ESAI: Efficient Split Artificial Intelligence via Early Exiting Using Neural Architecture Search

Behnam Zeinali, Di Zhuang, Member, IEEE, and J. Morris Chang, Senior Member, IEEE

Abstract—Deep neural networks have demonstrated superior performance in various computer vision tasks compared to traditional machine learning algorithms. However, deploying these models on resource-constrained mobile and IoT devices poses computational challenges. Many devices resort to cloud computing, where complex deep learning models analyze data on servers. This approach increases communication costs and hampers system efficiency in the absence of a network connection. In this paper, we introduce a novel framework for deploying deep neural networks on IoT devices. This framework leverages both cloud and on-device models by extracting meta-information from each sample’s classification result. It assesses the classification’s performance to determine whether sending the sample to the server is necessary. Extensive experiments on CIFAR10 and CINIC10 datasets reveal that only 45% of CIFAR10 and 60% of CINIC10 test data need to be transmitted to the server using this technique. The overall accuracy of the framework is 94% and 89%, respectively, enhancing the accuracy of both client and server models.

Index Terms—Internet of Things, embedded deep learning, split artificial intelligence, neural architecture search, skin lesion analysis.

I. INTRODUCTION

With Advancements in deep learning have led to the application of deep neural networks in various real-world scenarios, including botnet detection [1], [2], community detection [3], active authentication [4], image classification [5], and facial recognition [6], [7]. Concurrently, the evolution of information technologies like cloud computing, wireless communication, and the Internet of Things (IoT) has facilitated the widespread adoption of mobile and IoT devices. Recent progress in mobile technology has made these devices more practical and desirable. Integrating deep learning models onto mobile and IoT devices has become essential, but it presents challenges. Many practical deep learning models are resource-intensive and power-consuming, making implementation on mobile devices impractical. However, opting for lighter models with reduced power consumption often sacrifices accuracy. Thus, achieving a balance between resource efficiency and prediction accuracy is crucial for such applications.

Cloud computing technology, exemplified by services like Google Cloud AutoML [8], offers a potential solution to these challenges through Machine Learning as a Service (MLaaS) systems. These systems utilize robust cloud servers for machine learning tasks, leveraging data collected from IoT devices. While enhancing accuracy and reducing computational energy consumption for mobile devices, this approach increases communication costs and requires consistent, reliable resources. However, in scenarios with limited or unavailable communication resources, such as remote or rural areas without access to satellite communication, alternative approaches are necessary to address communication constraints effectively.

Therefore, it is crucial to develop a framework that maximizes the strengths of both client and server sides. This framework, adopting a client-server architecture with the mobile device as the client and cloud computing resources as the server, entails an intelligent decision-making process. It determines whether to process each input sample using the client or server model, aiming to optimize accuracy efficiently by leveraging the advantages of both models.

However, developing such a system presents numerous challenges. Firstly, deploying an accurate, compact, and efficient deep neural network (DNN) model on the client side is crucial to maintain acceptable accuracy while minimizing power consumption, especially in areas without access to satellite networks. Furthermore, when the network is available, the framework must exhibit intelligence in selecting suitable samples for transmission to the cloud network. This strategic decision-making process is geared towards attaining higher accuracy while minimizing communication costs.

Compressing the client model for mobile deployment without accuracy loss, especially in satellite-unavailable areas, is challenging. Techniques like knowledge distillation transfer knowledge from a complex to a simpler model [9], [10], [11]. However, this limits the framework’s ability to use a highly accurate server model without computational constraints. The Split-DNN architecture [12], [13], [14], [15] also relies on satellite resources, rendering it ineffective when communication is disrupted. Overcoming these challenges is crucial for successful deployment in varied network conditions.

The paper presents an intelligent AI framework using DNN models, operating as a client-server system. It intelligently decides whether input data should go to the server or be classified by the client model. This decision considers precision level,

Received 25 August 2022; revised 25 April 2024; accepted 25 July 2024.

Digital Object Identifier 10.1109/TETCI.2024.3485677
device energy, and their trade-off, aiming for optimization. If data doesn’t need server processing and communication energy is low, the framework classifies it on the client side.

For a lightweight client model with acceptable accuracy, we utilized the Neural Architecture Search (NAS) morphism algorithm [16]. Employing the knowledge distillation technique [9], we introduced a novel search strategy to derive this model. Equipped with a decision unit, the output uses meta-information to independently determine whether to send data to the server or present the classification result to the user.

To evaluate the proposed method, we used the skin lesion 2019 dataset [17], [18], [19], along with the CINIC10 and CIFAR10 datasets. The skin lesion dataset comprises 25,331 images representing eight diseases, divided into client-train, decision unit train (du-train set), and test sets, accounting for 80%, 10%, and 10% of the dataset, respectively. For CIFAR10, we utilized 40,000 images for training, 10,000 images for training the decision unit, and 10,000 images for testing. Similarly, for CINIC10, 90,000 images were used for training, 90,000 images for training the decision unit, and 90,000 images for testing.

Our study delves into combining knowledge distillation [9] with morphism-based NAS [16] to create a compact yet accurate model suitable for IoT devices. We then explore meta-information’s role in training the decision unit, drawing from [20]’s concept of using statistical features to gauge model uncertainty. Through features like maximum probability, least confidence, entropy, and standard deviation, we aim to enhance our framework’s decision-making abilities.

Lastly, we evaluate the accuracy and efficiency of the multi-exit framework in conserving computational power. Our experiments demonstrate that the framework sends 40%, 45%，and 60% of the test data to the server, achieving accuracies of 92%, 94%，and 89% for the Skin Lesion, CIFAR10, and CINIC10 datasets, respectively.

In summary, our contributions are:

- We introduce a novel and intelligent Split-AI architecture designed to efficiently deliver accurate deep learning services for mobile/IoT-based applications, leveraging both client and server models.
- We enhance the morphism-based NAS design by integrating Knowledge Distillation techniques into its searching strategy, resulting in a lightweight model deployable on mobile devices.

