

日立制御エッジコンピュータ

CE50-10A ユーザーズガイド

CC-5-0179

■ 対象製品

CE50-10A（適用 OS：Ubuntu 18.04 LTS（Linux カーネル 4.15））

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認のうえ、必要な手続きをお取りください。

なお、不明な場合は、お買い求め先にお問い合わせください。

■ 商標類

HITACHI は、株式会社 日立製作所の登録商標です。

Google Chrome は、Google LLC の商標です。

Intel および OpenVINO は、アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。

Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

Microsoft および Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

PostgreSQL は、PostgreSQL Community Association of Canada のカナダにおける登録商標およびその他の国における商標です。

Ubuntu は、Canonical Ltd. の登録商標または商標です。

その他記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。

■ ソフトウェアについて

本製品には、その機能を実現するため複数のオープンソースソフトウェアが搭載されています。搭載したオープンソースソフトウェアの著作権、所有権および知的財産権は作成者が保持しており、同梱されているライセンスに示された条件の下で使用できます。本製品はライセンスの内容をご確認の上、ライセンスに従って使用してください。詳細は本製品内の以下のファイルを参照してください。

/hitachi/licenses/copyright.txt

ライセンスにソースコードの提供が定められているオープンソースソフトウェアは、当社よりソースコードを提供いたします。希望される場合は当社窓口にご連絡ください。

当社は、オープンソースソフトウェアの保証は行いません。その使用に関し、損害賠償責任を含む一切の責任を負いません。

これは、この装置に対する製品責任を放棄する意味ではありません。

■ 注意

このマニュアルの内容はすべて著作権によって保護されています。このマニュアルの内容の一部または全部を無断で複製することはできません。このマニュアルの内容を、改良のため予告なしに変更することがあります。

マニュアルはよく読み、保管してください。操作する前に、安全上の指示をよく読み、十分理解してください。このマニュアルは、いつでも参照できるよう、手近な所に保管してください。

■ 発行

2020 年 12 月 CC-5-0179

■ 著作権

All Rights Reserved. Copyright (C) 2020, Hitachi, Ltd.

はじめに

このマニュアルは、日立制御エッジコンピュータ CE50-10A（以降、CE50-10A と略す）が提供する AI 映像アプリ開発実行処理機能（以降、AI 映像アプリ機能と略す）の概要、操作方法などについて説明したものです。

■ 対象読者

このマニュアルは、CE50-10A を操作する方（オペレーター、システムエンジニア、保守員）を対象としています。

■ マニュアルの構成

このマニュアルは、次に示す章と付録から構成されています。

第1編 解説編

第1章 機能概要

CE50-10A が提供する機能の概要について説明しています。

第2編 体験編

第2章 AI 映像アプリ機能体験ナビ

AI 映像アプリ機能を体験するためのセットアップ手順、サンプルプログラムの実行手順について説明しています。

第3編 構築編

第3章 AI 映像アプリ機能のセットアップ

AI 映像アプリ機能のセットアップ手順について説明しています。

第4編 設計・機能編

第4章 AI 映像アプリ機能の設計

AI 映像アプリ機能をユーザーが独自に設定する方法、AI 映像アプリ機能の起動/停止方法などについて説明しています。

第5章 データ入力機能

データ入力機能の概要、カメラの接続方法、設定方法などについて説明しています。

第6章 データ管理機能

データ管理機能の概要、API およびライブラリの仕様、設定方法について説明しています。

第7章 推論実行機能

推論実行機能の概要、推論処理を組み込む流れ、サンプルプログラムについて説明しています。

第8章 推論開発機能

推論開発機能の概要、起動手順、Jupyter Notebook の利用方法について説明しています。

第9章 AI 映像アプリ機能向け RAS 機能

エラー検出、コンテナの再起動、稼働情報収集の設定方法について説明しています。

第5編 運用編

第10章 アップデート機能

アップデート機能の概要、アップデート方法について説明しています。

第11章 トラブルシューティング

AI映像アプリ機能が表示するエラーメッセージと、その対処方法について説明しています。

付録A CE50-10Aのインターフェース

CE50-10Aがサポートするインターフェースについて説明しています。

■ 関連マニュアル

関連マニュアルを次に示します。必要に応じてお読みください。

- 日立制御エッジコンピュータ CE50-10 取扱説明書 (CC-5-0171)

■ このマニュアルでの表記

このマニュアルでは、製品名称について次のように表記しています。

製品名称	略称	説明
日立制御エッジコンピュータ CE50-10	CE50-10	CE50-10 シリーズの総称です。
日立制御エッジコンピュータ CE50-10A	CE50-10A	CE50-10 シリーズの中で、AI映像アプリ機能を搭載するモデルです。

■ このマニュアルで使用する書式

このマニュアルで使用する書式を説明します。

書式	説明
<>	可変の値を示します。 (例) 日付は<yyyyMMdd>の形式で指定します。
[]	ウィンドウ、ダイアログボックス、メニュー、ボタンなどの画面上の要素名を示します。
[] キー	キーボードのキーを示します。

■ このマニュアルで使用する KB (キロバイト) などの単位表記

1KB (キロバイト)、1MB (メガバイト)、1GB (ギガバイト)、1TB (テラバイト) はそれぞれ 1,024 バイト、1,024² バイト、1,024³ バイト、1,024⁴ バイトです。

目次

第 1 編 解説編

1	機能概要	1
1.1	CE50-10A でできること	2
1.2	AI 映像アプリ機能	4
1.2.1	OpenVINO	4

第 2 編 体験編

2	AI 映像アプリ機能体験ナビ	7
2.1	AI 映像アプリ機能を体験する流れ	8
2.1.1	Docker 用パーティションを作成する	9
2.1.2	Docker 用パーティションを確認する	9
2.1.3	Docker 用パーティションにファイルシステムを作成する	10
2.1.4	Docker 用パーティションをマウントする	10
2.1.5	Docker サービスを起動する	10
2.1.6	Docker イメージを組み込む	11
2.2	サンプルプログラムの説明	12
2.2.1	サンプル動画入力のサンプルプログラムを実行する	12
2.2.2	USB カメラ入力のサンプルプログラムを実行する	14

第 3 編 構築編

3	AI 映像アプリ機能のセットアップ	17
3.1	AI 映像アプリ機能のセットアップ手順	18
3.1.1	Docker 用パーティションの作成/マウント	18
3.1.2	Docker サービスの起動	18
3.1.3	Docker イメージの組み込み	18
3.2	パーティションを表示する	19

第4編 設計・機能編

4	AI 映像アプリ機能の設計	21
4.1	Compose ファイルを利用した使用方法	22
4.2	OS 起動時に Docker コンテナを自動起動する手順	25
4.2.1	Docker サービスを自動起動する手順	25
4.2.2	Docker コンテナを自動起動する手順	25
4.3	AI 映像アプリ機能の起動/停止方法	26
4.3.1	Docker コンテナの起動方法	26
4.3.2	Docker コンテナの停止方法	26
4.3.3	Docker コンテナの状態確認方法	27
4.4	システム設計上の留意点	28
4.4.1	提供コンテナイメージ	28
4.4.2	Docker コンテナの自動再起動	28
4.4.3	取得フレーム数のチューニング	29
4.4.4	ファイアウォールの設定	29
4.4.5	セキュリティリスクとその対策	31
5	データ入力機能	33
5.1	データ入力機能の概要	34
5.2	カメラ接続方法	35
5.2.1	USB カメラを接続する場合	35
5.2.2	IP カメラを接続する場合	35
5.3	データ入力機能の設定方法	36
5.3.1	データ入力機能の設定内容	36
5.3.2	コンテナの設定内容	39
5.4	動作確認手順	41
6	データ管理機能	43
6.1	データ管理機能の概要	44
6.2	データ管理機能の API 仕様	46
6.2.1	ファイルをデータ管理機能に登録する (v1FilesPost)	46
6.2.2	ファイルの状態を更新する (v1FilesFileIdPut)	47
6.2.3	ファイルを取得する (v1FilesGet)	47
6.3	データ管理機能で利用できるライブラリ仕様	49
6.3.1	ライブラリの利用形態	49
6.3.2	ライブラリを利用するための設定内容	49
6.3.3	ライブラリをインポートする方法	50
6.3.4	Dataman クラスメソッド	50

6.3.5	ファイルステータスの定数	51
6.4	データ管理機能の設定方法	52
6.4.1	データ管理機能の設定ファイル	52
6.4.2	Compose ファイルの設定内容	52
7	推論実行機能	55
7.1	推論実行機能の概要	56
7.1.1	OpenVINO 用に提供されている学習済みモデルの入手方法	56
7.1.2	学習済みモデルを中間表現へ変換する	56
7.2	推論処理を組み込む流れ	58
7.2.1	推論処理の組み込み方法	58
7.3	サンプルプログラムの解説	63
7.3.1	サンプルプログラムの処理内容	63
8	推論開発機能	69
8.1	推論開発機能の概要	70
8.2	推論開発機能の起動手順	71
8.3	Jupyter Notebook の利用方法	72
8.3.1	Jupyter NoteBook の基本的な使用方法	72
8.3.2	Jupyter NoteBook 上で OpenVINO を使う手順	74
9	AI 映像アプリ機能向け RAS 機能	77
9.1	AI 映像アプリ機能向け RAS 機能の概要	78
9.2	エラー検出	79
9.2.1	Compose ファイルの設定内容	79
9.2.2	ログフォーマット	79
9.2.3	ログファイル容量/世代管理	79
9.3	コンテナの再起動	81
9.4	稼働情報収集登録方法	82
第 5 編 運用編		
10	アップデート機能	83
10.1	アップデート機能の概要	84
10.2	アップデートの流れ	85
10.2.1	アップデート方法	85

11	トラブルシューティング	89
11.1	docker-compose コマンド実行時のエラーメッセージ	90
11.2	データ入力機能のエラーメッセージ	91
11.3	データ管理機能のエラーメッセージ	92
11.4	推論実行機能のエラーメッセージ	93
11.5	推論開発機能のエラーメッセージ	94
付録		95
付録 A	CE50-10A のインタフェース	96
付録 A.1	サポート仕様	96

1

機能概要

CE50-10A が提供する機能の概要について説明します。

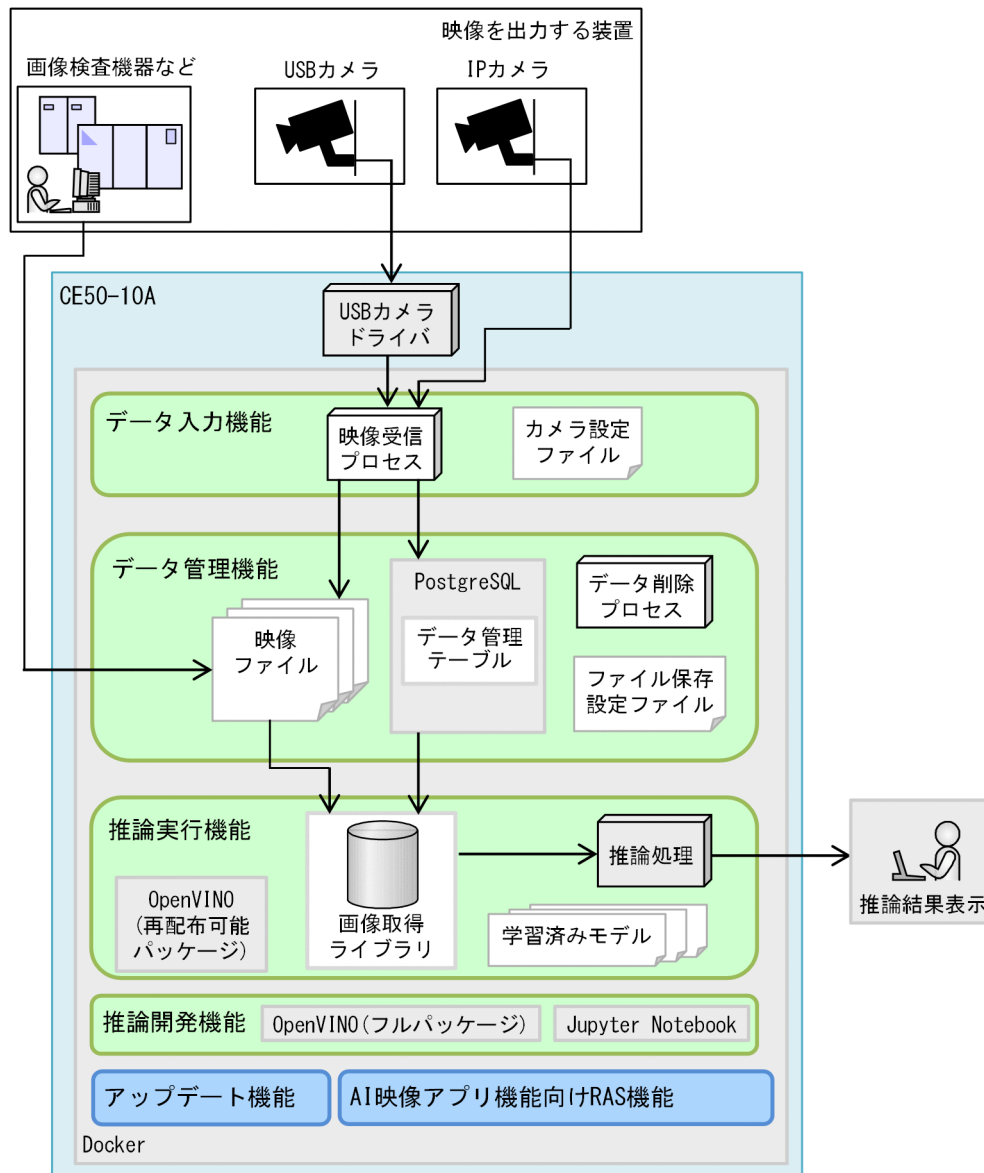
1.1 CE50-10A でできること

CE50-10A は、USB カメラ、IP カメラ、画像検査機器などから取得した映像データを使用して AI 推論処理（以降、推論処理と略す）を実施します。推論処理の結果を自動的にディスプレイに表示させることもでき、ユーザーは、その推論処理の結果から適切な判断を下せます。また、CE50-10A は、ユーザーが自由に推論処理の開発や推論結果を用いた独自処理の組み込みなどを実装できる環境を提供しています。

学習済みモデルを組み込んだ推論処理を活用し、様々なソリューションを短期間で構築することができます。

CE50-10A が提供する機能と映像データの処理の流れを、次の図に示します。

図 1-1 CE50-10A が提供する機能と映像データの処理の流れ



(凡例)

■ : 各機能のコンテナ ■ : CE50-10Aの運用を支援する機能

→ : 映像データの処理の流れ

Docker の説明は、「CE50-10 取扱説明書」の「コンテナ機能 (Docker)」を参照してください。

次に示す機能を総称して「AI 映像アプリ機能」と表記します。

- データ入力機能
- データ管理機能
- 推論実行機能
- 推論開発機能
- AI 映像アプリ機能向け RAS 機能
- アップデート機能

各機能の概要については、「1.2 AI 映像アプリ機能」を参照してください。

1.2 AI 映像アプリ機能

AI 映像アプリ機能の各機能の説明を、次の表に示します。

表 1-1 AI 映像アプリ機能

項番	機能名	内容	参照先
1	データ入力機能	USB カメラや IP カメラから取得した映像データを、動画ファイルや画像ファイル（以降、映像ファイルと略す）として生成します。設定ファイルに記載されたカメラ接続情報に従って映像ファイルを生成し、データ管理機能へ渡します。	5 データ入力機能
2	データ管理機能	データ入力機能から渡された映像ファイル、および画像検査装置などから転送された映像ファイルを管理します。 管理している映像ファイルを、推論処理からの要求に応じて推論実行機能へ渡します。	6 データ管理機能
3	推論実行機能	ユーザーが作成した学習済みモデルまたは OpenVINO 向けに提供されている学習済みモデルを使って、推論処理を実行します。	<ul style="list-style-type: none"> • 1.2.1 OpenVINO • 7 推論実行機能
4	推論開発機能	データ分析や AI 開発で標準的に使用されている Jupyter Notebook を使用して独自の推論処理を開発できます。	8 推論開発機能
5	AI 映像アプリ機能向け RAS 機能	AI 映像アプリ機能向け RAS 機能では、次の機能があります。 <ul style="list-style-type: none"> • エラー検出 • コンテナの再起動 • 稼働情報収集の設定 	9 AI 映像アプリ機能向け RAS 機能
6	アップデート機能	アップデート機能は、ユーザーが開発した推論処理のプログラム、およびソフトウェアをコンテナ単位でアップデートできます。	10 アップデート機能

