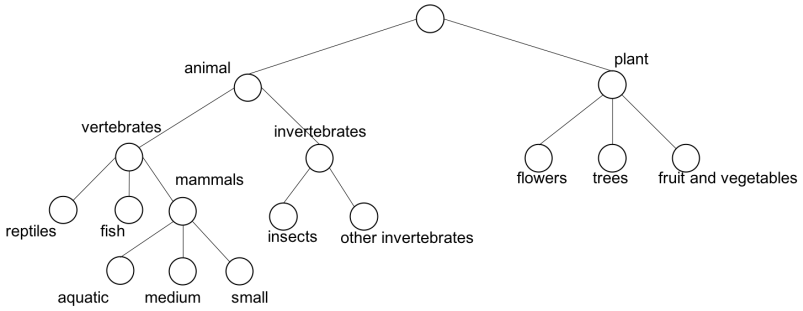

Figure 1: Hierarchy of CIFAR-100 dataset

2 Related Work

While machine learning becomes essential for lots of IoT applications, most of IoT devices make predictions via sending their measurement data to centralized clouds [1, 2]. Compared to centralized prediction, having edge devices make predictions locally avoids the latency of data transmission, and offers strong privacy protection [4]. One way to deploy machine learning onto multiple edge devices is distributed machine learning [5 - 8]. However, existing distributed machine learning, which aims to enable large-scale ML task, does not consider memory and computing constraints on the edge devices, thus cannot be used directly by edge devices.

There are many adaptive systems considering resource-constrained prediction, which ensure overall accuracy under budget constraints [9, 10]. Nan & Saligrama [11] propose an adaptive approximation approach for test-time resource-constrained prediction motivated by IoT. In [12], a random forest is pruned to optimize expected feature cost and accuracy for resource-constrained prediction.

Another set of related work designs resource-efficient small models which can run on IoT devices. Kumar et al. [4] develop a novel tree-based algorithm for efficient prediction on IoT devices. Gupta et al. [13] proposes ProtoNN, a novel kNN based algorithm for tiny resource-constrained devices to solve supervised learning problems. The Sparse Multiprototype Linear Learner (SMaLL) algorithm in [14] is used for training small resource-constrained predictors.

3 Method

The basic idea of our method is to group classes into some superclasses. And organize those superclasses as a hierarchical tree-structure. Then we assign one edge device to each vertex of the tree. The expectation is to reduce the complexity of models on each node, and increase the prediction accuracy. What is more, instead of building a classification model over the whole dataset, each individual edge node now works on a single group that contains only a subset of the total data. In that way, we can reduce the storage and communication overhead of the whole network.

For example, suppose we want to classify human images into 4 categories: women, girls, men and boys. And assume we only have 3 edge devices available. We can create a “females" superclass by grouping women and girls. Similarly, we group men and boys as a “males" superclass. Then, we can assign a edge device to determine if the image is female or male; assign one device to classify women and girls out of females; and use the last device for men and boys. Upon a request, the first node that works as the root of the tree will predict if the observation is female or male. Once it gets the result, it will forward the request to nodes in the next level. Eventually, the leave nodes will yield the final result. It is worthnoting that the way classes are grouped is generally not unique. For example, we can also merge women and men as an “adult" superclass; merge girls and boys as an “children" superclass. Other combination is also possible if that yields good classifiers.

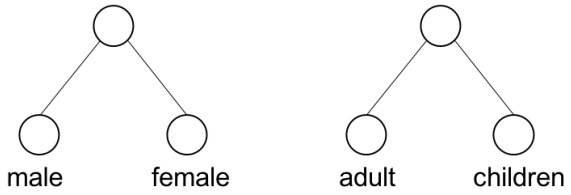

Figure 2: Two possible topology for 4 classes: women, girls, men and boys

As is discussed above, the topology of the groups can be organized as a tree (call it a t-Tree). Each edge node is associated with a t-Tree node, which trains a machine learning model to classify its children. On prediction, a query starts from the root, and goes all the way to the leaves. The prediction result is the class reported by the last t-Tree node on the path.

