electronics

MDPI

Article

A Cloud-Edge-Smart IoT Architecture for Speeding Up the Deployment of Neural Network Models with Transfer Learning Techniques

Tz-Heng Hsu [1,*], Zhi-Hao Wang [2] and Aaron Raymond See [3]

[1] Department of Computer Science and Information Engineering, Southern Taiwan University of Science and Technology, Tainan 710301, Taiwan
[2] Department of Information Management, Southern Taiwan University of Science and Technology, Tainan 710301, Taiwan; zhwang@stust.edu.tw
[3] Department of Electrical Engineering, Southern Taiwan University of Science and Technology, Tainan 710301, Taiwan; aaronsee@stust.edu.tw
* Correspondence: hsuth@mail.stust.edu.tw; Tel.: +886-6-253-3131 (ext. 3226)

Abstract: Existing edge computing architectures do not support the updating of neural network models, nor are they optimized for storing, updating, and transmitting different neural network models to a large number of IoT devices. In this paper, a cloud-edge smart IoT architecture for speeding up the deployment of neural network models with transfer learning techniques is proposed. A new model deployment and update mechanism based on the share weight characteristic of transfer learning is proposed to address the model deployment issues associated with the significant number of IoT devices. The proposed mechanism compares the feature weight and parameter difference between the old and new models whenever a new model is trained. With the proposed mechanism, the neural network model can be updated on IoT devices with just a small quantity of data sent. Utilizing the proposed collaborative edge computing platform, we demonstrate a significant reduction in network bandwidth transmission and an improved deployment speed of neural network models. Subsequently, the service quality of smart IoT applications can be enhanced.

Keywords: deep learning; transfer learning; lightweight neural network; edge computing

Citation: Hsu, T.-H.; Wang, Z.-H.; See, A.R. A Cloud-Edge-Smart IoT Architecture for Speeding up the Deployment of Neural Network Models with Transfer Learning Techniques. *Electronics* **2022**, *11*, 2255. https://doi.org/10.3390/electronics11142255

Academic Editors: Antoni Morell and Juan-Carlos Cano

Received: 1 June 2022
Accepted: 15 July 2022
Published: 19 July 2022

1. Introduction

With the rapid development of mobile broadband network communications and artificial intelligence technology, smart video networking services are becoming more and more popular. In order to satisfy the needs of large amounts of data and low transmission delays for video networking devices, edge computing architectures have emerged. The trend of artificial intelligence computing is moving from cloud computing to edge computing that uses a decentralized architecture to shorten the delay of network transmission and accelerate the processing speed of real-time computation. Different computing tasks are processed hierarchically and computing is completed close to the data source or client side to shorten network transmission delays and quickly obtain data analysis results. Edge computing speeds up the response speed of application services through the concept of computing layering and provides a better user experience. To meet the requirements of artificial intelligence (AI) services, the trend of AI computing is moving from cloud platforms to distributed edge computing. Existing edge computing frameworks, such as fog computing, cloudlet, and mobile edge computing, still have many shortcomings [1].

The need to deploy machine learning models on edge devices is growing rapidly. Edge AI allows inference to be run locally without connecting to the cloud, reducing the data transmission time and making the inference process of neural network models more efficient. Many vendors offer AI edge computing deployment platforms for IoT devices,

e.g., Microsoft Azure IoT Edge [2], AWS Greengrass [3] Google's Cloud IoT Edge [4], and IBM Watson IoT Platform Edge [5]. Microsoft Azure IoT Edge is built upon Azure IoT Hub that offers a centralized method of managing Azure IoT edge devices and deploying the neural network models to the edge devices. Users that prefer to perform AI tasks at the edge rather than in the cloud can use this service, in which IoT devices can spend less time sending data to the cloud server by shifting AI tasks to the edge. The workflow of deploying machine learning (ML) models on the Azure IoT Edge platform [6] is as follows: first, create Azure resources; second, configure Edge device; third, build the ML model into the docker image; fourth, deploy the ML model to the IoT Edge; fifth, test the ML module; and finally, tear down resources. In the third step, the user develops an ML model, builds it into a docker image, and registers it into an Azure Container Registry (ACR). In the fourth step, the user deploys modules (running containers from docker images registered in ACR) to the IoT edge device. In the Microsoft Azure IoT Edge architecture, when a neural network model is retrained, the entire neural network model file must be repackaged into a new docker image and then deployed to IoT devices, which is time consuming and requires a large data size to encapsulate the neural network model into the docker image file.

Larger and more diverse data are being gathered as various sensors are placed in mobile phones, vehicles, and buildings. How the diversity and large size of sensor data integrates into artificial intelligence (AI) solutions is an interesting topic of research [7]. Hence, how to provide a better service experience for users has become an important issue for intelligent AI services.

Traditional machine learning methods include supervised learning, unsupervised learning, and semi-supervised learning. One of the problems with the main visual neural network model of supervised learning is that training a new model from scratch requires a lot of data. If there are not enough training data, the trained neural networks are often prone to overfitting. However, in many application environments, due to various restrictions, it is hard to collect enough training samples for the neural network to effectively converge. Transfer learning technology can solve the overfitting problem of insufficient image samples. It usually starts by initializing a new neural network that retrieves the trained neural layer and weight information in advance from the publicly available high-performance neural network models, and then performs local weight updates. The pre-trained neural network can provide the trained feature weights as an advanced feature extractor. Transfer learning can also be applied to the training of lightweight neural network models, which can be implemented in hardware accelerators; the last layer of the target domain neural network can be quickly retrained and inferred on the hardware of the networked devices [8,9].

Transfer learning is the process of using feature representations from a pre-trained model to avoid having to train a new model from the ground up. Pre-trained models are typically trained on large datasets, which are common benchmarks in the fields of computer vision and natural language processing [10]. Transfer learning has a number of advantages when it comes to the construction of machine learning models. The weights calculated by the pre-trained models can be used as part of the training process for a new model. Generalized information can be transferred across the two neural network models if they are designed to accomplish similar tasks. When pre-trained models are used in a new model, the required computing resources and training time can be reduced [11]. Meanwhile, in the case of only having a tiny training dataset, transfer learning technologies can utilize the weights from the pre-trained models to train the new model and solve new challenges. Transfer learning has been effectively used in a variety of machine learning applications, including text sentiment classification [12] and image classification [13,14].