1.2.1 OpenVINO

CE50-10A は、OpenVINO を同梱しています。OpenVINO は、Intel アーキテクチャ上で推論処理ができるソフトウェアのことです。OpenVINO は、Model Optimizer と Inference Engine の 2 つから構成されています。それぞれの役割は、次のとおりです。

- Model Optimizer

Tensorflow、Caffe、mxnet など構築された学習済みモデルを、推論処理時に用いる OpenVINO 用のフォーマットに変換・最適化する機能です。

Model Optimizer によって、様々なフレームワークで構築した学習済みモデルを OpenVINO 上で動作させることができます。

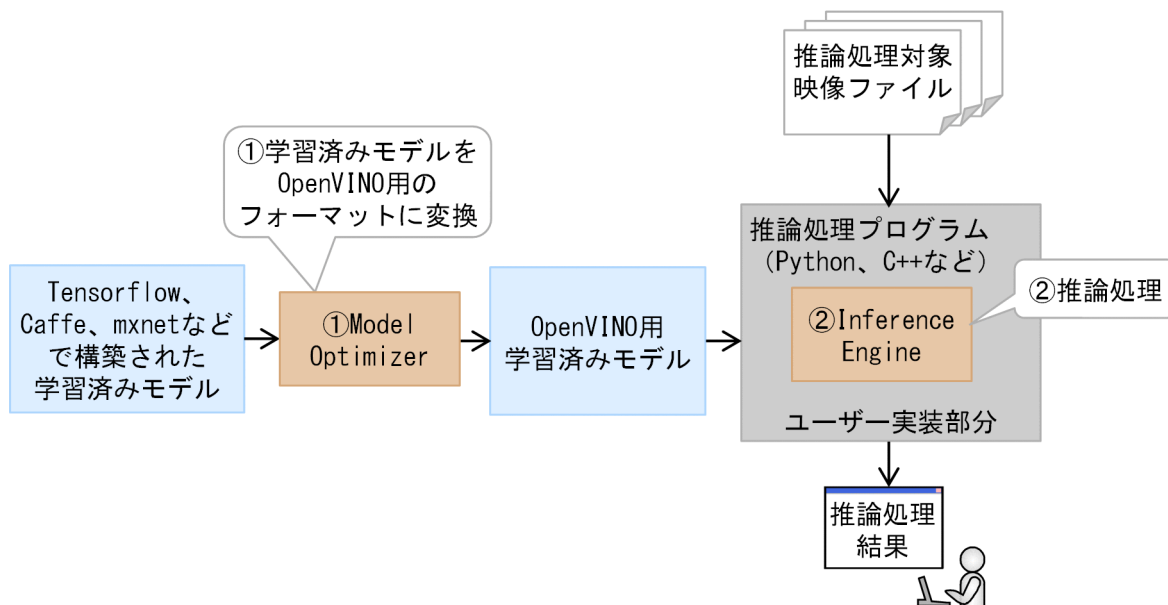
- Inference Engine

ニューラルネットワークの学習済みモデルを用いて、Intel アーキテクチャ上で高速に推論処理する機能です。OpenVINO は、推論処理を実行する Inference Engine ソフトウェアとそれを呼び出すライブ

ラリを同梱しています。ユーザーは、Python や C++ などから呼び出すことで推論処理の機能を使用できます。

OpenVINO に含まれる 2 つの機能を、次の図に示します。

図 1-2 OpenVINO に含まれる 2 つの機能



OpenVINO では、Intel が作成した学習済みモデルを提供しています。提供されるモデル例を、次の表に示します。

表 1-2 OpenVINO の学習済みモデル例

項番	分類	説明
1	Object Detection	物体を検出するモデルです。顔検出・人検出・歩行者/車検出などです。
2	Object Recognition	物体を認識するモデルです。人の年齢/性別認識・顔の向き認識・車のナンバープレート認識・感情認識などです。
3	Semantic Segmentation	画像内の全画素にラベルやカテゴリを関連づけるモデルです。
4	Human Pose Estimation	骨格を検出するモデルです。
5	Image Processing	画像に対して推論処理を行うモデルです。画像の高解像度化などです。
6	Text Detection	文字を検出するモデルです。
7	Text Recognition	文字を認識するモデルです。数字認識・アルファベット認識・日本語認識などです。
8	Action Recognition	行動を認識するモデルです。運転手の行動認識・手話認識などです。
9	Image Retrieval	画像を補完するモデルです。

2

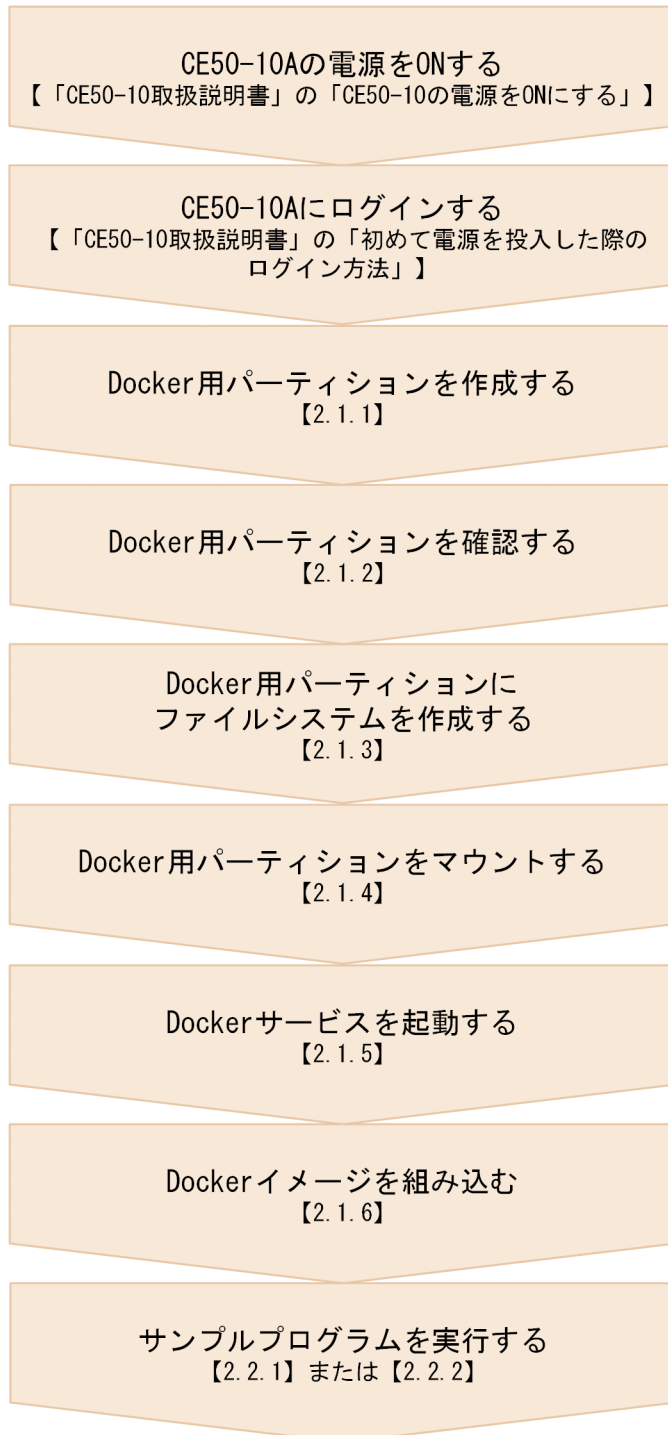
AI 映像アプリ機能体験ナビ

AI 映像アプリ機能を体験するためのセットアップ手順、サンプルプログラムの実行手順について説明します。

2.1 AI 映像アプリ機能を体験する流れ

AI 映像アプリ機能を体験する流れを説明します。

図 2-1 AI 映像アプリ機能を体験する流れ



(凡例)

【 】: 参照先を示す。

2.1.1 Docker 用パーティションを作成する

ここでは、サンプルプログラムを動作させるために 15GB の Docker 用パーティションを作成する手順を、次に示します。

1. `gdisk` コマンドを `/dev/sda` を引数にして起動します。

```
$ 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. 新規パーティションの作成コマンド `n` を入力します。

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

3. ここでは何も入力しないで `[Enter]` キーを押します。

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

4. ここでは何も入力しないで `[Enter]` キーを押します。

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

5. 15GB のサイズ指定をするため、`+15G` を入力して `[Enter]` キーを押します。

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

6. ここでは何も入力しないで `[Enter]` キーを押します。

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

7. ディスクへの反映コマンド `w` を入力します。

```
Command (? for help):w

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

8. 確認メッセージが表示されるので、`Y` を入力します。

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

9. ディスクへの反映が完了すると、次のメッセージを表示してコマンドが終了します。

```
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.
```

これで Docker 用パーティションが作成できます。

2.1.2 Docker 用パーティションを確認する

「2.1.1 Docker 用パーティションを作成する」で、作成した Docker 用パーティションを確認する手順を、次に示します。

1. 次のコマンドを実行して、パーティションの一覧を表示します。

```
$ ls /dev/sda*
/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4 /dev/sda5 /dev/sda6 /dev/sda7 /dev/sda8
```

2. 次のコマンドを実行して、OS にパーティション変更を通知します。

```
$ sudo partprobe
```

本コマンドによって OS がパーティションを認識するため、OS を再起動する必要はありません。

3. 次のコマンドを使用して、パーティションの一覧を表示します。

```
$ ls /dev/sda*
/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4 /dev/sda5 /dev/sda6 /dev/sda7 /dev/sda8 /dev/sda9
```

「2.1.1 Docker 用パーティションを作成する」で作成した Docker 用パーティションは、/dev/sda9 として OS に認識されます。

2.1.3 Docker 用パーティションにファイルシステムを作成する

「2.1.1 Docker 用パーティションを作成する」で、作成した Docker 用パーティションにファイルシステムを作成する手順を、次に示します。

1. 次のコマンドを実行して、EXT4 のファイルシステムを作成します。

```
$ 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 Docker 用パーティションをマウントする

「2.1.1 Docker 用パーティションを作成する」で、作成した Docker 用パーティションをマウントする手順を、次に示します。

1. docker を停止します。

すでに docker を停止している場合は、手順 2.へ進んでください。

```
$ sudo systemctl stop docker
```

2. 作成した Docker 用パーティションを、/var/lib/docker にマウントします。

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

2.1.5 Docker サービスを起動する

Docker サービスを起動する手順を、次に示します。

1. 次のコマンドを実行して、Docker を起動します。

```
$ sudo systemctl start docker
```

2.1.6 Docker イメージを組み込む

Docker イメージを組み込むための手順を、次に示します。

1. 次のコマンドを実行して、Docker イメージを組み込みます。

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

2.2 サンプルプログラムの説明

ここでは、サンプルプログラムおよびサンプル Compose ファイルを使って、AI 映像アプリ機能の使用方法を説明しています。

CE50-10A では、次のサンプルプログラムを用意しています。

- サンプル動画を入力とし、映っている人数を画面上に表示し、検出した人数によって AP ランプの点灯状態を制御する（AP ランプについては、「CE50-10 取扱説明書」の「RAS インジケーター（AP、E3、E2、E1）」を参照のこと）。
- USB カメラから取得した映像を入力とし、USB カメラに映っている人数を画面上に表示し、検出した人数によって AP ランプの点灯状態を制御する。

入力となるサンプル動画および USB カメラからの動画は、画面で確認できます。

サンプルプログラムは AP ランプの点灯状態を制御するため、独自のアプリケーションで AP ランプを制御している場合は十分注意してください。

また、AP ランプはサンプルプログラムの動作終了時の状態を保持しますので、必要に応じて次のコマンドを実行して AP ランプを消灯させてください。

```
$ sudo rasledctl -off
```


以降に、サンプルプログラムを実行する手順を説明します。

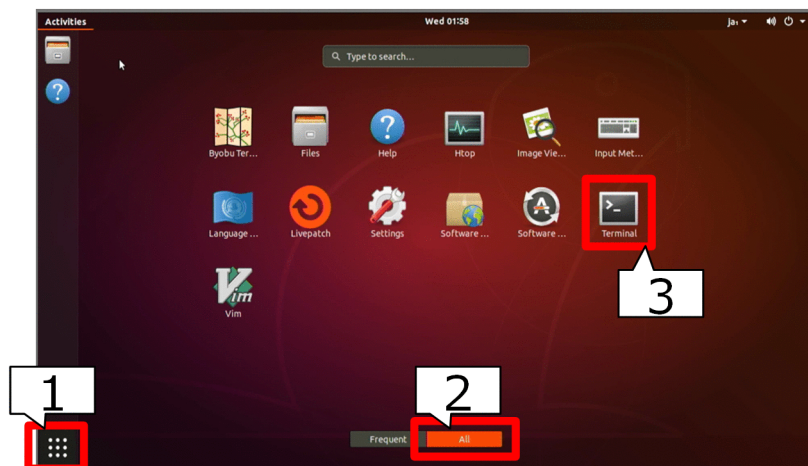
2.2.1 サンプル動画入力のサンプルプログラムを実行する

1. 次のコマンドを実行し、デスクトップを起動します。

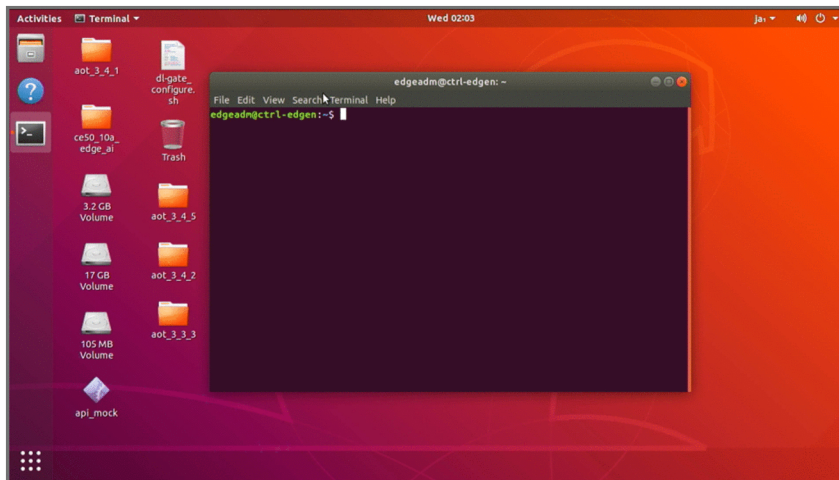
```
$ sudo systemctl start gdm3
```

2. 表示されたログイン画面から、edgeadm ユーザーでログインします。
3. 次に示す手順に従って Terminal を開きます。

1. 左下の  をクリック。
2. [All] をクリック。
3. Terminal をクリック。



Terminal が開きます。



4. 次のコマンドを実行し、Docker 上のプログラムが X サーバに接続することを許可します。