The remaining sections of this paper are structured as follows:

In Section II, we review existing efforts by researchers to implement deep neural networks on mobile devices. We also provide an overview of NAS algorithms and knowledge distillation techniques. Section III presents a detailed explanation of the proposed method, outlining each step of our approach. The experimental results are presented in Section IV, where we showcase the outcomes of our methodology. Finally, Section V offers concluding remarks for the paper.

II. RELATED WORK

In this section, first, a literature review about what has been done so far for reducing the network size and taking advantage of both a client and a server model has been done. Then, we review the idea of neural architecture search (NAS) and knowledge distillation techniques (KD) that have been harnessed in the proposed method.

A. Compact Deep Neural Networks

Compact Deep Neural Networks (Compact DNNs) enable real-world AI applications on mobile and IoT devices. They reduce redundancy in current DNN designs through innovative building blocks. For example, SqueezeNet [21] achieves AlexNet [22]-level accuracy with 50 times fewer parameters using 1 × 1 convolution filters. MobileNet [23], [24], [25] introduces the “inverted residual block” to reduce computational complexity without accuracy loss. YOLO [26] customizes architecture for real-time object detection with one-fourth of VGG16’s operations. EfficientNet [27] scales each dimension of DNN models uniformly for mobile and IoT devices.

Despite reducing complexity, such approaches require significant design insight. Compact DNN models may not match the performance of advanced server-side models, which could involve ensemble or fusion techniques.

B. Compressed Deep Neural Networks

DNN model compression [9], [10], [11], [28], [29], [30], [31], [32] provides an automated approach for efficient DNN design on mobile and IoT devices. Techniques like data quantization [30], network pruning [33], and knowledge distillation [9], [10], [11] compress models without relying on specific design principles. While compressed DNNs offer flexibility, relying solely on them may not exploit the benefits of advanced server-side models.

C. Split Deep Neural Networks

Split-DNN architectures [12], [13], [14], [15] divide a DNN into client-side and server-side components, maintaining lightweight feature extraction and data preprocessing on mobile or IoT devices while offloading complex DNN execution to powerful servers. For example, Osia et al. [15] propose a hybrid architecture where a DNN is split into a feature extraction network on the device and a classification network on the cloud. Both networks collaborate to execute the original DNN model. Matsubara et al. [34] introduce a KD-based Split-DNN framework to reduce communication costs between client and server. However, such approaches may not fully rely on the client-side model and fail if communication is impeded.

D. Multi-Exit Deep Neural Networks

Multi-exit Deep Neural Networks [35], [36], [37], [38], [39] offer DNNs with additional exits, choosing the best exit during testing based on criteria like accuracy and efficiency. For example, BranchyNet [35] augments standard DNN architectures like LeNet, AlexNet, and ResNet with early exiting structures. Multi-Scale DenseNet [36] provides a customizable multi-exit architecture, utilizing earlier exits as feature extractors. DNNNet [37]
E. Neural Architecture Search and Morphism-Based NAS

Since the advent of AlexNet in 2012 [22], human-made neural network architectures have seen remarkable progress. Researchers have developed various architectures like VGG19 [40], InceptionV3 [41], Xception [42], DenseNet [43], InceptionResNetV2 [44], and ResNet-152 [45]. These models, with diverse combinations of layers, are crafted by experts with extensive experience in machine learning and feature engineering. While time-consuming and intricate, they offer versatile solutions for different datasets and objectives, demanding specialized knowledge for tailored design.

To address this issue, Neural Architecture Search (NAS) has been introduced [46]. NAS automatically seeks optimal architectures for specific targets, potentially outperforming manually designed models in classification tasks. Two crucial aspects must be clarified for NAS implementation: the search space and the evaluation strategy for each architecture. For the search space, recent techniques like cell-based NAS algorithms [47] connect a predefined number of architecture cells to define the space, using evolutionary algorithms to evaluate architectures. Similarly, Liu et al. [48] utilized a bilevel optimization system for architecture evaluation.

The Neural Architecture Search (NAS) method based on the morphism algorithm was introduced by Elsken et al. [16]. In their approach, they define the search space using a morphism-based method. Initially, the algorithm sets an initial architecture, then extends it by randomly adding layers such as fully connected, convolutional, skip, and concatenation layers. To evaluate each new architecture, the algorithm employs a hill-climbing method. Hill climbing is a greedy algorithm that compares each new model with the previous one and replaces it if the new model demonstrates better performance.

F. Knowledge Distillation

The majority of models, whether manually designed or generated by NAS algorithms, are often too cumbersome for IoT devices due to their high power and storage consumption. Energy usage in IoT devices primarily stems from memory references, making model size reduction crucial for minimizing storage and power consumption. Techniques like knowledge distillation [9] and model compression [30] offer solutions to reduce model size while maintaining accuracy. Knowledge distillation, particularly, has gained popularity for its simplicity and effectiveness [49]. This method trains a smaller model by transferring knowledge from a larger one, using softened labels generated by the softmax output of the larger model. This allows the smaller model to capture essential knowledge without noise or unnecessary data. The amount of information transferred can be controlled by a temperature parameter (T) in the softmax function.

\[
S_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}
\]  

which \(z_i\) and \(S_i\) are the output probability and the softened logit value for class \(i\), respectively. A bigger value for T means that more information is being transferred to the smaller model.

Our work: Our split-AI architecture leverages both compact and efficient DNNs on the client side and highly accurate DNNs on the server side, surpassing the performance of solely applying compact DNNs. For example, we developed a novel split-AI architecture to deliver efficient and accurate deep-learning services on both mobile/IoT devices and servers. For the client model, we integrated Knowledge Distillation techniques into the searching strategy of morphism-based NAS to generate lighter DNN models. Additionally, we proposed the use of multi-exit DNN architecture for flexible and efficient trade-off tuning between resource usage efficiency and prediction accuracy.