Algorithm 1 illustrates the process of prediction an observation. The complexity of the algorithm is bounded by the height of the t-Tree, which is roughly $O(\log(L))$, where L is the total number of classes. Though the prediction accuracy of each node is increased because of fewer classes, the error is propagating in a tree structure. For the purpose of reducing error and pipelining queries, we want the upper level nodes to have higher accuracy and shorter runtime. Moreover, we want the tree to be shorter such that the error does not accumulate alongside a long path.

Algorithm 1: Prediction Algorithm

```
function Predict (T, X);
Input  : A t-Tree node T; A sample X
Output : A class label
if T is leave then
    return T.classifier.predict(X)
else
    id = T.classifier.predict(X)
    T' = T.children(id)
    return Predict(T', X)
end
```

Generally, if we have k edge devices and L classes, our purpose is to find the optimal tree structure that maximizes the average prediction score. The average prediction score of a sub-tree rooted at T is defined recursively as follows. We define the score of whole system as the score of the root node.

$$\text{Score}(T) = \begin{cases} T.\text{accuracy, if } T \text{ is leave} \\ T.\text{accuracy} \times \frac{1}{|T|} \sum_{C \in T.\text{children}} |C| \text{Score}(C), \text{otherwise} \end{cases} \tag{1}$$

where T.accuracy is the training accuracy of the classifier inside T, which could be accuracy on the training data or that on the hold-out data. $|T|$ is the number of samples of node T, which is the sum of that of its children. The definition is simply the accuracy of the root, multiplied by the number-of-samples-weighted average accuracy of its children. An possible alternative is instead of the average performance, we define the score as per the worst case.

$$\text{Score}(T) = \min_{\Gamma} \prod_{C \in \Gamma} C.\text{accuracy} \tag{2}$$

where Γ is a path from T to some leave node. We use the first definition in this project because we are more focusing on the average performance of the system. The second definition can be a good reference if the requests are highly skewed or one has strict requirements on the minimal accuracy guarantee.

Ideally, we want to find a t-Tree structure that maximizes the average accuracy. The problem is however very hard considering the possibilities of tree topology. Without a constraint on the tree structure, the number of combinations can be exponential with respect to number of classes L. Our idea is thus to limit the topology a t-Tree can take, thus pruning the searching space. We will discuss two pruning algorithms in Section 3.1 and 3.2.

We generally do not set limitations for the types and complexities of individual classification models. It is possible to use totally different models for different nodes. In the case the same model is used across the tree, one can still set different values for the tuning variables of different nodes. However, for efficiency consideration, it is wise to constraint the choice to a small set of models.

3.1 A baseline algorithm

A naive idea is to constrain the topology as a 2-layer tree. Algorithm 2 yields a 2-layer t-Tree with precisely k nodes. The initialization needs to train $\binom{L}{2}$ models, which takes $O(L^2)$ time. For each iteration in the while loop, the algorithm compares all pairs of leaves, which again takes $O(L^2)$ time. Upon merging a pair of nodes, we need to train two classification models, one for the merged node, one for the parent (i.e. the root node). Other nodes' accuracy are unchanged. There are $L/2 - k$ iterations, thus the complexity is $O(L^3)$ times the runtime for training, which could also be quite huge.

Although the algorithm is extremely slow, it doesn't rely on any prior knowledge of the categories. The two-layer structure means there are precisely two hops before the final prediction result is generated. Thus, the error does not accumulate that much as in a deep-tree. Also, the runtime of the system is more predictable, which is an important property for a stable and reliable system.

Algorithm 2: Baseline Algorithm

function Build-Tree-Baseline $(k, data, class)$;
Input : Number of edge nodes k; n observations $data$; L class labels $class$
Output : A t-Tree T
For each pair of classes $(c1, c2)$ in $class$, create a node that classifies $c1$ and $c2$ over $data[\text{label} = c1] + data[\text{label} = c2]$, calculate the training accuracy.
Find the $L/2$ groups that cover all the classes and have the highest accuracy, create a leave node for each of them. Add them to a set S.
Create a root node T that points to all nodes in S. Train T over the $L/2$ super-classes.
Creates an empty hash table H.
while *len(S)* $> k - 1$ **do**
    For each pair (l_1, l_2) in S, if it is not in H, try merge them, calculate the average score for the new tree structure as in Eq. 1. Put the result (call it merging score) into H using (l_1, l_2) as key.
    Find in H the pair (l_1^*, l_2^*) with the highest merging score.
    Merge (l_1^*, l_2^*) to create and train a new node p on corresponding entries of $data$.
    T.children.remove($[l_1^*, l_2^*]$)
    T.children.add(p)
    Re-train the root T over $data$ and the new super-classes.
    S.remove($[l_1^*, l_2^*]$)
    H.remove((l_1^*, l_2^*))
end
return T;