```
$ xhost local:
```

5. 「7.2.1 推論処理の組み込み方法」を参照してサンプル動画をダウンロードし、ファイル名を「sample_video.mp4」として次のように配置します。

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

6. 次のコマンドを実行し、AP ランプ操作プロセスを起動します。

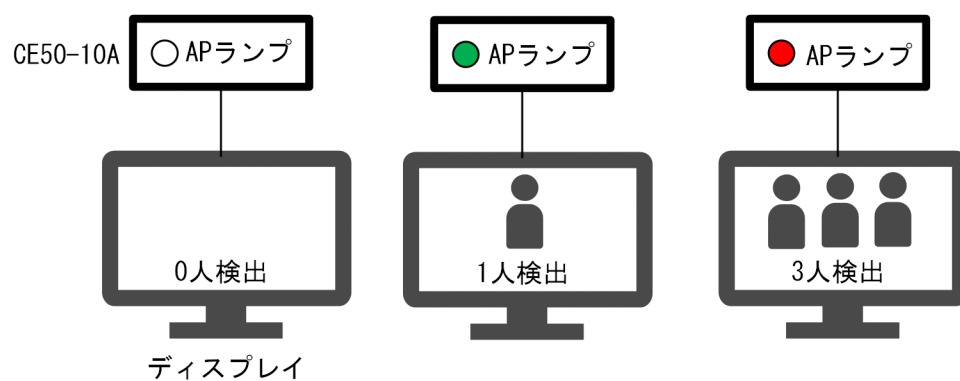
```
$ sudo /hitachi/ctrl_edge_ai/sample/people_count/start_AP
```

7. 次のコマンドを実行し、サンプルプログラムを含むコンテナを起動します。

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

8. 実行結果を確認します。

図 2-2 サンプルプログラム実行時のイメージ (サンプル動画)



9. サンプルプログラムを停止する場合、次のコマンドを実行してください。


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

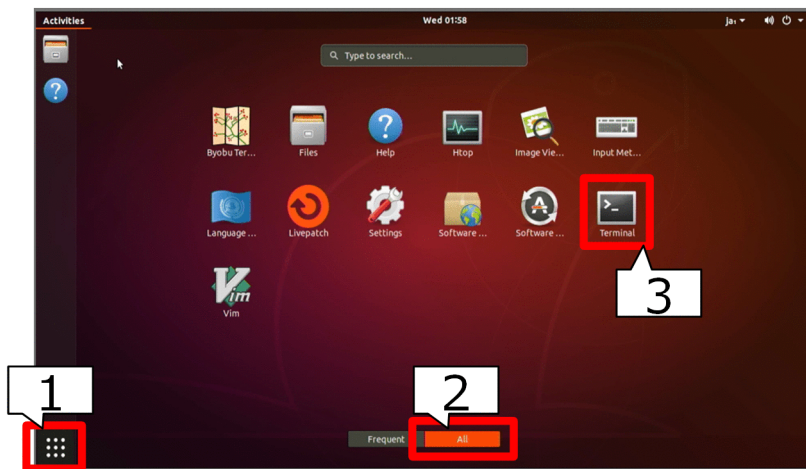
2.2.2 USB カメラ入力のサンプルプログラムを実行する

1. USB カメラのケーブルを CE50-10A の USB ポートに接続します。
2. 次のコマンドを実行し、デスクトップを起動します。

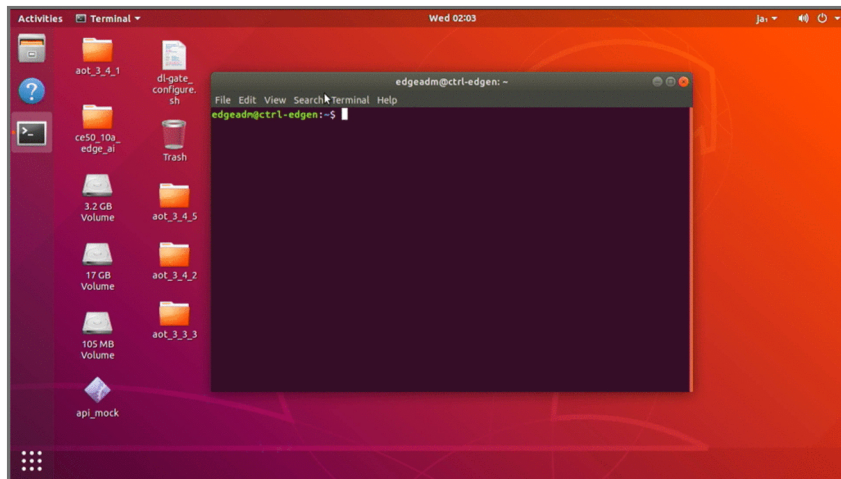
```
$ sudo systemctl start gdm3
```

3. 表示されたログイン画面から、edgeadm ユーザーでログインします。
4. 次に示す手順に従って Terminal を開きます。

1. 左下の  をクリック。
2. [All] をクリック。
3. Terminal をクリック。



Terminal が開きます。



5. 次のコマンドを実行し、Docker 上のプログラムが X サーバに接続することを許可します。

```
$ xhost local:
```

6. 次のコマンドを実行し、AP ランプ操作プロセスを起動します。

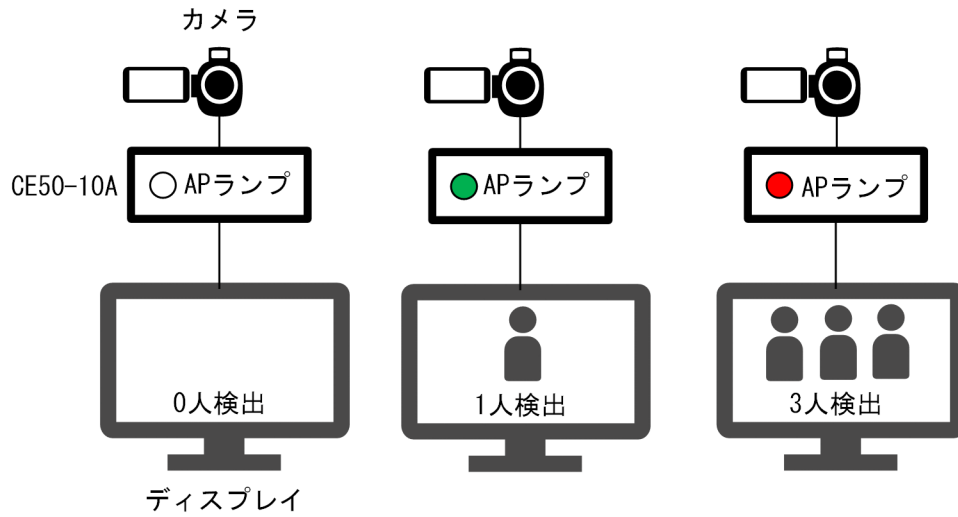
```
$ sudo /hitachi/ctrl_edge_ai/sample/people_count/start_AP
```

7. 次のコマンドを実行し、サンプルプログラムを起動します。

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

8. 実行結果を確認します。

図 2-3 サンプルプログラム実行時のイメージ (USB カメラ)



9. サンプルプログラムを停止する場合、次のコマンドを実行してください。

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

3

AI 映像アプリ機能のセットアップ

AI 映像アプリ機能のセットアップ手順について説明します。

3.1 AI 映像アプリ機能のセットアップ手順

AI 映像アプリ機能を使用するために必要なセットアップの手順について説明します。

3.1.1 Docker 用パーティションの作成/マウント

Docker を利用するために、Docker 用のパーティションを作成したあと、CE50-10A にマウントする必要があります。

(1) Docker 用パーティションの作成

Docker 用パーティションのサイズ指定では、15GB 以上を指定してください。

Docker 用パーティションの作成については、「CE50-10 取扱説明書」の「パーティションを作成する」を参照してください。

(2) 自動マウントするように定義する

CE50-10A を起動したときに、Docker 用のパーティションへ自動的にマウントするためには、マウント定義ファイル (/hitachi/etc/fsconf) に定義が必要です。

定義については、「CE50-10 取扱説明書」の「アプリケーション領域を自動マウントするように定義する」を参照してください。

3.1.2 Docker サービスの起動

Docker サービスは、デフォルトでは OS 起動時に停止しています。一時的に起動する場合は、次のコマンドを実行してください。

```
$ sudo systemctl start docker
```

上記コマンドを実行したあとに、OS を再起動した場合は、Docker サービスは停止状態となります。再度起動させるためには、上記コマンドを再度実行してください。

OS 起動時に Docker サービスを自動起動させる場合は、次のコマンドを実行してください。

```
$ sudo systemctl enable docker
```

上記コマンドを実行したあとは、OS を再起動した場合でも、Docker サービスは自動起動します。

3.1.3 Docker イメージの組み込み

AI 映像アプリ機能を搭載した Docker イメージを組み込むため、次のコマンドを実行してください。コマンドを実行するときは、通常モードの状態で行ってください。

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

Docker イメージが Docker 用のパーティションに組み込まれたあとは、Docker イメージを削除できます。ただし、Docker 用のパーティションの破損など障害が発生した場合に、再度 Docker イメージが必要となることがあります。そのため、Docker イメージの削除前には USB メモリなどの外部媒体に Docker イメージをバックアップしておくことを推奨します。

3.2 パーティションを表示する

ここでは「3.1.1(1) Docker用パーティションの作成」で作成したパーティションを表示して確認します。表示手順が2つあるため、それぞれの手順について説明します。

[表示手順 1]

1. `gdisk` コマンドを `/dev/sda` を引数にして起動します。

```
$ 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. パーティションの表示コマンド `p` を入力します。

```
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
```

注

下線部は、パーティション情報です。

3. `q` を入力して終了します。

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

[表示手順 2]

1. `gdisk` コマンドを `-l` オプション付きで `/dev/sda` を引数にして起動します。

`gdisk` のプロンプトは表示されず、パーティションを表示してコマンドが終了します。

```
$ 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
```

注

下線部は、パーティション情報です。

4

AI 映像アプリ機能の設計

AI 映像アプリ機能をユーザーが独自に設定する方法、AI 映像アプリ機能の起動/停止方法などについて説明します。

4.1 Compose ファイルを利用した使用方法

Compose ファイルの各項目を自由に設定することで、AI 映像アプリ機能を様々な場面で使用することができます。Compose ファイルは、YAML フォーマットです。YAML ファイルでは、データの階層構造はインデントを使用して表現されます。

Compose ファイルの記載例

行の右にある#付の数字は、参照先となる「表 4-1 Compose ファイルの設定内容の説明」の項番を示します。

```

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
```

Compose ファイルの設定内容の説明を、次の表に示します。

表 4-1 Compose ファイルの設定内容の説明

項番	設定内容	説明
1	version:	docker-compose コマンドで使用する Compose ファイルのバージョンを指定します。 製品に搭載している docker-compose のバージョンは 3 です。
2	services:	サービス定義であることを表します。
3	data_input: data_manager: openvino_prod:	サービス名を任意の名前で定義します。 左記の例では、3つのサービスを定義しています。 <ul style="list-style-type: none"> データ入力機能：data_input データ管理機能：data_manager 推論実行機能：openvino_prod
4	ipc:	共有メモリを使用する際、同じメモリ空間を共有する先を指定します。X 転送を行う際に使用します。 AI 映像アプリ機能は、host (ホストと共有) だけサポートします。
5	image: <イメージ名>:<タグ名>	イメージ名、タグ名を指定します。
6	network_mode: host	仮想ネットワークモードを指定します。 左記の例では、ホスト側と共有します。
7	restart: on-failure[:max_retries]	再起動の動作を指定します。 on-failure では、コンテナが異常終了 (終了ステータスが 0 以外) のとき、最大で max_retries 回数の再起動を行います。 max_retries に指定がない場合は、回数制限なく再起動を行います。
8	environment:※	環境変数を指定します。 前述の「Compose ファイルの記載例」にある例では、コンテナ上の GUI 表示をホスト側に転送 (X 転送) するため、ホスト側の環境変数 DISPLAY をコンテナへ引き渡します。
9	volumes:※	ボリュームを定義、またはマウントするボリュームを指定します。 "ボリューム名:" とすると、定義した名称のボリュームを使用します。 <ul style="list-style-type: none"> ホスト側パス:コンテナ側パス[:アクセスモード] を定義することで、ホストとコンテナとのファイル共有が可能となります。 ボリューム名:コンテナ側パス[:アクセスモード] を定義することで、コンテナ間での名前付きボリュームを介したファイル共有が可能となります。 例では、X 転送するため、X11-unix ディレクトリの共有、およびファイル共有のための share ディレクトリを設定しています。
10	depends_on:	コンテナの依存関係を指定します。 例では、data_manager > openvino_prod > data_input の順で起動します。

項番	設定内容	説明
11	logging:	ログに関する指定をします。 「9 AI 映像アプリ機能向け RAS 機能」を参照してください。
12	devices:	ホストのデバイスをコンテナのデバイスに割り当てます。 <ul style="list-style-type: none"> ホスト側デバイス:コンテナ側デバイス 例では、data_input コンテナは、GPU (Graphical Processing Unit) と USB カメラを、openvino_prod コンテナは、GPU を使用します。
13	volumes:	ボリューム定義であることを表します。
14	input_volume: storage_voume:	ボリュームの名称を定義します。 左記の例では、次の 2 つのボリュームを作成しています。 <ul style="list-style-type: none"> input_volume:データ入力機能→データ管理用のボリューム storage_volume:データ管理機能→推論実行用のボリューム
15	driver:	ボリュームドライバを指定します。 AI 映像アプリ機能は local (ローカルボリューム) だけサポートしています。
16	driver_opts:	ドライバオプションを指定します。 次の設定を追加することで、ボリュームのデータ保存先を主メモリに設定します。 type: tmpfs device: tmpfs o: "size=512m" size=512m は、格納可能なデータの上限を 512MB と設定しています。 o:の行は省略可能ですが、主メモリにデータを保存し過ぎることを防ぐため、サイズ上限を設定することを推奨します。

※注

コンテナ上の GUI 表示をホスト側に転送し、ホスト側で表示させる場合は、Compose ファイルの対応するコンテナに次に示す記述を追加してください。

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


4.2 OS 起動時に Docker コンテナを自動起動する手順

OS 起動時に Docker コンテナを自動起動する手順を説明します。

4.2.1 Docker サービスを自動起動する手順

[3.1.1 Docker 用パーティションの作成/マウント]を参照し、fsmount サービスを使って Docker 用パーティションを自動マウントする設定を実施してください。

[3.1.2 Docker サービスの起動]を参照し、Docker サービスを自動起動させるコマンドを実行してください。

4.2.2 Docker コンテナを自動起動する手順

OS 起動時に Docker コンテナを自動起動させる場合は、systemd を使用するため、次の手順を実施してください。

1. systemd スクリプトファイルを作成します。

次に例を示します。各項目の詳細は「CE50-10 取扱説明書」の「アプリケーションプログラムの起動と停止を設定する」を参照してください。

ファイル名は、実行するアプリケーションに合わせて設定してください（例: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
```

Compose ファイル（上記の例では、/home/edgeadm/work/docker-compose.yaml）を、docker-compose コマンドの-f オプションで絶対パスで指定してください。

2. 自動起動を有効に設定します。

作成した systemd スクリプトファイルを/lib/systemd/system に格納し、次のコマンドを実行してください。

```
$ sudo systemctl enable <systemdスクリプトファイル名(.service)は省略可>
```

実行例

```
$ sudo systemctl enable people_count
```

自動起動を無効に設定するには、次のコマンドを実行してください。

```
$ sudo systemctl disable <systemdスクリプトファイル名(.service)は省略可>
```

4.3 AI 映像アプリ機能の起動/停止方法

AI 映像アプリ機能を搭載した Docker コンテナの起動/停止方法について説明します。

Docker コンテナの起動/停止は、docker-compose コマンドによって実行します。

docker-compose コマンドを実行する場合、Compose ファイルが必要です。

デフォルトではカレントの Compose ファイル (docker-compose.yaml、または docker-compose.yml) を読み込み、Compose ファイルに記述されている内容に従って Docker コンテナを起動/停止します。Compose ファイルについては、「6.4.2 Compose ファイルの設定内容」を参照してください。

4.3.1 Docker コンテナの起動方法

docker-compose コマンドの up や start を使って Docker コンテナを起動します。

up は、Docker コンテナを生成および起動します。start は、生成済みの Docker コンテナを起動します。

Compose ファイル内に記述した全 Docker コンテナを生成および起動する場合

すべての Docker コンテナをデーモン化して起動する場合、-d オプションを指定してください。

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

Compose ファイル内に記述した Docker コンテナ (サービス名) を指定して Docker コンテナを生成および起動する場合

Docker コンテナをデーモン化して起動する場合、-d オプションを指定してください。

```
$ sudo docker-compose up [-d] <サービス名>
```

起動中の Docker コンテナに接続し、Docker コンテナ上でコマンドを実行する場合

```
$ sudo docker-compose exec <サービス名> <コマンド>
```

停止中のすべての Docker コンテナを起動する場合

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

OS を reboot したときは、起動しているすべての Docker コンテナが停止状態となります。Docker コンテナの自動起動が無効の場合、reboot したあとは start によって再開してください。

4.3.2 Docker コンテナの停止方法

docker-compose コマンドの down や stop を使って Docker コンテナを停止します。

down は、すべての Docker コンテナを停止および削除します。stop は、特定の Docker コンテナ、またはすべての Docker コンテナを停止します。

Compose ファイル内に記述したすべての Docker コンテナを停止し削除する場合

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

Compose ファイル内に記述した Docker コンテナ（サービス名）を指定して停止する場合