The main purpose of this paper is to develop a collaborative edge computing platform that aims to solve the problems of (1) deploying large numbers of neural network models to IoT devices and (2) reducing the computing resource requirements of cloud servers. With the advance of deep learning algorithms, the file size of the trained model is growing larger and more parameterized in order to increase the prediction accuracy, which creates a deployment issue. When a large number of edge AI computing devices update the new

models, the amount of data transmission required is considerable. For neural network models trained with transfer learning technologies, these models will share some of the same weights and parameters. Based on this characteristic, we propose a new model deployment and updating mechanism for solving the problems of deploying large numbers of neural network models to IoT devices. Whenever a new model is trained, the proposed mechanism will compare the feature weight and parameter difference between the old and new models. After the new neural network model is trained, the feature weights are encapsulated into a neural network model patch file after processing through a difference comparison (diff). By dispatching the neural network model patch file to all edge IoT devices for new model updating, only a small amount of data are required to be transmitted to the edge device, and the entire neural network model can be updated. As a result, the amount of data transmission can be significantly decreased for updating edge devices that already have the same shared pre-trained models. A collaborative model publish system is also proposed to assist the delivery and update of neural network models, reducing backbone bandwidth requirements. In order to evaluate the performance of the proposed model deployment and update mechanism, two different neural network models are used to evaluate the performance of the proposed system. The experimental results show that the proposed deployment and update mechanism can significantly lower the bandwidth requirement in model data transmission volume. Meanwhile, the computing resource requirements of cloud servers can be reduced with the proposed collaborative edge computing platform.

2. Related Works

Commercial cloud-based deep learning systems, such as Amazon Rekognition, usually require users to upload their personal data to a remote server for high-quality inference processing. However, uploading all data to a remote location may result in massive data transfers [15] and potential privacy risks [16]. Edge computing is a distributed computing architecture that moves the operations of applications, data, and services from the central node of the network to the edge nodes on the network logic for processing. Edge computing includes the following elements: (1) Proximity is in the Edge: Communication between edge nodes is faster and more efficient than communication with remote servers. (2) Intelligence is in the Edge: With the continuous enhancement of the computing power of sensors and smart video networking devices, edge nodes can make autonomous decisions and make immediate responses to the sensed data. (3) Trust is in the Edge: Much sensitive data are usually stored in personal devices (edge nodes). Therefore, trust relationships and sensitive data must also be managed manually at the Edge. (4) Control is in the Edge: Computing, data synchronization, or storage can be selectively allocated or delegated to other nodes or cores, and controlled by edge devices. (5) Humans are in the Edge: Human-centered design should place humans in a control loop, allowing users to control their data. Edge computing decomposes large-scale services that were originally handled by central nodes, cuts them into smaller and easier-to-manage parts, and distributes them to edge nodes for processing. As the edge node is closer to the user terminal device, it can speed up the processing and transmission of data and reduce transmission delay [17].

There are three main types of edge computing implementation architecture, which can be classified into fog computing, cloudlets, and mobile edge computing (MEC) [18]. Fog computing is a decentralized computing architecture based on fog computing nodes. Fog nodes consist of multiple elements, including routers, switches, network access points, IoT gateways, and set-top boxes. These nodes can be deployed anywhere between network architectures. Because these nodes are close to the edge of the network, fog computing can provide good real-time transmission quality. The abstract layer of fog computing can mask the differences between the devices, unify the resources of the devices, and form a resource pool that can be used by the upper layer. The edge network is very close to the end user, and the sensing data will be calculated directly at the fog computing nodes without uploading to the cloud computing center. Fog computing makes full use of a large number

of smart devices located at the edge of the network. Although the computing resources of individual devices are limited, a large number of devices can play a significant role in a centralized manner. Fog computing can use various heterogeneous networks, such as wired, wireless, and mobile networks, to connect different network devices, which can solve the network delay problem of cloud computing [19]. Scholars such as Beck introduced the classification and architecture topology of Mobile Edge Computing [20], as well as emerging technologies and applications of Edge content delivery and aggregation.

Many mobile devices now incorporate a large number of artificial intelligence application services, such as face recognition and voice translation. However, these mobile devices have very limited resources, such as limited battery power, network bandwidth, and computing storage capacity. Most artificial intelligence application services upload data to a cloud computing server, use the powerful computing and storage capabilities of the cloud computing center to process and store the data, and then send the computing results back to the mobile devices. However, with the development of the Internet of Things and the rapid growth of mobile devices, the transmission of large amounts of data to the cloud computing center can easily cause backbone network congestion, and network latency has become the bottleneck of cloud computing. Therefore, some scholars have proposed the concept of cloudlets. Cloudlets are resource-rich hosts or clusters of hosts. They are placed at the edge of the network and placed on the network closest to the mobile devices [21]. In this way, data can be sent to cloudlets for processing and return the results. At the same time, calculations that cannot be completed by the cloud nodes can be transferred to the cloud computing center for processing by the cloud server. Cloudlets have the following characteristics: (1) Proximity: cloudlets are very close to mobile devices and can be reached through one hop of the network. (2) Resource-Rich: compared with fog computing nodes, cloudlets are specially deployed computing or data storage nodes, and their computing and data storage capabilities are much higher than those of fog computing [21].

Edge computing architectures such as fog computing, cloudlets, and MEC still have much room for improvement in the implementation of artificial intelligence application services. The traditional edge computing architecture and collaborative algorithms do not support the training update of deep neural network models and the pre-trained network models; therefore, how to provide an efficient edge server that supports various neural network model updates has become a major issue for the successful deployment of large-scale artificial intelligence application services.

