Design and Implementation of Distributed Image Recognition App with Federal Learning Techniques

Yu-Wei Chan[1], Bo-You Wu[2], Yi-Ming Huang[2], and Chao-Tung Yang[2,3](✉)

[1] Department of Information Management, Providence University, Taichung, Taiwan
ywchan@gm.pu.edu.tw

[2] Department of Computer Science, Tunghai University, No. 1727, Sec. 4, Taiwan Boulevard, Taichung 407224, Taiwan, ROC
{s07351035,s07351048,ctyang}@thu.edu.tw

[3] Research Center for Smart Sustainable Circular Economy, Tunghai University, No. 1727, Sec. 4, Taiwan Boulevard, Taichung 407224, Taiwan, ROC

Abstract. In recent years, machine learning technology has been widely used in many fields, such as smart transportation, smart healthcare, smart finance and smart cities. Although machine learning technology has brought people a lot of convenience, the privacy problem of user data has also emerged [1]. Considering that users are not necessarily willing to upload personal privacy data to the cloud for deep learning training, therefore, instead of consuming a lot of bandwidth to upload data to the cloud, it is better to train on the local device and then use the model parameters obtained after training. (For example: weights and bias, etc.) upload to the server for aggregation. This emerging machine learning technology is called federated learning. In this way, the privacy and security of data can be guaranteed, and the purpose of decentralized learning can be achieved through aggregation. This study uses the architecture of federated learning technology and convolutional neural network algorithms to implement distributed image recognition mobile applications. This application allows users to use their mobile devices and the central servers for repeated training. After multiple rounds of repeated training, the convergence will be stabilized, and the accuracy will be significantly improved. At the same time, it can take into account privacy and achieve the machine the purpose of learning.

Keywords: Federated learning · Convolutional neural network · Image recognition · Decentralized deep learning · Machine learning

1 Introduction

In recent years, with the emerging development of artificial intelligence (AI) technology, deep learning (DL) technology has become a popular branch in the field of machine learning [2]. Deep learning technology has been successfully used in various fields, such

Y.-B. Lin et al. (Eds.): SGIoT 2021, LNICST 447, pp. 98–110, 2022.
https://doi.org/10.1007/978-3-031-20398-5_8

as image recognition[3], action recognition[4], speech recognition[5], natural language processing (NLP)[6] and network applications[7].

In the traditional machine learning architecture, a central server manages the trained model. First, the sensors or edge devices return the collected data to the server and perform training in the server, and then send the training results to the edge devices or mobile devices. However, this method will cause data privacy and security problems, and will occupy a large amount of network bandwidth and results in network delay problems. To solve the above mentioned problems, Federated Learning (FL) approach has been proposed [8, 9]. In the FL approach, the central server is connected with each client device (edge device/mobile device), and the client devices use local data to training for building a model, and then send the model weights obtained after training back to the central server for aggregation [10]. In the process of model training, there is no need to send clients' data back to the server. Thus, the FL approach can solve the problem of data privacy of users while it can achieve the accuracy similar to that of a centralized machine learning architecture.

Currently, there are many applications using federated learning technology. For example, if multiple hospitals want to cooperate to train a model, but because the patient's private information is not convenient to provide to other hospitals, through the framework of federated learning technology, machine learning training is performed in each hospital before the training. The latter weights are provided to the central server. In addition, Google's own input method Gboard allows each user's mobile phone to download a set of models, and then train the local model according to the user's usage behavior, and then send it back to the central server so that the user's input words will be sent back to Google, and the candidate words can appear more accurately. The voice assistant Siri on the iPhone, through the neural network engine on the CPU, allows the phone to train the user's voice, and the audio recorded by Siri never leaves the local device.

The purpose of this study is to implement the development of image recognition apps by using the federated learning technology, such that users can use their mobile devices for training to perform image recognition applications while ensure the data privacy of users.

2 Background Review and Related Works

In this section, we review some background knowledges for later use of system design and implementation. Several techniques which are applied as the methods in this work, such as federated learning, convolutional neural networks (CNNs) and Kotlin are introduced in the following.

2.1 Federated Learning

Federal learning is a decentralized machine learning architecture proposed by Google in 2016 [8]. This architecture is composed of a central server and multiple client devices that participate in training in different places. The client devices first train with their own data, and then upload their respective model weights to the central server after the training, and then the central aggregates the weights. Repeat the above steps until convergence is reached. The architecture is performed in the form of client-server, in which each client uses local data for training, and then uploads the updated model parameters to the server, so as to optimize all models simultaneously. Thus, even if the information can not be shared, the learning strategy still can be carried out by taking into account the problems of privacy and model bias [9].

