AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

OPENFL-2 2023

INSTYTUT INFORMATYKI

Authors:	**Agnieszka Szynalik, Jakub Machalica, Mariusz Kądziela**
Field of study:	**Informatyka**

Kraków, 2023

1 Introduction

Federated learning is a machine learning approach that allows multiple parties to collaboratively train a shared model without directly sharing their raw data. Instead of sending data to a central server or cloud, federated learning enables training to take place locally on individual devices or edge nodes. The trained local models are then combined or aggregated to create a global model without exposing the underlying data.

In federated learning, the process typically involves the following steps:

1. Initialization: A global model is created by a central server or a trusted entity.
2. Distribution: The global model is sent to participating devices or edge nodes.
3. Local Training: Each device or node performs training using its local data while keeping the data locally.
4. Model Update: The locally trained models are sent back to the central server.
5. Aggregation: The central server combines the locally trained models to create an updated global model.
6. Iterative Process: Steps 2-5 are repeated for multiple iterations to refine the global model.

Federated learning offers several advantages:

- Privacy: Data remains on the local devices or edge nodes, reducing the risks associated with centralized data storage and transfer.
- Efficiency: By training models locally, federated learning reduces the need for large-scale data transfer, minimizing bandwidth usage and reducing latency.
- Scalability: Federated learning can accommodate a large number of devices or edge nodes, making it suitable for distributed environments.
- Customization: Local training allows for personalized model updates based on individual devices' data, leading to tailored models for specific contexts.
- Data Ownership: Participants retain control and ownership over their data, promoting data governance and compliance.

Federated learning finds applications in various domains. Some common use cases include:

- Healthcare: Federated learning enables collaboration among healthcare institutions while preserving patient privacy, allowing for the development of robust disease prediction models.
- Internet of Things (IoT): Devices in an IoT network can collaboratively train models to improve their local intelligence without sharing sensitive data.
- Financial Services: Federated learning enables financial institutions to analyze customer behavior and detect fraud patterns while maintaining data privacy.
- Smart Cities: Edge devices in a smart city environment can work together to train models for traffic management, energy optimization, and other urban services.

Overall, federated learning facilitates the collective intelligence of distributed devices or edge nodes while ensuring data privacy, making it an attractive approach for collaborative machine learning tasks.

2 Theoretical background/technology stack

Federated learning is a privacy-preserving machine learning approach that ensures that sensitive data remains on devices and is not exposed to unauthorized access. OpenFL provides a range of tools to manage data privacy, including encryption and differential privacy.

It relies on distributed systems, which are networks of devices that communicate with each other to perform a task. Federated learning frameworks like OpenFL are designed to manage the complexity of such systems and ensure that they work together efficiently.

Moreover, OpenFL supports machine learning frameworks such as TensorFlow and PyTorch. These frameworks enable developers to build and deploy machine learning models using Python, which is a popular programming language for scientific computing and data analysis.

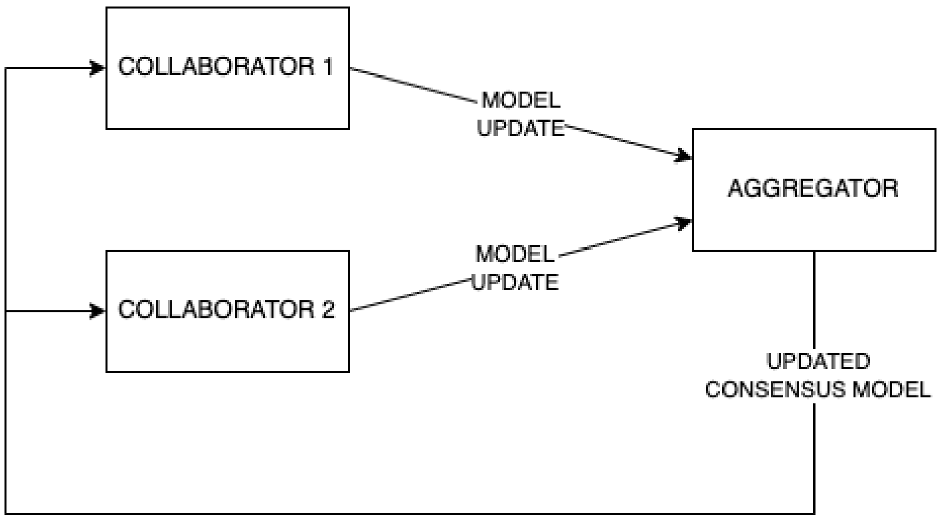
The infrastructure required to deploy a machine learning application depends on several factors, including the size of the data set, the complexity of the model, and the performance requirements of the application. AWS can be a great choice for running and hosting such application. It is a popular cloud computing platform that provides a wide range of services and tools. These services include Amazon Elastic Compute Cloud (EC2), Amazon S3, and Amazon Elastic Container Service (ECS).