3. System Architecture

Figure 1 shows the proposed Cloud-Edge Smart IoT architecture and model publish system for speeding up the deployment of neural network models with transfer learning techniques. The proposed cloud AI server uses transfer learning technology to retrieve the pre-trained neural layers and the weight information and then updates the local weights of the newly trained neural network model. A model publish system is developed to help the neural network model delivery. After the new feature weights have been trained, the cloud AI server uses the proposed model publish system to encapsulate the new feature weights into a neural network model patch file, i.e., the diff part. Then, the model patch file is transmitted to the edge servers. When an edge server receives the new model patch file, the edge server stores the patch file into its cache storage space and then notifies its nearby smart edge IoT devices for neural network model updating. When the nearby smart edge IoT devices receive the notification, the IoT devices then send model update requests to download the new model patch file and update their local neural network models if the old models exist. In this way, it is no longer necessary to transmit big-data-volume neural network model files; the cloud server only needs to transmit small-data-size neural network model patch files to smart edge IoT devices with the help of edge servers. When an edge IoT device receives the patch file, the weights can be updated in combination with the existing models within the edge IoT device. With the help of the proposed collaborative edge computing platform, network bandwidth transmission requirements can be greatly

reduced, and the deployment speed of neural network models can be improved. This section describes the proposed model publish system for speeding up the deployment of neural network models with transfer learning techniques.

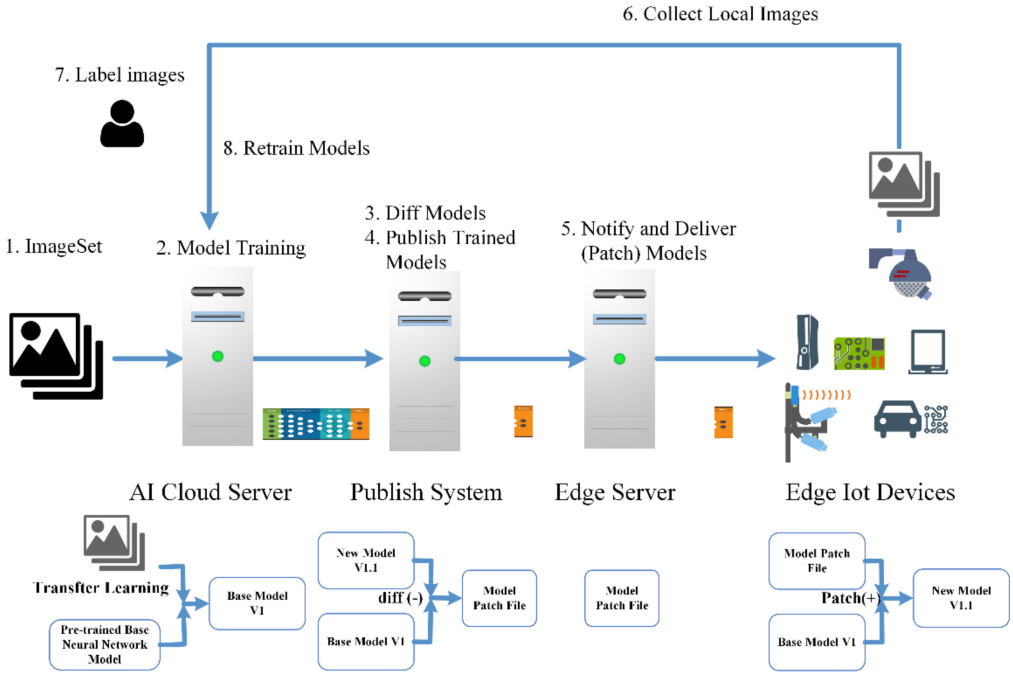


Figure 1. The proposed Cloud-Edge Smart IoT architecture and model publish system for speeding up the deployment of neural network models with transfer learning techniques.

3.1. The Model Publish System for Speeding Up the Deployment of Neural Network Models with Transfer Learning Techniques

Figure 2 illustrates the training mechanisms of deep neural networks (DNNs). One or more Hidden Layers (HLs) and Output Layer (Ols) are trained with the input dataset. Figure 2a represents the traditional machine learning (ML) mechanism, and various datasets are used to independently train separate models on given problems. Every trained model has a separate set of parameters, and no knowledge that can be shared between models. Subsequently, the entire neural network model must be retrained each time new data are collected, which takes a lot of training time and computing resources. Figure 2b represents the transfer learning mechanism; users can train new models using existing knowledge (features, weights, etc.) from pre-trained models and even solve problems such as having less data for the newer task. The most significant advantages of transfer learning are resource savings and increased efficiency while training new models [22]. Meanwhile, some weights and parameters could be shared by neural network models trained with transfer learning technologies. Based on this feature, we propose a new model deployment and updating mechanism to address the issues associated with deploying large numbers of neural network models to IoT devices.

In the proposed system architecture, once new data are collected and a neural network model needs to update its weights, the cloud AI servers use the same basic neural network as that on the edge IoT devices to perform weight update training through the transfer learning technologies to generate a new neural network model. After the new neural network model is trained, the feature weights that have been preprocessed and trained are encapsulated into a neural network model patch file after processing through a difference comparison (diff) with the help of the proposed model publish system. The model publish system will use the existing neural network model's weights to dispatch the neural network model patch file to all edge IoT devices for model updating through the help of edge servers.

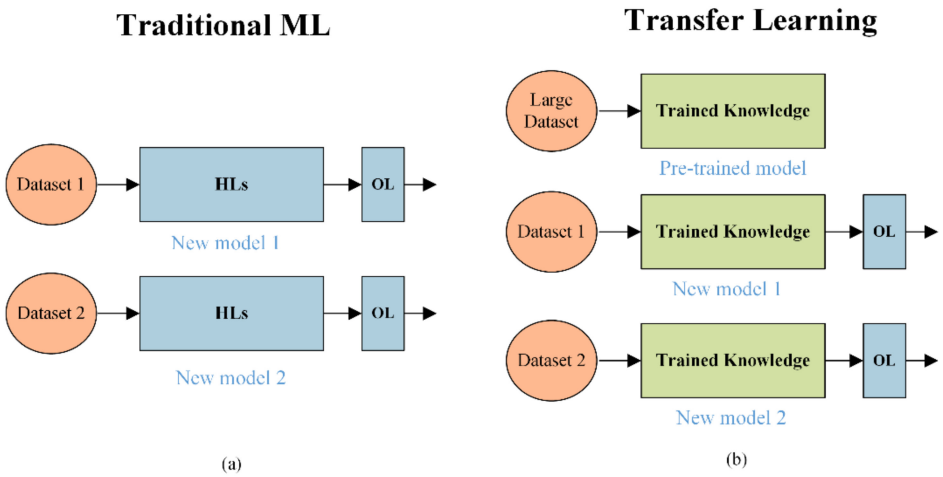


Figure 2. The training mechanisms of deep neural networks (DNNs): (**a**) traditional machine learning (ML) and (**b**) transfer learning.