The central server will update the global model based on the aggregated results, and return the updated global model to the clients participating in the FL architecture. The clients will update their local model and start the next round training. At the same time, the performance metrics of the overall model will be evaluated. When the performance metrics are converged to be stable, the iterative training of the FL model will be terminated.

2.2 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a backforward neural network. Its artificial neurons can respond to a part of the surrounding units in the coverage area, which is excellent for large-scale image processing. It has two important characteristics [11]: (1). It can effectively reduce a large amount of data to a small amount of data, (2). It can effectively retain the characteristics of the image, in line with the principles of image processing.

2.3 Kotlin

Kotlin is a programming language developed by JetBrains. It is a statically typed programming language that runs on the Java Virtual Machine (JVM). It can be compiled into Java byte code, or it can be converted into using LLVM after compilation. It also provides native codes to execute, or compile into JavaScript language for browser to execute.

3 Background Review and Related Works

The overall architecture of this work is shown in Fig. 1. The system is mainly divided into two parts: the first one is the server side, and the second one is the client side. First, on the server side, the system uses Kotlin as the main programming language. The server side is mainly responsible for generating an initialized global model, and then sending the global model to each connected client. After the client receives the initialized global model, it trains with the model and then uploads the updated model weights to the server. After receiving the weights obtained from clients, the weights obtained by each client are aggregated to generate an updated global model, and then the updated global model is transmitted to the clients. The second part is the client side, which is mainly built in the Android operating system and uses Kotlin as the main App programming language. In addition, the DeepLearning4j software is deployed as the machine learning framework. The image recognition training will be performed on the client device itself, and then the related weights of the recognition images will be uploaded to the server.

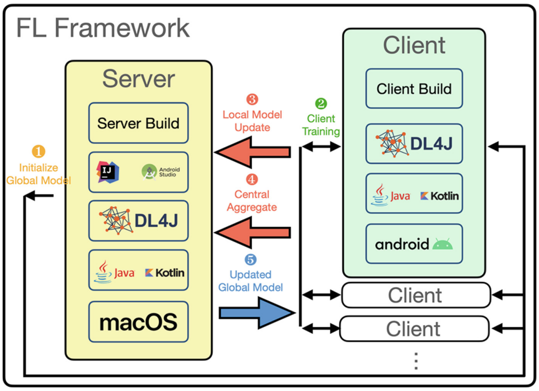

Fig. 1. The system architecture.

With regard to the server design and implementation of the system, the server is mainly responsible for starting a new round and accepting the weight of the client's return. In the system, we set a port and can use the browser to check whether the server is normal operation, its setting code is shown in Fig. 2. In addition, the server also needs to train the weights sent from the client. In this system, we place the model in the newly created main folder, which contains the weight of the client (currentRound), and the training is completed. The sub-folder of the model, and during training, the Server will read the parameter update file from the currentRound for training, and it can also open the DL4J UI interface to view the training process and results (as shown in Fig. 3).

```
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletContextHandler;
import org.eclipse.jetty.servlet.ServletHolder;
import org.glassfish.jersey.media.multipart.MultiPartFeature;
import org.glassfish.jersey.server.ResourceConfig;
import org.glassfish.jersey.servlet.ServletContainer;

public class JobQueueServer {

    public static void main(String[] args) {

        final ResourceConfig resourceConfig = new ResourceConfig(RestService.class);
        resourceConfig.register(MultiPartFeature.class);

        ServletHolder jerseyServlet = new ServletHolder(new ServletContainer(resourceConfig));

        Server jettyServer = new Server(9997);
        ServletContextHandler context = new ServletContextHandler(jettyServer, "/");
        context.addServlet(jerseyServlet, "/*");

        try {
            jettyServer.start();
            jettyServer.join();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            jettyServer.destroy();
        }
    }
}
```

Fig. 2. The code of server establishment.

In addition, with regard to the client system design and implementation, the system uses an Android phone as the client side, it will accept the model from the server, and then perform CNN training. After performing the images recognition, the obtained weights will be transmitted back to the server, instead of transmitting the raw image data.