Technolgy stack:

- **Programming language: Python 3**: Python is a versatile programming language widely used in data analysis and machine learning. Python 3 is the chosen version for this project, as it offers extensive libraries and frameworks for developing machine learning models. OpenFL, the federated learning framework used in this project, also supports Python 3.
- **Machine learning framework: TensorFlow**: TensorFlow is a popular open-source machine learning framework developed by Google. It provides a comprehensive set of tools and libraries for building and training machine learning models. TensorFlow offers support for various neural network architectures, including deep neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs). It integrates well with federated learning frameworks like OpenFL.
- **Federated learning framework: OpenFL**: OpenFL is an open-source federated learning framework that facilitates collaborative model training without exposing sensitive data. It allows organizations to build and deploy machine learning models on a distributed network of devices. OpenFL provides features for managing data privacy, version control, and model aggregation. It enables efficient coordination between the aggregator and collaborators in the federated learning process.
- **Infrastructure: Amazon Web Services (AWS)**: AWS is a leading cloud computing platform that offers a wide range of services and tools for building and deploying machine learning models. It provides scalable and cost-effective infrastructure to host the federated learning application. AWS services such as EC2 instances, S3 for data storage, and VPC for networking can be utilized to create a reliable and secure environment for the project.
- **Virtualization: Docker**: Docker is a popular platform that enables developers to build, share, and run applications in a portable and scalable manner. It provides containerization technology, allowing the bundling of the application and its dependencies into isolated containers. Docker is commonly used in machine learning projects to manage software dependencies and ensure reproducibility across different environments.

3 Case study concept description

In our project, we are utilizing federated learning as the core approach. Federated learning allows us to train a shared model collaboratively without directly accessing or transferring raw data. The goal is to create an application that demonstrates the functionality of federated learning, showcasing an aggregator and two collaborators working together.



The envisioned application will have the following components:

1. Aggregator: This component serves as the central server that coordinates the federated learning process. It initiates the model and distributes it to the collaborators, receives the trained models from the collaborators, and performs the aggregation to update the global model. The Aggregator will be deployed as a Docker container using Docker Compose.
2. Collaborators: There will be two collaborators in our application. Each collaborator represents a device or edge node that participates in the federated learning process. The collaborators receive the global model from the aggregator, train the model locally using their respective datasets, and send back the locally trained models to the aggregator for aggregation. The Collaborators will also be deployed as Docker containers using Docker Compose.

The workflow of the application can be summarized as follows:

1. The aggregator initializes the global model and distributes it to the two collaborators.
2. Each collaborator trains the model locally using its own dataset, while keeping the data securely on their respective devices.
3. Once the local training is complete, the collaborators send their trained models back to the aggregator.
4. The aggregator aggregates the locally trained models to create an updated global model that incorporates the knowledge from both collaborators.
5. The updated global model is then distributed back to the collaborators, and the process repeats for subsequent iterations to further refine the model.

Security:
Our applicarion will use Transport Layer Security (TLS) encryption for network connections in federated learning. Therefore, security keys and certificates will need to be created for the aggregator and collaborators to negotiate the connection securely.