Figure 3 shows the workflow chart of the proposed model diff/patch mechanism. When new data are collected, the cloud server trains the new neural network model with system configurations. If a model is suitable for use with transfer learning technologies, then the model is trained with a configured pre-trained model. Otherwise, the model is trained from scratch. Once a new model has been trained, the system checks to see if the previous model already exists for the same AI task. The system employs the diff tool to identify the differences between the old and the newly trained models if an old model is present. Then, a model patch file is created to contain all of the model's different pieces. The model patch file is delivered to IoT devices through the help of edge servers. If no old model exists in the cloud storage system, the new trained model file is delivered. Finally, all IoT devices update the existing old model with the new model patch file or have a whole new model file and perform the new AI task.

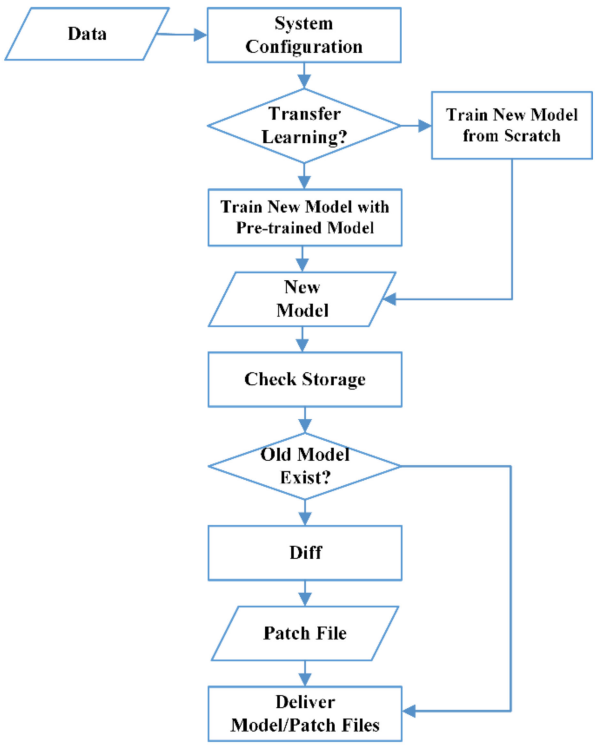


Figure 3. The workflow chart of the proposed model diff/patch mechanism.

3.2. The Cloud-Edge Smart IoT Architecture

This section explains the system model used in the proposed cloud-edge-Smart IoT architecture. Assume that there are N edge servers and each edge server has M child Smart IoT nodes in the cloud-edge Smart IoT network. Figure 4 shows the timing diagram of the proposed cloud-edge Smart IoT architecture, where TR_{upload} represents the time required by a user to upload the labeled data to the cloud AI server; TR_{train_model} represents the model training time required by the cloud AI server; T_{tm} represents the start time of model training on the cloud AI server; T_{dm} denotes the start time of the *diff* old model on the cloud AI server if an old model exists; TR_{diff_model} denotes the model *diff* time processed at the cloud AI server. Assume that an edge server's average available bandwidth for downloading a model from the cloud server is denoted as BW_{cloud_edge} and the average model size is denoted as $Model_{size}$. T_{ce} denotes the start time of dispatching a new model to an edge server. Let TR_{cloud_edge} denote the transmission time of dispatching a new model to the edge server, which is equal to:

$$TR_{cloud\_edge} = TR_{notify\_edge} + TR_{request\_edge} + \frac{Model_{size}}{BW_{cloud\_edge}} \quad (1)$$

where TR_{notify_edge} denotes that the cloud server notifies an edge server that a new model is trained completely, and $TR_{request\_edge}$ denotes that the edge server sends a new model update request to the cloud AI server. Assume that a Smart IoT node's average available bandwidth for downloading the model from the edge server is denoted as BW_{edge_IoT}, and T_{ei} denotes the start time of dispatching a new model from the edge server to a Smart IoT node. Let TR_{edge_IoT} denote the transmission time of dispatching a new model from the edge server to a Smart IoT node, which is equal to:

$$TR_{edge\_IoT} = TR_{notify\_IoT} + TR_{request\_IoT} + \frac{Model_{size}}{BW_{edge\_IoT}} \quad (2)$$

where TR_{notify_IoT} denotes that the edge server notifies a Smart IoT node that a new model is trained completely, and $TR_{request\_IoT}$ denotes that the Smart IoT node sends the new model update request to the corresponding edge server. T_{pi} denotes the start time of patching the new model in the Smart IoT node. The total model patch time in the Smart IoT node is denoted as TR_{patch_IoT}. In the proposed Cloud-Edge-Smart IoT architecture, the total time TR_{total} for updating a new model to N*M smart edge IoT devices is equal to:

$$TR_{total} = TR_{upload} + TR_{train\_model} + TR_{diff\_model} + N * TR_{cloud\_edge} + N * M * \left(TR_{edge\_IoT} + TR_{patch\_IoT}\right) \quad (3)$$

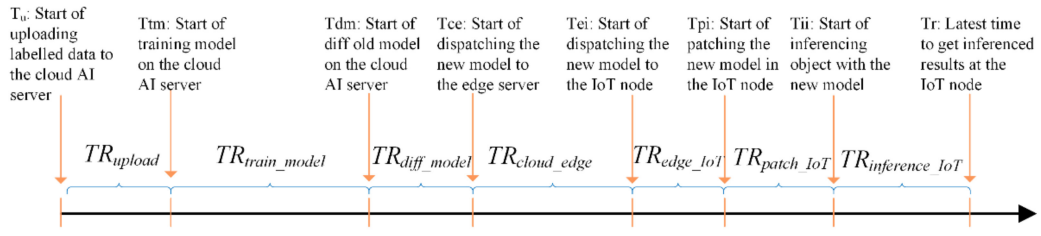


Figure 4. The timing diagram of the Cloud-Edge-SmartIot Architecture.