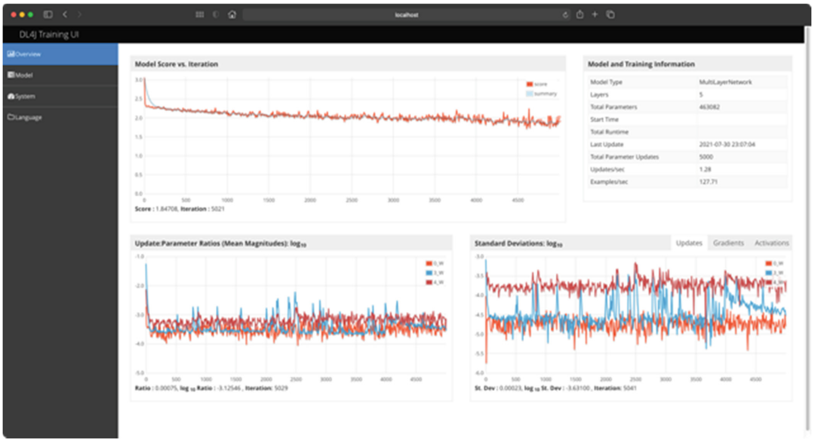

Fig. 3. The dashboard running in the server.

4 Experimental Results

4.1 The Hardware and Software Environment

The development software and hardware environment of this system is shown in Table 1. The system uses Android Studio IDE for server and mobile phone applications development. In addition, we use AVD emulator for application testing and execution in the system.

Table 1. The hardware and software equipment.

Hardware	**Server Side:** (1) Device: MacBook Pro (2) Operating System: MacOS 11.4 (3) CPU: Intel® Core™ i7-7820HQ 2.9 GHz (4) Memory (RAM): 16 GB (5) Hard Disk: 512 GB **Client Side:** (1) Device: Android Virtual Device Google Pixel 4 XL (2) Operating System: Android 10 (3) CPU: 4 Core CPU (4) Memory (RAM): 8 GB (5) Hard Disk: 20 GB
Software	(1) Programming Language: Kotlin v1.3.20 (JAVA v1.8.0) (2) Integrated Development Environment: Android Studio v4.1.3, IntelliJ IDEA 2020.2.4 (Community Edition)

4.2 Datasets

In this work, we use the Cifar-10 dataset [12] as the training and testing dataset for the development of the system. The dataset mainly consists of 10 categories of images which are all presented in the form of 32 × 32 pixels (as shown in Fig. 4). The categories are the airplanes, cars, birds, cats, deer, dogs, frogs, horses, boats, and trucks. The dataset contains a total of 60,000 images, each category contains 6,000 pictures, of which 50,000 pictures are used as the training set and 10,000 are used as the testing set. It is a collection of images often used in machine learning.

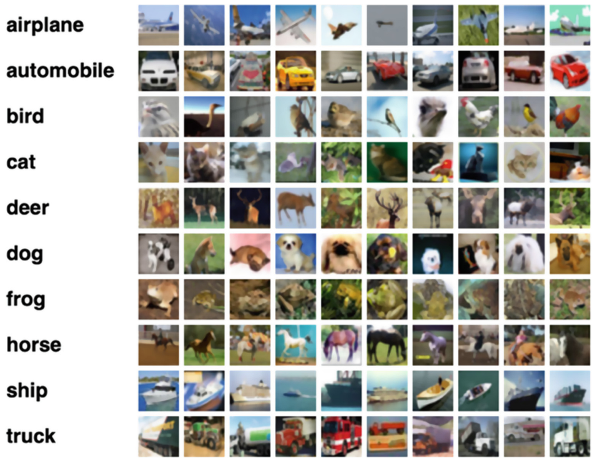

Fig. 4. The Cifar-10 dataset [12].

4.3 Implementation of App System

This app we develop will firstly use the camera software to capture images, recognize them and then store the classification results in the dataset for use in the next round of training. The obtained weights are then sent back to the server for training. In the following, we will present the main functions of the developed App system.

- The main screen of the system:

  The snapshot of the main screen in the system is shown in Fig. 5. From the figure, we see that when the model is loaded correctly, the program will pop up a prompt stating that "the model has been loaded". When the model has been loaded, the the toolbar icon will appear in the upper right corner of the screen, which means that you can use the previously captured images for training. You can also directly click " " to capture an image.

- The preliminary recognition function of the system:

  After the image is captured, the application will perform a preliminary recognition and will display which category the image belongs to. The snapshot of recognition results of the system is shown in Fig. 6. The user can choose whether to continue training or save the images. If you want to continue training, click " ". If you want to save the image, click " " to save the classification results.