Deployment:

To deploy the federated learning application on AWS, we can leverage a combination of AWS services, Docker, and Docker Compose. The application will be hosted on an Amazon EC2 instance, taking advantage of the scalability, reliability, and flexibility offered by the AWS cloud infrastructure.

With this deployment approach, users can securely access the application through an SSH connection. This provides a secure channel to interact with the application, start experiments, and access logs and generated files.

By utilizing AWS services, we ensure that the application benefits from the robust infrastructure and services provided by AWS, such as high availability, auto-scaling, and seamless integration with other AWS offerings. This enables efficient collaboration among the aggregator and collaborators while maintaining the privacy and security of their data.

4 Solution Architecture

The solution architecture for the federated learning project consists of various technologies and components working together to enable collaborative model training. Here is a short overview:

4.1 Determining the Demo Implementation Technology

The technology chosen for this federated learning project is Docker. It allows us to package the application, including the aggregator and collaborators, along with their dependencies, into containers. By utilizing Docker, we can ensure consistency and portability of the application across different environments, making it easier to deploy and showcase the federated learning functionality.

The use of Docker Compose, a tool for defining and managing multi-container Docker applications, further simplifies the deployment process. Docker Compose allows us to specify the configuration of the aggregator and collaborator containers, their network connections, and any other necessary settings. It enables us to start and manage the entire federated learning system with just a few commands.

By leveraging Docker and Docker Compose, we can create an environment that encapsulates the federated learning application, making it easy to deploy on AWS or any other platform that supports Docker. This approach ensures consistency, scalability, and reproducibility of the demo implementation, providing a seamless experience for showcasing the federated learning capabilities.

4.2 Data and Training Model

For the federated learning project, the CIFAR-10 dataset is utilized as the training data. It consists of 50,000 training images and 10,000 test images, categorized into 10 classes. Each image has a size of 32x32 pixels with three color channels (RGB).

The following code snippet demonstrates the model architecture used for training:

```
model = Sequential()
model.add(Conv2D(16, kernel_size=(4, 4), strides=(2, 2),
                 activation='relu', input_shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=(4, 4), strides=(2, 2),
                 activation='relu'))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss=ke.losses.categorical_crossentropy,
              optimizer=ke.optimizers.Adam(), metrics=['accuracy'])
```

This model architecture consists of several layers, including convolutional layers, dropout layers, max pooling layers, and fully connected layers. The Conv2D layers perform convolution operations on the input images, extracting important features. Dropout layers help prevent overfitting by randomly dropping out a fraction of the input units during training. Max pooling layers downsample the feature maps, reducing the spatial dimensions. The Flatten layer converts the multidimensional feature maps into a one-dimensional vector. The Dense layers provide the final classification output.

The model is compiled with the categorical cross-entropy loss function, the Adam optimizer, and the accuracy metric.

You can further customize the model architecture or experiment with different hyperparameters to improve the model's performance on the CIFAR-10 dataset.

5 Environment Configuration

To run the federated learning application, the following environment configuration is required:

5.1 Makefile

A Makefile is used to manage the execution of Docker commands. The Makefile includes the following targets:

- **start**: Starts the Docker containers using `docker compose up -d`.
- **stop**: Stops the Docker containers using `docker compose down`.
- **logs**: Displays the logs of all Docker containers using `docker compose logs`.
- **ps**: Shows the status of all Docker containers using `docker compose ps`.
- **logs_service**: Displays the logs of a specific service/container using `docker compose logs -follow -no-log-prefix -timestamps $(service)`.
- **logs_container**: Displays the logs of a specific container using `docker logs -follow -timestamps $(container)`.

5.2 docker-compose.yml

The `docker-compose.yml` file defines the services for the federated learning application. It includes the following services:

- **aggregator**: Builds and runs the aggregator component using the specified `${AGGREGATOR}` build path. It is configured to use the host network mode.
- **coll_1**: Builds and runs the first collaborator component using the specified `${COLL1}` build path. It is configured to use the host network mode.
- **coll_2**: Builds and runs the second collaborator component using the specified `${COLL2}` build path. It is configured to use the host network mode.