Fig. 3 shows an example with $k = 3, L = 6$. It takes $\binom{6}{2} = 15$ comparisons to go from state I to II. Three superclasses are generated after the first merging. A root node is created to classify over the 3 groups. To further reduce the number of nodes to $k = 3$, one more step is required. This takes another $\binom{3}{2} = 3$ comparisons. Finally, the "bird+flower" node and the "tree+fruit" node is merged. The resulting tree in state III has precisely 3 nodes.

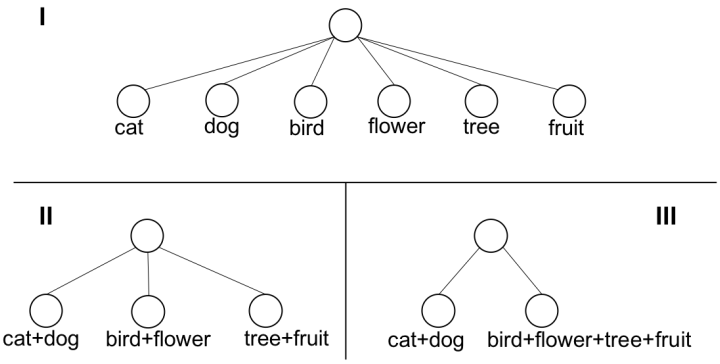

Figure 3: Apply the baseline algorithm to a 6-class example. I: initial layout. II: after initialization. III: after first iteration, also the final structure.

3.2 Taxonomy based algorithm

The baseline algorithm is straightforward yet very inefficient. In this section, we will propose a new algorithm that utilizes taxonomy as prior knowledge.

In fact, for most classification problems, the class labels are highly hierarchical. For example, biology taxonomy classifies species with a 9-layer tree: life, domain, kingdom, phylum, class, order, family, genus and species. So instead of building everything from scratch, we can create a t-Tree by modifying the "taxonomy tree". We argue that this method is reasonable because: 1) Taxonomy is a science that groups together classes with similar properties, which is usually what machine learning algorithms do; 2) There are many existing algorithms specifically designed for some groups in the taxonomy, which can be reused.

Algorithm 3: Taxonomy based Algorithm

function Build-Tree-Taxonomy $(K, data, G)$;
Input : Number of edge nodes K; n observations $data$; Pre-built taxonomy tree G
Output : A tree structure T with at most K nodes.
$T = G$
while $|T| > k$ **do**
  For each new pair (l_1, l_2) with the same parent, try merge them.
  Compute the change of the average score. Store the result into some global table.
  Find in the table the pair (l_1^*, l_2^*) with the highest merging score.
  Merge (l_1^*, l_2^*), insert the new node l_{new} under $p = l_1^*.parent$.
  if *p has only one child* **then**
    Replace p with l_{new}
  end
end
return T

In algorithm 3, a node can only merge with its leave-siblings. In the case when a new node becomes an only child, its parent is replaced by the new node. This is the only situation where the height of the tree decreases. After each iteration, the number of nodes will decrease by at least one. The algorithm will therefore stop after a finite number of iterations. The exact decrease of the computation time is depending on the structure of the taxonomy tree. Generally, the runtime gets smaller as the tree has more branches. Assume the tree is balanced, and suppose the L classes are evenly divided into L/ϵ groups, the runtime for each iteration is at most $O(\frac{L}{\epsilon}\binom{\epsilon}{2})$, which is greatly less than the baseline algorithm situation $O(\binom{L}{2})$.

