

Hitachi Industrial Edge Computer

CE50-10A User's Guide

CC-65-0179

■ Product covered by this manual

CE50-10A (Operating system: Ubuntu 18.04 LTS (Linux kernel 4.15))

■ Export restrictions

If you export this product, please check all restrictions (for example, Japan's Foreign Exchange and Foreign Trade Law, and USA export control laws and regulations), and carry out all required procedures.

If you require more information or clarification, please contact your Hitachi sales representative.

■ Trademarks

HITACHI is a registered trademark of Hitachi, Ltd.

Google Chrome is a trademark of Google LLC.

Intel and OpenVINO are trademarks of Intel Corporation in the U.S. and/or other countries.

Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

PostgreSQL is a registered trademark or trademark of the PostgreSQL Community Association of Canada in Canada and/or other countries.

Ubuntu is a registered trademark or trademark of Canonical Ltd.

Other company and product names mentioned in this document may be the trademarks of their respective owners.

■ Software

This product contains some open source software to perform its functions. The property rights, copyrights, and other intellectual property rights of the open source software belong to the author, and can be used under the conditions stated in the bundled license. For details, see the following file in this product:

/hitachi/licenses/copyright.txt

For open source software whose license specifies that the source code is to be provided, Hitachi will provide the source code of the open source software. Contact your Hitachi sales representative or distributor if you require such source code. Hitachi assumes no liability, including liability for damages, for open source software and its use.

This does not mean that Hitachi disclaims product liability for this unit.

■ Precautions

All contents of this manual are protected by copyright. No part of this manual may be reproduced in any form or by any means without permission in writing from the publisher. Information in this manual is subject to change without notice.

Read this manual carefully and keep it. Before using the unit, make sure that you thoroughly read and understand all the safety instructions.

Keep this manual in a place where it can be readily accessed as necessary.

■ Issued

June 2021: CC-65-0179

■ Copyright

All Rights Reserved. Copyright (C) 2021, Hitachi, Ltd.

Preface

This manual provides an overview, operation procedures, and other information about the AI image application development and execution functionality (abbreviated hereinafter to *AI image application functionality*) provided by the Hitachi Industrial Edge Computer CE50-10A (abbreviated hereinafter to *CE50-10A*).

■ Intended readers

This manual is intended for those who use the CE50-10A (operators, system engineers, and maintenance personnel).

■ Organization of this manual

This manual is organized into the following chapters and an appendix:

PART 1: Description

1. Functional overview

This chapter provides an overview of the functions that are provided by the CE50-10A.

PART 2: Experience

2. Quick guide for using AI image application functionality

This chapter provides a quick guide for setting up AI image application functionality and running sample programs for users who have not yet experienced the AI image application functionality.

PART 3: Setup

3. Setting up the AI image application functionality

This chapter describes the procedure for setting up the AI image application functionality.

PART 4: Design and Functionality

4. Design of the AI image application functionality

This chapter describes how to customize the AI image application functionality and how to start and stop the functionality.

5. Data input function

This chapter provides an overview of the data input function. This chapter also describes how to connect a camera to the function and how to set up the function.

6. Data management function

This chapter provides an overview of the data management function. This chapter also describes the API and library specifications, and how to set up the function.

7. Inference execution function

This chapter provides an overview of the inference execution function, shows the general procedure for implementing inference processing, and describes sample programs.

8. Inference development function

This chapter provides an overview of the inference development function. This chapter also describes how to start the function and how to use Jupyter Notebook.

PART 5: Operation

9. RAS techniques for the AI image application functionality

This chapter describes how to specify the settings for error detection, container restart, and collection of operating information.

10. Updater

This chapter provides an overview of the updater and describes the update procedure.

11. Troubleshooting

This chapter describes the error messages that are displayed by the AI image application functionality and the actions to be taken when the error messages are displayed.

Appendix

A. Interfaces of the CE50-10A

This appendix describes the interfaces that are supported by the CE50-10A.

■ Related publications

A manual related to this manual is as follows. Refer to this manual when necessary.

- Hitachi Industrial Edge Computer CE50-10 Instruction Manual (CC-65-0171)

■ Conventions: Abbreviations for product names

This manual uses the following abbreviations for product names:

Full name	Abbreviation	Meaning
Hitachi Industrial Edge Computer CE50-10	CE50-10	A generic name for the CE50-10 series models
Hitachi Industrial Edge Computer CE50-10A	CE50-10A	A CE50-10 series model that has the AI image application functionality

■ Conventions: Fonts and symbols

The following table explains the text formatting conventions used in this manual:

Text formatting	Convention
Bold	<p>Bold characters indicate text in a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example:</p> <ul style="list-style-type: none"> • From the File menu, choose Open. • Click the Cancel button. • In the Enter name entry box, type your name.
<i>Italic</i>	<p>Italic characters indicate a placeholder for some actual text to be provided by the user or system. For example:</p> <ul style="list-style-type: none"> • Write the command as follows: <code>copy source-file target-file</code> • The following message appears: A file was not found. (file = <i>file-name</i>) <p>Italic characters are also used for emphasis. For example:</p> <ul style="list-style-type: none"> • Do <i>not</i> delete the configuration file.
Monospace	<p>Monospace characters indicate text that the user enters without change, or text (such as messages) output by the system. For example:</p> <ul style="list-style-type: none"> • At the prompt, enter <code>dir</code>. • Use the <code>send</code> command to send mail. • The following message is displayed: <code>The password is incorrect.</code>

■ Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is $1,024$ bytes.
- 1 MB (megabyte) is $1,024^2$ bytes.
- 1 GB (gigabyte) is $1,024^3$ bytes.
- 1 TB (terabyte) is $1,024^4$ bytes.

Contents

Part 1: Description

1	Functional overview	1
1.1	What can be done with CE50-10A	2
1.2	AI image application functionality	4
1.2.1	OpenVINO	4

Part 2: Experience

2	Quick guide for using the AI image application functionality	7
2.1	Overview of experiencing the AI image application functionality	8
2.1.1	Creating a Docker-only partition	9
2.1.2	Confirming the Docker-only partition	9
2.1.3	Creating a file system in the Docker-only partition	10
2.1.4	Mounting the Docker-only partition	10
2.1.5	Starting the Docker service	10
2.1.6	Installing a Docker image	10
2.2	Description of sample programs	12
2.2.1	Running the sample program that imports a sample video	12
2.2.2	Running the sample program that imports a live video from a USB camera	13

Part 3: Setup

3	Setting up the AI image application functionality	17
3.1	Procedure for setting up the AI image application functionality	18
3.1.1	Creating and mounting a Docker-only partition	18
3.1.2	Starting the Docker service	18
3.1.3	Installing the Docker image	18
3.2	Displaying the partition information	19

Part 4: Design and Functionality

4	Design of the AI image application functionality	21
	4.1 Using the Compose file to customize the functionality	22
	4.2 Procedure for setting up the automatic startup of Docker containers during OS startup	25
	4.2.1 Procedure for setting up the automatic startup of the Docker service	25
	4.2.2 Procedure for enabling the automatic startup of Docker containers	25
	4.3 How to start and stop the AI image application functionality	26
	4.3.1 How to start Docker containers	26
	4.3.2 How to stop Docker containers	26
	4.3.3 How to check the status of the Docker containers	27
	4.4 Notes on designing a system	28
	4.4.1 Provided container images	28
	4.4.2 Automatic restart of Docker containers	28
	4.4.3 Tuning the number of obtained frames	29
	4.4.4 Firewall settings	29
	4.4.5 Security risks and countermeasures	31
5	Data input function	33
	5.1 Overview of the data input function	34
	5.2 How to connect a camera	35
	5.2.1 Connecting a USB camera	35
	5.2.2 Connecting an IP camera	35
	5.3 How to specify the settings of the data input function	36
	5.3.1 Settings of the data input function	36
	5.3.2 Container settings	39
	5.4 Procedure for verifying operation	41
6	Data management function	43
	6.1 Overview of the data management function	44
	6.2 API specifications of the data management function	46
	6.2.1 Registering a file in the data management function (v1FilesPost)	46
	6.2.2 Updating file status (v1FilesFileIdPut)	47
	6.2.3 Receiving files (v1FilesGet)	47
	6.3 Specifications of the library available for the data management function	49
	6.3.1 Library usage	49
	6.3.2 Settings for using the library	49
	6.3.3 How to import the library	50
	6.3.4 Dataman class methods	50
	6.3.5 Constants for file statuses	51

6.4	How to specify the settings of the data management function	52
6.4.1	Settings file for the data management function	52
6.4.2	Compose file settings	52
7	Inference execution function	55
7.1	Overview of the inference execution function	56
7.1.1	How to obtain the pre-trained models provided for use with OpenVINO	56
7.1.2	Converting a pre-trained model into an intermediate representation	56
7.2	Overview of implementing inference processing	58
7.2.1	How to implement inference processing	58
7.3	Description of the sample program	62
7.3.1	Processing performed by the sample program	62
8	Inference development function	67
8.1	Overview of the inference development function	68
8.2	How to start the inference development function	69
8.3	How to use Jupyter Notebook	70
8.3.1	Basic operations in Jupyter Notebook	70
8.3.2	Procedure for using OpenVINO on Jupyter Notebook	72
9	RAS techniques for the AI image application functionality	75
9.1	Overview of the RAS techniques for the AI image application functionality	76
9.2	Error detection	77
9.2.1	Settings in the Compose file	77
9.2.2	Log format	77
9.2.3	Maximum size and generation management of the log file	77
9.3	Container restart	79
9.4	Collection of operating information	80
Part 5: Operation		
10	Updater	81
10.1	Overview of the updater	82
10.2	Overview of the update procedure	83
10.2.1	Update procedure	83
11	Troubleshooting	87
11.1	Error messages that might be displayed when the docker-compose command is run	88

11.2 Error messages of the data input function	89
11.3 Error messages of the data management function	90
11.4 Error messages of the inference execution function	91
11.5 Error messages of the inference development function	92

Appendix 93

A. Interfaces of CE50-10A	94
A.1 Supported specifications	94

1

Functional overview

This chapter provides an overview of the functions that are provided by CE50-10A.

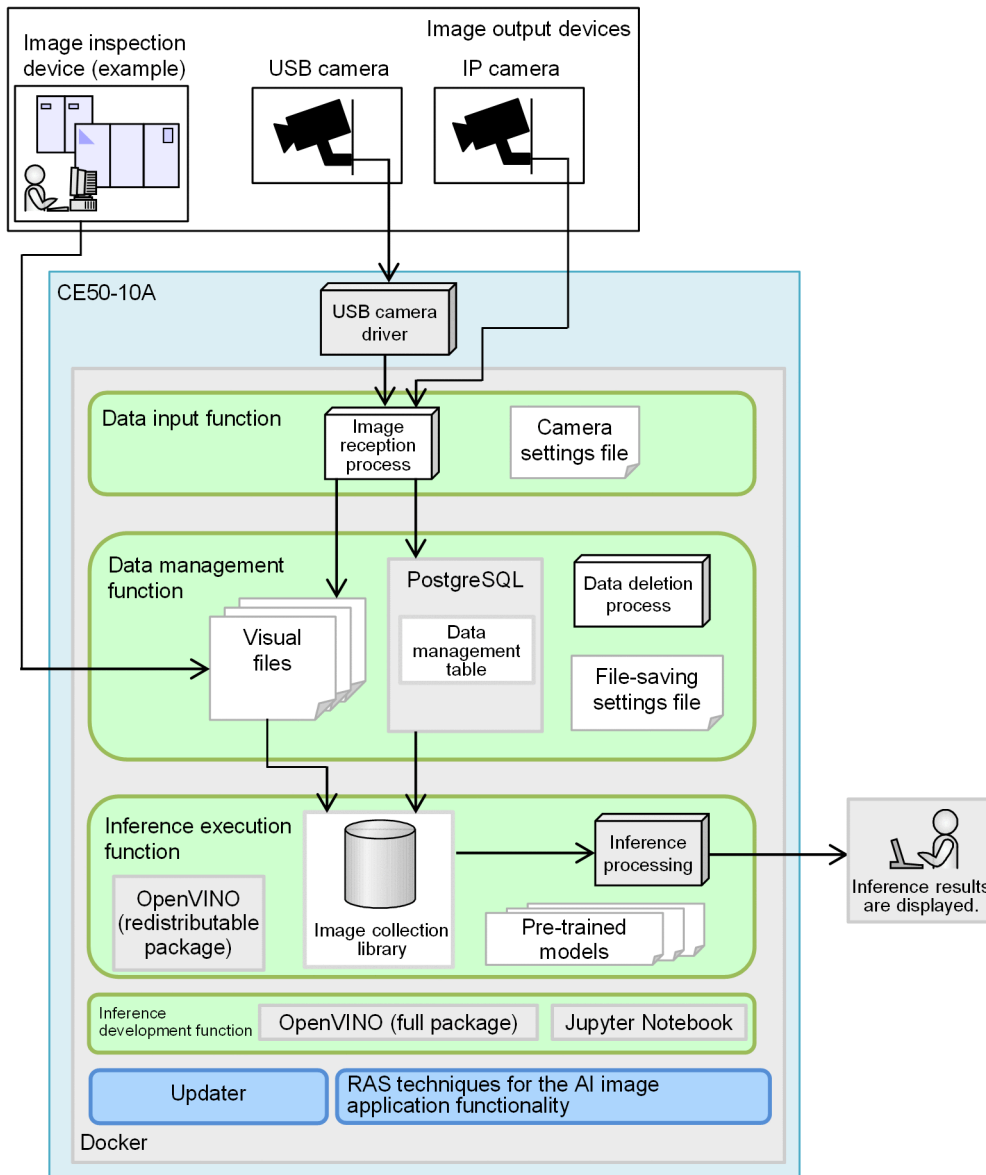
1.1 What can be done with CE50-10A

CE50-10A performs AI inference processing (abbreviated hereinafter to *inference processing*) by using image data obtained from USB cameras, IP cameras, image inspection devices, and other devices. CE50-10A can automatically display the results of inference processing and the user can make an appropriate decision from the displayed results. CE50-10A also provides an environment in which the user can freely develop inference processing or implement user processing that uses inference results.

The user can build a variety of solutions in a short period of time by using inference processing with pre-trained models.

The following figure provides an overview of the functions provided by CE50-10A and the flow of image data.

Figure 1–1: Functions provided by CE50-10A and the flow of image data



Legend:
 : Function container : Function assisting CE50-10A operation
 —> : Flow of image data processing

For details about Docker, see *Container function (Docker)* in the *CE50-10 Instruction Manual*.

The AI image application functionality consists of the following functions:

- Data input function
- Data management function
- Inference execution function
- Inference development function
- RAS techniques for the AI image application functionality
- Updater

For an overview of each function, see 1.2 AI image application functionality.

1.2 AI image application functionality

The following table describes the functions of the AI image application functionality.

Table 1–1: AI image application functionality

No.	Function	Description	Reference
1	Data input function	This function receives image data from a USB camera or IP camera and generates video (moving image) files and picture (still image) files from the data (hereinafter, video files and picture files are collectively referred to as <i>visual files</i>). The visual files are generated based on the camera connection information in the settings file. This function then passes the visual files to the data management function.	5. Data input function
2	Data management function	This function manages the visual files passed from the data input function and the visual files forwarded by devices such as the image inspection device. This function then passes the managed visual files to the inference execution function in response to inference processing requests.	6. Data management function
3	Inference execution function	This function performs inference processing by using a user-created pre-trained model or a pre-trained model provided for OpenVINO.	<ul style="list-style-type: none"> • 1.2.1 OpenVINO • 7. Inference execution function
4	Inference development function	Users can use Jupyter Notebook, which is a standard tool used for data analysis and AI development, to develop their own inference processing methods.	8. Inference development function
5	RAS techniques for the AI image application functionality	The following RAS techniques are available for the AI image application functionality: <ul style="list-style-type: none"> • Detecting errors • Restarting containers • Configuring the collection of operating information 	9. RAS techniques for the AI image application functionality
6	Updater	The updater can update user-developed inference processing programs and software on a container basis.	10. Updater

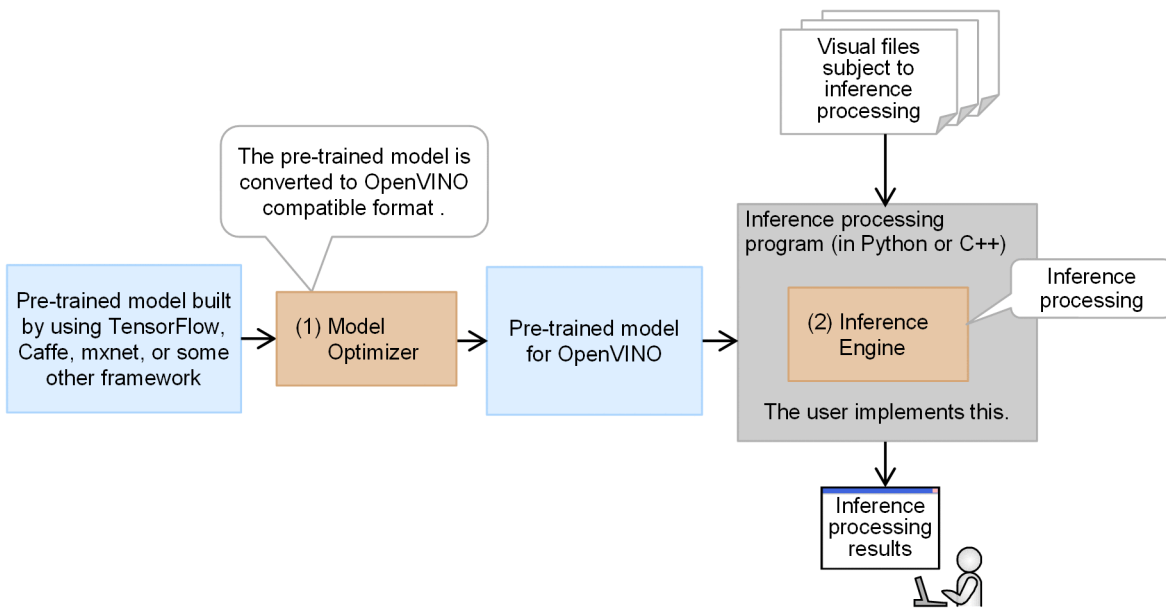
1.2.1 OpenVINO

OpenVINO is bundled with CE50-10A. OpenVINO is software that allows inference processing to run on the Intel architecture. OpenVINO consists of two components: Model Optimizer and Inference Engine. The roles of these components are as follows:

- **Model Optimizer**
This component optimally converts a pre-trained model (built by using TensorFlow, Caffe, mxnet, or some other framework) to an OpenVINO-compatible format for use during inference processing.
Pre-trained models that were built by using multiple frameworks can be made usable in OpenVINO by using Model Optimizer.
- **Inference Engine**
This component runs inference processing on the Intel architecture at high speed by using a pre-trained neural network model. OpenVINO is provided together with the Inference Engine software, which runs inference processing, and a library that calls the Inference Engine software. Users can call inference processing functions by using Python or C++.