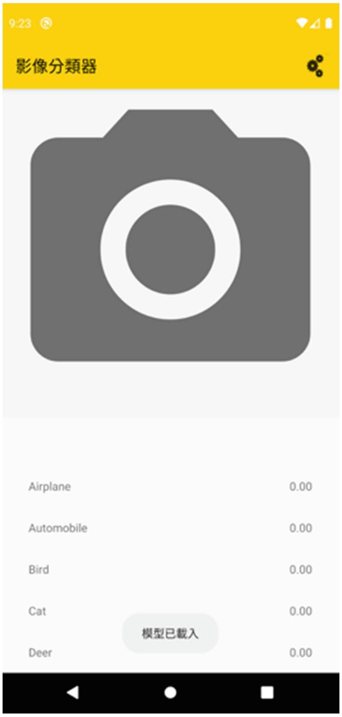

Fig. 5. The snapshot of the main screen of the system.

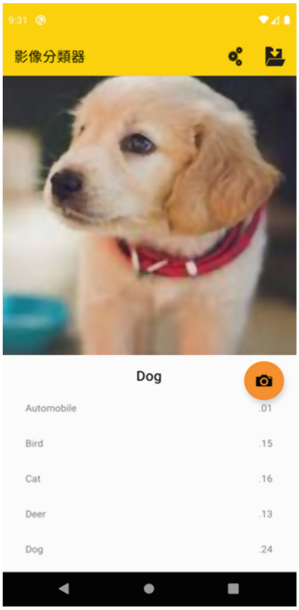

Fig. 6. The snapshot of preliminary recognition results of the system.

- The model training function of the system:
  After the images are taken, all the images are trained by the CNN model. The model has a total of four layers, and its architecture is shown in Fig. 7. In the system, we set the number of training (Epoch) to 100.

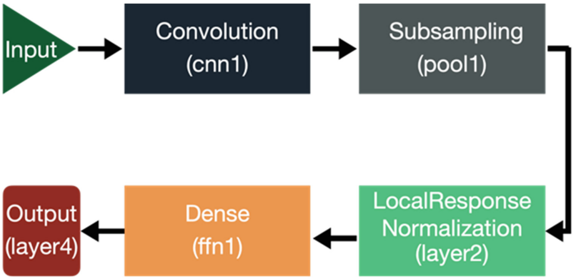

Fig. 7. The CNN model architecture used in this system.

Figure 8 is the snapshot of the model training in the system. From the figure, we see that the system provides the training data, model parameters and other related information. In addition, the training accuracy record of iterations are shown in the system.

Fig. 8. The snapshot of model training process.

In the CNN layer in the system, we use Adam's optimizer to update the weights of the neural network. The advantages are high computational efficiency and small memory requirements, and the ReLU activation function is used to solve the problem of the disappearance of the gradient, and to greatly reduce the amount of calculation. The pooling layer mainly adopts the MaxPooling method, such that when the entire image is shifted, the pixel judgment will not cause any influence which means that it has a good anti-noise function. Local Response Normalization (LRN) is a method to improve the accuracy of deep learning, generally after activation and pooling, the purpose of use is to reduce the error rate of the classifier.

When the training is finished, that indicates that the message "Model training completed" will be shown in the system (as shown in Fig. 9). Then, press the return button to show the recognition results. When the clients transmit the weight parameters to the server, the app will show the completed trained model which has been uploaded to the server (as shown in Fig. 10).

Fig. 9. The snapshot of model training completion.

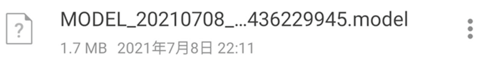

Fig. 10. The completed trained model uploaded to the server side.

After the first round of training, the images will be recognized. Then, the classification accuracy rate with respect to each category will be obtained. From Fig. 6, we see that the accuracy rate of the initial recognition with respect to each category are presented: 24% for Dog images, 16% for Cat images, 15% for Bird images, and so on. After multiple rounds of training, as shown in Fig. 11, the accuracy rate with respect to the Dog images is increased to 97%. From the implementation results, we see that after multiple rounds of training, the recognition results will tend to stabilize and converge. In addition, the accuracy rate will also be significantly increased.

- The server functions:

  In this work, we use the DL4J UI API [13] to provide an user interface in the browser to visualize the current network status and training process. First, in terms of model scores and iterative graphs which are shown in Fig. 12, the horizontal axis is the iteration, and the vertical axis is the model score. The trend of the number line in the figure increases with the iterations. Since we use the small batch gradient descent method, the model score is getting lower, and the fluctuations become smaller. In addition, the number line tends to converge, which means that the model training is more stable.