5.3 Dockerfiles

Two Dockerfiles are used to build the aggregator and collaborator components. Both Dockerfiles install TensorFlow 2.8.0 and create a directory for the application code. The aggregator Dockerfile sets the entrypoint to start the aggregator component, while the collaborator Dockerfile sets the entrypoint to start the collaborator component with specific parameters.

5.4 User Data Script for AWS EC2 Instance

The `user_data.sh` script is used for provisioning an AWS EC2 instance. It updates the system, installs Git, clones the federated learning application repository, installs Docker and Docker Compose, and grants the `ec2-user` access to the Docker daemon. Finally, it installs the `make` utility.

This environment configuration enables running the federated learning application locally or on an AWS EC2 instance.

6 Installation Method

To set up the federated learning application, follow these installation steps:

6.1 Prerequisites

Before starting the installation, ensure that the following prerequisites are met:

- Docker: Install Docker on your system by following the official Docker installation instructions for your operating system.
- Git: Install Git on your system to clone the federated learning application repository.
- AWS cli: Install the AWS Command Line Interface (CLI) on your system to interact with AWS services.

6.2 Installation Steps

Follow these steps to install and run the federated learning application:

1. Clone the repository: Use Git to clone the federated learning application repository to your local machine or the target environment.

   ```
   git clone https://github.com/syklzz/federated-learning.git
   ```

2. Configuration: Navigate to the repository directory and configure the necessary parameters, such as the dataset, model architecture, and training settings, according to your requirements.
3. Set up the environment: Copy the provided `.env` file and modify the necessary variables, such as `AGGREGATOR`, `COLL1`, and `COLL2`, to point to the appropriate directories containing the aggregator and collaborator code.

Following these installation steps will successfully set up the federated learning application for your use.

7 How to reproduce

1. Create an EC2 instance with user data:

   ```
   aws ec2 run-instances --image-id <ami_id> --instance-type
   <instance_type> --key-name <key_pair> --user-data file://user_data.sh
   ```

   Make sure to replace <ami_id> with the ID of the desired Amazon Machine Image for your instance, <instance_type> with the instance type you want to launch, and <key_pair> with the name of your key pair. The –user-data parameter should point to the location of your user data script.

2. Connect to an EC2 instance using SSH:

   ```
   ssh -i <path_to_key_pair_file> <username>@<public_dns_name>
   ```

   Replace the following placeholders with the appropriate values:

   - <path_to_key_pair_file>: The file path to the private key (.pem) associated with the key pair used to launch the EC2 instance.
   - <username>: The username associated with the operating system of the EC2 instance.
   - <public_dns_name>: The public DNS name or public IP address of the EC2 instance.

3. Navigate to application folder:

   ```
   cd app
   ```

4. Start the application:

   Use the provided Makefile to start the Docker containers. Run the following command:

   ```
   make start
   ```

   This command will bring up the aggregator and collaborator containers defined in the `docker-compose.yml` file.

5. Stop the application:

   When you're done using the application, use the provided Makefile target or Docker command to stop the containers. Run the following command:

   ```
   make stop
   ```

   This command will stop and remove the running containers.