III. METHODOLOGY

A. Overview

For a Split-AI Architecture, accurately deciding whether a sample should be sent to the server or classified using the client model is crucial. Ideally, with an 80% accurate client model, 20% of samples (misclassified) should be sent to the server. By sending data intelligently, the framework optimally utilizes the client model and incurs communication costs only for samples that require the better server model. This improves overall accuracy using both client and server models while minimizing communication costs.

The proposed decision unit in the framework identifies misclassified samples using meta-information. It is a 4-layer neural network with four inputs and a binary output, providing uncertainty values for classification performance. Training for the server, client, and decision unit is conducted once, after which they are integrated into the framework. To optimize the client model, our method enhances the NAS morphism algorithm’s search space using the knowledge distillation technique. Section III-B elaborates on our Split-AI framework, while Section III-C details our client model approach. Section III-D outlines the multi-exit concept.

B. Split-AI Framework

The Split-AI framework comprises two parts: a client framework and a high-accuracy model on the server. The client framework, implemented on edge devices, includes a light CNN model with acceptable accuracy, a feature extraction module, and a decision unit model. On the server side, a high-accuracy model has been trained and implemented. Fig. 1 illustrates the proposed framework.

Inspired by [20], the decision unit plays a crucial role in the framework. It is a 4-layer neural network responsible for identifying samples misclassified by the client model and sending them to the server. To train this model, after training the client model using the training set, the validation set is used for inference, and meta-information is extracted from the output logits. Since the labels of the validation set are available, the labels for the decision unit would be zero for samples misclassified by the client model and one for the others.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
The decision unit is a binary classifier trained using these two label data extracted from the validation set. For testing, misclassified samples (class 0) are separated and sent to the server. A parameter called the decision unit’s sensitivity, acting as a threshold for the decision, is defined and is a number between 0 and 1. According to Fig. 1, the decision unit will show the classification result to the user only when the output of its logit layer exceeds this sensitivity value; otherwise, it will send the data to the server.

1) Meta Information: Various types of meta-information can be extracted from the machine learning model output to identify the uncertainty of the results. In our work, we have extracted maximum probability, least confidence, entropy, and standard deviation as meta-information. The equations for each are defined below:

\[
MP = \max(P_i) \tag{2}
\]

\[
LC = (P_s(0) - P_s(1)) \tag{3}
\]

\[
Entropy = \sum_i P_i \log P_i \tag{4}
\]

\[
std = \sqrt{\frac{\sum_i (P_i - \mu)^2}{N}} \tag{5}
\]

where \(P_s = \text{sort}(P_i)\), \(P_i\) is the probability of the client output for each class \(i\), \(\mu\) is the mean value of \(P_i\) and \(N\) is the number of classes. Imagine a classifier with three classes and sample A and sample B have been applied to this classifier and we want to decide which one of them has more uncertainty and can be a better candidate to be sent to the server instead of on-device classification. Table I shows the extracted metainformation from two example samples. For example, the least confidence of sample A after being classified is \(0.9 - 0.09 = 0.81\) while sample B is \(0.5 - 0.3 = 0.2\) which means sample B has higher uncertainty and is more prone to be sent to the server. By computing other values (entropy, maximum probability, and standard deviation which are the last three rows of the Table I), sample B is an appropriate sample to be sent to the server.

2) Feature Extraction: After applying each sample of the validation set to the client model, meta-information from the output logits of samples has been extracted using equations from 2 to 5. Consequently, four features for each sample are available. Since the label is also available, the performance of the client model on that specific sample can be inferred. If the sample has been classified correctly, the label would be 1; otherwise, it would be 0. With these new labels and features, a four-layer neural network, which is a binary classifier, has been trained. The trained model is responsible for evaluating the results of the client model on the test data.

Fig. 2 depicts the distribution of all the features extracted using the equations 2 to 5 as meta-information, along with their labels, after applying all the validation set data to the client model. It is evident that the meta-information exhibits a good separability feature based on true and false classified samples. Our experimental results show that the performance of the framework by extracting meta-information as features is better than only using the maximum probability of the output logits. For example, the accuracy of the decision unit for separating samples that have been classified correctly by the client model is 86% for the CIFAR10 dataset, while using only the maximum probability as a confidence score, the accuracy is around 82%.

C. The Client Model

Developing neural networks for IoT devices has received limited attention in recent years. Researchers are endeavoring to

---

**TABLE I**

|          | sampleA | sampleB |
|----------|---------|---------|
| class1   | 0.9     | 0.2     |
| class2   | 0.09    | 0.3     |
| class3   | 0.01    | 0.3     |
| probability | 0.9   | 0.5     |
| least confidence | 0.81 | 0.2     |
| entropy  | -0.35   | -1.02   |
| standard deviation | 0.12 | 0.4     |

Probability of each sample for different classes and their related meta information using (2)-5.
devise new models with low storage and inference time, along with higher accuracy. Research in this area can be categorized into two main methodologies.

In the first approach, lightweight models with good accuracy have been designed automatically or by an expert to address classification, segmentation, and object detection problems. Recently introduced architectures include SqueezeNet [21], MobileNet [23], MobileNetV2 [24], EfficientNetB0 and B1 [27], and M NasNet [50]. In the second approach, researchers are aiming to reduce the model size of larger architectures while maintaining their accuracy. Techniques such as compression [51] and distillation [49] have been utilized in recent years to leverage well-known architectures.

However, these approaches introduce new challenges. For example, in the knowledge distillation technique, the student model needs to mimic the structure of the cumbersome model to achieve optimal performance. Designing a compact DNN model is also a tedious task and requires expertise to attain an acceptable model. Additionally, the input size of the dataset should be adjusted to match the predefined input size of the already designed architectures to utilize the full capacity of the network if the transfer learning method is being utilized. Incorporating a compression technique with a Neural Architecture Search (NAS) automatically can address these challenges and lead to design models appropriate for resource limited devices. We have addressed this demand by integrating the knowledge distillation technique with the NAS morphism algorithm [16].