```
$ sudo docker-compose stop <サービス名>
```

Compose ファイル内に記述したすべての Docker コンテナを停止する場合

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

stop によって停止した Docker コンテナは、OS を reboot したあとも停止状態のままとなります。

4.3.3 Docker コンテナの状態確認方法

Docker コンテナの状態は、ps オプションで確認できます。

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

up 実行後の例

Name	Command	State	Ports
edgeadm_test_1	/bin/bash	Up	

State が、Up になります。

down 実行後の例

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

ヘッダ表示だけとなります。

up→stop 実行後の例

Name	Command	State	Ports
edgeadm_test_1	/bin/bash	Exit	0

State が、Exit になります。

stop→start または up 実行後の例

Name	Command	State	Ports
edgeadm_test_1	/bin/bash	Up	

State が、Up になります。

4.4 システム設計上の留意点

4.4.1 提供コンテナイメージ

AI 映像アプリ機能として提供する各機能の Docker コンテナイメージを、次の表に示します。

表 4-2 提供コンテナイメージ

項番	機能名	リポジトリ名	イメージ名	タグ	再配布可否※
1	データ入力機能	ctrl_edge_ai	data_input	1.0	可
2	データ管理機能		data_manager	1.0	可
3	推論実行機能		openvino_prod	1.0	可
4	推論開発機能		openvino_devel	1.0	不可

注※

CE50-10A 内の OpenVINO は、CE50-10A の初回ログイン時に表示される EULA (End User License Agreement) に基づいてご利用ください。

OpenVINO のライセンスでは、OpenVINO の一部コンポーネントを除き再配布が禁止されています。例えば、開発したソフトウェアを CE50-10A に搭載し、顧客に納入することは再配布に当たります。再配布する際は、次のコマンドを実行して再配布不可のイメージを削除してください。

```
$ 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 Docker コンテナの自動再起動

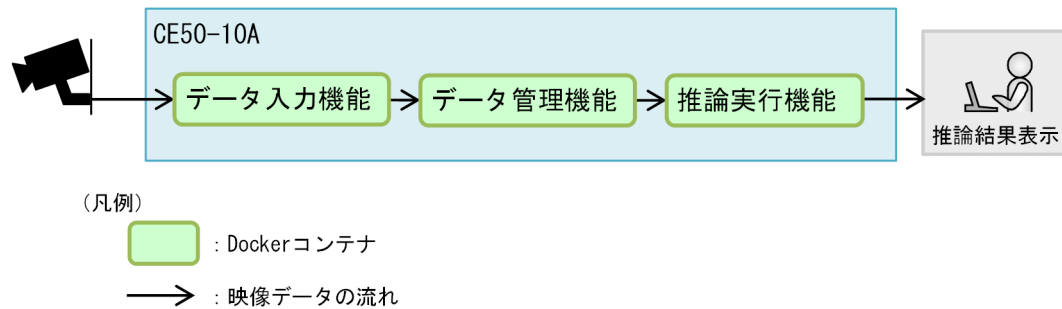
CE50-10A の運用時、各機能（データ入力機能、データ管理機能、推論実行機能）で異常が発生した場合、基本的に各機能のコンテナは、エラーが発生したときに終了します。ただし、Compose ファイルに restart オプションを設定することによって、エラーが発生して終了したコンテナを自動的に再起動させ、各機能を再開することができます。

restart オプションの設定方法は、「4.1 Compose ファイルを利用した使用方法」を参照してください。

ただし、エラー原因がコンテナの再起動で改善しない場合（例えば、カメラが故障し、データ入力機能では映像データが取得できない場合など）、各機能のコンテナは、設定した回数再起動を繰り返して終了します。この場合は、発生したエラーの原因を取り除いてから手動でコンテナを再起動してください。

CE50-10A 運用時のコンテナ構成を、次の図に示します。

図 4-1 CE50-10A 運用時のコンテナ構成



エラーが発生してから回復するまでの各機能の動作と、その間の映像データの状態を説明します。

- データ入力機能コンテナでエラーが発生した場合

カメラから映像が取得できない状態の場合など、データ入力機能コンテナで何らかのエラーが発生したとき、データ入力機能コンテナは終了します。データ管理機能コンテナ、推論実行機能コンテナは、映像データが渡されるまで待機状態となります。再起動によって発生したエラーが回復すれば処理を再開します。ただし、エラー発生時から再開するまでの間の映像データは失われます。
- データ管理機能コンテナでエラーが発生した場合

データ管理機能コンテナで何らかのエラーが発生した場合、データ管理機能コンテナは終了します。データ管理機能コンテナが終了すると、データ入力機能コンテナ、推論実行機能コンテナもデータ管理機能コンテナにアクセスができなくなり、各機能のコンテナは終了します。再起動によって発生したエラーが回復すれば処理を再開します。ただし、エラー発生時から再開するまでの間の映像データは失われます。
- 推論実行機能コンテナでエラーが発生した場合

推論実行機能コンテナで何らかのエラーが発生した場合、推論実行機能コンテナは終了します。データ管理機能コンテナは映像データを蓄積しているため、推論実行機能コンテナが再起動したあと、データ管理機能コンテナに登録されたファイルの古い順に推論を再開します。

4.4.3 取得フレーム数のチューニング

推論実行機能が、映像データを推論できる FPS (Frame Per Second) の設定値よりも、処理するデータ入力機能の FPS の設定値の方が大きい場合、推論実行機能がすべてのフレームを処理できなくなり、データ管理機能の容量がいっぱいになってしまうことが考えられます。

データ管理機能は、設定された上限値を超えた映像データを古い順に削除します。そのため、データ管理機能でエラーが発生することはありません。しかし、入力された映像データが CE50-10A で処理されず破棄されることは、効率的ではありません。そのため、運用前 (開発テスト段階など) に推論実行機能の性能を確認し、データ入力機能がデータ管理機能に登録するときの FPS (「5.3.1 データ入力機能の設定内容」に示す output > video > fps の値) の設定値を、推論実行機能の FPS の設定値よりも小さく設定する (チューニングする) ことを推奨します。

4.4.4 ファイアウォールの設定

AI 映像アプリ機能では、次の表に示す通信を行う必要があります。

表 4-3 AI 映像アプリ機能に必要な通信

項番	機能名	必要な通信	通信の向き (送信/受信)	備考
1	データ入力機能 (IP カメラ使用時)	IP カメラの RTSP ポート※ へのアクセス	送信	USB カメラを用いる場合 不要です。
2	データ管理機能	API 用ポート (TCP/ 13579)	送信	—
3	サンプルアプリケーション	API 用ポート (TCP/5000)	送信	サンプルアプリケーション 使用時以外は不要で す。
4	推論開発機能	Jupyter Notebook 用の ポート (8888/tcp)	受信	—

(凡例)

— : 該当しない。

注※

IANA (Internet Assigned Numbers Authority) の規定では、554/tcp、554/udp を用いると定められています。

しかし、市販の IP カメラはポート番号への攻撃軽減策として、異なるポート番号を使用している場合があります。

ファイアウォールを設定する場合の考慮点を次に示します。

- IP カメラを用いる場合は、IP カメラの RTSP ポート宛の送信を許可する必要があります (デフォルトでは許可していません)。
IP カメラの RTSP ポートは、IP カメラの取扱説明書などで確認してください (変更可能な場合もあります)。
- データ管理機能の API、サンプルアプリケーションの API は、自ホスト宛送受信がデフォルトで有効なためユーザー側で許可設定をする必要はありません。

表 4-4 ファイアウォール機能一覧

項番	ルール名	ルールの概要	デフォルト設定
1	ポート受信許可	指定したポートの受信を許可し、それ以外はすべてブロックします。 外部からアクセスが必要なポートだけオープンにすることでセキュリティが向上します。	無効
2	ポート送信許可	指定したポート宛の送信を許可し、それ以外はブロックします。	次に示すポート宛の送信を許可 <ul style="list-style-type: none"> • SSH (22/tcp) • DNS (53/udp) • HTTP (80/tcp) • NTP (123/tcp) • HTTPS (443/tcp)

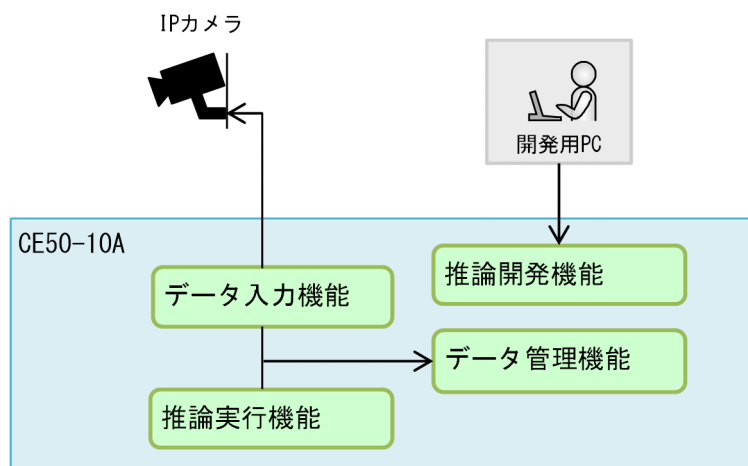
項番	ルール名	ルールの概要	デフォルト設定
3	ポート受信許可 (制限付)	指定したポートの受信を許可しますが、一定時間の接続可能数を制限します。 次の攻撃に対する防御として有効です。 <ul style="list-style-type: none"> • DoS (Denial of Service attacks) 攻撃 • DDoS (Distributed Denial of Service attacks) 攻撃 • ブルートフォース攻撃 	1 分間に 10 回まで SSH (22/tcp) の受信を許可
4	自ホスト宛送受信	自ホスト (127.0.0.1) が宛先となっている通信は、ポートを問わず許可します。	有効

4.4.5 セキュリティリスクとその対策

推論処理プログラムの処理内容や運用環境に応じて、新たなセキュリティリスクが発生して、求められる対策が変化する場合があります。そのような場合は、ユーザー側で十分な検討が必要となります。

CE50-10A のネットワーク通信のアクセス経路を、次の図に示します。

図 4-2 CE50-10A のネットワーク通信のアクセス経路



(凡例)

→ : アクセス経路

CE50-10A 使用時に存在するセキュリティリスクと、CE50-10A 上で実施している対策を、次の表に示します。

表 4-5 構成要素ごとのセキュリティリスクとその対策

項番	構成要素	データの種類	セキュリティリスク			セキュリティ対策・考慮点
			盗み見されたら	改ざんされたら	消失したら	
1	推論開発機能	Jupyter Notebook を使用するための通信	動画/画像などが漏えいします。	プログラムが正常に動作しなくなります。	推論開発機能が使用不能になります。	<ul style="list-style-type: none"> Jupyter Notebook ログイン時にワンタイムパスワードによる認証があります。 通信路での暗号化はサポートしていません。自社ネットワーク内など盗聴や改ざんのリスクの低いネットワーク上で使用することを推奨します。
2	IP カメラ	動画/画像	動画/画像などが漏えいします。	推論実行機能が誤った推論結果を出力するおそれがあります。	動画/画像の取得ができなくなります。	<ul style="list-style-type: none"> IP カメラ側で対策が必要です (CE50-10A 側の対策はありません)。 ID やパスワードによる認証が可能な IP カメラを使用ください。データ入力機能は ID やパスワードによる認証に対応しています。 通信路での暗号化はサポートしていません。自社ネットワーク内など盗聴や改ざんのリスクの低いネットワーク上で使用することを推奨します。
3	データ管理機能	CE50-10A が生成するデータ	動画/画像などが漏えいします。	推論実行機能が誤った推論結果を出力するおそれがあります。	取得した動画/画像が失われます。	<ul style="list-style-type: none"> データ管理機能は認証機能がありません。 データ管理機能へのアクセスはファイアウォールでブロックします。

5

データ入力機能

データ入力機能の概要、カメラの接続方法、設定方法などについて説明します。

5.1 データ入力機能の概要

データ入力機能には、次の機能があります。

- IP カメラ、USB カメラからの映像データを取得し、その映像データを映像ファイルとして出力します。出力した映像ファイルをデータ管理機能に渡します。
- 接続したカメラの映像をディスプレイにプレビューして、動作を確認することができます。

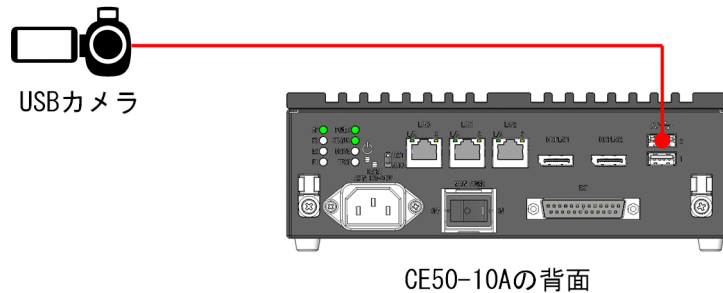
5.2 カメラ接続方法

CE50-10A にカメラを接続する方法を示します。

5.2.1 USB カメラを接続する場合

USB カメラを使用する場合、CE50-10A の USB ポートにケーブルを接続してください。

図 5-1 USB カメラの接続方法



5.2.2 IP カメラを接続する場合

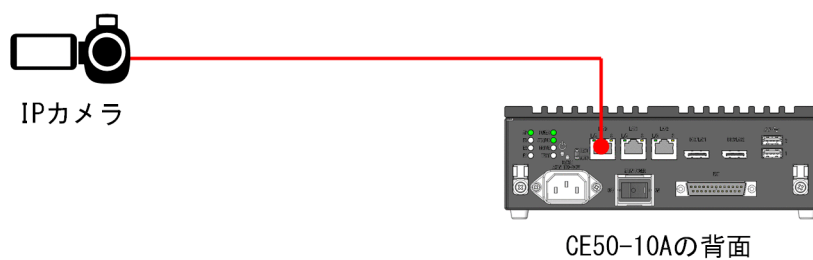
IP カメラを接続する場合、次の 2 つの接続方法があります。

- IP カメラの LAN ポートと、CE50-10A の LAN ポートを直接 LAN ケーブルで接続する方法
- IP カメラと CE50-10A をスイッチングハブやルータを介して直接通信できるネットワークに接続する方法

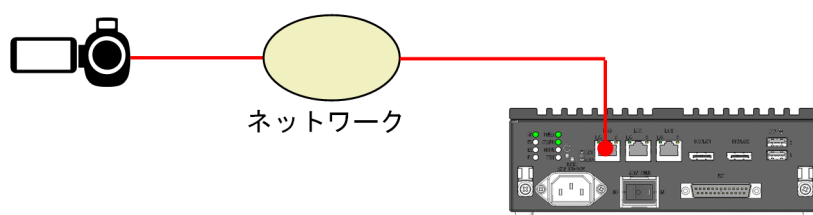
IP カメラに対しては、IP アドレスを別途付与してください。

図 5-2 IP カメラの接続方法

●直接LANポートに接続する



●ネットワークを介して接続する



5.3 データ入力機能の設定方法

映像データを撮影するカメラの設定内容、出力した映像ファイルに関する設定内容、カメラの映像をプレビューする設定内容について説明します。

5.3.1 データ入力機能の設定内容

データ入力機能の設定内容は、json 形式で記載します。データ入力機能コンテナ上では、次のパスで参照可能なように Docker に設定してください。

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

コンテナ側の設定は、「5.3.2 コンテナの設定内容」を参照してください。

設定する内容を、以降の表に示します。

設定ファイルに項目を重複して設定した場合、下部に記載した項目が設定値として反映されます。

表 5-1 データ入力機能での設定内容

項番	項目	設定要否	設定内容
1	camera	必須	カメラに関する設定をします。 詳細項目については、「表 5-2 camera の詳細項目」を参照してください。
2	output	任意	取得した映像データをデータ管理機能に格納する設定をします。省略時は出力しません。 詳細項目については、「表 5-6 output の詳細項目」を参照してください。
3	preview	任意	カメラの映像データをプレビューする設定をします。 省略時はプレビューしません。 詳細項目については、「表 5-8 preview の詳細項目」を参照してください。