The following figure shows how the two components work in OpenVINO.

Figure 1–2: How the two components work in OpenVINO



OpenVINO provides pre-trained models that were created by Intel. The following table shows some of the pre-trained models.

Table 1–2: Some of the pre-trained models provided by OpenVINO

No.	Model	Description
1	Object Detection	This model detects objects. For example, faces, people, pedestrians, and vehicles can be detected.
2	Object Recognition	This model recognizes objects. For example, the age and gender of a person, the direction of a face, the number on the registration plate of a vehicle, and the emotion of a person can be recognized.
3	Semantic Segmentation	This model associates a label or category with all of the pixels in an image.
4	Human Pose Estimation	This model detects the skeleton.
5	Image Processing	This model performs inference processing for an image. For example, the resolution of images can be enhanced by using this model.
6	Text Detection	This model detects text.
7	Text Recognition	This model recognizes text. For example, numbers, alphabetic characters, and Japanese characters can be recognized.
8	Action Recognition	This model recognizes actions. For example, the actions of a driver and sign language can be recognized.
9	Image Retrieval	This model completes an image.

2

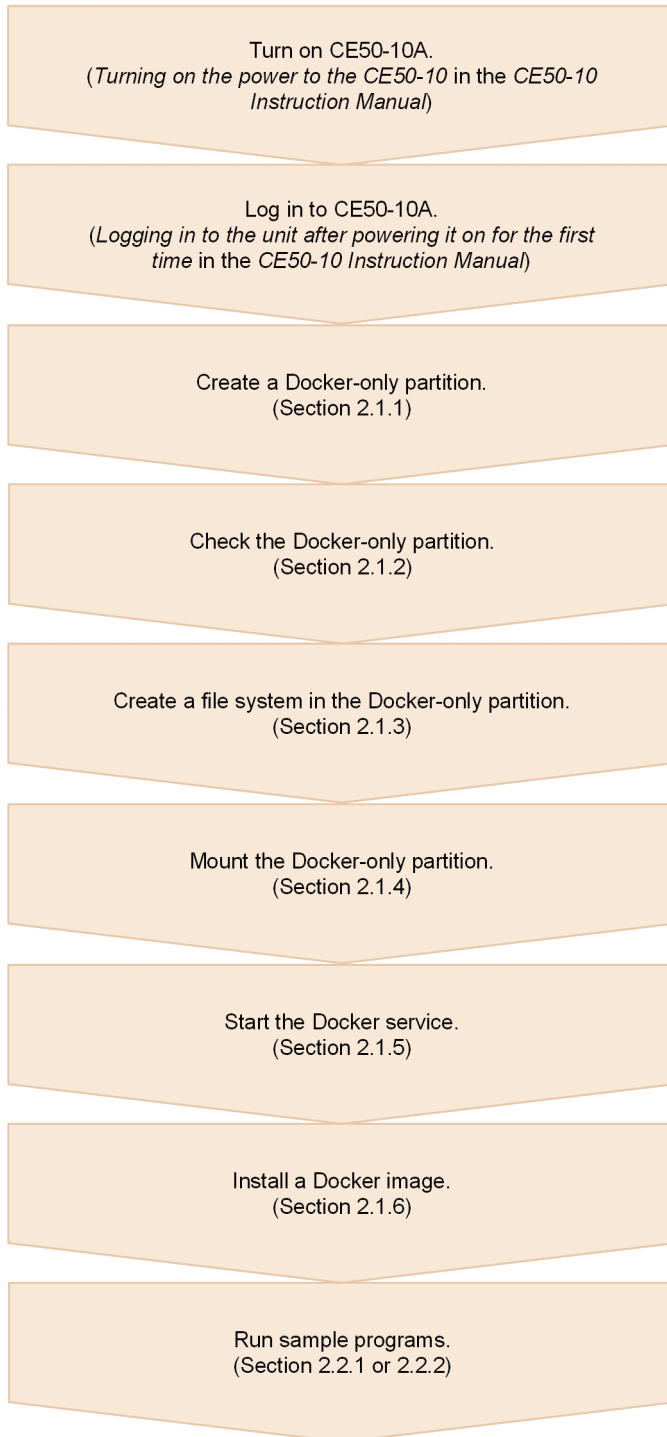
Quick guide for using the AI image application functionality

This chapter provides a quick guide for setting up the AI image application functionality and running sample programs for users who do not have experience in using the AI image application functionality.

2.1 Overview of experiencing the AI image application functionality

This section provides an overview of experiencing the AI image application functionality.

Figure 2–1: Overview of experiencing the AI image application functionality



Legend:

(...): Details are provided in the section enclosed in parentheses.

2.1.1 Creating a Docker-only partition

Perform the following procedure to create a 15-GB partition that will be used by Docker only. This partition is required to run sample programs.

1. Run the `gdisk` command with `/dev/sda` specified as an argument.

```
$ sudo gdisk /dev/sda
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.

Command (? for help):
```

2. Enter the `n` command, which creates a new partition.

```
Command (? for help):n
```

3. When the following appears, press the **Enter** key without entering anything:

```
Partition number (1-128, default 9):
```

4. When the following appears, press the **Enter** key without entering anything:

```
First sector (34-2047, default = 34) or {+-}size{KMGTP}:
```

5. Enter `+15G`, which specifies that a 15-GB partition will be created, and then press the **Enter** key.

```
Last sector (34-2047, default = 2047) or {+-}size{KMGTP}:+15G
```

6. When the following appears, press the **Enter** key without entering anything:

```
Current type is 'Linux filesystem'
Hex code or GUID (L to show codes, Enter = 8300):
```

7. Enter the `w` command, which starts a data write to the disk.

```
Command (? for help):w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!
```

8. When the following confirmation message appears, enter `Y`:

```
Do you want to proceed? (Y/N):Y
```

9. When the disk write is completed, the command displays the following message and then ends:

```
OK; writing new GUID partition table (GPT) to /dev/sda.
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot or after you
run partprobe(8) or kpartx(8)
The operation has completed successfully.
```

A Docker-only partition has been created.

2.1.2 Confirming the Docker-only partition

In 2.1.1 Creating a Docker-only partition, you created a Docker-only partition. Perform the following procedure to confirm that the created partition exists:

1. Run the following command to display a list of partitions as follows:

2. Quick guide for using the AI image application functionality

```
$ ls /dev/sda*
/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4 /dev/sda5 /dev/sda6 /dev/sda
7 /dev/sda8
```

2. Run the following command to notify the OS of the current partition configuration:

```
$ sudo partprobe
```

The OS recognizes the current partition configuration when the preceding command is run. You do not need to restart the OS.

3. Run the following command to display a list of partitions as follows:

```
$ ls /dev/sda*
/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4 /dev/sda5 /dev/sda6 /dev/sd
a7 /dev/sda8 /dev/sda9
```

If the partition that you created in 2.1.1 Creating a Docker-only partition is recognized by the OS as the Docker-only partition, `/dev/sda9` is included in the list of partitions.

2.1.3 Creating a file system in the Docker-only partition

Now you need to create a file system in the partition that you created in 2.1.1 Creating a Docker-only partition.

1. Run the following command to create an EXT4 file system as follows:

```
$ sudo mkfs -t ext4 /dev/sda9
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 4096 1k blocks and 1024 inodes

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
```

2.1.4 Mounting the Docker-only partition

Perform the following procedure to mount the Docker-only partition that you created in 2.1.1 Creating a Docker-only partition:

1. Stop Docker.

If Docker is already stopped, go to step 2.

```
$ sudo systemctl stop docker
```

2. Mount the Docker-only partition that you created to `/var/lib/docker`.

```
$ sudo mount /dev/sda9 /var/lib/docker
```

2.1.5 Starting the Docker service

Perform the following procedure to start the Docker service:

1. Run the following command to start Docker:

```
$ sudo systemctl start docker
```

2.1.6 Installing a Docker image

Perform the following procedure to install a Docker image:

1. Run the following command to install a Docker image:

```
$ sudo /hitachi/ctrl_edge_ai/bin/install_edge_ai.sh
```

2.2 Description of sample programs

The procedures for using the AI image application functionality are described later. In these procedures, you will use sample programs and sample Compose files.

CE50-10A comes with sample programs that perform the following processes:

- Import a sample video, display the number of people who appear in the video, and control the indication state of the AP indicator according to the number of people detected. (For details about the AP indicator, see *RAS indicator (AP, E3, E2, E1)* in the *CE50-10 Instruction Manual*.)
- Import a live video from a USB camera, display the number of people who appear in the video and control the indication state of the AP indicator according to the number of people detected.

The sample video and the live video from the USB camera can be viewed on a display.

The sample programs control the AP indicator, so confirm that no other process is controlling the AP indicator.

Note that the AP indicator stays in the state set by a sample program after the program ends. Therefore, if necessary, run the following command to turn off the AP indicator:

```
$ sudo rasledctl -off
```


See the next sections, which describe the procedures for running the sample programs.

2.2.1 Running the sample program that imports a sample video

1. Run the following command to start the desktop:

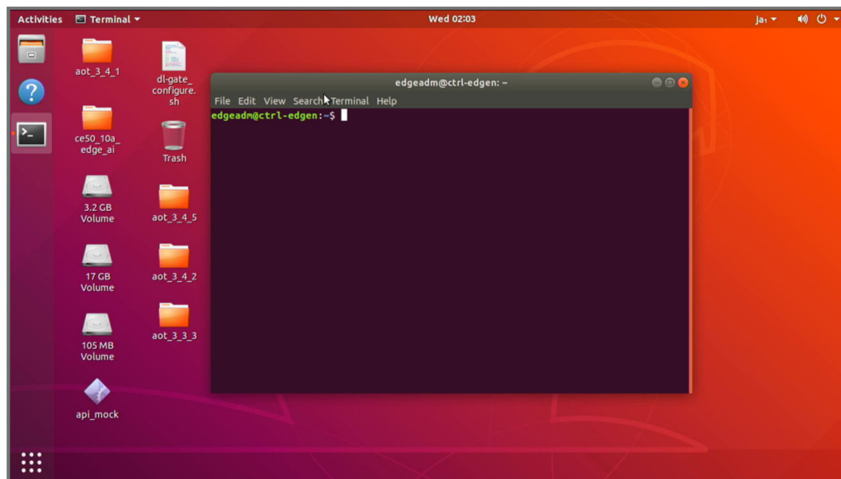
```
$ sudo systemctl start gdm3
```

2. In the login window that appears, log in as the user `edgeadm`.
3. Perform the following procedure to open Terminal:

1. Click  at the bottom left.
2. Click **All**.
3. Click **Terminal**.



Terminal opens.



4. Run the following command to permit the programs on Docker to connect to the X server:

```
$ xhost local:
```

5. Download a sample video file by referring to 7.2.1 How to implement inference processing, rename the downloaded file to `sample_video.mp4`, and then save the file in the following location:

```
/hitachi/ctrl_edge_ai/sample/people_count/people_count_sample_via_video/component/sample_video.mp4
```

6. Run the following command to start the AP indicator operation process:

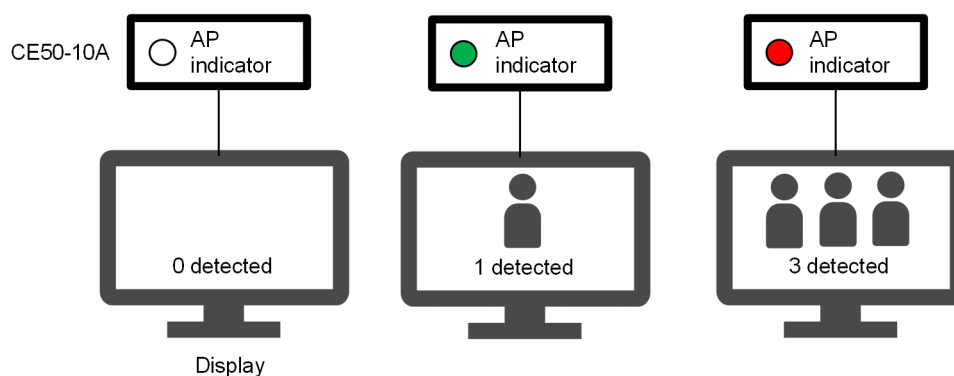
```
$ sudo /hitachi/ctrl_edge_ai/sample/people_count/start_AP
```

7. Run the following commands to start the container for the sample program:

```
$ cd /hitachi/ctrl_edge_ai/sample/people_count/people_count_sample_via_video
$ sudo docker-compose up -d
```

8. Check the execution results.

Figure 2–2: Examples of cases where the sample program is run (by using a sample video)



9. To stop the sample program, run the following commands:

```
$ sudo docker-compose down
$ sudo killall AP
```

2.2.2 Running the sample program that imports a live video from a USB camera


1. Connect the cable of the USB camera to a USB port of CE50-10A.
2. Run the following command to start the desktop:

2. Quick guide for using the AI image application functionality

```
$ sudo systemctl start gdm3
```

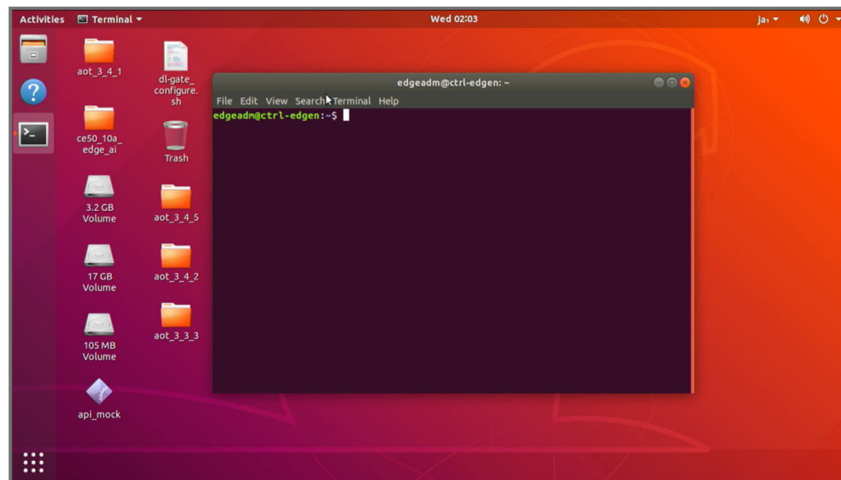
3. In the login window that appears, log in as user `edgeadm`.

4. Perform the following procedure to open Terminal:

1. Click  at the bottom left.
2. Click **All**.
3. Click **Terminal**.



Terminal opens.



5. Run the following command to permit the programs on Docker to connect to the X server:

```
$ xhost local:
```

6. Run the following command to start the AP indicator operation process:

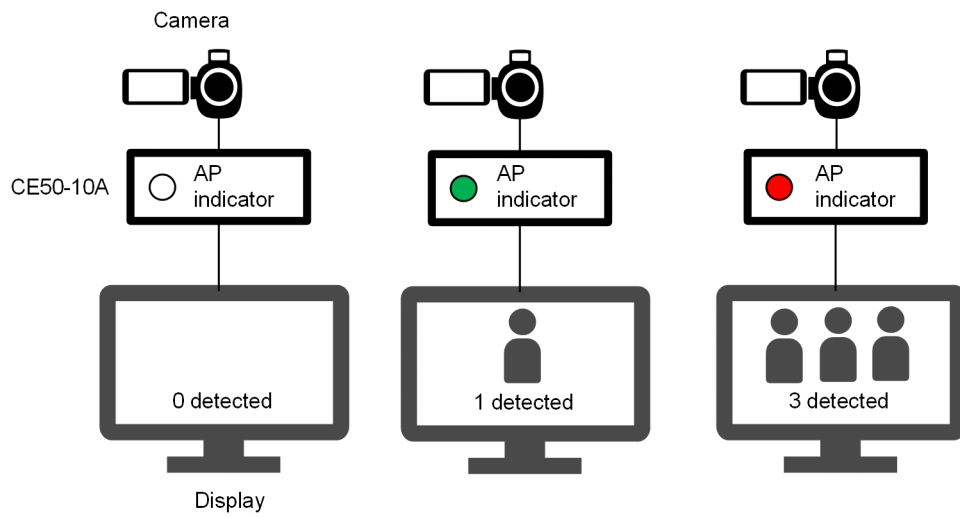
```
$ sudo /hitachi/ctrl_edge_ai/sample/people_count/start_AP
```

7. Run the following commands to start the sample program:

```
$ cd /hitachi/ctrl_edge_ai/sample/people_count/people_count_sample_via_usbcam  
$ sudo docker-compose up -d
```

8. Check the execution results.

Figure 2–3: Examples of cases where the sample program is run (by using a USB camera)



9. To stop the sample program, run the following commands:

```
$ sudo docker-compose down  
$ sudo killall AP
```

3

Setting up the AI image application functionality

This chapter describes how to set up the AI image application functionality.

3.1 Procedure for setting up the AI image application functionality

This section describes the how to perform the tasks that are required to use the AI image application functionality.

3.1.1 Creating and mounting a Docker-only partition

Before you can use Docker, you must create a Docker-only partition and then mount it on CE50-10A.

(1) Creating a Docker-only partition

Create a Docker-only partition whose size is 15 GB or more.

For details about how to create a Docker-only partition, see *Creating partitions* in the *CE50-10 Instruction Manual*.

(2) Setting up automatic mounting

The Docker-only partition can be automatically mounted when CE50-10A starts. To set up automatic mounting, you must specify the definition in the mount definition file (`/hitachi/etc/fsconf`).

For details about how to specify the definition, see *Specifying the definition to automatically mount the application area* in the *CE50-10 Instruction Manual*.