In the proposed method, as illustrated in Fig. 3, first, soft labels are generated using the method outlined in [52]. The ImageNet [53] pre-trained InceptionResNet model has been trained using the transfer learning method, and its logit layer is used to produce the softened labels. Then, a base teacher model is designed and trained using these soft labels as a teacher model. The architecture of this base model has been demonstrated in Table II. The extension procedure and evaluation are conducted in the search space, as depicted in Fig. 3. In each iteration, the algorithm extends the teacher model by randomly adding one of these procedures to the teacher model and creates student models:

- Deepen the student model by adding the convolutional layers in between randomly.
- Widen the student model by adding more filters to a randomly selected convolutional layer.

### Table II: Architecture of the First Teacher Model

| Stage | Operator | Filters | Channels(n) | Layer |
|-------|----------|---------|-------------|-------|
| 1     | Input(N*N) | -       | 3           | -     |
| 2     | Conv2D   | 3*3     | 32          | 1     |
| 3     | Conv2D   | 3*3     | 64          | 1     |
| 4     | Conv2D   | 3*3     | 128         | 3     |
| 5     | Conv2D   | 3*3     | 64          | 1     |
| 6     | Flatten  | -       | 128         | 1     |
| 7     | Dense    | -       | M           | 1     |

N = Input Size, M = Number of Classes.

- Make a concatenation between randomly selected layers.
- Make an add or skip connection like the idea of [45] between randomly selected layers.

After training each student model for 8 epochs, the student model with the least validation loss is chosen as a new teacher model. The algorithm continues for eight iteration or stops when the number of parameters reaches to a predefined number (5 * 10^6).

One of the advantages of the proposed method is that the resulting model is trained during the search time and does not need to be trained from scratch in each iteration. Consequently, for each student model, other than the newly added architecture part, other training parameters will be inherited from the previous teacher model. Also, unlike the knowledge distillation technique, the model does not need to follow any specific structure, and reaching a good accuracy with an acceptable model size is the final goal. Moreover, the distilled knowledge is transferred to the model in each iteration when the model is growing, leading to a model with a better performance. Additionally, unlike transfer learning methods, the input size of the network can be set to the size of the target dataset, and it does not need to be adjusted with the pre-trained network’s input size. The temperature value (T) is set to T = 20 according to (1) in the proposed algorithm.

### D. Multi-Exit Architecture

The proposed framework can be extended into three stages. The client model can be converted into a two-exit neural network, and a similar idea for the last exit can be applied to the earlier exit. The output logits of the earlier exits can be...
calculated after applying the model to the validation set dataset, and after extracting the meta-information, a separate neural network model can be trained as a decision unit for this stage. Based on meta-information, this time, correctly classified samples from the earlier exit can be determined by the decision unit and the classification result can be shown to the users. By doing so, a sample does not need to be calculated through all the layers of the neural network in the client model on IoT devices, which is computationally efficient. Consequently, the inference time can be reduced, and more energy can be saved.

IV. EXPERIMENTAL RESULT

To evaluate our proposed framework, different datasets have been used. Since the proposed framework can be used as a healthcare framework [54] implemented on mobile devices, the International Skin Imaging Collaboration Challenge 2019 (ISIC 2019) [18], [55], [56] was chosen to evaluate our proposed framework. The total number of images containing train and test images is 33,569 images. However, only the labels of the training data are available. In this paper, only the labeled images have been utilized to evaluate the proposed methods. The number of labeled images is 25,331 images of eight different skin diseases, which are basal cell carcinoma, benign keratosis, vascular lesion, melanoma, squamous cell carcinoma, melanocytic nevus, actinic keratosis, and dermatofibroma. The number of images for each class is 3,323, 2,624, 253, 4,522, 628, 12,875, 876, 239 respectively. We randomly split 80%, 10%, and 10% as client-training data, decision unit training data, and testing data respectively.

In addition to this dataset, the CINIC10 and CIFAR10 datasets are also utilized to evaluate the effectiveness of the proposed framework on complex datasets. The CIFAR10 dataset is balanced and contains images with the size of $32 \times 32$ in 10 different classes, which is smaller than the image size in the ISIC dataset. The dataset contains 50,000 images for training and 10,000 for testing. However, we have separated 10,000 samples from the training images for training the decision unit model. The CINIC dataset is similar to the CIFAR10 with more images. It contains 270,000 images of 10 classes that are evenly distributed between train, validation, and test sets. The resolution of the images is also $32 \times 32$ in this dataset.

To evaluate the proposed distillation-based NAS technique, the Imagenet16-120 dataset is utilized. This dataset contains the first 120 classes of the ImageNet dataset in the resolution of 16x16. It contains 150,000 images for the train set and 6,000 images for the test set. 6,000 images from the train set have been separated for the validation set.

A. Experimental Considerations

The experimental evaluation was conducted using the Python language. The implementation was carried out using the TensorFlow framework, and the results of the TensorFlow implementation were reported. A single GPU, specifically the Nvidia GeForce GTX 1080 Ti with 11 GB GDDR5X memory, was utilized for training all the models. A batch size of 32 was used for training. An initial learning rate of 0.001 was employed, and after completing the search, it was reduced after 10 epochs if there was no improvement in the learning process. The training objective function aimed to minimize the categorical cross-entropy loss for the predictions and labels. To ensure a fair comparison, identical hyperparameters were maintained across all models during the training phase.

B. Evaluating the Performance of KD-NAS

The results of applying the knowledge distillation technique to the search algorithm of the NAS morphism indicate that the proposed method is suitable for identifying a lightweight neural architecture model for implementation on edge devices. Experimental results on the skin lesion, CINIC10, and CIFAR10 datasets are shown in Tables III and IV. The model sizes demonstrate that our proposed KD-NAS model has comparable memory references in Table III and fewer memory references in Table IV. Since the same model serves as the decision unit for all networks, its size has not been considered for comparison. The TensorFlow Lite model size of the decision unit is 2 KB, which can be negligible in terms of efficiency on edge devices.