表 5-2 camera の詳細項目

項番	項目	設定要否	設定内容
1	name [*]	必須	カメラ ID です。カメラを一意に識別する ID を指定します。
2	type	必須	使用するカメラの種類を次から選択して指定します。 "usb" : USB カメラ "ip" : IP カメラ
3	connect	必須	カメラの接続情報を指定します。 カメラの接続情報の指定については、「表 5-3 connect の詳細項目 USB カメラ (type="usb") の場合」「表 5-4 connect の詳細項目 IP カメラ (type="ip") の場合」を参照してください。
4	video	必須	カメラから取得する動画の情報を指定します。 詳細項目については、「表 5-5 camera-video の詳細項目」を参照してください。

注※

カメラ ID に指定できる文字種は、英数字と_（アンダーバー）だけです。指定できる文字数は、31 文字以内です。

表 5-3 connect の詳細項目 USB カメラ (type="usb") の場合

項番	項目	設定要否	設定内容
1	device	必須	USB カメラのデバイスファイルへのパスを指定します。

表 5-4 connect の詳細項目 IP カメラ (type="ip") の場合

項番	項目	設定要否	設定内容
1	rtsp_url	必須	IP カメラの RTSP 用 URL を記載します。ID/パスワード認証が必要な場合は URL に含めます。 詳細は「IP カメラから映像データを取得する場合の設定例」を参照してください。
2	latency	必須	IP カメラから受信した映像データをバッファする時間を指定します（単位：ミリ秒）。

表 5-5 camera-video の詳細項目

項番	項目	設定要否	設定内容
1	width	必須	フレーム解像度・横幅を指定します（単位：ピクセル）※。
2	height	必須	フレーム解像度・高さを指定します（単位：ピクセル）※。
3	fps	必須	取得する映像データの 1 秒当たりのフレーム数を指定します※。
4	codec	必須	取得する動画のコーデックを次から選択して指定します。 <ul style="list-style-type: none"> • "YUYV"：非圧縮 • "MJPEG"：Motion-JPEG • "H264"：H.264

注※

- 設定可能な解像度/フレームレートの範囲は、次に示すとおりです。
 解像度：640×480～3840×2160
 フレームレート：1～30
- カメラが対応していない値を設定するとエラーが発生したり、設定値が無視され異なる設定値が指定されたりします。
 設定可能な値については、利用するカメラの取扱説明書などを参照してください。

表 5-6 output の詳細項目

項番	項目	設定要否	設定内容
1	type	必須	出力方法を次から選択して指定します。 <ul style="list-style-type: none"> • "frame"：動画を 1 フレームごとに画像ファイルとして出力します。

項番	項目	設定要否	設定内容
1	type	必須	<ul style="list-style-type: none"> "video": 一定時間ごとの動画ファイルとして出力します。
2	output_dir	必須	出力先ディレクトリパスを指定します。データ管理機能の入力用ボリュームをマウントしている個所を指定します。
3	duration [*]	type="video"時必須	1 個の動画ファイルに記録する動画の時間を指定します (単位: 秒)。
4	api_url	必須	データ管理機能の API の URL を指定します。 例) http://127.0.0.1:13579
5	video	必須	出力する動画の情報を指定します。 詳細項目については、「表 5-7 output-video の詳細項目」を参照してください。

注※

30 秒未満を設定した場合、動画記録時間が設定値と異なる場合があるため、30 秒以上の設定を推奨します。

表 5-7 output-video の詳細項目

項番	項目	設定要否	設定内容
1	width	必須	フレーム解像度・横幅を指定します (単位: ピクセル)。*
2	height	必須	フレーム解像度・高さを指定します (単位: ピクセル)。*
3	fps	必須	取得する映像データの 1 秒当たりのフレーム数を指定します。*
4	codec	必須	ファイルのコーデックを指定します。 <ul style="list-style-type: none"> "type"="frame"のときは"JPEG"だけ設定できます。 "type"="video"のときは"H264"だけ設定できます。

注※

- 設定可能な解像度/フレームレートの範囲は、次に示すとおりです。
解像度: 1×1~3840×2160
フレームレート: 1~30
- 解像度は、設定した値に合わせて拡大/縮小します。
FPS の設定値が入力よりも多い場合は同じフレームを繰り返し出力し、入力より少ない場合は 1 フレーム飛ばしなどフレームを間引いて合わせます。

表 5-8 preview の詳細項目

項番	項目	設定要否	設定内容
1	type	必須	プレビューする方式を指定します。 <ul style="list-style-type: none"> "x11": X 転送によってプレビューします。

USB カメラから映像データを取得する場合の設定例

次の設定例の内容は、次のパスにサンプルファイルとして配置しています。

```
/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"
}
```

IP カメラから映像データを取得する場合の設定例

次の設定例の内容は、次のパスにサンプルファイルとして配置しています。

```
/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 コンテナの設定内容

データ入力機能のコンテナに必要な設定について説明します。コンテナの構築に関する説明は「4.1 Compose ファイルを利用した使用方法」を参照してください。

GPU の使用設定

データ入力機能コンテナは、取得した動画のデコード/エンコードを行うため、GPU を使用します。

データ入力機能コンテナ上で GPU 処理を使用するため、次の設定を行ってください。

設定内容

Compose ファイルの設定で devices: の項目に /dev/dri を追加してください。

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

USB カメラの使用設定

USB カメラ使用時は、USB カメラのデバイスファイルをデータ入力機能コンテナからアクセスできるように設定してください。IP カメラを使用する場合は、この設定は不要です。

接続された USB カメラに対応するデバイスファイルは、/dev 以下に video0, video1, . . . という名前でホスト上に生成されます。

ls コマンドなどで USB カメラのデバイスファイルのファイル名を確認して、データ入力機能コンテナ上で利用できるように設定してください。

設定例

Compose ファイルの設定でデータ入力機能コンテナの設定に確認した USB カメラのパスを追加してください。

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

設定ファイルのホスト/コンテナの共有

[5.3.1 データ入力機能の設定内容] で示した設定ファイルを、データ入力機能コンテナ上から参照できるようにします。

ホスト上に配置した設定ファイル (data_input.json) のパスを、データ入力機能コンテナ上の /config/data_input.json にマウントします。

```
volumes:
  - "<ホスト上のdata_input.jsonのパス>:/config/data_input.json"
```

設定例

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

5.4 動作確認手順

データ入力機能とカメラが接続できているか確認する手順を説明します。

1. CE50-10A にアクセスします。

CE50-10A にディスプレイ、キーボード、マウスを直接接続してアクセスしてください。

2. 「CE50-10 取扱説明書」の「デスクトップの起動」を参照し、デスクトップを起動します。
3. デスクトップが起動したら、sudo 権限を持つユーザー（例：edgeadm）でログインします。

4. 設定ファイル（data_input.json）を作成して、OS 上の任意のパスに配置します。

設定ファイルの記載方法は、「5.3.1 データ入力機能の設定内容」を参照してください。ここでは"/home/edgeadm/preview"に配置した場合の例を説明します。

USB カメラの映像データを取得する場合の"data_input.json"の記載例

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

5. 設定ファイル（data_input.json）を作成して、OS 上の任意のパスに配置します。

設定ファイルの記載方法は、「5.3.1 データ入力機能の設定内容」を参照してください。ここでは"/home/edgeadm/preview"に配置した場合の例を説明します。

6. Compose ファイルを作成して、OS 上の任意のパスに配置します。

Compose ファイルの記載方法は、「4.1 Compose ファイルを利用した使用方法」および「5.3.2 コンテナの設定内容」を参照してください。

ここでは、/home/edgeadm/preview に配置した場合の例を説明します。

USB カメラの映像データをプレビューする場合の記載例

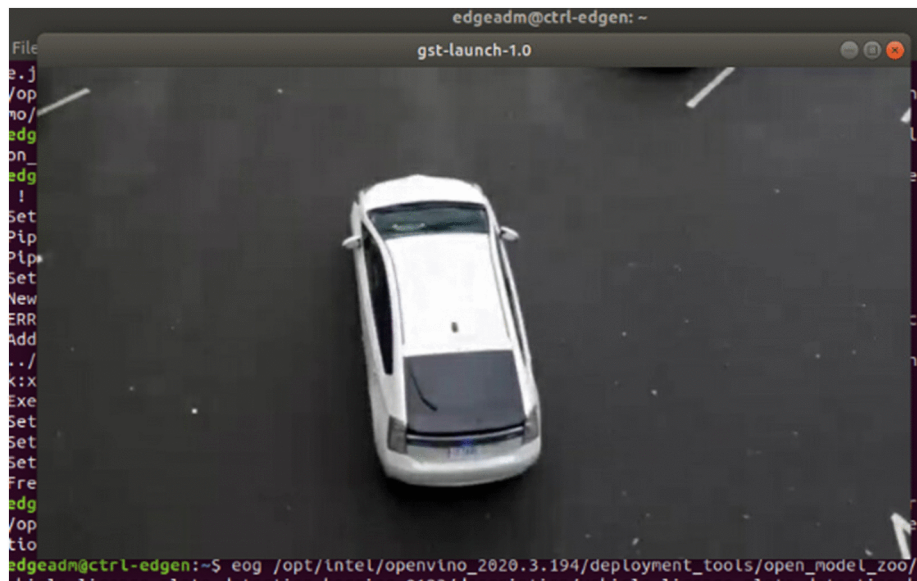
```
version: '3'
services:
  preview:
    ipc: host
    image: ctrl_edge_ai/data_input:1.0
    network_mode: "host"
    devices:
      - "/dev/dri:/dev/dri"
      - "/dev/video0:/dev/video0"
    volumes:
      - "./data_input.json:/config/data_input.json"
      - "/tmp/.X11-unix:/tmp/.X11-unix"
    environment: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

7. 次のコマンドを実行します。

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

data_input.json の内容に従って、カメラの動画が画面に表示されます。

図 5-3 表示例



サンプル動画の出典

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

8. 確認を終了するときは、ウィンドウの右上にある [×] ボタンをクリックします。

6

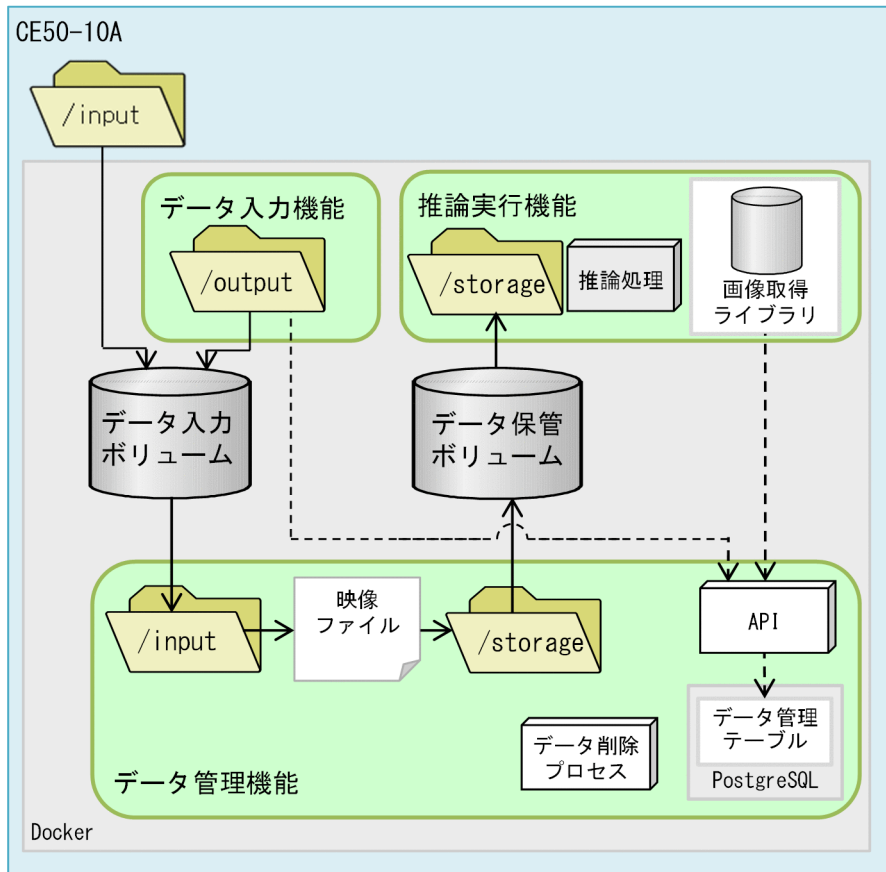
データ管理機能

データ管理機能の概要、API およびライブラリの仕様、設定方法について説明します。

6.1 データ管理機能の概要

データ管理機能は、データ入力機能と推論実行機能の間で映像ファイルを一時的に保持するキューのような役割をします。映像ファイルや管理情報の流れを、次の図に示します。

図 6-1 データ管理機能での映像ファイルや管理情報の流れ



(凡例)

- : 各機能のコンテナ
- : ディレクトリ
- : 映像ファイルの流れ
- - -> : 管理情報の流れ

データ管理機能が持つ機能を、次に示します。

ファイル登録

推論対象の映像ファイルを受け付け、データ管理テーブルに登録します。

この機能は、データ入力機能によって自動的に行われるため、データ入力機能を使用する場合はユーザー側の操作は不要です。画像検査装置など、カメラ以外の装置からファイルとして連携される画像や動画をデータ管理機能コンテナにファイルとして登録する場合は、下記の API 操作を実行するプログラムを作成する必要があります。API の使用方法は、「6.2 データ管理機能の API 仕様」を参照してください。

データ入力機能コンテナとデータ管理機能コンテナ間では、ファイルを共有するためボリュームを用います。

データ管理機能にファイルを登録する手順を、次に示します。

1. データ管理機能コンテナのディレクトリ上にファイルを配置します。
2. 配置したファイルパスを含むリクエストをAPI (/v1/files) にPOSTし、登録します。
APIは、ファイル名をデータ管理テーブルに登録します。このとき当該ファイルは"未推論"の状態として登録されます。
APIは、/inputディレクトリに配置されたファイルを/storageディレクトリへ移動します。

ファイル取得/状態変更

推論対象の映像ファイルを識別して、取り出します。推論実行機能、推論開発機能に搭載されている画像取得ライブラリは、自動的に次に示す操作を行うためユーザーの操作は不要です。画像取得ライブラリを使用せずに画像を取得する場合のファイル取得の手順を、次に示します。

1. API (/v1/files) をGETし、"未推論"状態のファイル取得をリクエストします。
APIは、データ管理テーブルから"未推論"のファイル名をオプションに従い1個応答します。
2. APIで、ファイル状態変更をリクエストして、1.で取得したファイルの状態を"推論中"に変更します。
3. 取得したファイルパスのファイルの推論を実行します。
4. 推論が完了したら、API (/v1/files/<file_id>) でファイル状態変更をリクエストし、1.で取得したファイルの状態を"推論済み"に変更します。

推論実行機能、推論開発機能に搭載されている画像取得ライブラリを使用すると、上記手順の1.~4.を意識せずに使用することができます。

ファイル削除

データ管理機能は、データ管理テーブルの情報を参照し、登録されたファイルを自動的に削除します。"未推論"と"推論済み"の状態のファイル容量の上限を、データ管理機能の設定ファイルで設定します。上限を超えた場合、古いファイルから順に削除されます。

6.2 データ管理機能の API 仕様

データ管理機能でファイル登録やファイル取得などを実行する API は、HTTP による REST API として提供します。API は、データ管理機能コンテナの 13579 番ポートを使用します。

アクセス URL 例

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

以降の説明では、「<プロトコル>://<ホスト名>:13579」をベース URL と表記します。

6.2.1 ファイルをデータ管理機能に登録する (v1FilesPost)

ファイルパスに指定されたファイルをデータベースに登録します。

リクエストライン

```
POST <ベースURL>/v1/files
```

リクエストメッセージ

ボディ

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

項番	属性	データ型	説明
1	camera_id	String	(任意) カメラ ID を指定します。 例: camera_1
2	input_path	String	(任意) 登録先のパスを指定します。 例: /path/to/file
3	status	String	(任意) ファイルの状態を指定します。 例: added

レスポンスメッセージ

ボディ

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

項番	属性	データ型	説明
1	file_id	String	割り当てられたファイル ID です。 例: cf16fe52-3365-3a1f-8572-288d8d2aaa46

ステータスコード

項番	ステータスコード	説明
1	200	登録に成功しました。
2	400	不正なリクエストです。
3	503	サーバ側でエラーが発生しました。

6.2.2 ファイルの状態を更新する (v1FilesFileIdPut)