8 Demo deployment steps

8.1 Configuration set-up

Configuration in this project can be considered in two ways. There is a configuration strictly for OpenFL, separate for aggregator and every collaborator. It consists of various options, ranging from file paths to networking, as specified in the official documentation. Example from aggregator:

```yaml
aggregator:
  settings:
    best_state_path: save/keras_cnn_cifar_best.pbuf
    db_store_rounds: 2
    init_state_path: save/keras_cnn_cifar_init.pbuf
    last_state_path: save/keras_cnn_cifar_last.pbuf
    rounds_to_train: 10
    write_logs: true
  template: openfl.component.Aggregator
assigner:
  settings:
    task_groups:
    - name: train_and_validate
      percentage: 1.0
      tasks:
      - aggregated_model_validation
      - train
      - locally_tuned_model_validation
  template: openfl.component.RandomGroupedAssigner
collaborator:
  settings:
    db_store_rounds: 1
    delta_updates: false
    opt_treatment: RESET
  template: openfl.component.Collaborator
compression_pipeline:
  settings: {}
  template: openfl.pipelines.NoCompressionPipeline
data_loader:
  settings:
    batch_size: 256
    collaborator_count: 2
    data_group_name: cifar
  template: src.tfcifar_inmemory.TensorFlowCIFARInMemory
network:
  settings:
    agg_addr: localhost
    agg_port: 60413
    cert_folder: cert
    client_reconnect_interval: 5
    disable_client_auth: false
    hash_salt: auto
    tls: true
  template: openfl.federation.Network
task_runner:
  settings: {}
  template: src.keras_cnn.KerasCNN
tasks:
  aggregated_model_validation:
    function: validate
    kwargs:
      apply: global
      batch_size: 32
      metrics:
      - accuracy
  locally_tuned_model_validation:
    function: validate
    kwargs:
      apply: local
      batch_size: 32
      metrics:
      - accuracy
  settings: {}
  train:
    function: train
    kwargs:
      batch_size: 32
      epochs: 1
      metrics:
      - loss
```

There is also a .env file, used for configuration of Docker Compose, where they are used as containers' names, with default contents:

```
AGGREGATOR="aggregator"
COLL1="collaborator_1"
COLL2="collaborator_2"
```

8.2 Data preparation

The data preparation in the cifar10.py class involves several steps. Firstly, the image data is converted to grayscale using the Image.fromarray(image).convert('L') method. This conversion transforms the RGB color image into a grayscale image, where each pixel is represented by a single intensity value.

By converting the images to grayscale, the color information is discarded, and only the grayscale intensity values are retained. This can be beneficial for certain tasks or models that do not require color information or when working with limited computational resources.

After the conversion to grayscale, the data is split among collaborators. This typically involves dividing the dataset into subsets that are allocated to different collaborators or workers. Each collaborator may receive a portion of the data for training, validation, or testing purposes. This partitioning allows for distributed or parallel processing, where multiple collaborators can work on their assigned subset of data simultaneously.

The data splitting step is important for collaborative tasks, as it enables efficient distribution of the workload and promotes collaboration among multiple individuals or teams. Each collaborator can independently work on their assigned data subset, potentially training separate models or conducting specific analyses.

Overall, the data preparation in the cifar10.py class involves converting the images to grayscale and subsequently distributing the data among collaborators to facilitate collaborative efforts in training models or performing various tasks related to the CIFAR-10 dataset.

8.3 Execution procedure

1. After connecting and configuring EC2 instance, the experiment was started by the use of:

   ```
   make start
   ```

2. The deployment was validated with:

   ```
   make ps
   ```

3. All initial logs were viewed by:

   ```
   make logs
   ```

4. Particular service logs were checked with:

   ```
   make logs_service <service_name>
   ```

   e.g.:

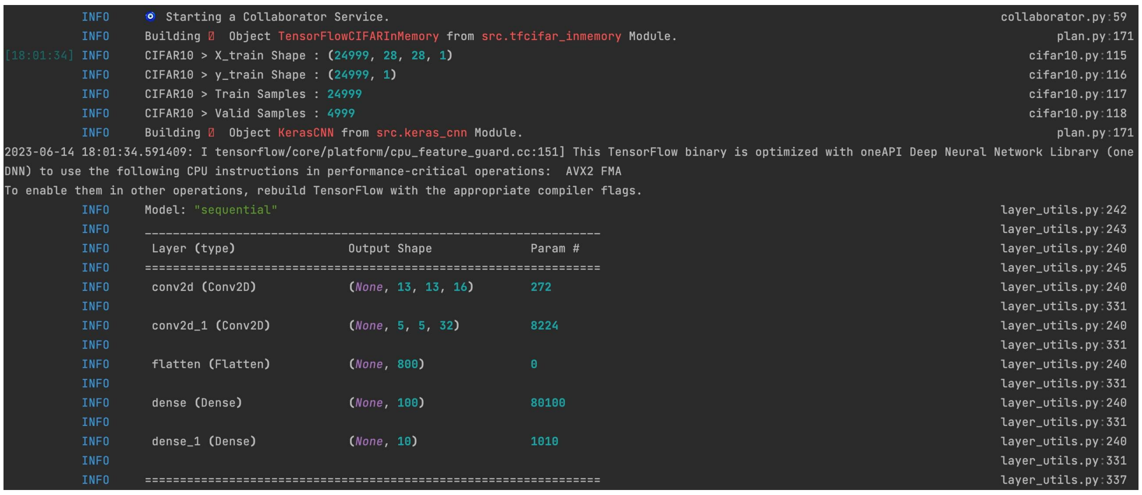
   ```
   make logs_service aggregator
   ```

5. Once the experiment ended the containers were stopped by:

   ```
   make stop
   ```

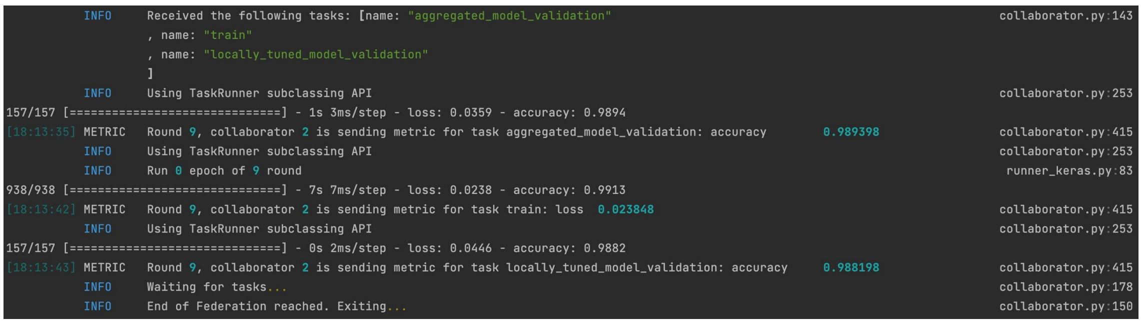
8.4 Results presentation

Once the experiment started, aggregator began coordination of collaborators and sent them the experiment plan. Collaborators built their private models and participated in federated learning.