3.1.2 Starting the Docker service

By default, the Docker service does not automatically start when the OS starts. To temporarily start the Docker service, run the following command:

```
$ sudo systemctl start docker
```

After you run this command, the Docker service will not start when the OS restarts. To start the Docker service again, run the preceding command again.

If you want the Docker service to start automatically when the OS starts, run the following command:

```
$ sudo systemctl enable docker
```

After you run this command, the Docker service will automatically start each time the OS restarts.

3.1.3 Installing the Docker image

Run the following command to install a Docker image that provides the AI image application functionality. When you run the command, make sure that the OS operation mode is set to normal mode.

```
$ sudo /hitachi/ctrl_edge_ai/bin/install_edge_ai.sh
```

After the Docker image is installed in the Docker-only partition, you can delete the Docker image. Note that you might need to use the Docker image again if an error (such as corruption) occurs on the partition. Therefore, we recommend that, before deleting the Docker image, you back it up to a USB memory drive or other external storage media.

3.2 Displaying the partition information

Display the partition information to confirm that the partition you created in (1) Creating a Docker-only partition exists. You can perform either of the following two procedures.

Procedure 1:

1. Run the `gdisk` command with `/dev/sda` specified as an argument.

```
$ sudo gdisk /dev/sda
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Command (? for help):
```

2. Enter the `p` command, which displays the partition information.

```
Command (? for help):p
Disk /dev/sda: 15728640 sectors, 7.5 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 6A423A3C-EFD1-448C-AA4F-70DB2D2D757B
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15728606
Partitions will be aligned on 2048-sector boundaries
Total free space is 2014 sectors (1007.0 KiB)
Number Start (sector) End (sector) Size Code Name
1 2048 15728606 7.5 GiB 8300 Linux filesystem
```

Note

The underlined portion is the partition information.

3. Enter `q` to quit the processing.

```
Command (? for help):q
```

Procedure 2:

1. Run the `gdisk` command with `/dev/sda` specified as an argument and with the `-l` option specified.

The `gdisk` command displays the partition information and ends immediately without displaying a prompt.

```
$ sudo gdisk /dev/sda -l
Disk /dev/sda: 15728640 sectors, 7.5 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 6A423A3C-EFD1-448C-AA4F-70DB2D2D757B
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 15728606
Partitions will be aligned on 2048-sector boundaries
Total free space is 2014 sectors (1007.0 KiB)
Number Start (sector) End (sector) Size Code Name
1 2048 15728606 7.5 GiB 8300 Linux filesystem
```

Note

The underlined portion is the partition information.

4

Design of the AI image application functionality

This chapter describes how to customize the AI image application functionality and how to start and stop the functionality.

4.1 Using the Compose file to customize the functionality

You can customize the settings in the Compose file so that the AI image application functionality can be used in different cases. The Compose file is in YAML format. In YAML format, indentation is used to express the hierarchical data structure.

Example of the Compose file

In the following example, the numbers #*n* along the right side correspond to the item numbers in Table 4–1: Setting items in the Compose file.

```

version: '3' #1
services: #2
  data_input: #3
    ipc: host #4
    image: ctrl_edge_ai/data_input:1.0 #5
    network_mode: host #6
    restart: on-failure:5 #7
    environment: #8
      DISPLAY: $DISPLAY
    volumes: #9
      - /tmp/.X11-unix:/tmp/.X11-unix
      - input_volume:/input
      - /home/edgeadm/share:/share
      - /home/edgeadm/data_input.json:/config/data_input.json
    depends_on: #10
      - openvino_prod
    logging: #11
      driver: syslog
      options:
        syslog-facility: daemon
        tag: docker-compose/{{.Name}}/{{.ID}}
    devices: #12
      - "/dev/dri:/dev/dri"
      - "/dev/video0:/dev/video0"
  data_manager:
    image: ctrl_edge_ai/data_manager:1.0
    network_mode: host
    restart: on-failure:5
    volumes:
      - input_volume:/input
      - storage_volume:/storage
      - /home/edgeadm/share:/share
    logging:
      driver: syslog
      options:
        syslog-facility: daemon
        tag: docker-compose/{{.Name}}/{{.ID}}
  openvino_prod:
    ipc: host
    image: ctrl_edge_ai/openvino_prod:1.0
    network_mode: host
    restart: on-failure:5
    environment:
      DISPLAY: $DISPLAY
    volumes:
      - /tmp/.X11-unix:/tmp/.X11-unix
      - storage_volume:/storage
      - /home/edgeadm/share:/share
    depends_on:
      - data_manager
    logging:
      driver: syslog
      options:
        syslog-facility: daemon
        tag: docker-compose/{{.Name}}/{{.ID}}
    devices:
      - "/dev/dri:/dev/dri"
  volumes: #13
    input_volume: #14
      driver: local #15
      driver_opts: #16
        type: tmpfs
        device: tmpfs
        o: "size=512m"
    storage_volume:
      driver: local

```

The following table describes the items set in the Compose file.

Table 4–1: Setting items in the Compose file

No.	Setting item	Description
1	<code>version:</code>	Specifies the Compose file version used by the <code>docker-compose</code> command. The <code>docker-compose</code> command provided by this product is compatible with version 3.
2	<code>services:</code>	Indicates that subsequent lines specify service definitions.
3	<code>data_input:</code> <code>data_manager:</code> <code>openvino_prod:</code>	Items at this hierarchical level define service names. You can define any names you like. In this example, three services are defined: <ul style="list-style-type: none"> • Data input function: <code>data_input</code> • Data management function: <code>data_manager</code> • Inference execution function: <code>openvino_prod</code>
4	<code>ipc:</code>	If shared memory is to be used, this item specifies what the same memory space is to be shared with. This item is used when X forwarding is to be performed. The AI image application functionality supports only the specification of <code>host</code> (meaning that the memory is shared with the host).
5	<code>image: image-name:tag-name</code>	Specifies an image name and tag name.
6	<code>network_mode: host</code>	Specifies the virtual network mode. In this example, data is shared with the host.
7	<code>restart: on-</code> <code>failure[:max_retries]</code>	Specifies how the functionality behaves when it restarts. As in this example, if <code>on-failure</code> is specified, the functionality attempts to restart if the container terminates abnormally (if the end status is not 0). The maximum number of times the functionality attempts to restart is specified by <code>max_retries</code> . If <code>max_retries</code> is not specified, the functionality continues to attempt to restart until it succeeds.
8	<code>environment:#</code>	Specifies environment variables. In the example of the Compose file shown earlier, the host environment variable <code>DISPLAY</code> is passed to the container so that the content displayed by the GUI in the container is forwarded (by X forwarding) to the host.
9	<code>volumes:#</code>	Defines volumes or specifies the volumes to be mounted. If the name of a volume is defined in <code>volume-name</code> format, the defined volume will be used. <ul style="list-style-type: none"> • A definition in <code>host-path:container-path[:access-mode]</code> format enables file sharing between the host and container. • A definition in <code>volume-name:container-path[:access-mode]</code> format enables file sharing between containers via the specified volume. In the example, the <code>share</code> directory is set to share the <code>X11-unix</code> directory and files so that X forwarding can be performed.
10	<code>depends_on:</code>	Specifies the dependency of the container. In the example, <code>data_manager</code> , <code>openvino_prod</code> , and <code>data_input</code> are started in this order.
11	<code>logging:</code>	Specifies logging-related settings. For details, see 9. RAS techniques for the AI image application functionality.
12	<code>devices:</code>	Allocates host devices to a container.

4. Design of the AI image application functionality

No.	Setting item	Description
12	<code>devices:</code>	<ul style="list-style-type: none"> <code>host-device: container-device</code> <p>In the example, the <code>data_input</code> container uses a GPU (graphical processing unit) and USB camera, and the <code>openvino_prod</code> container uses a GPU.</p>
13	<code>volumes:</code>	Indicates that subsequent lines specify volume definitions.
14	<code>input_volume:</code> <code>storage_volume:</code>	<p>Items at this hierarchical level define volume names.</p> <p>In the example, the following two volumes are created:</p> <ul style="list-style-type: none"> <code>input_volume</code>: Volumes used by the data input function to manage data <code>storage_volume</code>: Volume used by the data management function to perform inference
15	<code>driver:</code>	<p>Specifies a volume driver.</p> <p>The AI image application functionality supports only the specification of <code>local</code> (the local volume).</p>
16	<code>driver_opts:</code>	<p>Specifies driver options.</p> <p>The main memory is set as the data storage destination by additionally setting the following options:</p> <pre>type: tmpfs device: tmpfs o: "size=512m"</pre> <p>The <code>size=512m</code> setting limits data storage to a maximum of 512 MB.</p> <p>The <code>o:</code> option can be omitted. However, we recommend that you set an upper limit to prevent data from consuming too much of the main memory.</p>

#

To forward the content displayed by the GUI in a container to the host so that the content is displayed on the host, add the following settings to the appropriate container in the Compose file:

```
ipc: host
environment:
  DISPLAY: $DISPLAY
volumes:
  - /tmp/.X11-unix:/tmp/.X11-unix
```

4.2 Procedure for setting up the automatic startup of Docker containers during OS startup

This section describes how to set up the automatic startup of Docker containers during OS startup.

4.2.1 Procedure for setting up the automatic startup of the Docker service

To set up the automatic startup of the Docker service, use `fstmount` to specify the settings to automatically mount the Docker-only partition. For details, see 3.1.1 Creating and mounting a Docker-only partition.

Then, run the command that enables the automatic startup of the Docker service. For details, see 3.1.2 Starting the Docker service.

4.2.2 Procedure for enabling the automatic startup of Docker containers

Use the following procedure to set up the automatic startup of Docker containers during OS startup. Note that this procedure uses `systemd`.

1. Create a `systemd` script file.

The following is an example script. For details about the setting items to be specified, see *Setting start and stop of application programs* in the *CE50-10 Instruction Manual*.

When you assign a name to the script file, make sure that the name is related to the functionality of the application to be run (for example, `people_count.service`).

```
[Unit]
Description=Docker container
Requires=docker.service
After=docker.service
[Service]
ExecStart=/usr/local/bin/docker-compose -f /home/edgeadm/work/docker-compose.yaml up
Type=simple
[Install]
WantedBy=edge-normal.target
```

For the `-f` option of the `docker-compose` command, specify the absolute path of the Compose file. (In the preceding example, the absolute path of the Compose file is `/home/edgeadm/work/docker-compose.yaml`.)

2. Enable automatic startup.

After you have created the `systemd` script file, store it in `/lib/systemd/system`, and then run the following command:

```
$ sudo systemctl enable systemd-script-file-name (.service is omissible)
```

Example:

```
$ sudo systemctl enable people_count
```

To disable automatic startup, run the following command:

```
$ sudo systemctl disable systemd-script-file-name (.service is omissible)
```

4.3 How to start and stop the AI image application functionality

This section describes how to start and stop the Docker container that provides the AI image application functionality.

To start and stop the Docker container, use the `docker-compose` command.

To run the `docker-compose` command, a Compose file is required.

By default, the command reads the current Compose file (`docker-compose.yaml` or `docker-compose.yml`), and then starts or stops the Docker container as specified in the file. For details about the Compose file, see 6.4.2 Compose file settings.

4.3.1 How to start Docker containers

You can start Docker containers by running the `docker-compose` command with `up` or `start` specified.

To generate and start a new Docker container, specify `up`. To start a Docker container that has already been generated, specify `start`.

To generate and start all Docker containers specified in the Compose file:

To start all Docker containers as daemons, use the `-d` option.

```
$ sudo docker-compose up [-d]
```

To generate and start a Docker container specified (as a service name) in the Compose file:

To start a Docker container as a daemon, use the `-d` option.

```
$ sudo docker-compose up [-d] service-name
```

To connect to a Docker container that is already running and then run the command on the Docker container:

```
$ sudo docker-compose exec service-name command
```

To start all Docker containers that are stopped:

```
$ sudo docker-compose start
```

When the OS restarts, all Docker containers that are running will be stopped. If the automatic startup of Docker containers is disabled, after the OS restarts, specify `start` to restart the Docker containers.

4.3.2 How to stop Docker containers

You can stop Docker containers by running the `docker-compose` command with `down` or `stop` specified.

To stop and delete all Docker containers, specify `down`. To stop a specific Docker container or all Docker containers, specify `stop`.

To stop and delete all Docker containers specified in the Compose file:

```
$ sudo docker-compose down
```

To stop a Docker container specified (as a service name) in the Compose file:

```
$ sudo docker-compose stop service-name
```

To stop all Docker containers specified in the Compose file:

```
$ sudo docker-compose stop
```

The Docker containers stopped by specifying `stop` remain stopped after the OS restarts.

4.3.3 How to check the status of the Docker containers

You can check the status of the Docker containers by specifying the `ps` option.

```
$ sudo docker-compose ps
```

Example of execution results (returned when the command was run with `up` specified):

Name	Command	State	Ports
edgeadm_test_1	/bin/bash	Up	

`Up` is displayed in the `State` column.

Example of execution results (returned when the command was run with `down` specified):

Name	Command	State	Ports
------	---------	-------	-------

Only the header is displayed.

Example of execution results (returned after the command was run with `up` specified and then run with `stop` specified):

Name	Command	State	Ports
edgeadm_test_1	/bin/bash	Exit	0

`Exit` is displayed in the `State` column.

Example of execution results (returned after the command was run with `stop` specified and then run with `start` or `up` specified):

Name	Command	State	Ports
edgeadm_test_1	/bin/bash	Up	

`Up` is displayed in the `State` column.

4.4 Notes on designing a system

4.4.1 Provided container images

The following table lists the Docker container images for the functions provided by the AI image application functionality.

Table 4–2: Provided container images

No.	Function	Repository	Container image	Tag	Redistributable?#
1	Data input function	ctrl_edge_ai	data_input	1.0	Yes
2	Data management function		data_manager	1.0	Yes
3	Inference execution function		openvino_prod	1.0	Yes
4	Inference development function		openvino_devel	1.0	No

#

Before using OpenVINO provided by CE50-10A, you must agree to the end user license agreement (EULA) that appears when you first log in to CE50-10A.

The OpenVINO license prohibits the redistribution of OpenVINO with the exception of certain components. For example, suppose you develop software by using OpenVINO and then embed the software in CE50-10A. In that case, delivering CE50-10A to a customer is redistribution.

Before redistributing container images, delete any non-redistributable images by running the following commands:

```
$ sudo docker rmi ctrl_edge_ai/openvino_devel:1.0
$ sudo rm -f /hitachi/ctrl_edge_ai/docker_images/openvino_devel_1.0.tar.gz
```

4.4.2 Automatic restart of Docker containers

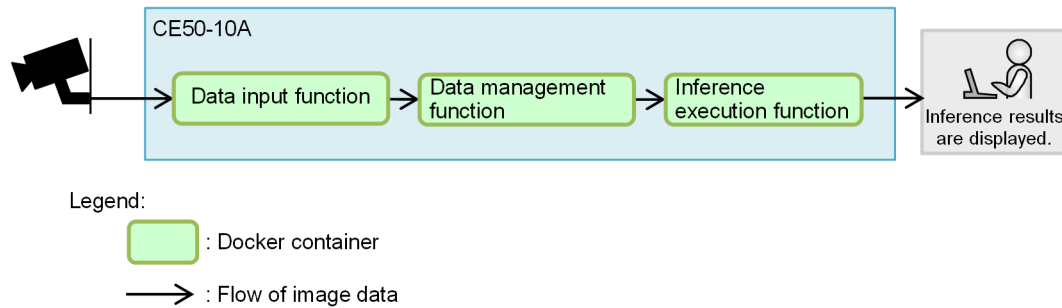
If an error occurs in the data input function, data management function, or inference execution function during operation of CE50-10A, the container of the relevant function ends. You can set the `restart` option in the Compose file so that a container that ends due to an error will automatically restart.

For details about how to set the `restart` option, see 4.1 Using the Compose file to customize the functionality.

However, sometimes an error cannot be corrected by restarting the container (for example, if the camera fails, and the data input function cannot obtain image data). In such a case, the container of the function attempts to restart for the preset number of times and then ends. If you encounter such a case, remove the cause of the error that occurred, and then manually restart the container.

The following figure shows the configuration of containers used during operation of CE50-10A.

Figure 4-1: Configuration of the containers used during operation of CE50-10A



The following describes how each function behaves before it recovers from an error that occurred and the status of image data in the period before recovery.

- If an error occurs in the container of the data input function
If an error (such as one that prevents the reception of image data from a camera) occurs in the container of the data input function, the container ends. The containers of the data management function and the inference execution function are placed on standby until they can receive image data. After the error is corrected by restarting the container, processing restarts. Any image data in the period of time from when the error occurred until the container was restarted will be lost.
- If an error occurs in the container of the data management function
If an error occurs in the container of the data management function, the container ends. If the container of the data management function ends, the containers of the data input function and the inference execution function will also end because they will no longer be able to access the container of the data management function. After the error is corrected by restarting the container, processing restarts. Any image data in the period of time from when the error occurred until the container was restarted will be lost.
- If an error occurs in the container of the inference execution function
If an error occurs in the container of the inference execution function, the container ends. While the container of the inference execution function is stopped, the container of the data management function continues to accumulate image data. Therefore, after the container of the inference execution function restarts, it starts performing inference for the files in the container of the data management function in the order in which they were registered.

4.4.3 Tuning the number of obtained frames

If the frame per second (FPS) of the data input function is greater than the maximum FPS that the inference execution function can handle, the inference execution function will not be able to process all of the frames. As a result, the capacity of the data management function might be exceeded. If the amount of data accumulated by the data management function exceeds the preset maximum, the function deletes the data in order from the oldest data. This prevents errors in the data management function. However, it is inefficient to discard image data that was received but cannot be processed. Therefore, we recommend that, before starting operation of CE50-10A (for example, in the development and testing phase), you check the performance of the inference execution function and configure (tune) the FPS settings of the data input function and inference execution function. Specifically, make sure that the FPS at which the data input function registers data in the data management function is less than the FPS of the inference execution function. For details about the FPS at which the data input function registers data, see the description of the `output > video > fps` setting in 5.3.1 Settings of the data input function.

4.4.4 Firewall settings

The following table shows the communications that the AI image application functionality must perform.