登録されているファイルの状態を変更します。

リクエストライン

```
PUT <ベースURL>/v1/files/<file_id>
```

項番	属性	データ型	説明
1	file_id	String	(必須) ファイル ID を指定します。 例: cf16fe52-3365-3a1f-8572-288d8d2aaa46

リクエストメッセージ

ボディ

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

項番	属性	データ型	説明
1	status	String	(必須) ファイルの状態です。 例: completed

レスポンスメッセージ

ステータスコード

項番	ステータスコード	説明
1	200	更新に成功しました。
2	400	不正なリクエストです。
3	404	<file_id>が見つかりません。
4	503	サーバ側でエラーが発生しました。

6.2.3 ファイルを取得する (v1FilesGet)

データ管理機能が保持しているファイルを取得します。

リクエストライン

GET <ベースURL>/v1/files

リクエストメッセージ

クエリパラメータ

項番	属性	データ型	説明
1	num	integer	(任意) 取得するファイルの数を指定します。省略した場合、1 になります。
2	camera_id	string	(任意) カメラ ID を指定します。省略した場合、すべてのカメラ ID のファイルが対象となります。複数のカメラ ID を指定する場合は、"?camera_id=camera_1&camera_id=camera_2..."の形式で指定してください。
3	order	string	(任意) ファイルの取得順を指定します。 LIFO: 新しいファイルの順に取得します (デフォルト)。 FIFO: 古いファイルの順に取得します。

レスポンスメッセージ

ボディ

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

項番	属性	データ型	説明
1	file_id	String	ファイル ID です。
2	path	String	登録先のパスです。
3	camera_id	String	カメラ ID です。

ステータスコード

項番	ステータスコード	説明
1	200	取得に成功しました。
2	400	不正なリクエストです。
3	503	サーバ側でエラーが発生しました。

6.3 データ管理機能で利用できるライブラリ仕様

データ管理機能を Python プログラムから利用する画像取得ライブラリの使用方法について説明します。

画像取得ライブラリは、Dataman という名称のモジュールとして用意しています。

6.3.1 ライブラリの利用形態

Dataman は、推論実行機能、および推論開発機能で利用できます。

推論実行機能の推論処理プログラム、推論開発機能の Jupyter Notebook で、データ管理機能からフレームを取得します。

設定項目でデバッグを有効にすると、デバッグ動作となります。このとき、データ管理機能にはアクセスせず、指定されたデバッグ用の画像/動画ファイルからフレームを取得します。

6.3.2 ライブラリを利用するための設定内容

Dataman を使用するには、設定ファイルを初期化時に読み込み動作を設定する必要があります。

設定ファイルのサンプルへのパス

```
/hitachi/crtl_edge_ai/sample/dataman/dataman.json
```

表 6-1 設定項目

項番	設定項目	内容
1	api_url	データ管理機能の API の URL を指定します。
2	frame_order	画像の取得順を設定します。 <ul style="list-style-type: none"> "LIFO": 新しい画像から順に取得します (初期値)。 "FIFO": 古い画像から順に取得します。
3	target_camera	取得対象のカメラ ID を指定します。 設定例 ["camera_1","camera_2"] すべてのカメラ ID の画像を取得対象とする場合は、[]と指定してください。 初期値は[]です。
4	debug	推論実行機能をデバッグするときに指定します。 debug が指定されている場合、api_url、frame_order、target_camera の設定はすべて無効になります。 詳細項目については、「表 6-2 debug の詳細項目」を参照してください。

表 6-2 debug の詳細項目

項番	設定項目	内容
1	file_path	動画/画像ファイルを指定します。

設定例 1

```
{
  "api_url": "http://127.0.0.1:13579",
  "frame_order": "LIFO",
```

```
    "target_camera": []
}
```

設定例 2

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

6.3.3 ライブラリをインポートする方法

推論実行機能上の Python プログラム、または推論開発機能の Jupyter Notebook では、ライブラリを次のようにインポートして使用します。Dataman モジュールは、ce50.ai パッケージに含まれています。

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

6.3.4 Dataman クラスメソッド

データ管理機能で利用する Dataman クラスメソッドの仕様について説明します。

(1) class Dataman.Dataman(conf_path)

ベースクラス：object

データ管理機能操作クラス

コンストラクタ

パラメータ：conf_path (str)

設定ファイルのパス

サンプル

```
>>> # Datamanクラスのインスタンスを生成
>>> dm=Dataman("/hitachi/people_count_sample/dataman/dataman.json")
```

(2) get_frame()

推論対象となる画像を 1 フレーム分取得して、cv::Mat 型で返します。OpenCV の VideoCapture クラスの read() と同様に利用できます。

デバッグ動作時は、デバッグ用の画像/動画ファイルのフレームを取得します。ファイルの末尾まで読み取ると、再度 1 フレーム目から読み取ります。画像ファイルの場合は、常に同じ画像を取得します。

戻り値

return：フレーム取得の成否（成功：True、失敗：False）

info：フレーム情報

frame：画像データ

戻り値の型：(bool、dict、cv::Mat)

サンプル

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

(3) set_target_camera(camera_id)

指定したカメラ ID の画像だけ get_frame の取得対象とします。

ただし、デバッグ動作時は実行しても何もありません。

パラメータ：camera_id_list (array)

カメラ ID

サンプル

```
>>> # camera_1の画像だけ取得対象とする。
>>> dm.set_target_camera(["camera_1"])
>>> # すべてのカメラIDの画像を取得対象とする。
>>> dm.set_target_camera([])
```

(4) update_frame_status(info, status)

指定したフレーム ID のステータスを変更します。

ただし、デバッグ動作時は実行しても何もありません。

パラメータ

info (dict)：変更対象のフレームのフレーム情報

status (str)：変更後のステータス

サンプル

```
>>> # 推論が完了したフレームを推論済みと設定する。
>>> dm.update_frame_status(info, Dataman.STATUS_COMPLETED)
```

6.3.5 ファイルステータスの定数

Dataman クラスで定義しているファイルステータスを表す定数を、次の表に示します。

表 6-3 ファイルステータスの定数

項番	定数名	意味
1	STATUS_ADDED	動画のステータスを設定するための定数です。 推論が未着手であることを示します。
2	STATUS_PROCESSING	
3	STATUS_COMPLETED	

6.4 データ管理機能の設定方法

データ管理機能の設定ファイル、Compose ファイルの設定内容について説明します。

6.4.1 データ管理機能の設定ファイル

データ管理機能の設定ファイルは、データ管理機能のコンテナ上の次のパスに配置してください。

設定ファイルがない場合は、デフォルト値が設定されます。

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

表 6-4 設定ファイルの設定項目

項目	設定項目	設定要否	設定内容
1	delete	必須	削除に関する設定です。 詳細項目については、「表 6-5 delete の詳細項目」を参照してください。

表 6-5 delete の詳細項目

項目	設定項目	設定要否	設定内容
1	unfinished_limit	必須	推論済みの状態になっていないファイルの合計サイズの上限值（単位：KB）です。 デフォルト値は、1048576KB です。 データ保管ボリュームの空き容量を超える場合エラーとなります。
2	finished_limit	必須	推論済みの状態になっているファイルの合計サイズの上限值（単位：KB）です。 デフォルト値は、0KB です。 データ保管ボリュームの空き容量を超える場合エラーとなります。

設定例

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

6.4.2 Compose ファイルの設定内容

Compose ファイルの設定で考慮する点を説明します。

(1) ボリュームの作成での考慮点

CE50-10A では、データを共有するため次のコンテナ間にボリュームを作成する必要があります。

- データ入力機能コンテナとデータ管理機能コンテナ
- データ管理機能コンテナと推論実行機能コンテナ

各ボリュームをそれぞれのコンテナ上に同じパスにマウントします。

データを保持するボリュームの作成先は、次の2つから選択できます。

1. SSD (Docker 用パーティション) 上
2. 主メモリ上

通常は、1.に保存するように設定します。

1.と2.のメリットとデメリットを、次の表に示します。

ボリュームの作成先	メリット	デメリット
1.SSD (Docker 用パーティション) 上	データが永続化できます。電源断など異常が発生してもデータを残すことができます。	<ul style="list-style-type: none"> • 書き込み速度が主メモリに比べると遅くなります。 • 大量のデータの書き換えを行うと SSD の寿命が短くなります。
2.主メモリ上	書き込み速度が高速です。	<ul style="list-style-type: none"> • データが永続化しません。異常時やコンテナ停止時にデータが消失します。 • 大量のデータを配置すると主メモリが不足します。

上記の表のメリットとデメリットを考慮して、次のように設定することを推奨します。

データを共有するコンテナ間	ボリュームの作成先
データ入力機能コンテナとデータ管理機能コンテナ	2.主メモリ上
データ管理機能コンテナと推論実行機能コンテナ※	1.SSD (Docker 用パーティション) 上

注※

書き込み量が 50GB/日以上発生する場合は、SSD が製品寿命よりも早く寿命を迎える可能性があります。そのため、システムの性質（要求される可用性、寿命、メンテナンス頻度など）を考慮した上で、主メモリ上にボリュームを構築することも検討してください。

設定例

各機能のコンテナでデータを共有するボリュームを、次のとおり作成します。

- データ入力機能コンテナとデータ管理機能コンテナ：input_volume（主メモリ上）
マウント先のパス：/input
- データ管理機能コンテナと推論実行機能コンテナ：storage_volume（SSD 上）
マウント先のパス：/storage

次の設定例では、関連する個所だけを記載しています。

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

(2) ポートの設定

データ管理機能コンテナの API のポートを他の機能のコンテナからアクセスできるように設定してください。

次の設定例では、関連する箇所だけを記載しています。

設定例

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

7

推論実行機能

推論実行機能の概要、推論処理を組み込む流れ、サンプルプログラムについて説明します。

7.1 推論実行機能の概要

データ入力機能が取り込んだ映像データは、データ管理機能でボリュームに映像ファイルとして登録されます。推論実行機能は、推論対象の映像ファイルをボリュームから取り出し、推論処理を実行します。

推論実行機能は、Python で構築された推論処理を実行する機能です。

ユーザーは、推論処理を実行するために、次の 2 つを準備する必要があります。

- 推論処理の実行プログラム
- 学習済みモデルを変換した中間表現（OpenVINO では IR（Intermediate Representation）と呼ばれる）

推論処理の実行プログラムと学習済みモデルの処理の概要を、次の表に示します。

表 7-1 推論処理の実行プログラムと学習済みモデルの処理の概要

項番	項目	処理の概要
1	推論処理の実行プログラム	学習済みモデルを読み込み、推論処理を実行させるための初期設定、読み込んだ画像の型変換の前処理、推論結果の表示などを行います。
2	学習済みモデル	入力情報を基に推論し、その結果を出力します。推論処理を実行できる画像の型、推論結果（検出人数や検出した座標など）は使用する学習済みモデルで異なります。

7.1.1 OpenVINO 用に提供されている学習済みモデルの入手方法

学習済みモデルは、次に示すサイトから入手できます。

学習済みモデルのダウンロードサイト

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

学習済みモデルは、上記のサイトに直接アクセスしてダウンロードできます。また、OpenVINO の機能の 1 つである、model downloader を利用してダウンロードすることもできます。

model downloader の使用方法は、次に示す URL を参照してください。

model downloader の使用方法の記載先

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

注※

このマニュアルでは、「2020.3」のバージョンを基に説明しています。OpenVINO のバージョンがアップデートされた場合は、そのバージョンに対応する情報を Intel の Web サイトなどで確認してください。

7.1.2 学習済みモデルを中間表現へ変換する

ユーザーが Tensorflow や Caffe を利用して作成したモデルを OpenVINO で利用する場合、学習済みモデルを中間表現へ変換する必要があります。

OpenVINO には、ユーザーが独自に作成した学習済みモデルを中間表現へ変換する機能 (Model Optimizer) が搭載されています。この Model Optimizer を利用することで、学習済みモデルの中間表現を作成できます。中間表現への変換方法は、次に示す URL を参照してください。

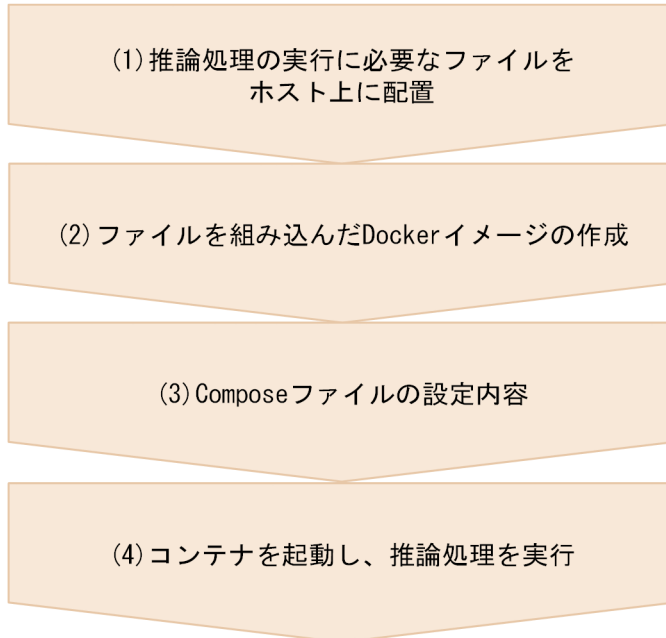
中間表現への変換方法の記載先

https://docs.openvino toolkit.org/2020.3/_docs_MO_DG_prepare_model_convert_model_Converting_Model.html

7.2 推論処理を組み込む流れ

ユーザーが独自に開発した推論処理を、推論実行機能に組み込み実行する方法を説明します。推論処理を組み込む流れを、次の図に示します。

図 7-1 推論処理を組み込む流れ



7.2.1 推論処理の組み込み方法

推論処理の組み込み方法について説明します。

(1) 推論処理の実行に必要なファイルをホスト上に配置

推論処理を実行するために準備するファイルは、「推論処理の実行プログラム」と「学習済みモデル」の2つです。

例として、CE50-10A に配置してあるサンプルプログラム一式を、次に示します。

なお、ファイル保存ディレクトリは一例です。任意のディレクトリに保存することができます。

```

/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

```

サンプルプログラムの構成要素を、次の表に示します。

表 7-2 サンプルプログラムの構成要素

項目	ファイル名	内容
1	AP	サンプルデモ実行時に AP ランプを点灯するために、HTTP サーバを起動するプログラムです。
2	Dockerfile	サンプル動画、または USB カメラからの情報を入力情報として、人数をカウントするデモを実行するイメージを作成するファイルです。docker のビルド時に利用します。
3	docker-compose.yaml	作成したイメージからコンテナを起動するための設定ファイルです。
4	people_count_demo.py	推論処理の実行プログラムです。
5	model_path.json	学習済みモデルまでのパスを記載します。 people_count_demo.py 実行時に読み込みます。
6	sample_video.mp4 ^{*1}	デモに使用する動画を、sample_video.mp4 という名称で配置してください。
7	dataman.json	データ管理機能からのデータの取得方法を記述したファイルです。
8	face-detection-retail-0004.bin ^{*2}	学習済みモデルのうち、重み情報を記述したファイルです。
9	face-detection-retail-0004.xml ^{*2}	学習済みモデルのうち、ネットワーク情報を記述したファイルです。

注※1

- 動画ファイルに必要とされる特性

サンプル動画を使うデモでは、人間の顔を検出して検出人数を表示、または、プレビュー表示で検出した顔を四角で囲って表示します。これらの機能を動作させるために、次のような特性を満たす動画を配置することを推奨します。

- 人の顔が映っていること
- 映っている顔の数が増えること

- 動画ファイルの入手先

一例として、次の URL からダウンロードできる動画がデモに利用できます。この動画ファイルを用いることで、デモの中で顔の検出と、検出人数に応じた人数表示の変化を確認することができます。

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

- 動画の配置方法

ダウンロードした動画のファイル名を「sample_video.mp4」へ変更し、「7.2.1(1) 推論処理の実行に必要なファイルをホスト上に配置」に記載のディレクトリに配置してください。

なお、デモ実行時に配置した動画をエンドユーザへ再頒布する際には動画の著作権や肖像権に留意し、必要に応じてエンドユーザへ頒布する前に配置した動画を削除してください。