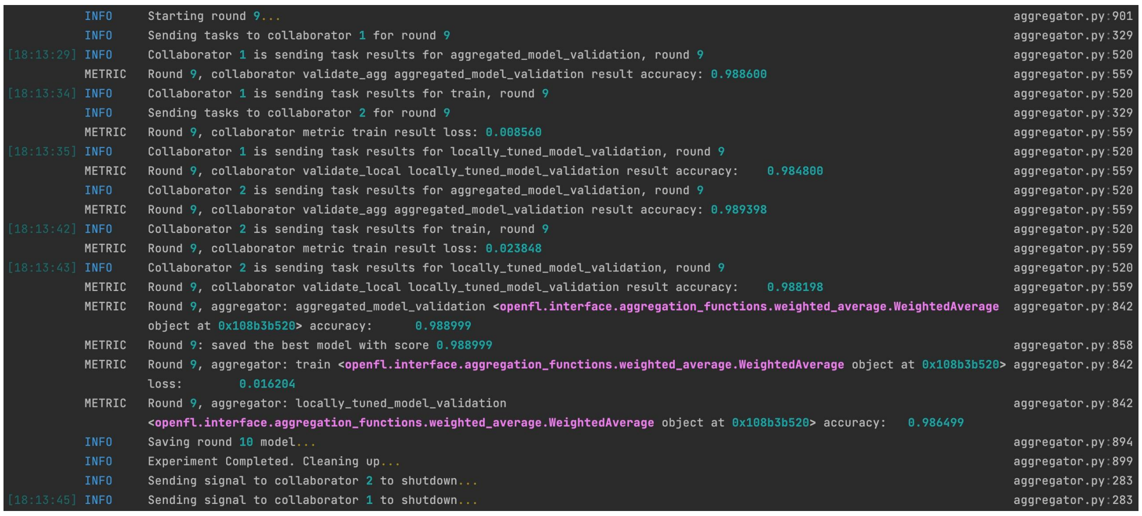
```
           INFO     🧿 Starting a Collaborator Service.                                                                   collaborator.py:59
           INFO     Building 🗄  Object TensorFlowCIFARInMemory from src.tfcifar_inmemory Module.                                plan.py:171
[18:01:34] INFO     CIFAR10 > X_train Shape : (24999, 28, 28, 1)                                                               cifar10.py:115
           INFO     CIFAR10 > y_train Shape : (24999, 1)                                                                       cifar10.py:116
           INFO     CIFAR10 > Train Samples : 24999                                                                            cifar10.py:117
           INFO     CIFAR10 > Valid Samples : 4999                                                                             cifar10.py:118
           INFO     Building 🗄  Object KerasCNN from src.keras_cnn Module.                                                       plan.py:171
2023-06-14 18:01:34.591409: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (one
DNN) to use the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
           INFO     Model: "sequential"                                                                                   layer_utils.py:242
           INFO     _________________________________________________________________                                     layer_utils.py:243
           INFO      Layer (type)                Output Shape              Param #                                        layer_utils.py:240
           INFO     =================================================================                                     layer_utils.py:245
           INFO      conv2d (Conv2D)             (None, 13, 13, 16)        272                                            layer_utils.py:240
           INFO                                                                                                           layer_utils.py:331
           INFO      conv2d_1 (Conv2D)           (None, 5, 5, 32)          8224                                           layer_utils.py:240
           INFO                                                                                                           layer_utils.py:331
           INFO      flatten (Flatten)           (None, 800)               0                                              layer_utils.py:240
           INFO                                                                                                           layer_utils.py:331
           INFO      dense (Dense)               (None, 100)               80100                                          layer_utils.py:240
           INFO                                                                                                           layer_utils.py:331
           INFO      dense_1 (Dense)             (None, 10)                1010                                           layer_utils.py:240
           INFO                                                                                                           layer_utils.py:331
           INFO     =================================================================                                     layer_utils.py:337
```

When the desired number of rounds was reached and there weren't any new tasks, collaborators exited.



```
           INFO     Received the following tasks: [name: "aggregated_model_validation"                                  collaborator.py:143
                    , name: "train"
                    , name: "locally_tuned_model_validation"
                    ]
           INFO     Using TaskRunner subclassing API                                                                    collaborator.py:253
157/157 [==============================] - 1s 3ms/step - loss: 0.0359 - accuracy: 0.9894
[18:13:35] METRIC   Round 9, collaborator 2 is sending metric for task aggregated_model_validation: accuracy        0.989398      collaborator.py:415
           INFO     Using TaskRunner subclassing API                                                                    collaborator.py:253
           INFO     Run 0 epoch of 9 round                                                                               runner_keras.py:83
938/938 [==============================] - 7s 7ms/step - loss: 0.0238 - accuracy: 0.9913
[18:13:42] METRIC   Round 9, collaborator 2 is sending metric for task train: loss  0.023848                            collaborator.py:415
           INFO     Using TaskRunner subclassing API                                                                    collaborator.py:253
157/157 [==============================] - 0s 2ms/step - loss: 0.0446 - accuracy: 0.9882
[18:13:43] METRIC   Round 9, collaborator 2 is sending metric for task locally_tuned_model_validation: accuracy     0.988198      collaborator.py:415
           INFO     Waiting for tasks...                                                                                collaborator.py:178
           INFO     End of Federation reached. Exiting...                                                               collaborator.py:150
```