After the new model is patched completely, the edge IoT device can start the AI inference tasks. Let T_{ii} denote the start time of the inferencing object with the new patched model in the Smart IoT node. The total inferencing time in the Smart IoT node is denoted as $TR_{inference\_IoT}$; T_r denotes the latest time to obtain the inferenced result at the Smart IoT node.

4. System Evaluations

In the proposed collaborative edge computing platform, we use the Django framework [23] to implement the neural network model publish server and use the Celery framework [24] to perform the model *diff* task in the background; the Redis broker [25] server is used to store the Celery tasks. In Celery, task queues are used to distribute work across threads and machines. Dedicated worker processes constantly monitor task queues for new work to perform. Celery usually uses a broker, i.e., Redis broker, to help clients and workers talk to each other through messages. A Celery system can have multiple workers and brokers, allowing for high availability and horizontal scaling.

Using a publish/subscribe model, the MQTT protocol [26] provides a lightweight method of messaging. The MQTT protocol is appropriate for Internet of Things messaging, such as with low-power sensors and mobile devices. In the proposed collaborative edge computing platform, when a neural network model *diff* task is completed, the Mosquito MQTT broker [27] server is used to notify the edge servers to download and update the existing model. After the edge server downloads the model, it will notify the edge IoT devices to update the local neural network model through the Mosquito MQTT broker server. Each IoT device in the proposed platform comes with a pre-installed client system, which will automatically monitor the MQTT messages, update the AI model, and hand off control to the AI task for making inferences after the model is updated. Figure 5 shows the detailed workflow of the neural network model publish process in the proposed collaborative edge computing platform.

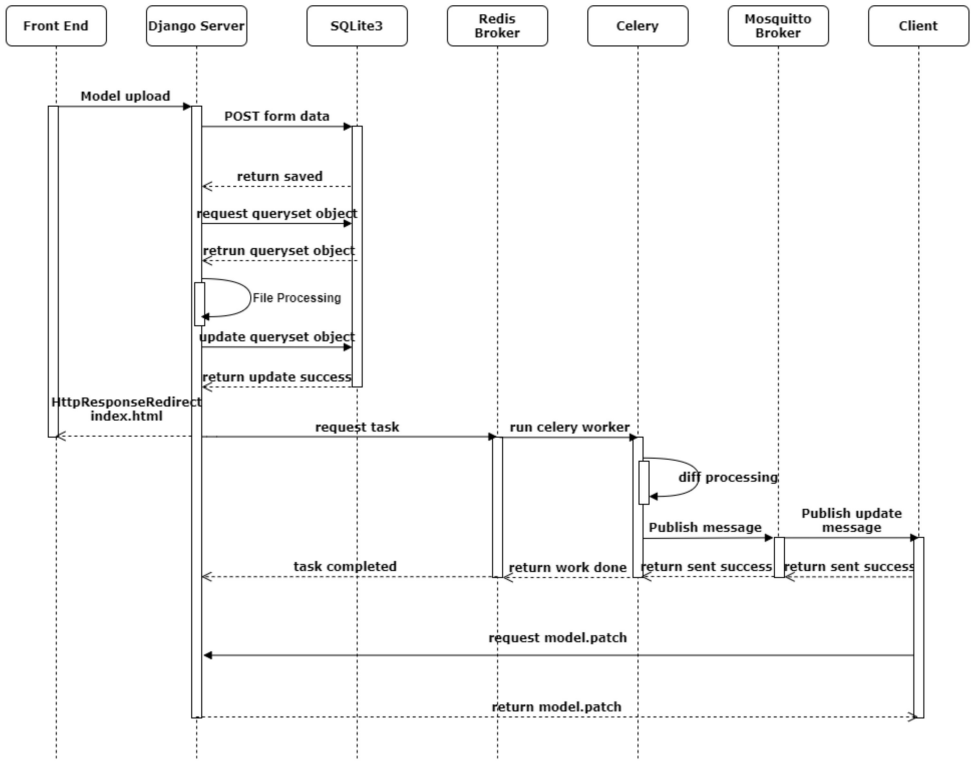
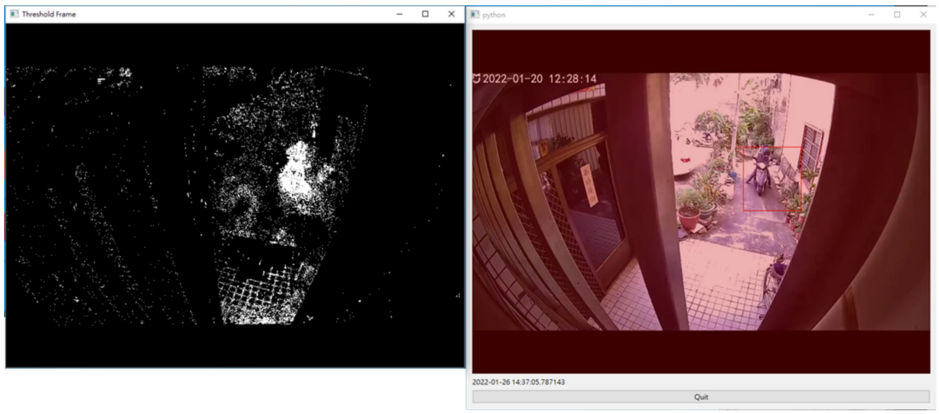


Figure 5. The detailed workflow of the neural network model publish process in the proposed collaborative edge computing platform.

In the experiment, a model publish server used in the experiment utilized an Intel(R) Core (TM) i9-9900K CPU @ 3.60 GHz, 64 GB RAM, a GeForce RTX 2080 Ti video card, a 512 GB SSD, and the operating system of Ubuntu 20.04.3 LTS. The experiment uses the Google Colab (Colaboratory) deep learning platform for model development with TensorFlow 2.5 and the Keras framework. The basic neural network test models used are MobileNet v2 [28] and VGG-19. MobileNet v2 enhances mobile model performance across

many benchmarks and model sizes, which is built on an inverted residual structure. In the intermediate expansion layer, MobileNet v2 filters features using lightweight depthwise convolutions [28]. The VGG-19 is a deep learning convolutional neural network (CNN) architecture for image classification, with 16 convolutional layers and 3 fully connected layers [29]. Common convolutional neural network models, such as MobileNet v2 and VGG-19, are frequently utilized as pre-training network models for various transfer learning tasks. Consequently, these two network pre-training models are employed to implement image classification tasks in the experiment with the dataset of the custom six persons' dataset captured by a home camera, as shown in Figure 6.