注※2

学習済みモデルのダウンロードサイトを、次に示します。

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

(2) ファイルを組み込んだ Docker イメージの作成

次の手順に従って、Docker イメージを作成します。

1. 次のコマンドを実行して、任意のディレクトリに Dockerfile を作成します。

```
$ sudo vi Dockerfile
```

Dockerfile 内に COPY コマンドを使って、コンテナ内にコピーする必要のあるファイルのホスト上のパスと、コンテナ上の配置先を記載します。

Dockerfile の記載例

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

各コマンドの説明

- FROM
ベースのイメージ名を記載します。推論実行機能の場合、"ctrl_edge_ai/openvino_prod:1.0"を記載します。
- CMD
コンテナ実行時に実行する推論処理の python ファイルパス（コンテナ上のパス）を記載します。

2. Dockerfile のビルドを行って、"people_count"という名前の Docker イメージを作成します。

Dockerfile を配置したパスで次のコマンドを実行してください。

```
$ cd <Dockerfileまでのパス>
$ sudo docker build . -t people_count:latest
```

(3) Compose ファイルの設定内容

「(2) ファイルを組み込んだ Docker イメージの作成」で作成した Docker イメージからコンテナを作成したあと、推論処理が実行されるように Compose ファイルを作成します。Compose ファイルの作成については、「4.1 Compose ファイルを利用した使用方法」を参照してください。

Compose ファイルの記載例

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

なお、推論実行時に GPU を利用する場合は、Compose ファイル内の people_count コンテナに"devices"オプションを記載してください。

GPU を利用する場合の Compose ファイルの記載例

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

GPU 実行時は学習済みモデルを GPU 用に変換するため、学習済みモデルの読み込みに数分程度の時間が掛かります。

環境変数 `cl_cache_dir` にパスを指定することで、指定したパスに学習済みモデルを GPU 用に変換したキャッシュが保存されます。次回の学習済みモデルの読み込み時に保存されたキャッシュを利用することで、学習済みモデルの読み込み時間を短縮できます。

上記の設定を行う Compose ファイルの記載方法を、次に示します。なお、必要な部分だけ記載していません。

環境変数 `cl_cache_dir` を使用する場合の Compose ファイルの記載例

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

なお、一般ユーザーでコンテナを起動している場合、キャッシュを残すためには環境変数で指定したパスの権限を変更する必要があります。キャッシュを残すディレクトリの権限は、すべてのユーザーで「読み込み」「書き込み」「実行」のすべてを可能としてください。この変更を行うために、Dockerfile のビルド時にキャッシュを残したいパスへの権限を変更する処理を加える必要があります。

ディレクトリ権限を変更する Dockerfile の記載例を、次に示します。

ディレクトリ権限を変更する場合の Dockerfile の記載例

```
FROM ctrl_edge_ai/openvino_prod:1.0

# Dockerfile内で作業を行うユーザー (root) を指定
USER root

# キャッシュを保存するディレクトリを作成し、ディレクトリの権限を変更
RUN mkdir /cl_cache
RUN chmod 777 /cl_cache

# コンテナ起動時に一般ユーザーを指定
USER openvino

# 環境変数の設定
ENV cl_cache_dir=/cl_cache

# コンテナ起動時に実行するプログラムを設定
CMD [ "/hitachi/people_count_sample/people_count_sample.py" ]
```

また、キャッシュを本番用のイメージに組み込む場合、ホスト上にキャッシュを保存し、Dockerfile のビルド時にホスト上のキャッシュをイメージに組み込む方法が考えられます。

一般ユーザーで起動しているコンテナから、ホスト上のディレクトリをマウントし、コンテナを削除したあともホスト上にキャッシュを残すためには、コンテナ上のディレクトリの権限だけではなく、ホスト上のディレクトリの権限も変更する必要があります。

ホスト上のディレクトリ権限の変更方法の実行例と、ホスト上に残したキャッシュをイメージに組み込む場合の Dockerfile の記載例を、次に示します。

ホスト上のディレクトリの権限を変更するコマンド実行例

```
# コンテナにマウントされるホスト上のディレクトリの権限を変更
$ sudo chmod 777 /home/edgeadm/cl_cache

# /home/edgeadm/cl_cache をマウントしたコンテナを起動し、ホスト上にキャッシュの生成
$ sudo docker-compose up -d
```

ホスト上に残したキャッシュをイメージに組み込む場合の Dockerfile の記載例

```
FROM ctrl_edge_ai/openvino_prod:1.0
```

```
# Dockerfile内で作業を行うユーザー (root) を指定
USER root

# ホスト上に保存されているキャッシュをイメージに組み込む
COPY <cl_cacheディレクトリへの相対パス>/* /cl_cache

# ディレクトリの権限を変更
RUN chmod 777 /cl_cache

# コンテナ起動時に一般ユーザーを指定
USER openvino

# 環境変数の設定
ENV cl_cache_dir=/cl_cache

# コンテナ起動時に実行するプログラムを設定
CMD [ "/hitachi/people_count_sample/people_count_sample.py" ]
```

(4) コンテナを起動し、推論処理を実行

次のコマンドを実行して、コンテナを起動し推論処理を実行します。

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

7.3 サンプルプログラムの解説

CE50-10A ではサンプルプログラムを用意しています。ソースコードの全文を確認する場合は、次に示すファイルを参照してください。

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

7.3.1 サンプルプログラムの処理内容

CE50-10A が提供するサンプルプログラム (people_count_demo.py) の処理内容を説明します。サンプルプログラム内で利用している学習済みモデルは、"face-detection-retail-0004"です。

"face-detection-retail-0004"に対応する入力情報と出力情報は、次のとおりです。

- 入力情報：[1×3×300×300] (画像のバッチ数×配色×画像の高さ×画像の幅)
- 出力情報：[1, 1, N, 7] (N には検出人数や検出した座標などを含む)

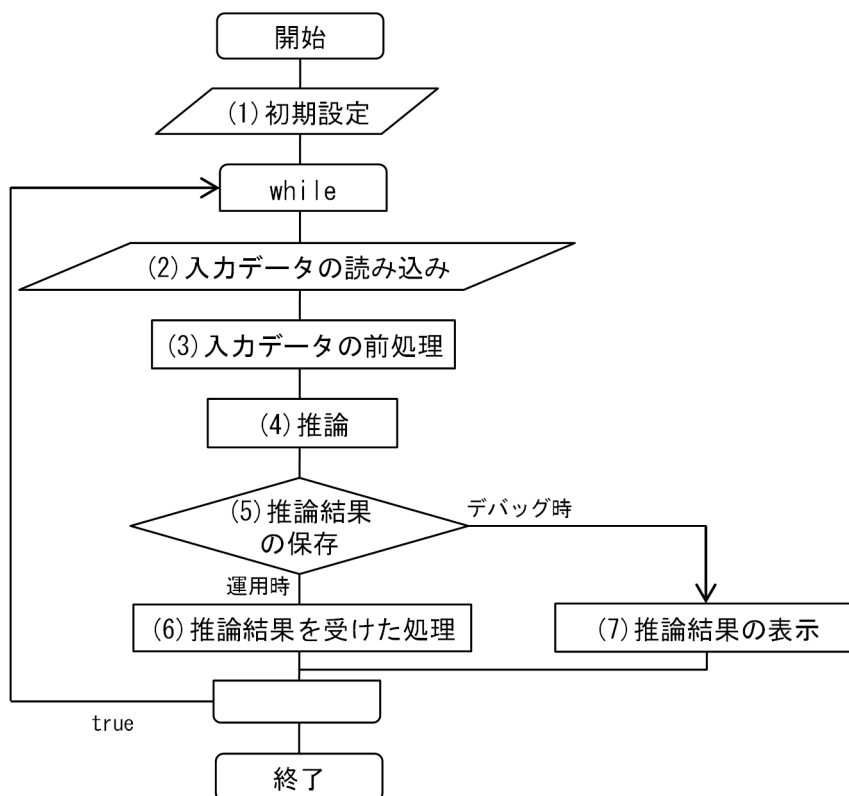
入出力情報の詳細は、次の URL を参照してください。

```
https://docs.opencv.org/2020.3/_models_intel_face_detection_retail_0004_description_face_detection_retail_0004.html
```

提供するサンプルプログラムは、推論を実行するための前処理と推論の実行、AP ランプを点灯させる処理などを行います。

提供するサンプルプログラムの処理の流れを、次の図に示します。

図 7-2 サンプルプログラムの処理の流れ



(1) 初期設定

初期設定では、次の6つを設定します。

①ライブラリの読み込み

サンプルプログラムでは、次の5つのライブラリを読み込み、推論で利用しています。

- cv2：画像処理用のライブラリ（OpenCVのPython用ラッパー）
- json：json形式を扱うライブラリ
- numpy：数値計算用のライブラリ
- IECore：OpenVINOの推論エンジン用のライブラリ
- Dataman：データ管理機能コンテナから画像を取得するためのライブラリ

該当のソースコードを次に示します。

```
# ライブラリの読み込み
import cv2
import numpy as np
import json
import requests

from opencv.inference_engine import IECore
# データ管理機能ライブラリ
from ce50.ai import *
```

②IECore と Dataman のクラスからインスタンス作成

①で読み込んだライブラリから、IECore と Dataman のインスタンスを作成します。

IECore クラスの詳細については、次の URL を参照してください。

```
https://docs.opencv.org/2020.3/ie_python_api/classie__api_1_1IECore.html#afe73d64ddd115a41f5acc0d31031f52b
```

該当のソースコードを次に示します。

```
# Inference EngineのIECoreクラスのインスタンスieを生成
ie = IECore()
# Datamanクラスのインスタンスdmを生成
dm = Dataman("/hitachi/people_count_sample/dataman.json")
```

③学習済みモデルのパス指定と学習済みモデルの読み込み

利用する学習済みモデル（xml ファイルと bin ファイル）のパスを指定し、②で作成したインスタンスを利用して学習済みモデルを読み込みます。

該当のソースコードを次に示します。

```
# model_path.jsonの読み込み
json_open = open('/hitachi/people_count_sample/model_path.json', 'r')
json_load = json.load(json_open)
# jsonファイルからIRまでのパスを読み込み
xml = json_load['model_path']['xml']
bin = json_load['model_path']['bin']
# 読み込んだjsonファイルを閉じる
json_open.close()
net = ie.read_network(model=xml, weights=bin)
```

④読み込んだ学習済みモデルを、選択したデバイス（CPU または GPU）に対応した実行可能オブジェクトに変換

③で読み込んだ学習済みモデル（今回は"net"）を、指定したデバイスに対応した推論処理実行可能オブジェクトに変換します。

該当のソースコードを次に示します。

```
# 読み込んだモデルを選択したデバイス（CPUまたはGPU）で実行可能オブジェクトに変換
exec_net = ie.load_network(network=net, device_name="CPU")
```

⑤読み込む学習済みモデルに合わせて、入力情報と出力情報を取得

読み込んだ学習済みモデルを実行できる入出力情報の型を取得します。

入出力情報に設定する項目として、次の項目があります。

- 入力情報：画像の幅や高さ、配色、一度に処理する画像数、画像の精度
- 出力情報：画像の精度

学習済みモデルに対応する入出力情報と画像の精度に関しては、次の URL を参照してください。

学習済みモデルについて：

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

精度について：

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

該当のソースコードを次に示します。

```
# 読み込む学習済みモデルに合わせて、入力情報と出力情報の設定
for input_key in net.inputs:
    if len(net.inputs[input_key].layout) == 4:
        n, c, h, w = net.inputs[input_key].shape
```

⑥検出する閾値の設定

顔領域と判定するための閾値を設定します。詳細は、「(5) 推論結果の保存」を参照してください。

該当のソースコードを次に示します。

```
# 検出する閾値の設定
DETECTION_THRESHOLD = 0.5
```

(2) 入力データの読み込み

データ管理機能コンテナから入力情報を取得します。データ管理機能コンテナから入力情報を取得するためには、Dataman クラスの get_frame 関数を利用します。

該当のソースコードを次に示します。

```
# retはフレーム取得の成否 (True or False) を格納
# infoはフレーム情報を格納
# frameは入力画像データを格納
ret, info, frame = dm.get_frame()
```

(3) 入力データの前処理

読み込んだ学習済みモデルの推論実行条件に合わせて入力情報の変換を行います（推論実行条件は「(1) 初期設定」の⑤で指定）。入力情報（画像）は「高さ」「幅」「配色」という情報を3次元の行列で表現しています。ここでは、学習済みモデルが求めている構造に入力情報を変換して、変換したあとの入力情報を "images" に格納します。

該当のソースコードを次に示します。

```
# 入力データを学習済みモデルが実行できるサイズ（今回は高さ×幅を300×300）へ変換
if (ih, iw) != (h, w):
    image = cv2.resize(image, (w, h))
# データの型を(h, w, c)から(c, h, w)への変換
image = image.transpose((2, 0, 1))
images = image
```

(4) 推論

「(3) 入力データの前処理」で読み込んだ学習済みモデルで実行可能な型へ変換した入力情報 (images) を"data"に辞書型で格納します。格納した入力情報を基に推論処理を実行して、結果を"res"に取得します。

OpenVINO の推論処理に関しては次の URL を参照してください。

```
https://docs.opencv.org/2020.3/ie_python_api/classie__api_1_1InferRequest.html#aac8de3eea8b2eec962bd5b64a176f618
```

該当のソースコードを次に示します。

```
# images (変換処理済みの画像) をdataに格納
data = {}
data[input_name] = images

# 推論
res = exec_net.infer(inputs=data)
```

(5) 推論結果の保存

推論結果を"res"に格納します。

"res"には、入力情報から検出した顔領域の件数分、次の情報が格納されています。

- 顔の領域を示す座標 (左上座標 (x,y)、右下座標 (x,y))
- その領域が顔を示していることの確からしさの度合い (0~1)

ここでは、推論結果から、検出した顔領域のデータを for で結果を 1 件ずつ取り出し、「(1) 初期設定」の⑥で設定した閾値と、顔を示していることの確からしさを比較します (⑥で設定した閾値以上の確からしさを持つ推論結果だけを顔領域とし、処理しやすいように boxes、classes という配列に格納しています)。

該当のソースコードを次に示します。

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

(6) 推論結果を受けた処理

運用時には推論結果から、設定に応じた処理を実行します。

例えば、推論の結果から異常を検出した場合、その異常を管理者へ通知することなどです。

サンプルプログラムでは、検出人数に応じてランプを点灯させてユーザーに通知する処理を行っています。

次にサンプルプログラムの当該箇所を示します。

ホスト上で AP ランプを操作するプロセス (AP) を起動しておき、そのプロセスに HTTP で指定した色で AP ランプを点灯させるようにリクエストを送信しています。

```
if len(boxes) == 0:
    people_num = 0
    print("people=0")
else:
    people_num=len(boxes[imid])
    print("people="+str(people_num))
# 検出人数が0人の場合は「off」、1人~2人の場合は「green」、3人以上の場合は「red」を「ap_color」へ格納
if people_num >= 3:
```

```

    ap_color="red"
elif people_num >= 1 :
    ap_color="green"
else:
    ap_color="off"

# ap_colorをURLの末尾に指定して、APプロセスへリクエストを送信する
if ap_color != current_ap_color:
    requests.get('http://127.0.0.1:5000/' + ap_color)
    current_ap_color=ap_color

```

推論完了をデータ管理機能へ通知

データ管理機能に当該フレームの推論が完了したことを通知します。

入力元が画像ファイルであればその時点で、動画ファイルであれば末尾を処理した時点でデータ管理テーブルが更新されます。推論処理の実行プログラムでは、ファイルをどこまで推論したかを意識する必要はありません。

```

# 当該フレーム
dm.update_frame_status(info, Dataman.STATUS_COMPLETED)

```

(7) 推論結果の表示

作成した学習済みモデルの精度確認などを目的に、デバッグ時は画面上に推論結果を表示する必要があります。

推論結果を表示するには、「cv2.rectangle」で推論結果から得られた検出箇所を四角で囲み、「cv2.imshow」で画面に表示します。

サンプルコードに使用している「cv2.rectangle」は、次の引数を利用します。

- 第1引数：入力画像
- 第2引数：検出した顔領域の左上の座標（X座標、Y座標）
- 第3引数：検出した顔領域の右下の座標（X座標、Y座標）
- 第4引数：描画する線の配色
- 第5引数：描画する線の太さ

「cv2.rectangle」の詳細については、次のURLを参照してください。

https://docs.opencv.org/3.4.0/d6/d6e/group_imgproc_draw.html#ga346ac30b5c74e9b5137576c9ee9e0e8c