4. Design of the AI image application functionality

Table 4–3: Communications the AI image application functionality must perform

No.	Function	Necessary communication	Direction of communication (send/receive)	Remarks
1	Data input function (if an IP camera is used)	Access to the RTSP port [#] of the IP camera	Send	Unnecessary if using a USB camera
2	Data management function	API port (TCP/13579)	Send	--
3	Sample application	API port (TCP/5000)	Send	Required only when using the sample application
4	Inference development function	Jupyter Notebook port (8888/tcp)	Receive	--

Legend:

--: Not applicable

#

The Internet Assigned Numbers Authority (IANA) has stipulated that 554/tcp or 554/udp be used.

However, as a countermeasure against port attacks, some commercially available IP cameras use port numbers other than the stipulated port numbers.

If you use a firewall, consider the following points:

- To use an IP camera, transmission to the RTSP port of the IP camera must be permitted. (Transmission is prohibited by default.)
For the port number of the RTSP port of the IP camera, see the documentation for the IP camera. (It might be possible to change the port number.)
- For the APIs of the data management function and sample application, transmission to the local host and reception from the local host are permitted by default. Therefore, you do not need to specify settings to permit these communications.

Table 4–4: List of firewall functions

No.	Rule	Summary	Default
1	Permit reception through ports	Permits reception through specified ports and blocks all the other communications. Security is enhanced by opening only the ports that need external access.	Disabled
2	Permit transmission through ports	Permits transmission through specified ports and blocks all the other communications.	Transmission through the following ports is permitted: <ul style="list-style-type: none"> • SSH (22/tcp) • DNS (53/udp) • HTTP (80/tcp) • NTP (123/tcp) • HTTPS (443/tcp)
3	Permit reception through ports (Restricted)	Permits reception through specified ports but limits the maximum number of connections that can be established within a certain period of time. This rule is effective as protection from the following attacks: <ul style="list-style-type: none"> • DoS (Denial of Service) attacks • DDoS (Distributed Denial of Service) attacks 	Limits incoming SSH (22/tcp) connections to a maximum of 10 per minute

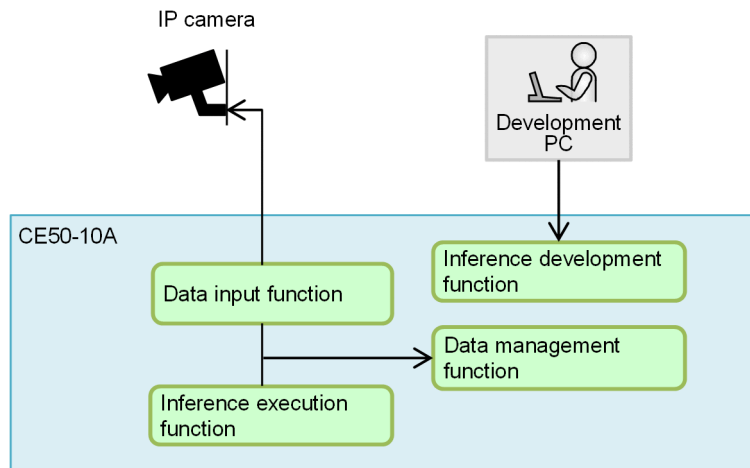
No.	Rule	Summary	Default
3	Permit reception through ports (Restricted)	<ul style="list-style-type: none"> Brute-force attacks 	Limits incoming SSH (22/tcp) connections to a maximum of 10 per minute
4	Permit transmission and reception by the local host	Permits communications with the local host (127.0.0.1) via any ports.	Enabled

4.4.5 Security risks and countermeasures

Depending on the data to be processed by the inference processing program or the environment in which CE50-10A is used, security risks that have not yet been encountered might arise. The countermeasures that must be taken might change according to the new risks. If you encounter a new risk, consider all possible countermeasures.

The following figure shows the access routes for the network communications of CE50-10A.

Figure 4-2: CE50-10A network communication access routes



Legend:

→ : Access route

The following table shows the security risks that might arise during use of CE50-10A and the countermeasures to be taken for CE50-10A.

Table 4-5: Security risks and countermeasures for each component

No.	Component	Data type	Security risk			Measures and considerations
			If intercepted	If altered	If lost	
1	Inference development function	Communication for using Jupyter Notebook	Videos, images, and other types of data might be revealed.	The program might no longer work correctly.	The inference development function would become unusable.	<ul style="list-style-type: none"> Authentication using a one-time password is performed when the user logs in to Jupyter Notebook. Encryption for communication routes is not supported. We recommend using this function in an internal network or other network in which wiretapping and

4. Design of the AI image application functionality

No.	Component	Data type	Security risk			Measures and considerations
			If intercepted	If altered	If lost	
1	Inference development function	Communication for using Jupyter Notebook	Videos, images, and other types of data might be revealed.	The program might no longer work correctly.	The inference development function would become unusable.	falsification occur less frequently.
2	IP camera	Videos and images	Videos, images, and other types of data might be revealed.	The inference execution function might output incorrect results.	The reception of videos and images would become impossible.	<ul style="list-style-type: none"> Measures are needed on the IP-camera side. (Measures on the CE50-10A side are unnecessary.) Use an IP camera that supports authentication by using IDs and passwords. The data input function supports authentication by using IDs and passwords. Encryption for communication routes is not supported. We recommend using this function in an internal network or other network in which wiretapping and falsification occur less frequently.
3	Data management function	Data generated by CE50-10A	Videos, images, and other types of data might be revealed.	The inference execution function might output incorrect results.	Received videos and images would be lost.	<ul style="list-style-type: none"> The data management function does not provide an authentication function. Access to the data management function is blocked by a firewall.

5

Data input function

This chapter provides an overview of the data input function. This chapter also describes how to connect a camera and set up the function.

5.1 Overview of the data input function

The data input function controls the following processes:

- The data input function obtains visual data from an IP camera or USB camera and outputs the data as visual files. It then passes the files to the data management function.
- The data input function provides a preview of the visual data captured by the connected camera on a monitor. This can be used to verify camera operation.

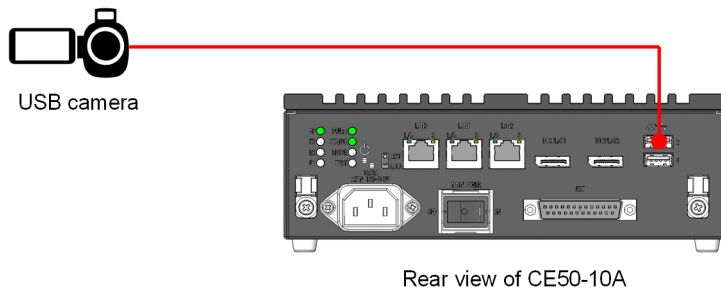
5.2 How to connect a camera

This section describes how to connect a camera to CE50-10A.

5.2.1 Connecting a USB camera

To use a USB camera, connect the camera cable to a USB port of CE50-10A.

Figure 5–1: Connecting a USB camera



5.2.2 Connecting an IP camera

An IP camera can be connected in the following two ways:

- Directly connect the LAN ports of an IP camera and CE50-10A with a LAN cable.
- Connect an IP camera and CE50-10A to a network so that they can directly communicate with each other through switching hubs and/or routers.

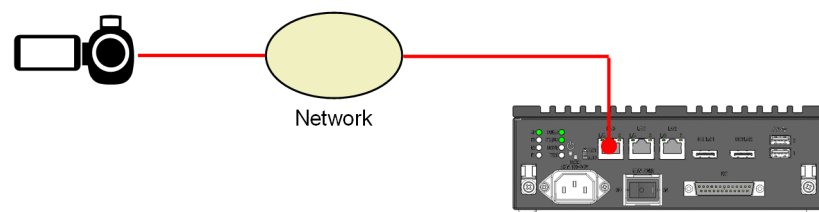
For an IP camera, assign an IP address.

Figure 5–2: Connecting an IP camera

- Direct connection between LAN ports



- Connection via a network



5.3 How to specify the settings of the data input function

This section describes the settings of the camera that captures visual data, the settings related to the generated visual files, and the settings for previewing the visual data captured by the camera.

5.3.1 Settings of the data input function

The settings of the data input function are specified in JSON format. In the container of the data input function, specify the settings for Docker so that they can be accessed at the following path:

```
/config/data_input.json
```

For details about the settings to be specified on the container side, see 5.3.2 Container settings.

The following table describes the setting items to be specified.

If multiple instances of the same item are specified in the settings file, the last specification overrides the others.

Table 5–1: Setting items of the data input function

No.	Item	Required or optional	Description
1	camera	Required	Specifies the settings related to the camera. For details about this item, see Table 5–2: Sub-items of camera.
2	output	Optional	Specifies the settings for storing received visual data in the data management function. If this item is omitted, the data input function does not output visual data. For details about this item, see Table 5–6: Sub-items of output.
3	preview	Optional	Specifies the settings for previewing the visual data captured by the camera. If this item is omitted, the data input function does not provide a preview. For details about this item, see Table 5–8: Sub-item of preview.

Table 5–2: Sub-items of camera

No.	Item	Required or optional	Description
1	name#	Required	Specifies a camera ID. Specify an ID that uniquely identifies the camera.
2	type	Required	Specifies the type of the camera to be used. Specify either of the following values: "usb": USB camera "ip": IP camera
3	connect	Required	Specifies the camera connection information. For details about how to specify the camera connection information, see Table 5–3: Sub-items of connect USB cameras (type="usb") or Table 5–4: Sub-items of connect for IP cameras (type="ip").
4	video	Required	Specifies the information about video data to be received from the camera. For details about this item, see Table 5–5: Sub-items of camera-video.

#

The camera ID can consist of alphabetic characters and underscores (_) only. The camera ID can have no more than 31 characters.

Table 5–3: Sub-items of connect USB cameras (type="usb")

No.	Item	Required or optional	Description
1	device	Required	Specifies the path of the device file for the USB camera.

Table 5–4: Sub-items of connect for IP cameras (type="ip")

No.	Item	Required or optional	Description
1	rtsp_url	Required	Specifies the RTSP URL of the IP camera. If authentication using an ID and password is performed, include the ID and password in the URL. For details, see <i>Example of the settings for receiving visual data from an IP camera</i> .
2	latency	Required	Specifies the buffer time (in milliseconds) for the visual data received from the IP camera.

Table 5–5: Sub-items of camera-video

No.	Item	Required or optional	Description
1	width	Required	Specifies the frame width (in pixels) that is used to determine the frame resolution.#
2	height	Required	Specifies the frame height (in pixels) that is used to determine the frame resolution.#
3	fps	Required	Specifies the number of visual data frames to be received per second.#
4	codec	Required	Specifies the codec to be used for the received video. Specify one of the following values: <ul style="list-style-type: none"> • "YUYV": Non-compressed • "MJPEG": Motion-JPEG • "H264": H.264

#

- The specifiable resolution and frame rate ranges are as follows:
Resolution: 640 x 480 to 3,840 x 2,160
Frame rate: 1 to 30
- If you specify a value that is not supported by the camera, the specification is ignored, and either an error occurs or a different value is set.
For the values that can be specified, see the documentation for the camera.

Table 5–6: Sub-items of output

No.	Item	Required or optional	Description
1	type	Required	Specifies the data output method. Specify either of the following values: <ul style="list-style-type: none"> • "frame": Outputs each frame of the video as a picture file. • "video": Outputs the video as video files each of a certain length.
2	output_dir	Required	Specifies the path of the output destination directory. Specify the path of the directory that is the mount point of the input volume for the data management function.

5. Data input function

No.	Item	Required or optional	Description
3	duration#	Required if type="video" is specified	Specifies the length (in seconds) of each video file.
4	api_url	Required	Specifies the URL of the API for the data management function. Example: http://127.0.0.1:13579
5	video	Required	Specifies information about the video to be output. For details about this item, see Table 5-7: Sub-items of output-video.

#

We recommend that you set 30 or more seconds. If you set a length shorter than 30 seconds, the actual length of each video file might differ from the length that you set.

Table 5-7: Sub-items of output-video

No.	Item	Required or optional	Description
1	width	Required	Specifies the frame width (in pixels) that is used to determine the frame resolution.#
2	height	Required	Specifies the frame height (in pixels) that is used to determine the frame resolution.#
3	fps	Required	Specifies the number of visual data frames to be received per second.#
4	codec	Required	Specifies the codec to be used for files. <ul style="list-style-type: none"> If "type"="frame" is specified, you can specify "JPEG" only. If "type"="video" is specified, you can specify "H264" only.

#

- The specifiable resolution and frame rate ranges are as follows:
Resolution: 1 x 1 to 3,840 x 2,160
Frame rate: 1 to 30
- The resolution is increased or decreased according to the values that are set.
If the fps value is larger than the number of incoming frames, the function continues outputting the same frame. If the fps value is smaller than the number of incoming frames, the function skips some frames. The data flow is adjusted in this way.

Table 5-8: Sub-item of preview

No.	Item	Required or optional	Description
1	type	Required	Specifies the preview method: <ul style="list-style-type: none"> "x11": Uses X forwarding to provide a preview.

Example of the settings for receiving visual data from a USB camera

The following shows the path of a sample file and the example settings included in the sample file.

```
/hitachi/ctrl_edge_ai/sample/data_input/config_usb_cam.json
```

```

{
  "camera": {
    "name": "camera_no1",
    "type": "usb",
    "connect": {
      "device": "/dev/video0"
    },
    "video": {
      "width": 1920,
      "height": 1080,
      "fps": 30,
      "codec": "MJPEG"
    }
  },
  "output": {
    "type": "frame",
    "output_dir": "/video",
    "video": {
      "width": 640,
      "height": 480,
      "fps": 30,
      "codec": "JPEG"
    }
  },
  "api_url": "http://127.0.0.1:13579"
}

```

Example of the settings for receiving visual data from an IP camera

The following shows the path of a sample file and the example settings included in the sample file.

```

/hitachi/ctrl_edge_ai/sample/data_input/config_ip_cam.json

```

```

{
  "camera": {
    "name": "camera_no1",
    "type": "ip",
    "connect": {
      "rtsp_url": "rtsp://id:password@192.168.10.129:48512/ipcam_h264.sdp",
      "latency": 2000
    },
    "video": {
      "width": 1920,
      "height": 1080,
      "fps": 30,
      "codec": "H264"
    }
  },
  "output": {
    "type": "video",
    "output_dir": "/video",
    "duration": 30,
    "video": {
      "width": 640,
      "height": 480,
      "fps": 30,
      "codec": "H264"
    }
  },
  "api_url": "http://127.0.0.1:13579"
}

```

5.3.2 Container settings

This section describes the settings that are required for the container of the data input function. For details about building a container, see 4.1 Using the Compose file to customize the functionality.

Settings for using a GPU

The container of the data input function uses a GPU to decode or encode the received video data.

To make GPU processing available for the container of the data input function, specify the following settings:

5. Data input function

Settings:

Add `/dev/dri` to the `devices` : item in the Compose file.

```
devices:
  - "/dev/dri:/dev/dri"
```

Settings for using a USB camera

To use a USB camera, specify the settings so that the device file for the USB camera can be accessed from the container of the data input function. This specification is unnecessary if you are using an IP camera.

The device files corresponding to connected USB cameras are generated on the host in the `/dev` directory. These device files are assigned names in `videoX` format (for example, `video0`, `video1`, ...).

Use the `ls` command or another means to check the name of the device file for the USB camera that you want to use, and then specify the settings so that the device file can be used in the container of the data input function.

Setting example:

In the Compose file, add the path of the device file that you checked to the settings of the container of the data input function.

```
devices:
  - "/dev/dri:/dev/dri"
  - "/dev/video0:/dev/video0"
```

Making the settings file on the host accessible to the container

Make sure that the settings file shown in 5.3.1 Settings of the data input function can be accessed from the container of the data input function. The procedure is as follows.

Mount the path of the settings file (`data_input.json`) on the host to `/config/data_input.json` in the container of the data input function.

```
volumes:
  - "path-of-data_input.json-on-the-host:/config/data_input.json"
```

Setting example:

```
volumes:
  - "/hitachi/ctrl_edge_ai/sample/data_input/usb_cam.json:/config/data_input.json"
```

5.4 Procedure for verifying operation

This section describes the procedure for verifying that the data input function has been connected to the camera.

1. Access CE50-10A.

Directly connect a display, keyboard, and mouse to CE50-10A, and then access CE50-10A.

2. Start the desktop. (For details, see *Starting the desktop* in the *CE50-10 Instruction Manual*.)

3. When the desktop appears, log in as a user with sudo privileges (for example, edgeadm).

4. Create the settings file (`data_input.json`), and then save it to a path of your choice on the OS.

For details about how to specify the settings in the settings file, see 5.3.1 Settings of the data input function. The following is an example when the settings file is saved in `/home/edgeadm/preview`.

Example of the settings in the `data_input.json` file for receiving visual data from a USB camera:

```
{
  "camera": {
    "name": "camera_no1",
    "type": "usb",
    "connect": {
      "device": "/dev/video0"
    },
    "video": {
      "width": 640, "height": 480, "fps": 30, "codec": "MJPEG"
    }
  },
  "preview": {
    "type": "x11"
  }
}
```

5. Create the settings file (`data_input.json`), and then save it to a path of your choice on the OS.

For details about how to specify the settings in the settings file, see 5.3.1 Settings of the data input function. The following is an example when the settings file is saved in `/home/edgeadm/preview`.

6. Create a Compose file, and then save it to a path of your choice on the OS.

For details about how to specify the settings in a Compose file, see 4.1 Using the Compose file to customize the functionality and 5.3.2 Container settings.

The following is an example when the Compose file is saved in `/home/edgeadm/preview`.