Fig. 4 applies the taxonomy based algorithm to the same example as in section 3.1. We need to compare $2 \times \binom{3}{2} = 6$ pairs from state I to state II, and another $2 \times \binom{2}{2} = 2$ pairs from state II to state II. The total number of comparisons is 8, which is a huge decrease compared to the 18 comparisons taken by the baseline case.

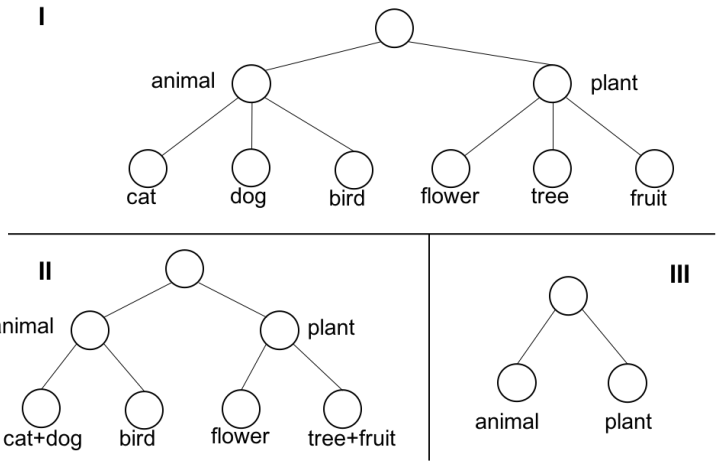

Figure 4: Apply the taxonomy based algorithm to a 6-class example. I: initial layout. II: after first two merge. III. after all level-two nodes are merged, also the final structure.

4 Dataset

CIFAR-100:

It contains 100 classes with 600 32x32 color images each. The 100 classes are then grouped into 20 superclasses. We chose this dataset because there exists a hierarchy of all the classes and can serve as prior knowledge to our model.

We do not need to pre-process the dataset, but we choose images of animals and plants out of the whole dataset for training and testing, due to time and computing resource restrictions. We also need to identify a hierarchical structure different from the existing "superclass-subclass" structure. The inputs are all training and testing images of animals and plants as numpy arrays and the outputs are the classification results of both the class and superclass.

5 Experiment & Result

5.1 Training

We train our models using proposed taxonomy-based algorithm on CIFAR-100 dataset. We set the number of available IoT devices as eight, which is the number of devices we have for testing. At each round, we calculate the accuracy loss when merging two node, then make the merging decision with the lowest accuracy loss. The iteration ends when the number of nodes left in the tree is equal to the number of available devices (e.g., eight in our case). For each node, we use Keras to train a CNN model. We save all these pre-trained models for testing. Each model file is around 18 MB, which is suitable for a IoT device to run on.

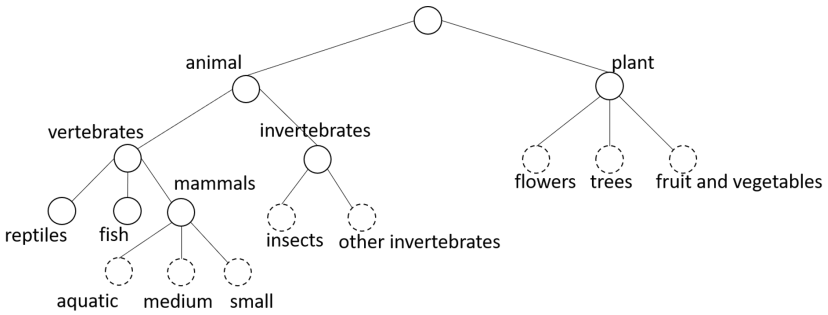

Figure 5: Output taxonomy-based tree

Table 1: Accuracy of each node

Name	# of samples	Accuracy
All (root node)	25000	87%
Animal	17500	76%
Plant	7500	95%
Vertebrates	12500	80%
Invertebrates	5000	91%
Reptiles	2500	82%
Fish	2500	87%
Mammals	7500	94%

The new taxonomy-based tree is shown in Figure 5. Each solid node in the tree represents a classifier. During the execution of the algorithm, the nodes are merged in the following order:

1. insects + other invertebrates -> invertebrates
2. aquatic + small -> new node A
3. trees + fruit&veges -> new node B
4. B + flowers -> plant
5. A + small -> mammals

Here, the nodes on the left hand side of the arrow are the nodes to be merged and the right hand side is the merging result. For example, in the first iteration, we merge insects with other invertebrates; since invertebrates only contains insects and other invertebrates, the merging result is the invertebrate node itself. The training result for the final tree is shown in Table 1.