As the performance of the decision unit heavily relies on the extracted meta-information from the client model, having a client model with acceptable accuracy is crucial. However, since the model is intended for implementation on edge devices, the model size should also be considered. The proposed KD-NAS method offers the ability to strike a balance between accuracy and model size. For training other models in the table and to achieve the highest possible accuracy, we utilized ImageNet pre-trained architectures [53] and trained the models using the transfer learning method.

When deploying state-of-the-art models on edge devices using the transfer learning method, regardless of the size of the input data, it needs to be adjusted to the input size of the network to achieve maximum accuracy. However, with the proposed method, the input size of the network can be adapted to the input size of the dataset. By comparing the results in Tables III and IV, it can be observed that the model size of the state-of-the-art models is almost the same for skin lesion, CINIC10, and CIFAR10 datasets. However, for the CIFAR10 and CINIC10 datasets, the models obtained by using the proposed method have comparable accuracy with fewer memory references, demonstrating the applicability of the proposed method for edge devices.

The InceptionResNet [44] model has been utilized as a server model in this paper. Other models are state-of-the-art models proposed by researchers for deployment on mobile and IoT devices. They are trained using the transfer learning method. The CINIC10 dataset includes the CIFAR10 dataset and 32x32 dimensions of some parts of the ImageNet dataset. Using the transfer learning method on the ImageNet pre-trained model leads to higher performance for this dataset since these pre-trained models have already been trained on some part of the CINIC10 data images with better resolution. Comparable accuracy can be achieved by training these models from scratch. For instance, the test accuracy of training MobileNet and MobileNetV2 on the CINIC10 dataset from scratch is 57% and 59%, respectively, while the KD-NAS model’s accuracy is 75%.
TABLE III
KD-NAS AND OTHER STATE-OF-THE-ART ARCHITECTURES FOR SKIN LESION DATASET

| Architecture       | Accuracy | Number of Parameters | TensorFlow Model Size (MB) |
|--------------------|----------|----------------------|---------------------------|
| EfficientNet-B0 [27] | 81.5%    | 4,059,812            | 16 MB (2KB)               |
| EfficientNet-B1 [27] | 83%      | 6,585,680            | 26 MB (2KB)               |
| MobileNet [23]      | 80.5%    | 3,237,064            | 12.8 MB (2KB)             |
| MobileNetV2 [24]    | 76%      | 2,288,232            | 9 MB (2KB)                |
| InceptionResNet [44]| 85.8%    | 53,877,672           | 200 MB                    |
| NAS [16]            | 80%      | 1,090,312            | 7 MB (2KB)                |
| KD-NAS              | 82%      | 4,969,032            | 19 MB (2KB)               |
| ESAI Framework      | 92%      | -                    | -                         |

The number in the parenthesis shows the tensorflow lite model size of the decision unit.

TABLE IV
KD-NAS AND OTHER STATE-OF-THE-ART ARCHITECTURES FOR CIFAR10 AND CINIC10 DATASETS

| Architecture       | CIFAR10 Dataset | CINIC10 Dataset |
|--------------------|-----------------|-----------------|
|                    | Accuracy | # of Parameters | TF Model Size | Accuracy | # of Parameters | TF Model Size |
| EfficientNet-B0 [27] | 85%      | 4,380,070       | 17.4 MB (2KB) | 84.80%    | 4,380,070       | 17.4 MB (2KB) |
| EfficientNet-B1 [27]| 86.5%    | 6,905,738       | 27.4 MB (2KB) | 85.92%    | 6,905,738       | 27.4 MB (2KB) |
| MobileNet [23]      | 84%      | 3,493,834       | 13.9 MB (2KB) | 82.1%     | 3,493,834       | 13.9 MB (2KB) |
| MobileNetV2 [24]    | 83%      | 2,588,490       | 10.2 MB (2KB) | 81.68%    | 2,588,490       | 10.2 MB (2KB) |
| InceptionResNet [44]| 89%      | 54,932,778      | 202 MB        | 86%       | 54,932,778      | 202 MB        |
| NAS [16]            | 81%      | 745,866         | 3 MB (2KB)    | 58.36%    | 958,858         | 3.8 MB (2KB)  |
| KD-NAS              | 84%      | 893,962         | 3.6 MB (2KB)  | 75%       | 2,952,138       | 11.8 MB (2KB) |
| ESAI Framework      | 94%      | -                | 89%           |

The number in the parenthesis shows the tensorflow lite model size of the decision unit.

TABLE V
COMPARISON OF NAS MORMPHISM [16] METHOD AND THE PROPOSED METHOD

| Method                 | Dataset     | Training Time (Minute) | Number of Parameters (Million) | Search Time (GPU Day) | Accuracy (%) |
|------------------------|-------------|------------------------|---------------------------------|------------------------|--------------|
| NAS Method [16]        | CIFAR10     | 46                     | 2.6                             | 0.4                    | 77.6         |
|                        | ImageNet16-120 | 56                    | 1.2                             | 0.6                    | 24.2         |
| Proposed Method        | CIFAR10     | 52                     | 2.8                             | 0.1                    | 81.6         |
|                        | ImageNet16-120 | 80                   | 2.9                             | 0.3                    | 29.0         |

C. Distillation-Based Nas Morphism

In this paper, we introduce an enhanced approach to neural architecture morphism that leverages the knowledge distillation technique. Building upon the existing framework, our method aims to improve the efficiency and performance of neural architecture search (NAS) morphism [16] by incorporating knowledge distillation. This process transfers knowledge from a larger, teacher model to a smaller, student model. By integrating this technique into the NAS algorithm, we achieve superior results in terms of search time, accuracy, and suitability for deployment on resource-constrained devices.