Example of the settings for providing a preview of visual data from a USB camera:

```
version: '3'
services:
  preview:
    ipc: host
    image: ctrl_edge_ai/data_input:1.0
    network_mode: "host"
    devices:
      - "/dev/dri:/dev/dri"
      - "/dev/video0:/dev/video0"
    volumes:
      - "./data_input.json:/config/data_input.json"
      - /tmp/.X11-unix:/tmp/.X11-unix
    environment:
      DISPLAY: $DISPLAY
```

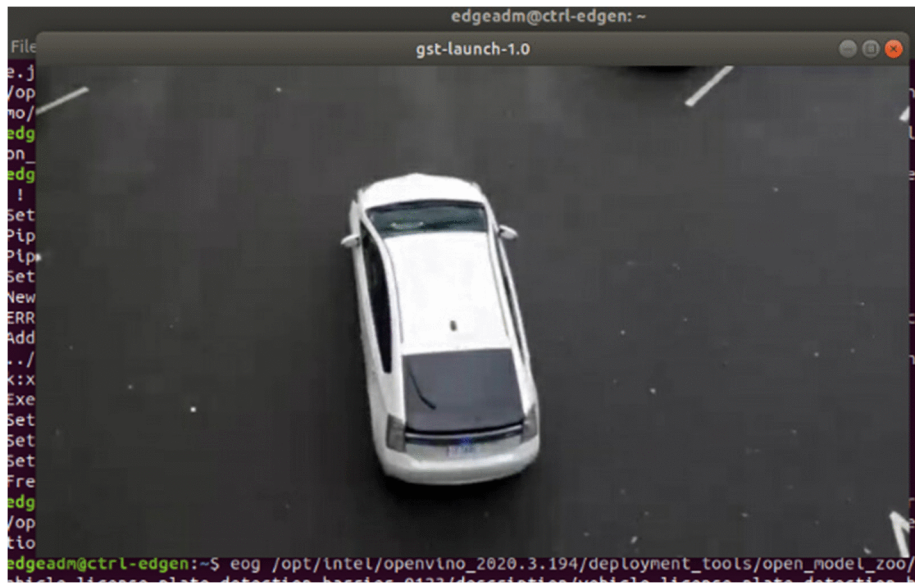
7. Run the following command:

```
$ sudo docker-compose up
```

The video captured by the camera is displayed according to the settings specified in the `data_input.json` file.

5. Data input function

Figure 5–3: Example of displayed video



The source of the sample video is as follows:

<https://github.com/intel-iot-devkit/sample-videos/raw/master/car-detection.mp4>

8. To end the preview, click the close (X) button at the top right of the window.

6

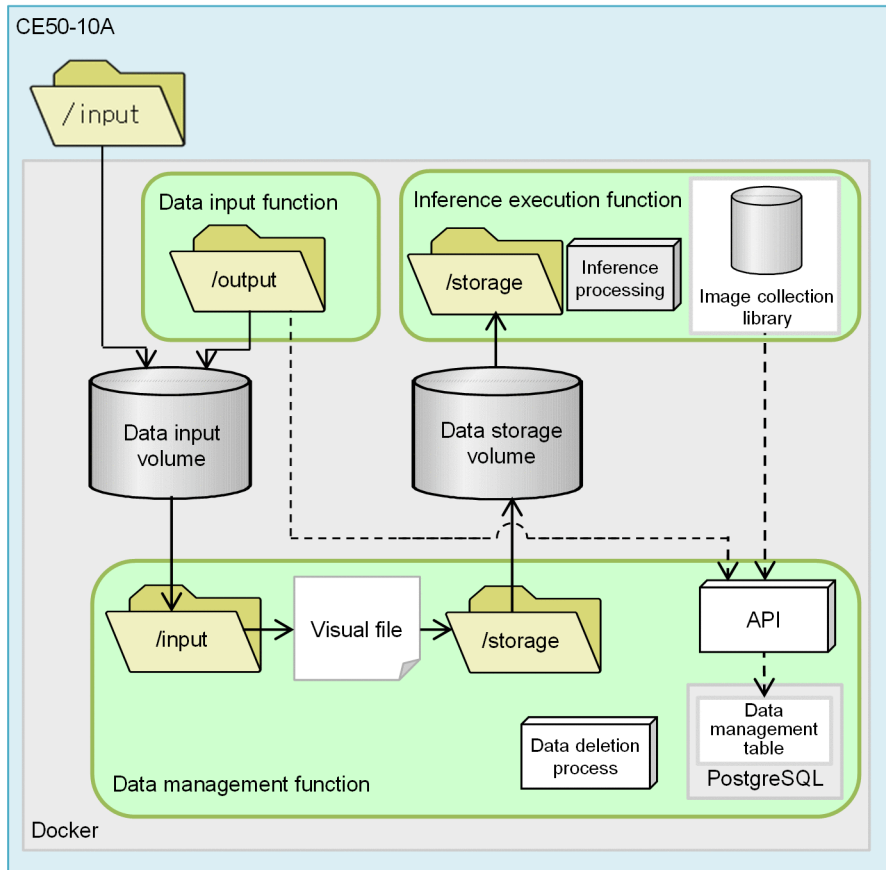
Data management function

This chapter provides an overview of the data management function. This chapter also describes the API and library specifications, and how to set up the function.





6.1 Overview of the data management function

The data management function acts as a queue that temporarily holds visual files between the data input function and the inference execution function. The following figure shows the flow of visual files and management information.

Figure 6–1: Flow of visual files and management information handled by the data management function



Legend:

-  : Container of a function
-  : Directory
-  : Flow of visual files
-  : Flow of management information

The processing performed by the data management function is as follows.

Registering files:

This processing accepts inference-target visual files and registers them in a data management table.

The data input function registers files automatically, so you do not need to do anything in particular. Note, however, that there might be cases where visual data passed as a file by an image inspection device or other non-camera device is to be registered as a file in the container of the data management function. In such a case, you will need to create a program that performs the API operations described later. For details about how to use the API, see 6.2 API specifications of the data management function.

The containers of the data input function and data management function use volumes to share files.

The following shows the procedure for registering a file in the data management function:

1. Save the target file in the `/input` directory of the container of the data management function.

2. Use the `POST` method to issue a request to the API (`/v1/files`) to register the file, where the request includes the path of the saved file.
The API registers the file name in the data management table. At this time, the file is registered as a file in the *ADDED* status.
The API then moves the file from the `/input` directory to the `/storage` directory.

Receiving files and changing the status:

This processing identifies and retrieves inference-target visual files. The image collection library of the inference execution function or inference development function automatically performs the following procedure, so you do not need to do anything in particular. Perform the following procedure if you receive visual data without using the image collection library:

1. Use the `GET` method to issue a request to the API (`/v1/files`) to receive a file that is in the *ADDED* status.
The API returns the name of a file in the *ADDED* status from the data management table according to the specified option.
2. Issue a request to the API to change the status of the file received in step 1 to *PROCESSING*.
3. Perform inference for the file at the returned path.
4. When inference is complete, issue a request to the API (`/v1/files/file_id`) to change the status of the file received in step 1 to *COMPLETED*.

If the image collection library of the inference execution function or inference development function is used, the preceding procedure (steps 1 to 4) are performed automatically.

Deleting files:

This processing reads the information in the data management table and automatically deletes files registered in the table. The upper limit on the total size of files in the *ADDED* and *COMPLETED* statuses is specified in the settings file of the data management function. If the upper limit is exceeded, registered files are deleted in order from the oldest files.

6.2 API specifications of the data management function

The API methods that register files, receive file, and perform other operations in the data management function are provided by the HTTP REST API. This API uses port 13579 of the container of the data management function.

Example of the access URL:

```
http://127.0.0.1:13579/v1/files
```

In the descriptions in the subsequent sections, the *protocol://host-name:13579* portion of the URL is referred to as the *base-URL*.

6.2.1 Registering a file in the data management function (v1FilesPost)

The method that registers a file at a specified path in the database is as follows.

Request line

```
POST base-URL/v1/files
```

Request message

Body

```
{
  "camera_id" : "camera_1"
  "input_path" : "/path/to/file"
  "status" : "added"
}
```

No.	Attribute	Data type	Description
1	camera_id	String	(Optional item) Specify the camera ID. Example: camera_1
2	input_path	String	(Optional item) Specify the path to which the file is to be registered. Example: /path/to/file
3	status	String	(Optional item) Specify the status of the file. Example: added

Response message

Body

```
{
  "file_id" : "cf16fe52-3365-3a1f-8572-288d8d2aaa46"
}
```

No.	Attribute	Data type	Description
1	file_id	String	Returns the assigned file ID. Example: cf16fe52-3365-3a1f-8572-288d8d2aaa46

Status codes

No.	Status code	Description
1	200	Registration was successful.
2	400	Invalid request

No.	Status code	Description
3	503	An error occurred on the server side.

6.2.2 Updating file status (v1FilesFileIdPut)

The method that changes the status of a registered file is as follows.

Request line

```
PUT base-URL/v1/files/file_id
```

No.	Attribute	Data type	Description
1	file_id	String	(Required item) Specify the file ID. Example: cf16fe52-3365-3a1f-8572-288d8d2aaa46

Request message

Body

```
{
  "status" : "completed"
}
```

No.	Attribute	Data type	Description
1	status	String	(Required item) Specifies the file status. Example: completed

Response message

Status codes

No.	Status code	Description
1	200	Update was successful.
2	400	Invalid request
3	404	<i>file_id</i> was not found.
4	503	An error occurred on the server side.

6.2.3 Receiving files (v1FilesGet)

The method that receives files held by the data management function is as follows.

Request line

```
GET base-URL/v1/files
```

Request message

Query parameters

No.	Attribute	Data type	Description
1	num	integer	(Optional item) Specify the number of files to be received. The default is 1.
2	camera_id	string	(Optional item) Specify the camera IDs. If this item is omitted, the method assumes that all camera IDs are specified. To specify multiple camera IDs, use the following format: camera_id=camera_1&camera_id=camera_2...
3	order	string	(Optional item) Specify the order in which the files are to be received. LIFO: Receive newer files first (default). FIFO: Receive older files first.

Response message

Body

```
[
  {
    "file_id" : "cf16fe52-3365-3a1f-8572-288d8d2aaa46",
    "path" : "/path/to/file",
    "camera_id" : "camera_2"
  }
]
```

No.	Attribute	Data type	Description
1	file_id	String	Returns the file IDs.
2	path	String	Returns the registration destination path.
3	camera_id	String	Returns the camera IDs.

Status codes

No.	Status code	Description
1	200	Reception was successful.
2	400	Invalid request
3	503	An error occurred on the server side.

6.3 Specifications of the library available for the data management function

This section describes how to use the image collection library that uses the data management function from a Python program.

The image collection library is provided as a module named `Dataman`.

6.3.1 Library usage

`Dataman` can be used by the inference execution and inference development functions.

The inference processing program used by the inference execution function and Jupyter Notebook used by the inference development function receive frames from the data management function.

You can specify settings to enable debug mode. In this mode, the aforementioned components receive frames from the visual file specified for debugging, without accessing the data management function.

6.3.2 Settings for using the library

Before `Dataman` can be used, you must set up an operation to load the settings file during initialization.

Path of the sample settings file:

```
/hitachi/crt1_edge_ai/sample/dataman/dataman.json
```

Table 6–1: Setting items

No.	Setting item	Description
1	<code>api_url</code>	Specifies the URL of the API for the data management function.
2	<code>frame_order</code>	Specifies the order in which the images are to be received. <ul style="list-style-type: none"> "LIFO": Receives newer images first (initial value). "FIFO": Receives older images first.
3	<code>target_camera</code>	Specifies the IDs of the cameras from which images are to be received. Setting example: <code>["camera_1", "camera_2"]</code> To receive images from all cameras, specify <code>[]</code> . The initial value is <code>[]</code> .
4	<code>debug</code>	Specify this item to debug the inference execution function. If you specify <code>debug</code> , the <code>api_url</code> , <code>frame_order</code> , and <code>target_camera</code> setting items are disabled. For details about this item, see Table 6–2: Sub-item of debug.

Table 6–2: Sub-item of debug

No.	Item	Description
1	<code>file_path</code>	Specifies the visual file to be used.

Setting example 1:

```
{
  "api_url": "http://127.0.0.1:13579",
  "frame_order": "LIFO",
  "target_camera": []
}
```

Setting example 2:

```
{
  "debug":{
    "file_path": "/hitachi/people_count_sample/sample_video.mp4"
  }
}
```

6.3.3 How to import the library

For the Python program used by the inference execution function or Jupyter Notebook used by the inference development function, the library to be used is imported as follows. The `Dataman` module is included in the `ce50.ai` package.

```
from ce50.ai import *
```

6.3.4 Dataman class methods

This section describes the specifications of the `Dataman` class methods used in the data management function.

(1) class `Dataman.Dataman(conf_path)`

Base class: `object`

Class for operating the data management function

Constructor

Parameter: `conf_path` (str)

Path of the settings file

Sample:

```
>>> # To generate an instance of the Dataman class.
>>> dm=Dataman("/hitachi/people_count_sample/dataman/dataman.json")
```

(2) `get_frame()`

This method acquires an image frame subject to inference and returns it as a `cv: :Mat` type. This method can be used in the same way as the `read()` method in the `VideoCapture` class of OpenCV.

In debug mode, this method acquires the picture or video frames of the visual file specified for debugging. After the method finishes reading all of the frames in the file, it rereads the file starting from the first frame. If the specified file is a picture file, the method always acquires the same image.

Return values:

`return`: Whether frame acquisition was successful (success: True, failure: False)

`info`: Frame information

`frame`: Image data

Types of return values: `bool`, `dict`, and `cv: :Mat`

Sample:

```
>>> ret, info, frame = db.get_frame()
```

(3) `set_target_camera(camera_id)`

This method specifies for the `get_frame` method to acquire images from cameras whose IDs were specified.

If this method is run in debug mode, the method does not do anything.

Parameter: `camera_id_list` (array)

Camera IDs

Sample:

```
>>> # To receive images from only camera_1.
>>> dm.set_target_camera(["camera_1"])
>>> # To receive images from all cameras.
>>> dm.set_target_camera([])
```

(4) `update_frame_status(info, status)`

This method changes the status of the frame that has the specified frame ID.

If this method is run in debug mode, the method does not do anything.

Parameters:

`info` (dict): Information about the target frame

`status` (str): New status

Sample:

```
>>> # To change the status of an inferred frame to "COMPLETED".
>>> dm.update_frame_status(info, Dataman.STATUS_COMPLETED)
```

6.3.5 Constants for file statuses

The following table shows the constants that indicate the file statuses defined in the `Dataman` class.

Table 6–3: Constants for the file statuses

No.	Constant	Description
1	<code>STATUS_ADDED</code>	These constants are used to indicate the status of a video. For example, <code>STATUS_ADDED</code> indicates that inference has not started yet.
2	<code>STATUS_PROCESSING</code>	
3	<code>STATUS_COMPLETED</code>	

6.4 How to specify the settings of the data management function

This section describes the settings file and Compose file for the data management function.

6.4.1 Settings file for the data management function

If you want to use a settings file for the data management function, save the file to the following path in the container of the data management function.

If no settings file exists, default settings are used.

```
/config/data_manager.json
```

Table 6–4: Setting item in the settings file

No.	Item	Required or optional	Description
1	delete	Required	Specifies the settings related to deletion. For details about this item, see Table 6–5: Sub-items of delete.

Table 6–5: Sub-items of delete

No.	Item	Required or optional	Description
1	unfinished_limit	Required	Specifies the upper limit on the total size (in KB) of files whose status is not <i>COMPLETED</i> . The default is 1048576 (KB). If you specify a value greater than the size of free space on the data storage volume, an error occurs.
2	finished_limit	Required	Specifies the upper limit on the total size (in KB) of files whose status is <i>COMPLETED</i> . The default is 0 (KB). If you specify a value greater than the size of free space on the data storage volume, an error occurs.

Setting example:

```
{
  "delete":{
    "unfinished_limit":1048576,
    "finished_limit":0
  }
}
```

6.4.2 Compose file settings

When you specify the settings of a Compose file, take the following into consideration.

(1) Considerations for creating volumes

For CE50-10A, volumes must be created to share data between containers as follows:

- A volume between the container of the data input function and that of the data management function
- A volume between the container of the data management function and that of the inference execution function

Make sure that each volume is mounted to the same path on both containers.

The volumes for holding data can be created in either of the following locations:

1. SSD (Docker-only partition)
2. Main memory

In general, create the volumes in location 1.

The following table shows the advantages and disadvantages of each of the two locations.

Location	Advantage	Disadvantage
1. SSD (Docker-only partition)	Data can be held persistently. The data is retained even if an abnormality such as a power failure occurs.	<ul style="list-style-type: none"> • The write speed is slower than the speed of writing to main memory. • Repeatedly writing large amounts of data will shorten the lifetime of the SSD.
2. Main memory	Data is written at a high speed.	<ul style="list-style-type: none"> • The data is not persistent. If an abnormality occurs or the container stops, data will be lost. • Storing a large amount of data reduces the amount of usable main memory.

Based on the advantages and disadvantages in this table, we recommend that you create the volumes as follows:

Containers that will share data	Where to create the volume
The containers of the data input function container and data management function	2. Main memory
The containers of the data management function container and inference execution function [#]	1. SSD (Docker-only partition)

#

If the amount of data written per day is 50 GB or more, the actual lifetime of the SSD might be shorter than the expected lifetime. Therefore, you might need to consider creating the volume on main memory depending on the requirements for the system (such as the required availability, lifetime, and frequency of maintenance).

Setting example:

In this example, you will create volumes as follows for the containers of the functions that share data:

- Volume between the containers of the data input function and data management function: `input_volume` (in the main memory)
Path of the mount point: `/input`
- Volume between the containers of the data management function and inference execution function: `storage_volume` (on an SSD)
Path of the mount point: `/storage`

The following is an example of the settings to be specified. Note that this example is an excerpt of only the relevant entries.

```

services:
  data_input:
    image: ctrl_edge_ai/data_input:1.0
    volumes:
      - input_volume :/input
  data_manager:
    image: ctrl_edge_ai/data_manager:1.0
    volumes:
      - input_volume:/input
      - storage_volume:/storage
  people_count:
    image: people_count:latest
    volumes:
      - storage_volume:/storage

```

6. Data management function

```
volumes:
  input_volume:
    driver: local
    driver_opts:
      type: tmpfs
      device: tmpfs
      o: "size=512m"
  storage_volume:
    driver: local
```

(2) Port settings

Specify settings so that the port used by the API for the container of the data management function can be accessed from the containers of other functions.

The following is an example of the settings to be specified. Note that this example is an excerpt of only the relevant entries.

Setting example:

```
services:
  data_manager:
    network_mode: host
```

7

Inference execution function

This chapter provides an overview of the inference execution function, the general procedure for implementing inference processing, and sample programs.

7.1 Overview of the inference execution function

Visual data is imported by the data input function and then registered in a volume as visual files by the data management function. The inference execution function retrieves the inference-target visual files from the volume and then performs inference for those files.

The inference execution function performs inference processing created in Python.

To perform inference processing, you must prepare the following two items:

- The inference-processing execution program
- An intermediate representation (called *IR* in OpenVINO), which is the result of converting a pre-trained model

The following table provides an overview of the processing of an inference-processing execution program and a pre-trained model.