(**a**) (**b**)

Figure 6. The custom 6 persons' dataset captured by a home camera: (**a**) frame difference result and (**b**) motion detection result.

Figure 7 illustrates the structure and parameters of the neural network test models based on MobileNet v2 and VGG-19 models. The pre-trained model is reused by simply removing the final layer and using it to classify new image categories. Layers are added to a model that has already been trained and they are frozen to prevent losing the information they contain during future training processes. On top of the frozen layers, trainable layers are added. In the test model based on the MobileNet v2 pre-trained model, the total params are 2,626,854; the trainable params are 368,870 and the nontrainable params are 2,257,984. In the test model based on the VGG-19 pre-trained model, the total params are 20,172,070; the trainable params are 147,686 and the nontrainable params are 20,024,384. The trained neural network model is saved with the TensorFlow model file format, which includes variable index (variables.index), variable value (variables.data-00000-of-00001), and GraphDef (*.pb) files. The GraphDef (*.pb) format contains serialized data and calculation graphs of protobuf objects. The operating system of the computer equipment used by the client is ubuntu, and the file difference comparison (*diff*) and neural network model patch file merge test are performed. In the experiment, the difference comparison tool used is hdiffz, and the correction tool used in the neural network model patch file is hpatchz [30]. The experiment explores the original model size of the neural network model, the size of the neural network model patch file after the difference comparison, and the data saving ratio.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 7, 7, 512)         20,024,384
_________________________________________________________________
conv2d (Conv2D)              (None, 5, 5, 32)          147,488
_________________________________________________________________
dropout (Dropout)            (None, 5, 5, 32)          0
_________________________________________________________________
global_average_pooling2d (Gl (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 6)                 198
=================================================================
Total params: 20,172,070
Trainable params: 147,686
Non-trainable params: 20,024,384
```

(**a**)

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2,257,984
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 5, 32)          368,672
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 32)          0
_________________________________________________________________
global_average_pooling2d_2 ( (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 198
=================================================================
Total params: 2,626,854
Trainable params: 368,870
Non-trainable params: 2,257,984
```

(**b**)

Figure 7. The structure and parameters of the neural network test models based on MobileNet v2 and VGG-19 models. (**a**) The test model based on MobileNet v2 model. (**b**) The test model based on VGG-19 model.

The experiment needs to upgrade the related files of the neural network model, variable index (variables.index), variable value (variables.data-00000-of-00001), and GraphDef (*.pb) files. After the difference comparison of model files (*diff*), the model publish server generates neural network model-related patch files. Then, the neural network model-related patch files are packaged and transmitted to edge servers and edge IoT devices. After the new neural network model is updated on the edge IoT devices, the new neural network model can be loaded for inference and prediction. The experimental results are shown in Table 1.

Table 1 shows the relevant file sizes of the MobileNet v2 and VGG-19 neural network models after transfer learning with the six categories of person images. Among them, the neural network-related files after diff comparison are based on the MobileNet v2 neural network. With the transfer learning techniques, the models share the same weights of the basic neural network model, which can reduce the transmission data after differential comparison. The VGG-19 neural network model has up to 19 layers, and its basic neural network model variable value reaches 80,109,019 bytes. After the differential comparison, only 1,778,387 bytes need to be transmitted, which greatly reduces the need to transmit the total amount of model data. The new neural network model with MobileNet v2 transfer learning can save up to 72.93% of data, while the new neural network model with VGG-19 transfer learning can save data as much as 97.76%.

Table 1. The experimental results of image classification with diff and patch.

Image Classification (6 Persons)	
Number of images in dataset	289
Number of dataset categories	6
Training set/test set ratio	8:2
Basic Neural Network Model-mobilenetv2	
Basic neural network model pb model size	3,217,201
Basic neural network model variable index size	16,627
Basic neural network model variable value size	9,125,379
Model name-mobilenetv2_6persons (based on mobilenetv2)	
Model PB model size after transfer learning	4,350,337
Model variable index size after transfer learning	15,979
Model variable value size after transfer learning	15,979
Neural network pb model size after difference comparison	380,385
Neural network variable index size after difference comparison	4306
Neural network variable value size after difference comparison	4,464,323
The ratio of data saved after difference comparison	0.7293753813
The diff processing time	1.376 s
The patch processing time	0.004 s
Basic Neural Network Model-vgg19_base	
Basic neural network model pb model size	353,484
Basic neural network model variable index size	2285
Basic neural network model variable value	80,109,019
Model name-vgg19_6persons (based on vgg19_base)	
Model PB model size after transfer learning	537,634
Model variable index size after transfer learning	3434
Model variable value size after transfer learning	81,885,560
Neural network pb model size after difference comparison	64,741
Neural network variable index size after difference comparison	1243
Neural network variable value size after difference comparison	1,778,387
The ratio of data saved after difference comparison	0.9776240877
The diff processing time	8.439 s
The patch processing time	0.013 s

The generated neural network model-related files, i.e., variable index (variables.index), variable value (variables.data-00000-of-00001), and GraphDef (*.pb) files, are all compared with the original file with the SHA256 checksum, which verifies whether it is consistent or not after the patch update. The experimental verification monitors the checksum codes of the neural network model-related files, which are all correct, indicating that all the neural network model-related files are successfully deployed and updated. The time required for comparing (diff) the different file part among the new model with the six categories of person images and the original MobileNet v2 model is about 1.376 s and the patch time required for recovering (patch) the new model with the MobileNet v2 patch file is about 0.004 s in the experiment machine. The time required for comparing (diff) the difference among the new model with the six categories of person images and the original VGG-19 model is about 8.439 s and the patch time required for recovering (patch) the new model with the VGG-19 patch file is about 0.013 s in the experiment machine.