To evaluate the proposed algorithm, both the proposed method and the original method proposed in [16] are applied to CIFAR10 and ImageNet16-120 datasets. The same teacher model, as a base model, has been trained for both methods, and similar hyper-parameters and learning rates (0.001) have been considered for both methods to ensure a fair comparison. The architecture of the first teacher model is shown in Table II. N is the image resolution and M is the number of classes. 32 and 10 are the values for N and M for the CIFAR10 dataset and 16 and 120 for the Imagenet16-120 respectively. In the proposed method, unlike [16], the teacher model has not been considered as a new student model in each iteration to reduce searching time. Table V presents the average performance of each method on the ImageNet16-120 and CIFAR10 datasets after 5 runs. The algorithm will stop after N iterations or if the number of parameters of the model reaches 5 * 10^6. N is considered as 8 for ImageNet16-120 and 10 for CIFAR10. It can be observed from the table that the proposed method outperforms NAS morphism method in terms of accuracy and resource utilization.

D. Evaluate the Performance of Split-AI (Decision Unit)

The framework’s objective is to minimize communication while maintaining or enhancing accuracy. The decision unit is tasked with identifying misclassified samples and transmitting them to the server. Therefore, to evaluate the performance of our proposed Split-AI framework, we have concentrated on the performance of the decision unit. To accomplish this, following the training of the model using the training dataset (as presented in Table III), the client model’s output logits were computed by applying the model to the validation set data, meta-information...
TABLE VI
KD-NAS and OTHER STATE-OF-THE-ART ARCHITECTURES DECISION UNITS’ PERFORMANCE FOR SKIN LESION DATASET

| Architecture   | Accuracy | AUC  | Sensitivity | Specificity |
|----------------|----------|------|-------------|-------------|
| EfficientNet-B0 [27] | 81       | 0.79 | 0.91        | 0.32        |
| EfficientNet-B1 [27] | 82       | 0.78 | 0.94        | 0.19        |
| MobileNet [25]    | 75       | 0.80 | 0.95        | 0.26        |
| MobileNetV2 [24]  | 84       | 0.83 | 0.95        | 0.32        |

Decision units are being trained using the output logits of each model. The architecture is the same for all decision units.

was extracted from logits as features for training the decision unit model.

The True Positive (TP) for the decision unit means that the sample has been correctly classified by the client model and the result has been shown to the user. Likewise, True Negative (TN) means that the client model has misclassified the sample, but the decision unit has made the correct decision by sending the data to the server. False Positive (FP) means that a sample should have been sent to the server, but the classification result has been shown to the user. Similarly, False Negative (FN) means that the result should have been shown to the user, but the sample has been sent to the server (unnecessary communication). Furthermore, the accuracy, sensitivity, and specificity metrics can be defined as follows:

\[
\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \tag{6}
\]

\[
\text{Sensitivity} = \frac{TP}{TP + FN} \tag{7}
\]

\[
\text{Specificity} = \frac{TN}{TN + FP} \tag{8}
\]

Four trained models from Table III (EfficientNet-B0, EfficientNet-B1, MobileNet, MobileNetV2), as well as the KD-NAS model (proposed model), have been utilized to extract meta-information, and a separate decision unit classifier has been trained using each model’s meta-information for comparison.

The decision unit has a parameter called sensitivity, which indicates how confident the decision unit should be about each of its decisions. By adjusting this parameter, different numbers for sensitivity, specificity, and accuracy can be obtained. This sensitivity value can be set based on the communication availability, device battery status, and even the dataset. For example, in medical field datasets, a lower False Positive rate is required to minimize showing false results to the user, allowing the cumbersome model on the server with better accuracy to classify them. To achieve this, a high value must be set for the sensitivity.

The sensitivity for the decision unit has been set to 0.5 for the results in Table VI, meaning that if the output probability of the unit is higher than this number, it assumes that the client model has made the correct classification. According to Table VI, the decision unit trained using the output logits of our proposed client model is more capable of separating misclassified samples. To evaluate the performance of the decision unit for different sensitivity parameter values, the Area under the Curve (AUC) number has been calculated. The AUC numbers in Table III also demonstrate the superiority of this decision unit for different sensitivity values for the decision unit.

E. Client-Server Ensemble Model

Fig. 4 shows the overall accuracy and the ratio of locally classified samples for the skin lesion, CINIC10 dataset, CIFAR10 dataset, and Imagenet16-120 dataset, respectively. As it can be seen from Fig. 4(a), when the sensitivity is the lowest (0), it means that all the samples are being classified by the local model and the accuracy is around 82%. Similarly, when the sensitivity is the highest(1), all samples are being classified by the server model and the accuracy is around 86%. For the sensitivity between 0.5 and 0.9, the overall accuracy is better than the client and server models and 92% accuracy can be reached by only sending around 40% of the test samples to the server which saves more communication power and shows the effectiveness of the decision unit. Similar results can be interpreted from Fig. 4(b), and (c). It can be seen from Figures that the overall accuracy is improved since the client and the server models are working as an ensemble network in the platform. For Fig. 4(d), it can be observed that the decision unit has been perfectly able to detect misclassified samples for server inference. The sensitivity number of the decision unit can be adjusted based on the communication power and the desired accuracy. For calculating the overall accuracy this equation has been used:

\[
\text{Overall Accuracy} = \frac{TP + S_P}{S_T} \tag{9}
\]

Where \( TP \) is the true positive number of the decision unit, \( S_P \) is the number of samples that have been sent to the server and have been classified correctly and \( S_T \) is the total number of test samples. For the server model, the image-net pre-trained inceptionResnet [44] model has been used for figures in Fig. 4. Any models with better performances can be replaced by this model to get better accuracy in the infrastructure implementation.

F. Evaluating the Performance of Multi-Exit Architecture

For larger client models, a similar approach can be adopted by incorporating early exits within the model. These exits involve implementing separate decision unit models, where correctly classified samples are presented to the user, while misclassified ones are forwarded to subsequent exits for further inference.