Table 7–1: Overview of the processing of an inference-processing execution program and a pre-trained model

No.	Item	Processing overview
1	Inference-processing execution program	The inference-processing execution program specifies the initial settings that are required to read a pre-trained model and perform inference processing. The program also performs preprocessing to convert loaded images to another format and displays the inference results.
2	Pre-trained model	A pre-trained model performs inference based on the input information and outputs the inference results. The types of images for which inference processing can be performed and the inference results (such as the number of people detected and the detected coordinates) vary depending on the pre-trained model that is used.

7.1.1 How to obtain the pre-trained models provided for use with OpenVINO

You can obtain pre-trained models from the following website.

Website for downloading pre-trained models:

<https://download.01.org/opencv/>

You can download pre-trained models by directly accessing this website. You can also use Model Downloader, which is a function of OpenVINO.

For details about how to use Model Downloader, see the following webpage.

Webpage about how to use Model Downloader:

https://docs.openvino toolkit.org/2020.3/_tools_downloader_README.html#

#

The information in this manual is based on OpenVINO version 2020.3. If a newer version of OpenVINO is available, check the information about the new version. Such information can be found, for example, on the Intel website.

7.1.2 Converting a pre-trained model into an intermediate representation

Pre-trained models that users create by using TensorFlow or Caffe can also be used with OpenVINO. In this case, the pre-trained models must be converted into intermediate representations.

OpenVINO provides Model Optimizer, which is a function that converts a user-created pre-trained model into an intermediate representation. Model Optimizer can be used to create intermediate representations of pre-trained models. For details about how to convert a pre-trained model into an intermediate representation, see the following webpage:

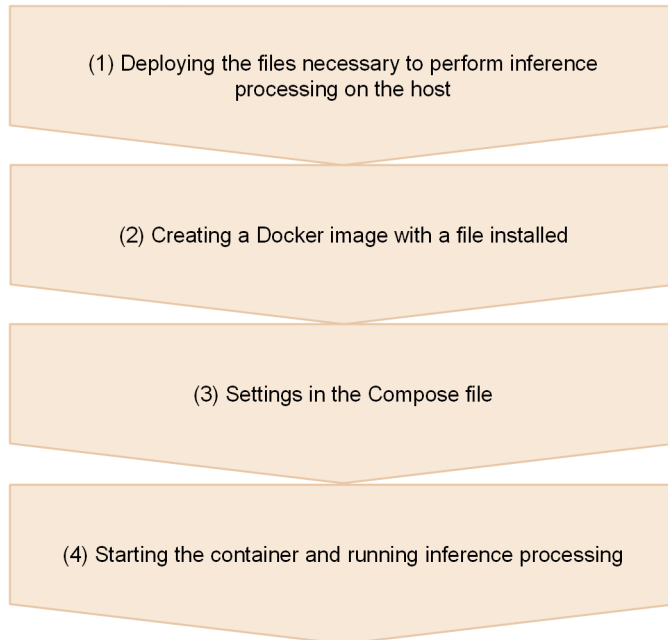
Webpage about how to convert a pre-trained model into an intermediate representation:

https://docs.openvinotoolkit.org/2020.3/_docs_MO_DG_prepare_model_convert_model_Converting_Model.html

7.2 Overview of implementing inference processing

This section describes how to implement user-developed inference processing in the inference execution function. The following figure provides an overview of implementing inference processing.

Figure 7–1: Overview of implementing inference processing



7.2.1 How to implement inference processing

This section describes how to implement inference processing.

(1) Deploying the files necessary to perform inference processing on the host

You need to prepare two types of files to perform inference processing: files for an inference-processing execution program and files for a pre-trained model.

Examples of these files are included in the following sample program provided with CE50-10A.

You can change the names of the directories in the following file structure as you like.

```

/hitachi/ctrl_edge_ai/sample/people_count/
|- AP
|- people_count_sample_via_video
|   |-- Dockerfile
|   |-- docker-compose.yaml
|   |-- component
|   |-- people_count_demo.py
|   |-- model_path.json
|   |-- sample_video.mp4
|   |-- dataman.json
|   |-- openvino_models
|   |-- face-detection-retail-0004.bin
|   |-- face-detection-retail-0004.xml
|- people_count_sample_via_usbcam
|   |-- Dockerfile
|   |-- docker-compose.yaml
|   |-- component
|   |-- people_count_demo.py
|   |-- model_path.json
|   |-- dataman.json
|   |-- openvino_models
|   |-- face-detection-retail-0004.bin
|   |-- face-detection-retail-0004.xml
  
```

The following table describes the items in this sample program.

Table 7–2: Items in the sample program

No.	File name	Description
1	AP	A program that starts an HTTP server so that the AP indicator lights up when the sample program is executed as a demo.
2	Dockerfile	A file for creating a Docker image that runs a demo to count the number of people in a sample video or the information received from a USB camera. This file is used during processing to build a Docker container.
3	docker-compose.yaml	A settings file for starting a container from the created Docker image.
4	people_count_demo.py	A program that performs inference processing.
5	model_path.json	A file in which the path of the pre-trained model is specified. This file is loaded when <code>people_count_demo.py</code> is run.
6	sample_video.mp4 ^{#1}	A video file used to perform the demo. The name of this file is fixed as <code>sample_video.mp4</code> .
7	dataman.json	A file in which the method for receiving data from the data management function is specified.
8	face-detection-retail-0004.bin ^{#2}	A pre-trained model file in which the weight information is specified.
9	face-detection-retail-0004.xml ^{#2}	A pre-trained model file in which the network information is specified.

#1

- Characteristics required of the video file
In a demo that uses a sample video, human faces are detected and the number of people detected is displayed. Alternatively, the detected faces are enclosed in a rectangular frame in the preview. For these functions to work properly, we recommend that you use a sample video that has the following characteristics:
 - The video includes people's faces.
 - The number of faces captured on video changes over the course of the video.
- Sample video file that can be used
You can download a sample video file from the following webpage and use it for demos. By using this video file for demos, you can check how faces are detected and how the number of detected faces changes.

```
https://github.com/intel-iot-devkit/sample-videos/raw/master/face-demographics-walking-and-pause.mp4
```

- How to deploy the sample video
After you have downloaded the video file, rename it to `sample_video.mp4`, and then copy it to the appropriate directory as shown in (1) Deploying the files necessary to perform inference processing on the host.
If you redistribute the sample program to end users after the demo, make sure that the sample program does not include any videos that infringe on someone's copyright or portrait right. If the sample program includes such videos, delete them before redistributing the sample program.

#2

The pre-trained model file can be downloaded from the following website:

```
https://download.01.org/opencv/2020/openvinotoolkit/2020.3/open_model_zoo/models_bin/1/face-detection-retail-0004/FP16/
```

(2) Creating a Docker image with a file installed

Perform the following procedure to create a Docker image:

1. Run the following command to create a Dockerfile in a directory of your choice.

```
$ sudo vi Dockerfile
```

In the Dockerfile, specify the location (path on the host) of the file to be copied by using the `COPY` command and the container to which the file is to be copied.

Example of the settings in the Dockerfile:

```
FROM ctrl_edge_ai/opencvino_prod:1.0
COPY ./component /hitachi/people_count_sample
CMD [ "/hitachi/people_count_sample/people_count_demo.py" ]
```

Commands used in the Dockerfile:

- `FROM`
Specify the name of the base Docker image. To use the inference execution function, specify `ctrl_edge_ai/opencvino_prod:1.0`.
- `CMD`
Specify the path (in the container) of the Python file that contains the inference processing to be performed when the container is run.

2. Change the current directory to the directory that contains the Dockerfile and build the Dockerfile into a Docker image named `people_count`. To do this, run the following commands:

```
$ cd path-of-the-directory-containing-the-Dockerfile
$ sudo docker build . -t people_count:latest
```

(3) Settings in the Compose file

In (2) Creating a Docker image with a file installed, you created a Docker image. Here, you will create a container from the Docker image, and then create a Compose file so that inference processing can be performed. For details about how to create a Compose file, see 4.1 Using the Compose file to customize the functionality.

Example of the settings specified in the Compose file:

```
services:
  people_count:
    image: people_count:latest
```

If a GPU is to be used during inference processing, add the `devices` option to the `people_count` container in the Compose file.

Example of the settings specified in the Compose file if a GPU is to be used:

```
services:
  people_count:
    image: people_count:latest
    devices:
      - "/dev/dri:/dev/dri"
```

If a GPU is used, the pre-trained model must be converted into a format compatible with the GPU. Therefore, it will take a few minutes to read the pre-trained model.

The conversion results can be cached in the directory specified in the `cl_cache_dir` environment variable. If the conversion results of a pre-trained model are cached, the next time the model needs to be read, it will take less time.

The following is an example of the settings to be specified in the Compose file if the cache is to be used. Note that this example is an excerpt of only the relevant entries.

Example of the settings specified in the Compose file if the `cl_cache_dir` environment variable is specified:

```
services:
  people_count:
    image: people_count:latest
    environment:
      cl_cache_dir: "/cl_cache"
    volumes:
      - /home/edgeadm/cl_cache:/cl_cache
```

Note that, if you start a container as a general user, you will need to change the permissions of the directory specified for the environment variable so that the cached data is retained. Make sure that all users are permitted to perform read, write, and run operations on the cache directory. To do this, add processing to change the permissions of the directory where you want to retain the cache when you build the Dockerfile.

The following is an example of the settings in a Dockerfile to which processing to change directory permissions has been added.

Example of the settings specified in the Dockerfile to change the directory permissions:

```
FROM ctrl_edge_ai/openvino_prod:1.0

# Specify the user (root) who can access the Dockerfile.
USER root

# Create a cache directory and change the permissions.
RUN mkdir /cl_cache
RUN chmod 777 /cl_cache

# Specify a general user when the container starts.
USER openvino

# Set the environment variable.
ENV cl_cache_dir=/cl_cache

# Set the program to be run when the container starts.
CMD [ "/hitachi/people_count_sample/people_count_sample.py" ]
```

If you want to include the cache in the Docker image in the production environment, you can save the cache on the host and then have the cache on the host included in the Docker image when the Dockerfile is built.

To mount a host directory from a container that was started by a general user so that cached data remains on the host even after the container is deleted, you must change not only the permissions for the directory in the container, but also the permissions for the host directory.

The following examples show the commands for changing the permissions for the host directory, and the settings in the Dockerfile when data cached on the host is to be included in the Docker image.

Example of the commands for changing the permissions for the host directory:

```
# Change permissions for the host directory to be mounted to a container.
$ sudo chmod 777 /home/edgeadm/cl_cache

# Start the container so that data is cached on the host.
$ sudo docker-compose up -d
```

Example of the settings in the Dockerfile when data cached on the host is to be included in the Docker image:

```
FROM ctrl_edge_ai/openvino_prod:1.0

# Specify the user (root) who can access the Dockerfile.
USER root

# Include the data cached on the host in the Docker image.
COPY relative-path-of-the-cl_cache-directory/* /cl_cache

# Change the permissions of the directory.
RUN chmod 777 /cl_cache

# Specify a general user when the container starts.
USER openvino

# Set the environment variable.
ENV cl_cache_dir=/cl_cache

# Set the program to be run when the container starts.
CMD [ "/hitachi/people_count_sample/people_count_sample.py" ]
```

(4) Starting the container and running inference processing

Run the following command to start the container and perform inference processing.

```
$ sudo docker-compose up -d
```

7.3 Description of the sample program

CE50-10A provides a sample program. The source code is in the following file:

```
/hitachi/ctrl_edge_ai/sample/people_count/people_count_sample_via_video/component/people_count_demo.py
```

7.3.1 Processing performed by the sample program

This section describes the processing performed by the sample program (`people_count_demo.py`) provided with CE50-10A. The pre-trained model used in the sample program is `face-detection-retail-0004`.

The information input and output for `face-detection-retail-0004` is as follows.

- Input information: `[1x3x300x300]` (= batch size x number of channels x image height x image width)
- Output information: `[1, 1, N, 7]` (N : number of detected bounding boxes)

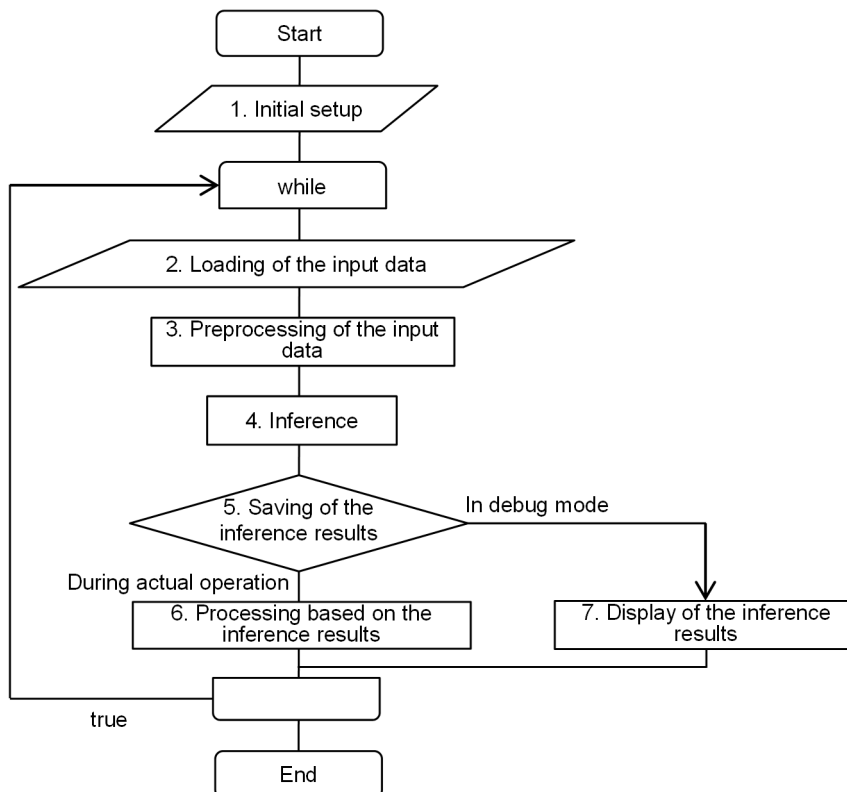
For details about the input information and output information, see the following webpage:

```
https://docs.openvinotoolkit.org/2020.3/\_models\_intel\_face\_detection\_retail\_0004\_description\_face\_detection\_retail\_0004.html
```

The sample program performs inference preprocessing and inference processing. It also turns the AP indicator on.

The following figure provides an overview of the processing performed by the provided sample program.

Figure 7-2: Overview of the processing performed by the provided sample program



(1) Initial setup

In this step, the sample program performs the following six operations:

[1] Read libraries.

The sample program reads the following five libraries, which are used for inference:

- `cv2`: Library for image processing (Python wrapper for OpenCV)
- `json`: Library for handling the JSON format
- `numpy`: Library for numerical calculation
- `IECore`: Library for the inference engine of OpenVINO
- `Dataman`: Library for retrieving images from the container of the data management function

The relevant source code is as follows:

```
# Read libraries.
import cv2
import numpy as np
import json
import requests

from openvino.inference_engine import IECore
# Data management function library
from ce50.ai import *
```

[2] Create instances from the `IECore` and `Dataman` classes.

The sample program creates an `IECore` instance and a `Dataman` instance from the libraries that were read in [1].

For details about the `IECore` class, see the following webpage:

https://docs.openvino toolkit.org/2020.3/ie_python_api/classie__api_1_1IECore.html#a fe73d64ddd115a41f5acc0d31031f52b

The relevant source code is as follows:

```
# Create instance ie from class IECore of Inference Engine.
ie = IECore()
# Create instance dm of class Dataman.
dm = Dataman ("/hitachi/people_count_sample/dataman.json" )
```

[3] Specify the paths of the pre-trained model files and read the pre-trained model.

The sample program uses the instances created in [2] to read the pre-trained model from the specified paths of the pre-trained model files (`.xml` and `.bin` files) to be used.

The relevant source code is as follows:

```
# Read model_path.json.
json_open = open('/hitachi/people_count_sample/model_path.json', 'r')
json_load = json.load(json_open)
# Read the paths to the IR files from the JSON file.
xml = json_load['model_path']['xml']
bin = json_load['model_path']['bin']
# Close the JSON file.
json_open.close()
net = ie.read_network(model=xml, weights=bin)
```

[4] Convert the pre-trained model that was read into an object that can run on the selected device (CPU or GPU).

The sample program converts `net`, the pre-trained model that was read in [3], into an object for which inference processing can be performed on the specified device.

The relevant source code is as follows:

```
# Convert the model into an object that can run on the selected device (CPU in this
case).
exec_net = ie.load_network(network=net, device_name ="CPU")
```

[5] Obtain the input information and output information according to the pre-trained model that was read.

The sample program obtains the types of input and output information needed to run the pre-trained model.

The items set in the input information and output information are as follows:

- Input information: Image width, image height, number of channels, number of images to be processed at one time, and image precision
- Output information: Image precision

For details about the input information, output information, and image precision, see the following webpages:

Webpage about the pre-trained model:

```
http://docs.openvinotoolkit.org/2020.3/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.htm
```

Webpage about precision:

```
https://docs.openvinotoolkit.org/2020.3/classInferenceEngine_1_1Precision.html
```

The relevant source code is as follows:

```
# Set input and output information according to the pre-trained model to be read.
for input_key in net.inputs:
    if len(net.inputs[input_key].layout) == 4:
        n, c, h, w = net.inputs[input_key].shape
```

[6] Set the detection threshold.

The sample program sets the threshold to be used as the basis for determining the face area. For details, see (5) Saving the inference results.

The relevant source code is as follows:

```
# Set the detection threshold.
DETECTION_THRESHOLD = 0.5
```

(2) Reading the input data

In this step, the sample program obtains input information from the container of the data management function. The program uses the `get_frame` function of the `Dataman` class when obtaining input information from the container of the data management function.

The relevant source code is as follows:

```
# "ret" contains whether frame acquisition was successful (True or False).
# "info" contains frame information.
# "frame" contains the input image data.
ret, info, frame = dm.get_frame()
```

(3) Preprocessing the input data

In this step, the sample program converts the input information based on the inference conditions for the pre-trained model that was read. Note that the inference conditions are specified in [5] in (1) Initial setup. The input information (image) is represented by using a three-dimensional array consisting of the height, width, and number of channels. The sample program converts the input information into the structure required for the pre-trained model and stores the conversion results in `images`.