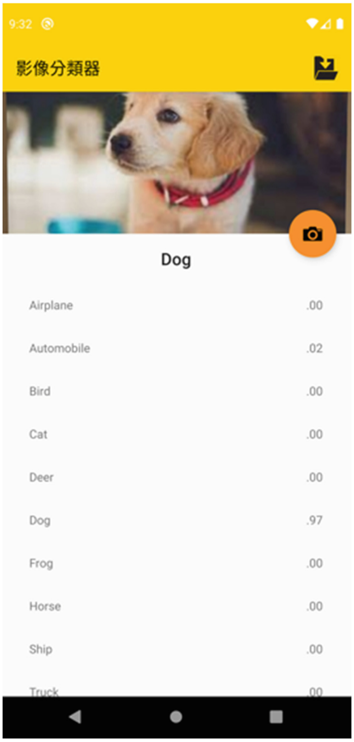

Fig. 11. The snapshot of accuracy results with respect to the dog images after multiple rounds of training.

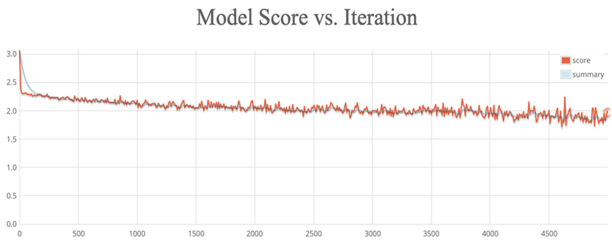

Fig. 12. The snapshot of the obtained model score of the system.

Figure 13 shows the updated parameter ratio. In the figure, the horizontal axis is the iteration and the vertical axis is the logarithm of parameter ratios. From the figure, we see that after the iteration starts at 50, the value tends to be stable, and there does not have more fluctuations.

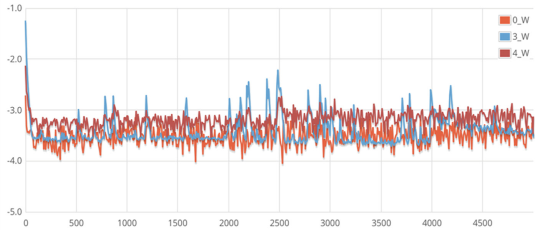

Fig. 13. The snapshot of the updated parameter ratio of the system.

5 Conclusion

In this work, we have successfully developed the App system based on the federated learning technology to perform real time image recognition with CNN models on the users' phones, and then returned the weight parameters after training to the central server for optimization. From the implementation results, we found that compared to the traditional machine learning approaches, the federated learning approach did not send raw images back to the server for model training. The implemented app let users to use their mobile devices and the central servers for iterative training. After multiple rounds of training, the convergence will be stabilized, and the accuracy will be significantly improved. Thus, the privacy issues of users could be solved and the amount of data transmission between the clients and the server could also be reduced significantly.

Acknowledgement. This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 110-2621-M-029-003-, 110-2221-E-029-002-MY3, 110-2221-E-126-004- and 110-2622-E-029-003-.

References

1. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. Proc. of the 2017 IEEE symposium on security and privacy (May 2017)
2. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553) (2015)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proc. of the IEEE conference on computer vision and pattern recognition (CVPR) (June 2016)
4. Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., Sheikh, Y.: OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields. In: IEEE Transactions on Pattern Analysis and Machine Intelligence **43**(1), pp. 172–186 (1 Jan. 2021)
5. Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., Stolcke, A.: The Microsoft 2017 conversational speech recognition system. In: Proc. of 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (April 2018)
6. Young, T., Hazarika, D., Poria, S., Cambria, E.: Recent trends in deep learning based natural language processing. IEEE Comput. Intell. Mag. **13**(3), 55–75 (2018)
7. Qian, M., Fei, H., Hao, Q.: Deep learning for intelligent wireless networks: a comprehensive survey. IEEE Communications Surveys & Tutorials **20**(4), 2595–2621 (2018)
8. Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency (2016). arXiv preprint arXiv: 1610.05492

9. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: concept and applications. ACM Trans. Intell. Sys. Technol. (TIST) **10**(2), 1–19 (2019)
10. Bonawitz, K., et al.: Towards federated learning at scale: System design (2019). arXiv preprint arXiv:1902.01046
11. Albawi, S., Mohammed, T.A., Al-Zawi, S.: Understanding of a convolutional neural network. In: Proc. of 2017 International Conference on Engineering and Technology (ICET) (Aug. 2017)
12. The CIFAR-10 dataset https://www.cs.toronto.edu/~kriz/cifar.html
13. Visualization - Deeplearning4j, https://deeplearning4j.konduit.ai/tuning-and-training/visualization