To establish the multi-exit architecture, the output probability of the initial exit’s decision unit is computed. If the certainty of the classification exceeds the decision unit’s sensitivity, the sample undergoes classification at the current exit; otherwise, it proceeds to the next exit. This process is reiterated until the final exit is reached. For samples designated to be transmitted to the server, the server model’s classification result is determined for each of them. The overall accuracy is then computed using the following equation:

\[
\text{Accuracy} = \frac{TP_1 + TP_2 + S_P}{S_T} \tag{10}
\]

where \( TP_1 \) is the true positive number of the first exit, \( TP_2 \) is the true positive number of the second exit, \( S_P \) is the number of samples that have been sent to the server, and have been classified correctly and \( S_T \) is the total number of test samples.
In this way, the whole accuracy of the framework has been calculated regardless of at which stage the samples are going to be classified.

The architecture of the Skin lesion dataset is treated as a large client model, and the multi-exit experiment is exclusively applied to this dataset. Figs. 5 and 6 illustrate the accuracy of the entire Split-AI framework and the number of samples classified by the local model and its early exit, respectively, across varying sensitivities of each decision unit. According to Fig. 5, the model accuracy is changing from 63.5% to 92%. If all the samples have been classified in the first exit, the accuracy is 63.5%. If they have been classified using only the second exit, the accuracy is 82%, and using the server model for all samples, the accuracy would be 85%.

When the accuracy of the model is around 90%, 60% of the test data has been classified in the client model and only 40% of them has been sent to the server. Consequently, not only has the accuracy been improved compared to the client and server models’ accuracy, but also only 40% of them are being sent to the server and our split framework is using fewer communication resources. From the samples that have been classified on the client, 18% of them have been classified in the first exit. Since the model size of the first exit is less than the whole model, less computational energy is being consumed by the edge devices.
V. CONCLUSION

In this paper, a novel and effective framework for mobile systems has been proposed. This framework is able to get the benefits of both models on the cloud server and the client model intelligently. To do so, a framework contains a decision unit model that is responsible for making a decision about each sample to whether to send it to the server or not. After classifying the model by the client model, the decision unit tries to intelligently evaluate the exactitude of the classification task and send the samples with high classification uncertainty. Early exiting classification has been produced by following a similar idea in which a decision unit makes the decision on the classified samples with a lower uncertainty in the earlier exit to reduce the computational power. To obtain an accurate and acceptable model, the knowledge distillation technique has been applied to the morphism-based neural architecture search methodology in a specific way and then defines an early exit before the last convolutional layers of the model. In our future work, we will try to enhance the framework by introducing better decision unit techniques and more powerful metainformation. Also, we will try to Improve the model parameters based on the last decision on the sample in an intelligent manner.

ACKNOWLEDGMENT

The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

REFERENCES

[1] D. Zhuang and J. M. Chang, “Peerhunter: Detecting peer-to-peer botnets through community behavior analysis,” in Proc. 2017 IEEE Conf. Dependable Secure Comput., 2017, pp. 493–500.
[2] D. Zhuang and J. Chang, “Enhanced PeerHunter: Detecting peer-to-peer botnets through network-flow level community behavior analysis,” IEEE Trans. Inf. Forensics Secur., vol. 14, no. 6, pp. 1485–1500, Jun. 2019.
[3] D. Zhuang, M. Chang, and M. Li, “DynaMo: Dynamic community detection by incrementally maximizing modularity,” IEEE Trans. Knowl. Data Eng., vol. 33, no. 5, pp. 1934–1945, May 2021.
[4] P.-Y. Wu, C.-C. Fang, J. M. Chang, and S.-Y. Kang, “Cost-effective kernel ridge regression implementation for keystroke-based active authentication system,” IEEE Trans. Cybern., vol. 47, no. 11, pp. 3916–3927, Nov. 2017.
[5] K. Chen, D. Zhuang, and J. M. Chang, “Discriminative adversarial domain generalization with meta-learning based cross-domain validation,” Neurocomputing, vol. 467, pp. 418–426, 2022.
[6] H. Nguyen, D. Zhuang, P.-Y. Wu, and M. Chang, “Autogated-based dimension reduction for privacy preservation,” Neurocomputing, vol. 384, pp. 94–103, 2020.
[7] D. Zhuang, S. Wang, and J. M. Chang, “FRIPAL: Face recognition in privacy abstraction layer,” in Proc. 2017 IEEE Conf. Dependable Secure Comput., 2017, pp. 441–448.
[8] “Google cloud automl,” Accessed: Nov. 9, 2018. [Online]. Available: https://cloud.google.com/automl/
[9] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 1–15.
[10] J. Ba and R. Caruana, “Do deep nets really need to be deep?,” in Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 2654–2662.
[11] A. Polino, R. Pancasa, and D. Alistarh, “Model compression via distillation and quantization,” in Proc. Int. Conf. Learn. Representations, 2018.
[12] H.-J. Jeong, J. Jeong, H.-J. Lee, and S.-M. Moon, “Computation offloading for machine learning web apps in the edge server environment,” in Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst., 2018, pp. 1492–1499.
[13] Y. Kang et al., “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” ACM SIGARCH Comput. Archit. News, vol. 45, no. 1, pp. 615–629, 2017.
[14] N. D. Lane et al., “DeepX: A software accelerator for low-power deep learning inference on mobile devices,” in Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw., 2016, pp. 1–12.
[15] S. A. Osia et al., “A hybrid deep learning architecture for privacy-preserving mobile analytics,” IEEE Internet Things J., vol. 7, no. 5, pp. 4505–4518, 2020.
[16] T. Elsken, J.-H. Metzen, and F. Hutter, “Simple and efficient architecture search for convolutional neural networks,” in Proc. Int. Joint Conf. Artif. Intell., 2017, pp. 1–7.
[17] D. Gutman et al., “Skin lesion analysis toward melanoma detection: A challenge at the international symposium on biomedical imaging (ISBI) 2016, hosted by the international skin imaging collaboration (ISIC),” 2016, arXiv:1605.01397.
[18] N. C. Codella et al., “Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (ISBI), hosted by the international skin imaging collaboration (ISIC),” in Proc. IEEE 15th Int. Symp. Biomed. Imag., 2018, pp. 168–172.
[19] N. C. Codella et al., “Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (ISIC),” IEEE Trans. Med. Imag., vol. 38, no. 12, pp. 2793–2803, Dec. 2019.
[20] S. Tong, Active Learning: Theory and Applications, vol. 1. Stanford, CA, USA: Stanford Univ., 2001.
[21] 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.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Special Operations Command.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Zeinali et al.  EIAI: Efficient Split Artificial Intelligence Via Early Exiting Using Neural Architecture Search 971

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. Neural Inf. Process. Syst., 2012, pp. 1097–1105.