The relevant source code is as follows:

```
# Convert the input data at a size that can be handled by the pre-trained model (height x width = 300 x 300).
if (ih, iw) != (h, w):
    image = cv2.resize(image, (w, h))
# Convert the data type from (h, w, c) to (c, h, w).
image = image.transpose((2, 0, 1))
images = image
```

(4) Inference

In (3) Preprocessing the input data, the sample program converted the input information into a format that can be handled by the pre-trained model that was read by the program, and then stored the conversion results in `images`. In this step, the sample program stores the converted input information to `data` as dictionary-type data, performs inference processing based on the stored input information, and then stores the processing results in `res`.

For details about the inference processing performed by OpenVINO, see the following webpage:

```
https://docs.openvinotoolkit.org/2020.3/ie_python_api/classie__api_1_1InferRequest.htm
l#aac8de3eea8b2eec962bd5b64a176f618
```

The relevant source code is as follows:

```
# Store "images" (converted image) to "data".
data = {}
data[input_name] = images

# Perform inference.
res = exec_net.infer(inputs=data)
```

(5) Saving the inference results

In this step, the sample program re-stores the inference results in `res`.

In `res`, the following information items are stored for each face area detected in the input information:

- Top-left coordinates (x, y) and bottom-right coordinates (x, y) of the face area
- Degree of certainty that the area is a face (0 to 1)

The sample program retrieves data from the inference results for each detected face area by using a `for` loop. Then, the program compares the detection threshold (set in [6] in (1) Initial setup) with the degree of certainty. (To make processing easier, only inference results with a degree of certainty greater than or equal to the threshold are assumed to be face areas and stored in arrays `boxes` and `classes`.)

The relevant source code is as follows:

```
res = res[out_blob]
boxes, classes = {}, {}
data = res[0][0]
for number, proposal in enumerate(data):
    if proposal[2] > DETECTION_THRESHOLD:
```

(6) Processing based on the inference results

During actual operation, the inference results are processed according to the specified settings.

For example, if an error is detected from the inference results, the administrator might be notified of the error.

The sample program notifies the users by turning on an indicator according to the number of detected people.

The following code is the relevant part of the sample program.

The sample program starts process AP (which operates the AP indicator) on the host. In the following code, the program requests the process to turn on the AP indicator in the color specified via HTTP.

```
if len(boxes) == 0:
    people_num = 0
    print("people=0")
else:
    people_num=len(boxes[imid])
    print("people="+str(people_num))
# Store a value in "ap_color" according to the number of detected people as follows: 0
# = "off", 1 or 2 = "green", 3 or more = "red".
if people_num >= 3:
    ap_color="red"
elif people_num >= 1 :
    ap_color="green"
else:
    ap_color="off"

# Send a request to process AP specifying the URL followed by ap_color.
if ap_color != current_ap_color:
    requests.get('http://127.0.0.1:5000/'+ap_color)
    current_ap_color=ap_color
```

Notifying the data management function that inference finished

The sample program notifies the data management function that inference of the relevant frame has finished.

7. Inference execution function

If the input source is a picture file, the program immediately updates the data management table. If the input source is a video file, the program updates the data management table when processing reaches the last frame. The program does not need to track the progress of inference processing for the target file.

```
# Relevant frame
dm.update_frame_status(info, Dataman.STATUS_COMPLETED)
```

(7) Displaying the inference results

In debug mode, the inference results need to be displayed on a monitor so that the user can check the precision of the pre-trained model that was created.

To display the inference results, use `cv2.rectangle` to enclose the areas detected based on the inference results and use `cv2.imshow` to display those areas on a monitor.

In the sample code, `cv2.rectangle` uses the following arguments:

- Argument 1: Input image
- Argument 2: Top-left coordinates (x, y) of the detected face area
- Argument 3: Bottom-right coordinates (x, y) of the detected face area
- Argument 4: Color of the line to be drawn
- Argument 5: Thickness of the line to be drawn

For details about `cv2.rectangle`, see the following webpage:

```
https://docs.opencv.org/3.4.0/d6/d6e/group\_imgproc\_\_draw.html#ga346ac30b5c74e9b5137576c9ee9e0e8c
```

The following is sample code that displays inference results.

```
# Perform the following processing in debug mode only.
# To enter debug mode, add the -O option (example: $ python -O demo.py).
if not __debug__ :
    # To display inference results together with the input information, use cv2.rectangle
    # to rectangularly enclose detected coordinates and save them in tmp_image.
    for imid in classes:
        for box in boxes[imid]:
            cv2.rectangle(tmp_image, (box[0], box[1]), (box[2], box[3]), (232, 35, 244
            ), 2)

    # Display detection results (arg 1: window name of type string, arg 2: image to show).
    cv2.imshow('demo', tmp_image)
    cv2.waitKey(1)
```

8

Inference development function

This chapter provides an overview of the inference development function. This chapter also describes how to start the function and how to use Jupyter Notebook.

8.1 Overview of the inference development function

CE50-10A provides the inference development function, which allows users to develop their own inference processing. Specifically, the inference development function provided is Jupyter Notebook, which is a standard tool used for data analysis and AI development.

For details about Jupyter Notebook, see the following website:

<https://jupyter.org/>

With Jupyter Notebook, a user can connect to CE50-10A via a network from a web browser on a PC to input and run Python programs. The user can then verify how such a program operates and debug it in an interactive manner.

8.2 How to start the inference development function

The following describes how to start the inference development function from CE50-10A or from a PC via a network connection:

1. Connect to CE50-10A by using SSH or another protocol from a PC that is connected to CE50-10A via a network, and then run the following commands to start the container of the inference development function. You can also run the commands directly on CE50-10A.

```
# Restart the Docker daemon (after stopping all running containers).
$ sudo systemctl restart docker

# Create a container specifying ports.
# Connect port 8888 of the host and port 8888 of the container.
$ sudo docker container run -it --rm -p 8888:8888 ctrl_edge_ai/openvino_devel:1.0
```

When you run these preceding commands, the following message appears. The token displayed in the message is the password for logging in to Jupyter Notebook. A new token is generated each time the container starts.

```
Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://0.0.0.0:8888/?token=9f64181ec9bb360b831f2ce6d3893c4e6b4f3e16a004aa99
```

2. On a PC that is connected to the network to which CE50-10A is connected, start a web browser, and then enter the following information in the address bar:

Address bar:

```
http://IP-address-of-the-CE50-10A:8888/?token=xxx
```

For xxx, specify the token displayed in step 1.

3. After entering the information shown in step 2 in the address bar, press the **Enter** key.

The Jupyter Notebook webpage appears in the web browser.

Figure 8–1: Jupyter Notebook page



4. After you have finished your work, close the container of the inference development function that you created on CE50-10A, and then close the ports that you opened on CE50-10A.

```
# While the container of the inference development function is displayed,
# press [Ctrl+C]. When "Shutdown this notebook server (y/[n])?" appears,
# enter [y] and then press the [Enter] key.
```

8.3 How to use Jupyter Notebook

This section describes basic Jupyter Notebook operations and how to use OpenVINO with Jupyter Notebook.

8.3.1 Basic operations in Jupyter Notebook

The following describes how to perform basic Jupyter Notebook operation.

(1) Logging in to Jupyter Notebook

When you start Jupyter Notebook, it generates a URL. When you access the URL from a web browser, the Jupyter Notebook home page appears.

For details about how to start Jupyter Notebook, see 8.2 How to start the inference development function.

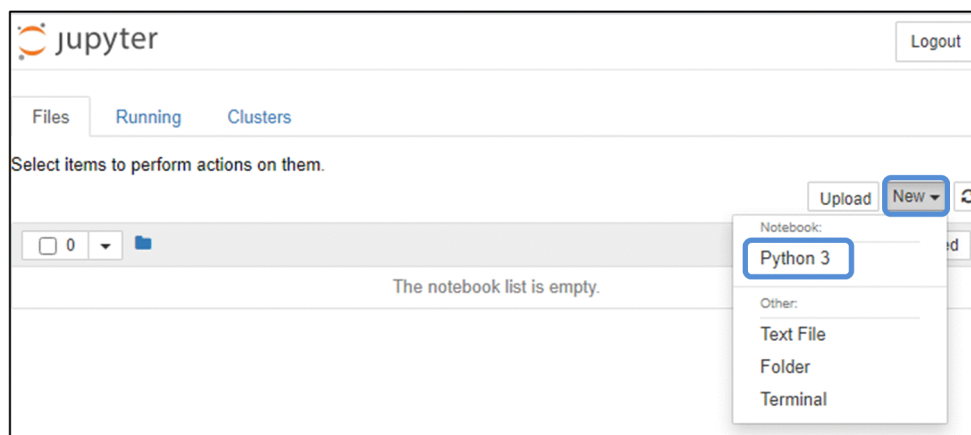
(2) Using Jupyter Notebook to create a new Python 3 notebook

The following describes how to use Jupyter Notebook to create a new Python 3 notebook:

1. On the Jupyter Notebook home page, click the **New** button.
2. In the menu that appears, click **Python 3**.

A new Python 3 notebook is created.

Figure 8–2: Creating a new notebook



(3) Running Python 3 (outputting a string to the standard output)

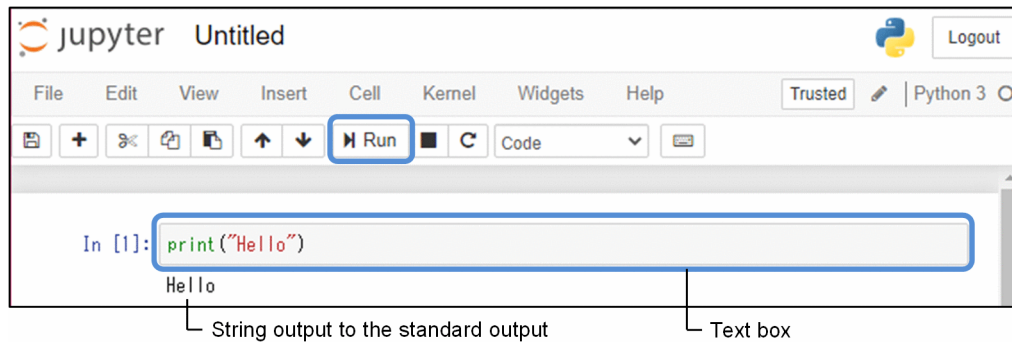
The following describes how to run Python 3 in Jupyter Notebook:

1. In the editor page of the notebook, enter Python source code in an **In** text box.
2. Click the **Run** button.

When you click the **Run** button, the code in the text box runs, and then the execution results are displayed below the text box. Note that all of the code in the **In** text boxes is executed.

The following is an example in which the `print` function is used to output a character string below the text box.

Figure 8–3: Example of outputting a character string to the standard output

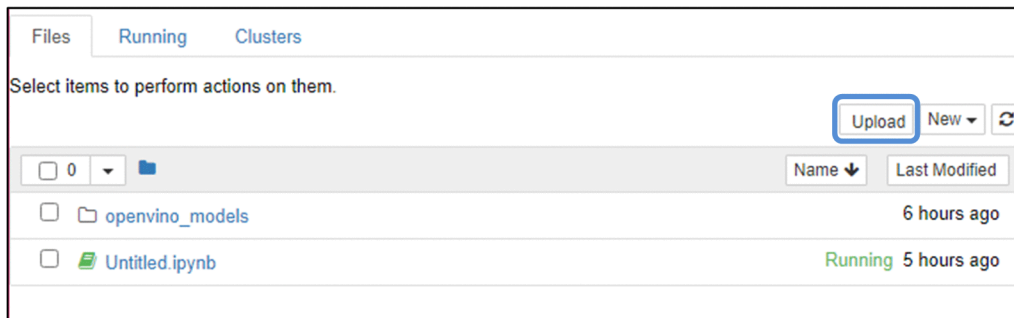


(4) Uploading and downloading files

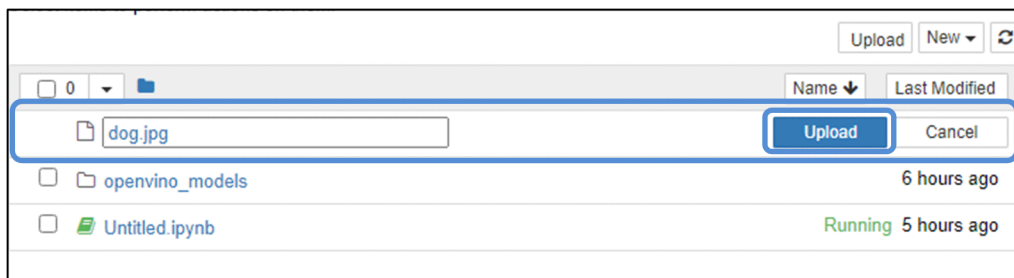
To upload a file:

The following describes how to use Jupyter Notebook to upload files:

1. On the Jupyter Notebook home page, click the **Upload** button.



2. In the Explorer window that appears, specify the files to be uploaded.
3. Confirm that the specified files are displayed on the home page, and then click the **Upload** button again.

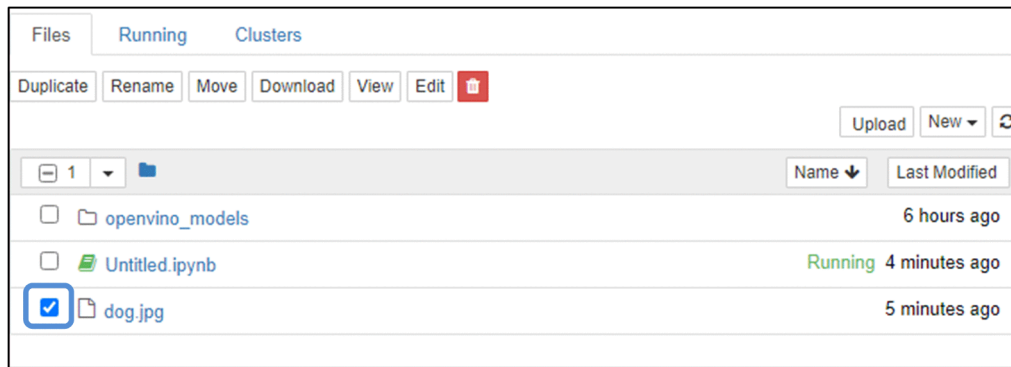


To download a file:

The following describes how to use Jupyter Notebook to download files:

1. On the Jupyter Notebook home page, select the check boxes of the files to be downloaded.

8. Inference development function



2. Click the **Download** button.



8.3.2 Procedure for using OpenVINO on Jupyter Notebook

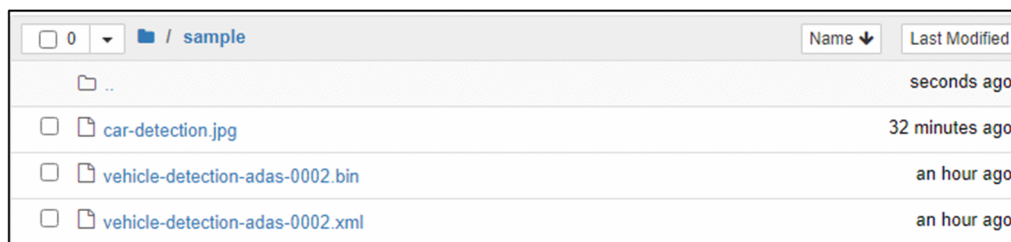
This section describes how to use OpenVINO to perform inference on Jupyter Notebook, using a case in which an automobile detection program is run as an example.

(1) Preparation

- Upload test data (`car-detection.jpg` in this example) and pre-trained model files (`.xml` and `.bin` files).

The source of the test data is as follows:

```
https://github.com/intel-iot-devkit/sample-videos/raw/master/car-detection.mp4
```



- Create a new Python 3 notebook, and then open the editor page.

(2) Procedure for using OpenVINO

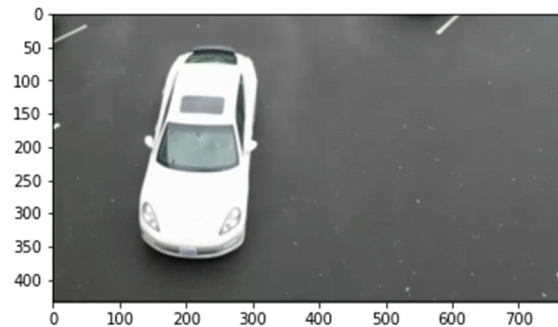
1. Load the necessary libraries.
If the libraries do not exist, install them.

```
In [1]: import cv2
import matplotlib.pyplot as plt
from openvino.inference_engine import IECore
import numpy as np
```

2. Load picture files to display them on the Jupyter Notebook page. (Images, figures, and tables are drawn in the **Out** fields.)

```
In [2]: img = cv2.imread("car-detection.jpg")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
Out [2]: <matplotlib.image.AxesImage at 0x7fb7efcc3a20>
```



3. Load the pre-trained model files.

```
In [3]: model_xml = "vehicle-detection-adas-0002.xml"
model_bin = "vehicle-detection-adas-0002.bin"
ie = IECore()
net = ie.read_network(model = model_xml, weights = model_bin)
```

4. Change the type and size of the input image to adapt the image to the pre-trained model.

```
In [4]: n, c, h, w = net.inputs['data'].shape
ih, iw = img.shape[:2]
input_img = cv2.resize(img, (w, h))
input_img = input_img.transpose((2, 0, 1))
data = {}
data['data'] = input_img
```

5. Perform inference.

```
In [5]: exec_net = ie.load_network(network=net, device_name="CPU")
res = exec_net.infer(inputs=data)
res = res['detection_out']
```

6. Prepare to output the inference results.

```
In [6]: data = res[0][0]
boxes = []
for number, proposal in enumerate(data):
    confidence = proposal[1]
    if confidence > 0.3:
        label = proposal[0]
        xmin = np.int(iw * proposal[3])
        ymin = np.int(ih * proposal[4])
        xmax = np.int(iw * proposal[5])
        ymax = np.int(ih * proposal[6])
        boxes.append([xmin, ymin, xmax, ymax])
```

7. The inference results are drawn on the page.

8. Inference development function

```
In [7]: for box in boxes:  
        img = cv2.rectangle(img, (box[0], box[1]), (box[2], box[3]), (232, 35, 244), 2)  
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
Out [7]: <matplotlib.image.AxesImage at 0x7fb7dbfdb390>
```



9

RAS techniques for the AI image application functionality

This chapter describes how to configure the settings for error detection, container restart, and the collection of operating information.