For investigating the system performance, the edge IoT node number is changed from 100 to 1000 for evaluating the simulation performance. In the simulation environment setup, the number of edge servers is set to 4; the available download bandwidth from a cloud server to edge server is set to 100 Mbps; the available download bandwidth from an edge server to an edge IoT is also set to 100 Mbps. The cloud to edge server's propagation delay is set to 300 ms and the edge server to edge IoT node's propagation delay is set to 30 ms. Figure 8 shows the simulation results. Label 'vgg_cloud' means that for a cloud

server to transmit the VGG-19 model to 1000 edge IoT devices, it needs about 13,237 s. For the proposed diff model publish system, it only needs about 225 s to deploy new VGG-19 models to 1,000 edge IoT devices. For MobileNet v2, a similar result can be observed. The simulation results show that the proposed collaborative edge computing platform can speed up the deployment of artificial intelligence services.

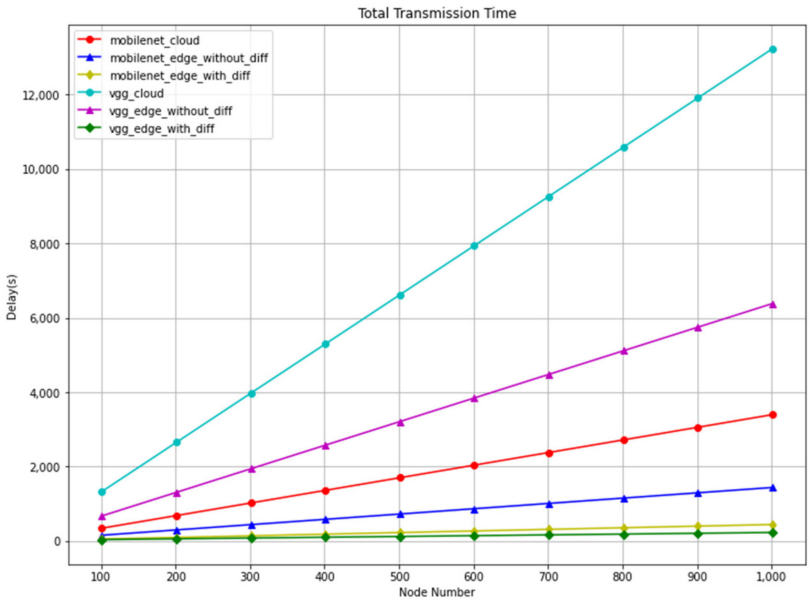


Figure 8. The simulation results of the proposed collaborative edge computing platform.

5. Discussion and Conclusions

Edge AI makes it possible to execute AI inference tasks locally without connecting to the cloud. Machine learning models must be quickly deployed on edge devices. Many cloud service providers provide AI edge computing deployment platforms for IoT devices, including Microsoft Azure IoT Edge [6], AWS Greengrass [3], Google's Cloud IoT Edge [4], and IBM Watson IoT Platform Edge [5].

When a neural network model is trained in the Microsoft Azure IoT Edge architecture, the complete neural model file must be repackaged into a new docker image before being deployed to IoT devices, which takes time and requires a sizable amount of data [6]. Machine learning models that are created, trained, and optimized in the cloud and run inference locally on devices are part of AWS IoT Greengrass [3]. Data gathered from the IoT devices can be sent back to AWS SageMaker where they can be used to continuously improve the quality of machine learning models [31]. AWS IoT Greengrass is capable of deploying components to IoT devices. Each IoT device operates on a combination of the software from the deployments that target the devices. Deployments to the same target device, however, overwrite any previous deployment components there. When a deployment for a target device is updated, the outdated components are swapped out for the updated components [32].

Google IoT Core contains two modules: (a) Device Manager allows users to set-up, authenticate, configure, and control IoT devices remotely; (b) Protocol Bridge operating with MQTT and HTTP protocols is in charge of service connectivity. Cloud Pub/Sub data are redirected to Google cloud services. For edge computing and AI in the Google IoT ecosystem, Google's Cloud IoT Edge is performed via its branded Edge TPU chip [33]. With the Coral platform for ML at the edge, Google's Cloud TPU and Cloud IoT are enhanced to offer an end-to-end (cloud-to-edge, hardware + software) infrastructure that makes it easier for clients to develop AI-based solutions. The Coral platform offers a full developer toolkit in addition to its open-source TensorFlow Lite programming environment, allowing users to create models or retrain a number of Google AI models for the Edge TPU, combining

Google's experience in both AI and hardware. However, the deployment method is not extensively defined [34].

IBM Edge Application Manager (IEAM) is designed specifically for edge node management to minimize deployment risks and to manage the service software lifecycle on edge nodes fully autonomously. Software developers develop and publish edge services to the management hub. Administrators define the deployment policies that control where edge services are deployed. IEAM publishes an existing Docker image as an edge service, creates an associated deployment pattern, and registers IoT edge nodes to run that deployment pattern. Similar to Microsoft Azure IoT Edge [6], the complete neural model file must be packaged into a new docker image before being deployed to IoT devices, which takes time and requires a sizable amount of data [35].

Existing edge IoT deployment platforms do not take into account the issue of sharing pre-trained models when deploying trained models, which leads to significant data consumption. Our work is different from the above system, and the proposed Cloud-Edge Smart IoT architecture and model publish system aims to solve the problems of deploying large numbers of neural network models to IoT devices and reducing the computing resource requirements of cloud servers. Only a small amount of data are required to be transmitted to all edge IoT devices for new model updating, and the entire neural network model can be updated. Existing edge computing architectures do not support updating the weights of various neural network models, nor are they optimized for storage, updating, and transmission of various neural network models for a large number of edge IoT devices. The proposed collaborative edge computing platform uses pre-trained neural layers and weight information to train and update the neural network model. Through differential comparison, the edge server only needs to send a small-part neural network model patch file to complete the deployment of new neural network models on IoT devices. The experimental results show that the proposed neural network model weight comparison mechanism can speed up the deployment of artificial intelligence services.

Author Contributions: Conceptualization, T.-H.H.; software, T.-H.H.; data curation, T.-H.H.; methodology, T.-H.H. and Z.-H.W.; validation, T.-H.H., Z.-H.W. and A.R.S.; writing—original draft, T.-H.H.; writing—review and editing, A.R.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Technology (MOST), Taiwan, Grant Nos. MOST 110-2221-E-218-018 and MOST 109-2221-E-218-019.