[23] A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861.

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 4510–4520.

[25] A. Howard et al., “Searching for mobilnetv3,” in Proc. IEEE Int. Conf. Comput. Vis., 2019, pp. 1314–1324.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 779–788.

[27] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. 35th Int. Conf. Mach. Learn., 2019, pp. 6105–6114. [Online]. Available: https://arxiv.org/abs/1905.11946

[28] L. N. Huyhn, Y. Lee, and R. K. Balan, “DeepMon: Mobile GPU-based deep learning framework for continuous vision applications,” in Proc. 15th Annu. Int. Conf. Mobile Syst., Appl., Serv., 2017, pp. 82–95.

[29] S. Han et al., “EIE: Efficient inference engine on compressed deep neural network,” ACM SIGARCH Comput. Archit. News, vol. 44, no. 3, pp. 243–254, 2016.

[30] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” in Proc. Int. Conf. Learn. Representations, 2016.

[31] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, “On-demand deep model compression for mobile devices: A usage-driven model selection framework,” (2018), in Proc. 16th Annu. Int. Conf. Mobile Syst., Appl., Serv., 2018, pp. 389–400.

[32] Z. Zhao, K. M. Barjough, and A. Gersflauer, “DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 37, no. 11, pp. 2348–2359, Nov. 2018.

[33] J.-H. Liao, J. Wu, and W. Lin, “ThiNet: A filter level pruning method for deep neural network compression,” in Proc. IEEE Int. Conf. Comput. Vis., 2017, pp. 5058–5066.

[34] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, “Distilled split deep neural networks for edge-assisted real-time systems,” in Proc. 2019 Workshop Hot Topics Video Anal. Intell. Edges, 2019, pp. 21–26.

[35] S. Teerapittayanon, B. McDaniel, and H.-T. Kung, “BranchyNet: Fast inference via early exiting from deep neural networks,” in Proc. IEEE 23rd Int. Conf. Pattern Recognit., 2016, pp. 2464–2469.

[36] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, “Multi-scale dense networks for resource efficient image classification,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 4310–4319.

[37] J. Kim, S. Hong, Y. Choi, and J. Kim, “Doubly nested network for resource-efficient inference,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 2740–2749.

[38] C. Zhang, M. Ren, and R. Uartsun, “Graph hypernetworks for neural architecture search,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 2548–2557.

[39] M. Phuong and C. H. Lampert, “Distillation-based training for multi-exit architectures,” in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2019, pp. 1355–1364.

[40] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Representations, 2015.

[41] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 2818–2826.

[42] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 1251–1258.

[43] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 770–778.

[44] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in Proc. AAAI Conf. Artif. Intell., 2017, pp. 4278–4284.

[45] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[46] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 8697–8710.

[47] C. Liu et al., “Progressive neural architecture search,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 19–34.

[48] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search,” in Proc. Int. Conf. Learn. Representations, 2019.

[49] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 6, pp. 1–23, Jun. 2020.

[50] M. Tan et al., “MnasNet: Platform-aware neural architecture search for mobile,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 2820–2828.

[51] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” IEEE Trans. Neural Networks Learn. Syst., vol. 31, no. 11, pp. 4749–4769, Nov. 2020.

[52] H. Zanddizari, N. Nguyen, B. Zeinali, and J. M. Chang, “A new preprocessing approach to improve the performance of CNN-based skin lesion classification,” Med. Biol. Eng. Comput., vol. 59, no. 5, pp. 1123–1131, 2021.

[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. 2009 IEEE Conf. Comput. Vis. Pattern Recognit., 2009, pp. 248–255.

[54] B. Farahani, F. Firouz, and K. Chakrabarty, “Healthcare IoT,” in Intelligent Internet of Things, B. V. K. Vijaya Kumar and M. A. B. Siddiqui, Eds. Springer, 2019, pp. 515–534.

[55] P. Tschandl, C. Rosenzlanti, and H. Kittler, “The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions,” Sci. Data, vol. 5, 2018, Art. 180161.

[56] M. Combalia et al., “BCN20000: Dermoscopic lesions in the wild,” J. Am. Acad. Dermatol., vol. 82, no. 5, pp. 1212–1215, May 2020.

Behnam Zeinali received the M.Sc. degree in electrical engineering from the Iran University of Science and Technology, Tehran, Iran, in 2013. He is currently working towards the Ph.D. degree with the University of South Florida, Tampa, FL, USA. From 2013 to 2019, he was with the industry as a Programmer, Researcher, and Developer in the field of AI. His research interests include machine and deep learning, computer vision, and mobile application programming.

Di Zhuang (Member, IEEE) received the B.E. degree in computer science and information security from Nankai University, Tianjin, China, and the Ph.D. degree in electrical engineering, from the University of South Florida, Tampa, FL, USA. His research interests include cyber security, social network science, privacy-enhancing technologies, machine learning, and deep learning.

J. Morris Chang (Senior Member, IEEE) He received the Ph.D. degree from North Carolina State University, Raleigh, NC, USA. His past industrial experiences include positions with Texas Instruments, Microelectronic Center of North Carolina, Durham, NC, USA, and AT&T Bell Labs. He is currently a Professor with the Department of Electrical Engineering, University of South Florida, Tampa, FL, USA. His research interests include cyber security and data privacy, machine learning, and mobile computing.

He was the recipient of the University Excellence in Teaching Award with the Illinois Institute of Technology, Chicago, IL, USA, in 1999. He was inducted into the NC State University ECE Alumni Hall of Fame in 2019. He is also a handling Editor of the Journal of Microprocessors and Microsystems and the Editor of IEEE IT Professional.