9.1 Overview of the RAS techniques for the AI image application functionality

The RAS techniques for the AI image application functionality are as follows:

- Error detection
If errors occur on function containers, log messages are output. These log messages are subject to generation management provided by CE50-10.
- Container restart
You can specify whether to restart a container that terminates abnormally.
- Collection of operating information
The log messages output by the Docker service and containers can also be collected by using the `eclogsave` command, which is provided by CE50-10 to collect maintenance information.
For details about the `eclogsave` command, see the section that describes the `eclogsave` command in the *CE50-10 Instruction Manual*. That section also describes how to use the `eclogsave` command to collect log files output by user-developed applications.

9.2 Error detection

The processes of the AI image application functionality output error messages if errors occur during operation. This section describes how to set the output destination (a log file) of the error messages in the Compose file. This section also describes the format, maximum size, and generation management of the log file.

9.2.1 Settings in the Compose file

To forward the messages output by processes running on function containers to `/var/log/docker.log` via the syslog, specify the following `logging:` setting items in the Compose file:

```
logging:
  driver: syslog
  options:
    syslog-facility: daemon
    tag: docker-compose/{{.Name}}/{{.ID}}
```

The following table describes these setting items.

Table 9–1: "logging:" setting items in the Compose file

No.	Item	Description
1	<code>driver:</code>	Specifies the logging driver. Specify <code>syslog</code> so that the messages are output to <code>/var/log/docker.log</code> via the syslog.
2	<code>options:</code>	Specifies logging-related options.
3	<code>syslog-facility:</code>	Specifies the syslog facility. Specify <code>daemon</code> so that the messages are output through a syslog filter to <code>/var/log/docker.log</code> .
4	<code>tag:</code>	Specifies the tags to be output at the beginning of each log message. Specify <code>docker-compose/{{.Name}}/{{.ID}}</code> so that the messages are output through a syslog filter to <code>/var/log/docker.log</code> . <ul style="list-style-type: none"> <code>{{.Name}}</code>: Container name <code>{{.ID}}</code>: First 12 characters of the container ID

9.2.2 Log format

The following shows the format of log messages that are output to `/var/log/docker.log`:

```
date/time host-name docker-compose/container-name/first-12-characters-of-the-container-ID[process-ID]: container-log-message
date/time host-name dockerd[process-ID]: Docker-log-message
date/time host-name containerd[process-ID]: Docker-log-message
```

9.2.3 Maximum size and generation management of the log file

The following table describes the maximum size, the number of generations that can be managed, and other information about the log file in the `/var/log` directory.

Table 9–2: Size and number of generations for the log file

No.	Log file name	Data to be logged	No. of generations	Maximum size (KB)	Collection interval	Size (KB) per interval (approx.)	Storage period (days) (approx.)
1	<code>docker.log</code>	Docker log	3	3,072 (1,024 x 3)	Undefined	13 [#]	236 [#]

#

The calculation assumes that 100 messages are output to the log per day and each message consists of a body of 50 bytes and tags of 80 bytes:

- Size of logged data per day: $(50 + 80) \times 100 = 13$ (KB/day)
- Storage period of logged data: $3,072 / 13 = 236$ (days)

Note that all messages output to the standard output of a Docker container are recorded in `docker.log`. Therefore, if you implement an application that continuously outputs information to the standard output, the log data storage period will be much shorter than expected. In such a case, you must consider the frequency and content of the messages to be logged when designing the application.

For details about other log files output to `/var/log`, see *Collecting maintenance information* in the *CE50-10 Instruction Manual*.

9.3 Container restart

If a function container ends abnormally, it can restart automatically. To enable automatic container restart, specify `restart:on-failure` in the Compose file. For details, see Table 4-1: Setting items in the Compose file.

By default, `restart:on-failure` is not specified in the Compose file. In other words, automatic container restart is disabled.

After the keyword `on-failure`, you can specify the maximum number of retries. If you want to specify this, determine the appropriate value by referring to the following table, which describes the advantages and disadvantages of specifying a larger value or a smaller value as the maximum number of retries.

Table 9-3: Advantages and disadvantages of specifying a larger value or a smaller value as the maximum number of retries

No.	Maximum number of retries	Advantages	Disadvantages
1	Larger	The likelihood of successful failure recovery increases.	The messages logged for repeated retries might cause the original error message to be deleted.
2	Smaller	The likelihood that the original error message will be deleted decreases.	<ul style="list-style-type: none"> The likelihood of successful failure recovery decreases. The retry counter is not reset each time the container restarts. Therefore, even in the case of a transient failure, the likelihood of successful failure recovery will decrease in a system that has been running for a long time.

9.4 Collection of operating information

CE50-10 provides the `eclogsave` maintenance information collection command. This command collects log data recorded and saved by the OS, log files registered in the definition file by the user, and system information required for failure analysis all at one time. In CE50-10A, the `eclogsave` command also collects data from `/var/log/docker.log` by default.

Furthermore, the `eclogsave` command can collect data from logs and other files created in containers by the user if they are visible to the host.

For example, you can make files in containers visible to the host by using a shared directory that is specified for `volumes` in the Compose file.

If you want user files to be collected by `eclogsave`, specify their absolute paths on the host in the user definition file (`/hitachi/etc/save_applog.def`).

Example of specifying user files in the user definition file (`/hitachi/etc/save_applog.def`):

```
/home/edgeadm/share/applog1  
/home/edgeadm/share/applog2
```

10 Updater

This chapter provides an overview of the updater and describes the update procedure.

10.1 Overview of the updater

With Docker, software and its prerequisite files are packaged in a container as a Docker image, which can be copied to and run on another host.

Normally, the following tasks must be performed to run a program on another host:

1. Install prerequisite libraries.
2. Deploy the files required to run the program.

To run a program on multiple hosts, the preceding tasks must be performed on each host, which is time-consuming work.

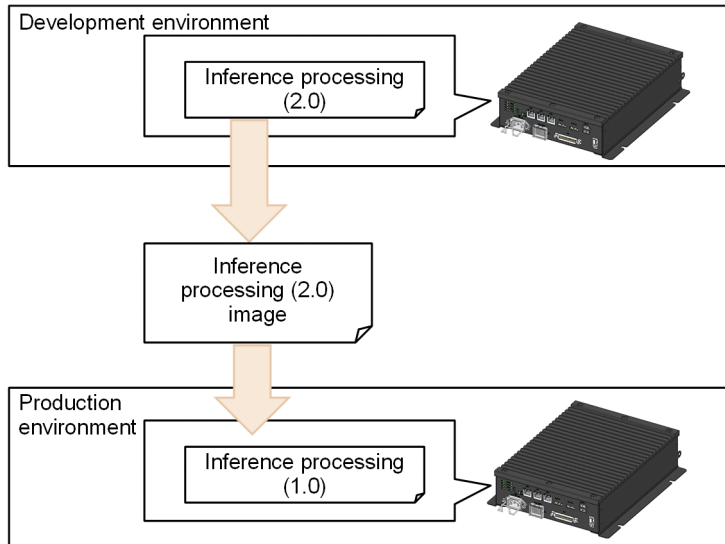
With Docker, you can handle the prerequisite libraries and files required for program execution as a single package called a Docker image. By deploying a Docker image on a host, you can easily run a program as a container from the Docker image. This greatly reduces the time required for the preceding tasks.

Docker also facilitates updates because you can update a program by simply replacing the image file.

10.2 Overview of the update procedure

The following figure shows an environment in which the update function is used.

Figure 10–1: Example of using the update function

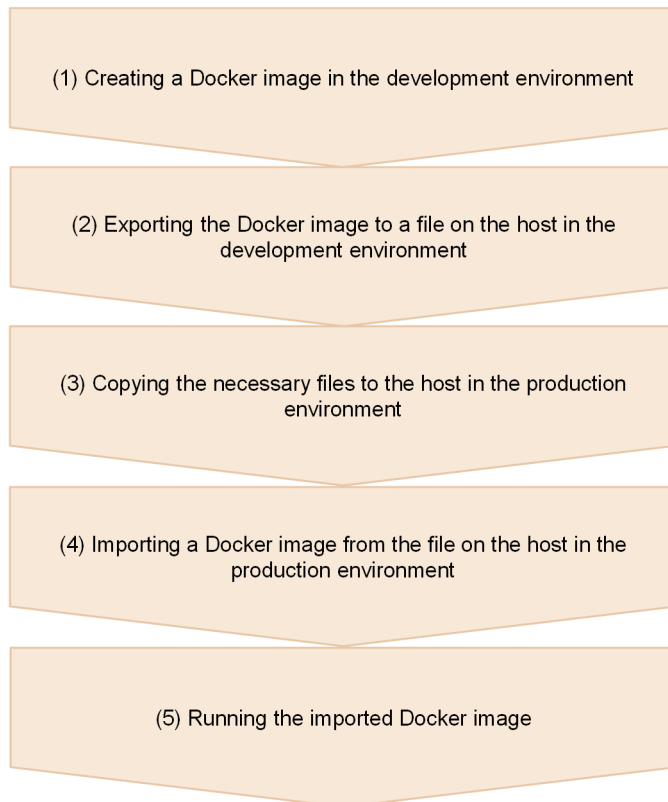


The description in this chapter assumes that inference processing version 1.0 is running on CE50-10A in the production environment and that inference processing version 2.0 is running on CE50-10A in the development environment.

10.2.1 Update procedure

The following figure provides an overview of how to update a program from version 1.0 to version 2.0.

Figure 10–2: Overview of updating a program



Although you can perform tasks (1) to (4) while the container of the inference execution function is running, we recommend that you stop the container before performing these tasks.

(1) Creating a Docker image in the development environment

After you have created a settings file and inference program, you can include them in a Docker image provided by CE50-10A and then register that Docker image in Docker as a new Docker image.

For details about the procedure, see (2) Creating a Docker image with a file installed.

The description in this section assumes that you are creating a Docker image of the container of the inference execution function. However, the same procedure applies when you create a Docker image of the container of the data management function or the container of the data input function.

(2) Exporting the Docker image to a file on the host in the development environment

Run the following command to export the Docker image to a file.

Example of the command to be run:

```
$ sudo docker save people_count:2.0 > people_count_2_0.tar
```

(3) Copying the necessary files to the host in the production environment

Copy the necessary files to the host. For example, you can do so by using SCP to transfer the files via a network or by using storage media such as a USB memory drive.

To copy the files via a network, run the following command on the host in the development environment.

Example of the command to be run:

```
$ scp ./people_count_2_0.tar edgeadm@IP-address-of-the-host-in-the-production-environment:image-file-transfer-destination-path
```

(4) Importing a Docker image from the file on the host in the production environment

Run the following command to import the Docker image to a file.

Example of the command to be run:

```
$ cd image-file-path
$ sudo docker load < people_count_2_0.tar
```

(5) Running the imported Docker image

If the Docker image name has been changed, change the `image:` value accordingly in the Compose file as follows.

Before the change:

```
services:
  people_count:
    image: people_count:1.0
```

After the change:

```
services:
  people_count:
    image: people_count:2.0
```

Run the following command to start the service for which the Docker image has been updated. (Internally, the current container stops and then a new container starts.)

```
$ cd path-of-the-directory-that-contains-docker-compose.yaml
$ sudo docker-compose up -d service-name
```

11

Troubleshooting

This chapter describes the error messages that are displayed by the AI image application functionality and the action to take for each error message.

11.1 Error messages that might be displayed when the docker-compose command is run

This section describes the error messages that might be displayed when the `docker-compose` command is run and the action to take for each error message.

Table 11–1: Error messages that might be displayed when the docker-compose command is run

No.	Message	Description	Action
1	<pre>ERROR: Can't find a suitable configuration file in this directory or any parent. Are you in the right directory? Supported filename s: docker-compose.yml, docker- compose.yml</pre>	The Compose file could not be found.	Specify the Compose file for the <code>-f</code> option. Alternatively, place the Compose file named <code>docker-compose.yml</code> or <code>docker-compose.yml</code> in the current directory.
2	<pre>Pulling XXXX (YYYY:ZZZZ) .. ERROR: Get https://registry-1.docker.io/v2/: dial tcp: lookup registry-1.docker.io: Temporary failure in _name resolution#</pre>	<p>The Docker image specified in the Compose file could not be found.</p> <p><i>XXXX</i>: Service name <i>YYYY</i>: Docker image name <i>ZZZZ</i>: Tag name</p>	Make sure that the Docker image name specified in the Compose file is correct.
3	<pre>ERROR: for XXXX Cannot start service XXXX: OCI runtime create failed: container_linux.go:345#: starting container process caused "exec: \"YYYY\": stat YYYY: no such file or directory": unknown#</pre>	<p>The command could not be run.</p> <p><i>XXXX</i>: Service name <i>YYYY</i>: Command</p>	Make sure that the command name and path are correct.
4	<pre>ERROR: Service 'XXXX' depends on service 'YYYY' which is undefined.</pre>	The <i>YYYY</i> service specified for <code>depends_on</code> of the <i>XXXX</i> service could not be found.	Revise the container dependency and specify a correct value for <code>depends_on</code> .
5	<pre>ERROR: No containers to start</pre>	There is no container to be started.	Check whether the container to be started by the <code>docker-compose ps</code> command exists. If it does not exist, run <code>docker-compose up [-d]</code> to generate a container, and then start the container.

#

The underlined portions might change depending on the network environment or the command that is run.

11.2 Error messages of the data input function

This section describes the error messages that might be displayed when the data input function is used and the action to take for each error message.

Table 11–2: Error messages of the data input function

No.	Message	Description	Action
1	Failed to parse <i>path-of-data_input.json</i>	The settings in the <i>data_input.json</i> file are incorrectly formatted.	Revise the settings in the <i>data_input.json</i> file.
2	Unable to set the pipeline to the playing state. (video_input:1): GStreamer-CRITICAL **: 22:56:50.374 : gst_element_get_bus: assertion 'GST_IS_ELEMENT (element)' failed Failed to get bus. or Error: Could not open resource for reading and writing. or Error: Unauthorized	A connection to the camera could not be established.	Make sure that the camera is correctly connected. In addition, revise the settings in the <i>docker-compose.yaml</i> file.
3	Error received from element ximagesink0: Could not initialise X output Debugging information: ximagesink.c(860): gst_x_image_sink_xcontext_get(): /GstPipeline:pipeline0/GstXImageSink:ximagesink0: Could not open display	The function failed to display a preview on a monitor.	Make sure that the X forwarding settings are correctly specified in the <i>docker-compose.yaml</i> file.
4	Error: Internal data stream error.	An error occurred on the V4L driver when a USB camera was used.	Make sure that the USB camera is correctly connected and that proper resolution and FPS values are set.

11.3 Error messages of the data management function

This section describes the error messages that might be displayed when the data management function is used and the action to take for each error message.

Table 11-3: Error messages of the data management function

No.	Message	Description	Action
1	Failed to parse <i>path-of-data_manager.json</i>	The settings in the <i>data_manager.json</i> file are incorrectly formatted.	Revise the settings in the <i>data_manager.json</i> file.
2	Failed to parse <i>path-of-dataman.json</i>	The <i>dataman.json</i> file could not be read. Alternatively, the format of the file is incorrect.	Revise the settings in the <i>dataman.json</i> file.
3	The value is over usable volume size.	The value of <i>unfinished_limit</i> or <i>finished_limit</i> exceeds the size of free space on the volume.	Revise the values so that they are less than the size of free space on the volume.
4	Failed to connect Data Manager API.	The function could not access the API.	Make sure that the URL specified in the <i>dataman.json</i> file is correct. In addition, make sure that the port settings for the container of the data management function in the <i>docker-compose.yaml</i> file are correct.

11.4 Error messages of the inference execution function

This section describes the error messages that might be displayed when the inference execution function is used and the action to take for each error message.

Table 11–4: Error messages of the inference execution function

No.	Message	Description	Action
1	Failed to connect Data Manager API.	After the container of the inference execution function started, no visual file could be received from the container of the data management function.	Check the status of the container of the data management function. If the container has not successfully started, start it by referring to 4.3.1 How to start Docker containers.
2	Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?	The Docker daemon is not running.	Start the Docker daemon by referring to 3.1.2 Starting the Docker service.
3	RuntimeError: Failed to create plugin /opt/intel/openvino/deployment_tools/inference_engine/lib/intel64/libclDNNPlugin.so for device GPU	The GPU could not start in the container.	Specify the settings that allocate a GPU device to the container in the <code>docker-compose.yaml</code> file by referring to 4.1 Using the Compose file to customize the functionality.

11.5 Error messages of the inference development function

This section describes the error messages that might be displayed when the inference development function is used and the action to take for each error message.

Table 11–5: Error messages of the inference development function

No.	Message	Description	Action
1	<pre>docker: Error response from daemon: driver failed programming external connectivity on endpoint ***: (iptables failed: iptables -wait -t filter -A DOCKER ! -i docker0 -o docker0 -p tcp -d *** --dport 8888 -j ACCEPT: iptables: No chain/target/match by that name.(exit status 1)).</pre>	The ports of CE50-10A and the container could not be connected.	Restart the Docker daemon.

Appendix

A. Interfaces of CE50-10A

This appendix describes the interfaces supported by CE50-10A.

A.1 Supported specifications

The following table shows the specifications supported by the AI image application functionality.

Table A–1: Supported specifications

No.	Item	Specifications
1	Camera	<ul style="list-style-type: none"> • USB camera connected via USB 2.0 or 3.0 (compatible with USB Video Class 1.0 or 1.1) • IP camera (RTSP) Codec: - USB camera: Uncompressed and Motion-JPEG - IP camera: Motion-JPEG and H.264
2	Input file format	JPEG, PNG, MPEG-2, and MPEG-4 (H.264)
3	Maximum resolution	3,840 x 2,160
4	Inference processing development language	Python
5	Pre-trained model	OpenVINO IR (extension: .xml or .bin) Precision: FP32, FP16, or INT8
6	Device that performs inference	CPU or GPU
7	Development environment	OpenVINO, Jupyter Notebook, or Python
8	PC used to perform operations via a network	<ul style="list-style-type: none"> • OS: Windows 10 version 1803 or later • Web browser: Google Chrome