The aggregator aggregated the final model and measured accuracy. The final accuracy was 0.986.



```
           INFO     Starting round 9...                                                                                         aggregator.py:901
           INFO     Sending tasks to collaborator 1 for round 9                                                                 aggregator.py:329
[18:13:29] INFO     Collaborator 1 is sending task results for aggregated_model_validation, round 9                             aggregator.py:520
           METRIC   Round 9, collaborator validate_agg aggregated_model_validation result accuracy: 0.988600                    aggregator.py:559
[18:13:34] INFO     Collaborator 1 is sending task results for train, round 9                                                   aggregator.py:520
           INFO     Sending tasks to collaborator 2 for round 9                                                                 aggregator.py:329
           METRIC   Round 9, collaborator metric train result loss: 0.008560                                                    aggregator.py:559
[18:13:35] INFO     Collaborator 1 is sending task results for locally_tuned_model_validation, round 9                          aggregator.py:520
           METRIC   Round 9, collaborator validate_local locally_tuned_model_validation result accuracy:    0.984800            aggregator.py:559
           INFO     Collaborator 2 is sending task results for aggregated_model_validation, round 9                             aggregator.py:520
           METRIC   Round 9, collaborator validate_agg aggregated_model_validation result accuracy: 0.989398                    aggregator.py:559
[18:13:42] INFO     Collaborator 2 is sending task results for train, round 9                                                   aggregator.py:520
           METRIC   Round 9, collaborator metric train result loss: 0.023848                                                    aggregator.py:559
[18:13:43] INFO     Collaborator 2 is sending task results for locally_tuned_model_validation, round 9                          aggregator.py:520
           METRIC   Round 9, collaborator validate_local locally_tuned_model_validation result accuracy:    0.988198            aggregator.py:559
           METRIC   Round 9, aggregator: aggregated_model_validation <openfl.interface.aggregation_functions.weighted_average.WeightedAverage  aggregator.py:842
                    object at 0x108b3b520> accuracy:      0.988999
           METRIC   Round 9: saved the best model with score 0.988999                                                           aggregator.py:858
           METRIC   Round 9, aggregator: train <openfl.interface.aggregation_functions.weighted_average.WeightedAverage object at 0x108b3b520> aggregator.py:842
                    loss:        0.016204
           METRIC   Round 9, aggregator: locally_tuned_model_validation                                                         aggregator.py:842
                    <openfl.interface.aggregation_functions.weighted_average.WeightedAverage object at 0x108b3b520> accuracy:   0.986499
           INFO     Saving round 10 model...                                                                                    aggregator.py:894
           INFO     Experiment Completed. Cleaning up...                                                                        aggregator.py:899
           INFO     Sending signal to collaborator 2 to shutdown...                                                             aggregator.py:283
[18:13:45] INFO     Sending signal to collaborator 1 to shutdown...                                                             aggregator.py:283
```

9 Summary

The project resulted in a successful implementation of a federated learning system using Intel OpenFL and AWS infrastructure. It demonstrated the potential of federated learning to leverage distributed data resources while maintaining privacy, which is especially important for industries dealing with sensitive data, such as healthcare or financial sectors.

The system allowed participants to collaborate on training machine learning models without sharing their raw data, thus addressing privacy concerns and enabling data-sensitive industries to benefit from the advantages of machine learning. The deployment involved configuring secure communication protocols such as TLS, ensuring encrypted and authenticated data transmission between the participating parties. For this purpose, certificates were used to establish trust and verify the authenticity of the parties involved.

Throughout the project, the team dedicated efforts to enhance the implemented model, aiming for optimal performance and accuracy. Regular experimentation and evaluation were conducted to measure the system's performance and assess its accuracy.

Overall, the federated learning project showcased the importance of data privacy in machine learning and provided a practical framework for secure and collaborative model training across multiple parties.

10 References

- https://en.wikipedia.org/wiki/Federated_learning
- https://www.intel.com/content/www/us/en/developer/articles/technical/how-openfl-boost-your-federated-learning-project.html