5.2 Testing

There are 8 Raspberry Pis for testing. Figure 6 shows the deployment of them in the experiment. First, we install Keras and Tensorflow on all our Raspberry Pis. Then, we set up a server. All devices are connected to that server, allowing communication between them via TCP socket.

Figure 6: Edge Machine Learning System of 8 Raspberry Pis

We load the pre-trained models onto these 8 Raspberry Pis, and each Pi acts as a solid node in Figure 5. Then, each parent node (Pi) tells its child nodes (Pis) which part of data they need to classify through the server. Remind that one advantage of our method is that it allows parallelization for fast classification. Table 2 shows the running time of each Pi for testing in experiment. Note that they run in parallel and the total testing completion time is 95 seconds, which is much faster than using a single Pi with a complicated model to classify all the data (150 seconds).

Table 2: Running time for testing

Name	# of samples	Running time (s)
All (root node)	5000	90
Animal	3000	61
Plant	2000	38
Vertebrates	2500	52
Invertebrates	500	8
Reptiles	100	2
Fish	2150	47
Mammals	250	5

6 Conclusion & Future Work

In this project, we propose a tree-based hierarchical classification model and design a pruning algorithm for edge computing with resource-constrained IoT devices. We use the algorithm to train models on CIFAR-100 dataset. We load these pre-trained models onto 8 Raspberry Pis for testing. The results show that, compared to using a complicated model to classify all the data, our method achieves faster image classification through parallelization.

For future work, we will consider the following directions:
(1) improve the accuracy of individual classifiers with sophisticated models;
(2) inspect the influence of the prior taxonomy structure on the final tree topology as well as the overall performance;
(3) extend the optimization for inhomogeneous edge devices.

References

[1] https://cloud.google.com/solutions/automating-iot-machine-learning.

[2] https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-deploy-machine-learning.

[3] https://microsoft.github.io/ELL/.

[4] Kumar, A., Goyal, S. & Varma, M. (2017). Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things. In *International Conference on Machine Learning*, pp. 1935-1944.

[5] Li, M., Andersen, D. G., Smola, A. J., & Yu, K. (2014). Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pp. 19-27.

[6] Hsieh, K., Harlap, A., Vijaykumar, N., Konomis, D., Ganger, G. R., Gibbons, P. B., & Mutlu, O. (2017). Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *NSDI*, pp. 629-647.

[7] Cui, H., Zhang, H., Ganger, G. R., Gibbons, P. B., & Xing, E. P. (2016). GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*.

[8] Xing, E. P., Ho, Q., Dai, W., Kim, J.K., Wei, J., Lee, S., Zheng, X., Xie, P., Kumar, A. & Yu, Y. (2015). Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2), pp. 49-67.

[9] Trapeznikov, K., & Saligrama, V. (2013). Supervised sequential classification under budget constraints. In *Artificial Intelligence and Statistics*, pp. 581-589.

[10] Xu, Z., Weinberger, K., & Chapelle, O. (2012). The greedy miser: Learning under test-time budgets. In *Proceedings of the 29th International Conference on Machine Learning*.

[11] Nan, F. & Saligrama, V. (2017). Adaptive Classification for Prediction Under a Budget. In *Advances in Neural Information Processing Systems*, pp. 4730-4740.

[12] Nan, F., Wang, J., & Saligrama, V. (2016). Pruning random forests for prediction on a budget. In *Advances in neural information processing systems*, pp. 2334-2342.

[13] Gupta, C., Suggala, A.S., Goyal, A., Simhadri, H.V., Paranjape, B., Kumar, A., Goyal, S., Udupa, R., Varma, M. & Jain, P. (2017). ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices. In *International Conference on Machine Learning*, pp. 1331-1340.

[14] Garg, V. K., Dekel, O., & Xiao, L. (2018). Learning SMaLL Predictors. arXiv preprint arXiv:1803.02388.