Acknowledgments: The authors wish to express their gratitude to the Ministry of Science and Technology (MOST) for support this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ai, Y.; Peng, M.; Zhang, K. Edge computing technologies for Internet of Things: A primer. *Digit. Commun. Netw.* **2018**, *4*, 77–86. [CrossRef]
2. IoT Edge | Cloud Intelligence | Microsoft Azure. Available online: https://azure.microsoft.com/en-us/services/iot-edge/ (accessed on 22 June 2022).
3. AWS IoT Greengrass Documentation. Available online: https://docs.aws.amazon.com/greengrass/index.html (accessed on 22 June 2022).
4. Google Cloud IoT-Fully Managed IoT Services. Available online: https://cloud.google.com/solutions/iot (accessed on 22 June 2022).
5. IBM Watson IoT Platform. Available online: https://internetofthings.ibmcloud.com/internetofthings.ibmcloud.com (accessed on 22 June 2022).
6. Zhang, Y. Deploy Machine Learning Models on Azure IoT Edge. Available online: https://github.com/microsoft/deploy-MLmodels-on-iotedge/commits?author=YanZhangADS (accessed on 22 June 2022).
7. Weiss, K.; Khoshgoftaar, T.M.; Wang, D. A Survey of Transfer Learning. *J. Big Data* **2016**, *3*, 9. [CrossRef]
8. Leroux, S.; Bohez, S.; Verbelen, T.; Vankeirsbilck, B.; Simoens, P.; Dhoedt, B. Transfer Learning with Binary Neural Networks. *arXiv* **2017**, arXiv:1711.10761.
9. Pan, S.J.; Yang, Q. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 1345–1359. [CrossRef]
10. Transfer Learning for Machine Learning. Available online: https://www.seldon.io/transfer-learning (accessed on 21 June 2022).

11. Transfer Learning Guide: A Practical Tutorial with Examples for Images and Text in Keras. Available online: https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras (accessed on 21 June 2022).
12. Wang, C.; Mahadevan, S. Heterogeneous Domain Adaptation Using Manifold Alignment. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Spain, 16 July 2011; AAAI Press: Palo Alto, CA, USA, 2011; Volume 2, pp. 1541–1546.
13. Li, W.; Duan, L.; Xu, D.; Tsang, I.W. Learning with Augmented Features for Supervised and Semi-Supervised Heterogeneous Domain Adaptation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2014**, *36*, 1134–1148. [CrossRef] [PubMed]
14. Zhu, Y.; Chen, Y.; Lu, Z.; Pan, S.J.; Xue, G.-R.; Yu, Y.; Yang, Q. Heterogeneous Transfer Learning for Image Classification. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 7 August 2011; AAAI Press: Palo Alto, CA, USA, 2011; pp. 1304–1309.
15. Shi, S.; Wang, Q.; Xu, P.; Chu, X. Benchmarking State-of-The-Art Deep Learning Software Tools. In Proceedings of the 2016 7th International Conference on Cloud Computing and Big Data (CCBD), Macau, China, 16–18 November 2016.
16. Zhang, Q.; Yang, L.T.; Chen, Z.; Li, P.; Deen, M.J. Privacy-Preserving Double-Projection Deep Computation Model with Crowdsourcing on Cloud For Big Data Feature Learning. *IEEE Internet Things J.* **2017**, *5*, 2896–2903. [CrossRef]
17. Garcia Lopez, P.; Montresor, A.; Epema, D.; Datta, A.; Higashino, T.; Iamnitchi, A.; Barcellos, M.; Felber, P.; Riviere, E. Edge-centric Computing: Vision and Challenges. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 37–42. [CrossRef]
18. Dolui, K.; Datta, S.K. Comparison of Edge Computing Implementations: Fog Computing, Cloudlet and Mobile Edge Computing. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017.
19. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and its Role in the Internet of Things. In Proceedings of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012.
20. Beck, M.T.; Werner, M.; Feld, S.; Schimper, S. Mobile Edge Computing: A Taxonomy. In Proceedings of the 6th International Conference on Advances in Future Internet, Lisbon, Portugal, 16–20 November 2014.
21. Satyanarayanan, M.; Bahl, P.; Caceres, R.; Davies, N. The Case for Vm-Based Cloudlets in Mobile Computing. *IEEE Pervasive Comput.* **2009**, *8*, 14–23. [CrossRef]
22. Sarkar, D. (DJ) A Comprehensive Hands-On Guide to Transfer Learning with Real-World Applications in Deep Learning. Available online: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a (accessed on 22 June 2022).
23. Django. Available online: https://djangoproject.com (accessed on 5 December 2020).
24. Celery-Distributed Task Queue. Available online: https://docs.celeryproject.org/en/stable/#celery-distributed-task-queue (accessed on 8 November 2020).
25. Redis. Available online: https://redis.io/ (accessed on 9 March 2021).
26. MQTT Version 3.1.1. Available online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (accessed on 4 July 2022).
27. Eclipse Mosquitto. Available online: https://mosquitto.org/ (accessed on 10 March 2021).
28. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
29. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015.
30. HDiffPatch. Available online: https://github.com/sisong/HDiffPatch (accessed on 10 May 2021).
31. AWS Greengrass Machine Learning Inference-Amazon Web Services. Available online: https://aws.amazon.com/greengrass/ml/ (accessed on 25 June 2022).
32. Deploy AWS IoT Greengrass Components to Devices-AWS IoT Greengrass. Available online: https://docs.aws.amazon.com/greengrass/v2/developerguide/manage-deployments.html (accessed on 25 June 2022).
33. Making Sense of IoT Platforms: AWS vs. Azure vs. Google vs. IBM vs. Cisco. AltexSoft. Available online: https://www.altexsoft.com/blog/iot-platforms/ (accessed on 25 June 2022).
34. Edge TPU-Run Inference at the Edge. Available online: https://cloud.google.com/edge-tpu (accessed on 25 June 2022).
35. Transform Image to Edge Service. Available online: https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/eam/4.3?topic=SSFKVV_4.3/OH/docs/developing/transform_image.html (accessed on 25 June 2022).