推論結果を表示するサンプルコードを、次に示します。

```

# デバッグ時だけ次の処理を実行する
# デバッグを行いたい場合は実行時に「-0」をオプションで追加する（デバッグの実行例：$ python -0 demo.py）。
if not debug :
    # 推論結果を画面上に入力情報とともに表示する場合「cv2.rectangle」を利用し、
    # 検出座標を四角で囲み、tmp_imageとして保存
    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)

    # 検出結果の表示(第1引数：文字列型で指定するウィンドウ名、第2引数：表示する画像)
    cv2.imshow('demo', tmp_image)
    cv2.waitKey(1)

```

8

推論開発機能

推論開発機能の概要、起動手順、Jupyter Notebook の利用方法について説明します。

8.1 推論開発機能の概要

CE50-10A は、ユーザーが独自の推論処理を開発できる推論開発機能を提供します。

具体的には、データ分析や AI 開発で標準的に使用されている Jupyter Notebook を提供します。

Jupyter Notebook については、次の URL を参照してください。

<https://jupyter.org/>

Jupyter Notebook を使用して、CE50-10A と同一ネットワークに接続された PC の Web ブラウザから CE50-10A に接続し、入力した Python プログラムをインタラクティブに実行して動作確認やデバッグを行うことができます。

8.2 推論開発機能の起動手順

CE50-10A、または CE50-10A と同一ネットワークに接続された PC から、推論開発機能を起動する手順を、次に示します。

1. CE50-10A とネットワーク接続された PC から CE50-10A に SSH など接続し、次のコマンドを実行して、推論開発機能コンテナを起動します。なお、次のコマンドは CE50-10A を直接操作して、実行することもできます。

```
# dockerデーモンの再起動を行う（起動しているコンテナがある場合は停止する）
$ sudo systemctl restart docker

# ポートを指定し、コンテナを作成する
# ホスト上の8888番ポートと、コンテナの8888番ポートを接続する
$ sudo docker container run -it --rm -p 8888:8888 ctrl_edge_ai/openvino_devel:1.0
```

コマンド実行後、次のようなメッセージが表示されます。表示されるトークンは、Jupyter Notebook にログインする際のパスワードであり、コンテナ起動のたびに表示されるトークンは異なります。

```
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. 同一ネットワークに接続された PC の Web ブラウザを起動して、アドレスバーに次の情報を入力してください。

アドレスバー

```
http://CE50-10AのIPアドレス:8888/?token=<xxx>
```

<xxx>には、手順 1. で表示されたトークンの番号を入力してください。

3. アドレスバーに手順 2. の情報を入力後、[Enter] キーを押します。

Jupyter Notebook 画面が Web ブラウザ上に表示されます。

図 8-1 Jupyter Notebook の画面



4. 操作を終了したあと、CE50-10A で作成した推論開発機能コンテナを閉じて、CE50-10A でオープンしたポートをクローズします。

```
# 推論開発機能コンテナが画面に表示されている状態で、
# [Ctrl] + [c] キーを押すと、Shutdown this notebook server (y/[n])?と表示されるので、
# yを入力し、[Enter] キーを押す。
```

8.3 Jupyter Notebook の利用方法

ここでは、Jupyter NoteBook の基本的な使い方を説明したあと、Jupyter Notebook で OpenVINO を使う手順を説明します。

8.3.1 Jupyter NoteBook の基本的な使用方法

Jupyter Notebook の基本的な操作手順を説明します。

(1) Jupyter Notebook にログインする

Jupyter Notebook 起動時に生成される URL に Web ブラウザでアクセスすると、Jupyter Notebook のホーム画面が表示されます。

操作手順については、「8.2 推論開発機能の起動手順」を参照してください。

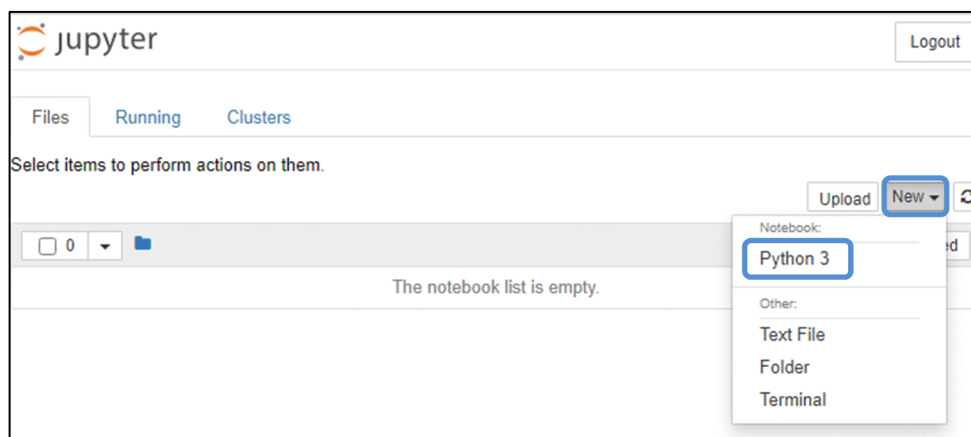
(2) Jupyter Notebook で Python3 のノートブックを新規作成する

Jupyter Notebook で Python3 のノートブックを新規作成する手順を、次に示します。

1. Jupyter Notebook のホーム画面で [New] ボタンをクリックします。
2. 表示されるメニューから [Python3] をクリックします。

Python3 のノートブックが新規作成されます。

図 8-2 ノートブックの新規作成



(3) Python3 を実行する（文字列の標準出力）

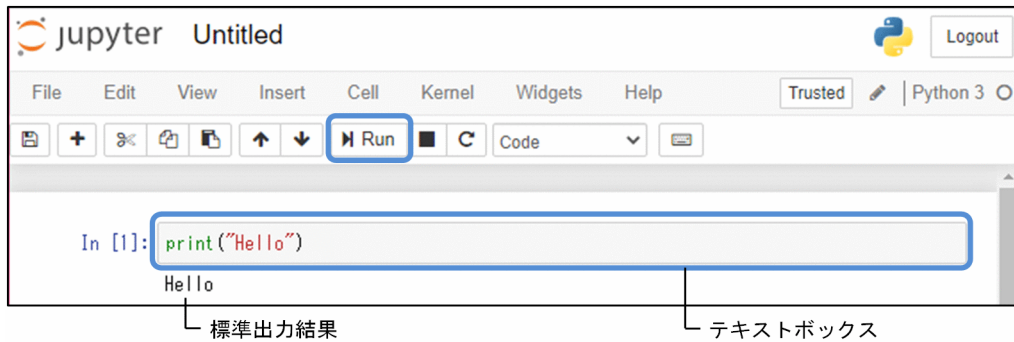
Jupyter Notebook で Python3 を実行する手順を、次に示します。

1. ノートブックの編集画面で "In" と表示されているテキストボックスに Python のソースコードを入力します。
2. [Run] ボタンをクリックします。

[Run] ボタンをクリックしたあと、テキストボックス内のプログラムが実行され、出力結果がテキストボックスの下に表示されます ("In" のテキストボックス単位でまとめて実行できます)。

print 関数を使うと、次の図に示すテキストボックスの下に文字列が出力されます。

図 8-3 文字列の標準出力例

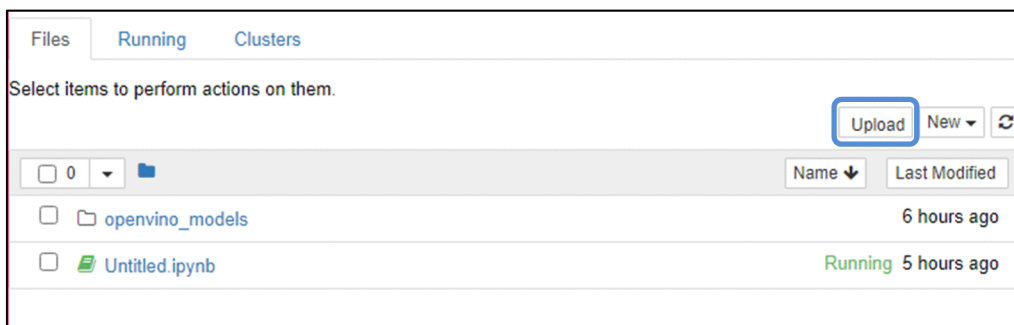


(4) ファイルをアップロード/ダウンロードする

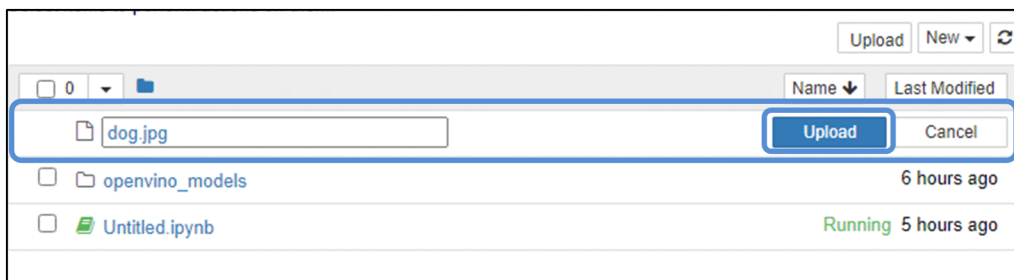
ファイルをアップロードする場合

Jupyter Notebook でファイルをアップロードする手順を、次に示します。

1. Jupyter Notebook のホーム画面で [Upload] ボタンをクリックします。



2. 表示されたエクスプローラーで、アップロードするファイルを指定します。
3. 指定したファイルがホーム画面に表示されたあと、再度 [Upload] ボタンをクリックします。



ファイルをダウンロードする場合

Jupyter Notebook でファイルをダウンロードする手順を、次に示します。

1. Jupyter Notebook のホーム画面でダウンロードするファイルのチェックボックスにチェックを付けます。



2. [Download] ボタンをクリックします。



8.3.2 Jupyter Notebook 上で OpenVINO を使う手順

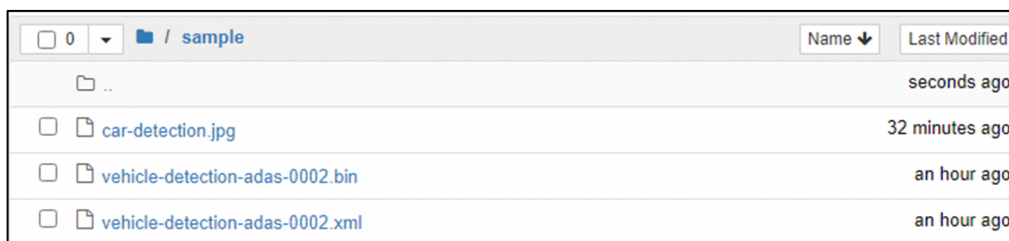
ここでは、Jupyter Notebook 上で OpenVINO を使った推論を行う際の手順を、自動車検出プログラムの実行例に沿って説明します。

(1) 前準備

- テストデータ（この例では car-detection.jpg）と学習済みモデル（.xml と .bin）をアップロードします。

画像出典

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



- Python3 のノートブックを新規作成し、編集画面を開きます。

(2) OpenVINO を使う手順

1. 必要なライブラリを読み込みます。

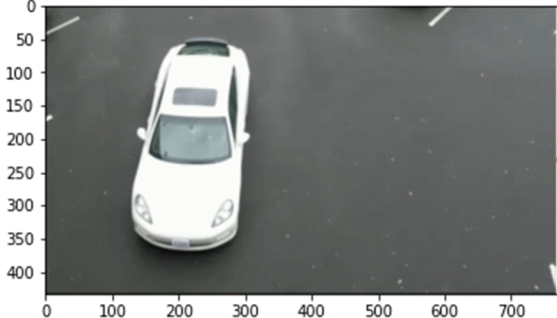
ライブラリが存在しない場合は、インストールしてください。

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

2. 画像ファイルを読み込んで、画像を Jupyter Notebook の画面に表示します（画像、図、表の描画は「Out」の部分に表示されます）。

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

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



3. 学習済みモデルを読み込みます。

```
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. 学習済みモデルに適応させるために入力画像の型とサイズを変換します。

```
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. 推論を実行します。

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

6. 推論結果の出力準備をします。

```
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. 推論結果を画面に描画します。

```
In [7]: for box in boxes:
img = cv2.rectangle(img, (box[0], box[1]), (box[2], box[3]), (255, 0, 0), 2)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

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



9

AI 映像アプリ機能向け RAS 機能

エラー検出、コンテナの再起動、稼働情報収集の設定方法について説明します。

9.1 AI 映像アプリ機能向け RAS 機能の概要

AI 映像アプリ機能向けの RAS 機能を、次に示します。

- エラー検出

各機能のコンテナ上で異常が発生した場合、ログメッセージを出力して、CE50-10 で提供しているログ世代管理の対象とします。

- コンテナの再起動

コンテナが異常終了した場合、コンテナを再起動するかどうかを設定します。

- 稼働情報収集の設定

Docker サービスおよび Docker コンテナが出力するログを CE50-10 が提供している保守情報一括収集コマンド (eclogsave) の対象とします。保守情報一括収集コマンド (eclogsave) については、「CE50-10 取扱説明書」の「eclogsave」を参照してください。

また、ユーザーが開発したアプリケーションが出力するログファイルを、eclogsave で取得する方法について記載します。

9.2 エラー検出

AI 映像アプリ機能に不具合が発生した場合でも、そのエラーを検出できるよう、各プロセスが出力したメッセージの出力先を Compose ファイルに設定できます。また、出力するログのフォーマットやログファイルの容量/世代管理について説明します。

9.2.1 Compose ファイルの設定内容

各機能のコンテナ上のプロセスが出力するメッセージを、syslog 経由で/var/log/docker.log に出力するためには、Compose ファイルに次の logging: を指定してください。

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

設定項目について、次の表に示します。

表 9-1 Compose ファイルの logging: の設定項目の説明

項番	設定項目	説明
1	driver:	ロギングドライバを指定します。 syslog 経由で/var/log/docker.log に出力するため、指定値は"syslog"固定です。
2	options:	ログに関するオプションを指定します。
3	syslog-facility:	syslog のファシリティを指定します。 syslog フィルタリングによって/var/log/docker.log に出力するため、指定値は"daemon"固定です。
4	tag:	ログの先頭に出力するタグを指定します。 syslog フィルタリングによって/var/log/docker.log に出力するため、指定値は"docker-compose/{{.Name}}/{{.ID}}"固定です。 <ul style="list-style-type: none"> • {{.Name}}: コンテナ名 • {{.ID}}: コンテナ ID の冒頭 12 文字

9.2.2 ログフォーマット

/var/log/docker.log に出力するログメッセージのフォーマットを、次に示します。

```
<記録日時> <ホスト名> docker-compose/<コンテナ名>/<コンテナID(12文字)>[<プロセスID>]: <コンテナログメッセージ>
<記録日時> <ホスト名> dockerd[<プロセスID>]: <dockerログメッセージ>
<記録日時> <ホスト名> containerd[<プロセスID>]: <dockerログメッセージ>
```

9.2.3 ログファイル容量/世代管理

/var/log 配下のログファイルの容量や管理できる世代数などを、次の表に示します。

表 9-2 ログファイル容量および世代管理

項番	ログファイル名	ログ内容	世代管理	容量 [KB]	収集周期	サイズ [KB] / 周期 (概算)	ログ保存期間 [日] (概算)
1	docker.log	Docker ログ	3	3,072 (1,024×3)	随時	13*	236*

注※

1 メッセージを 50 [B]、タグ部を 80 [B]、100 [件/日] のログ出力があることを想定して算出しています。

1 日当たりのログ容量：(50 + 80) [B/ログ] × 100 [件/日] = 13 [KB/日]

ログ保存期間：3,072 [KB] ÷ 13 [KB/日] = 236 [日]

なお、Docker コンテナ内の標準出力はすべて docker.log へ記録されるため、継続的に標準出力を繰り返すアプリケーションを実装した場合、ログの保存期間は想定よりも大幅に少なくなります。そのため、ログへ記録する頻度や内容は、アプリケーション側で適切に設計する必要があります。

上記以外の/var/log 配下のログファイルについては「CE50-10 取扱説明書」の「保守情報を採取する」を参照してください。

9.3 コンテナの再起動

機能のコンテナが異常終了した際に、コンテナを自動で再起動させたい場合は、「表 4-1 Compose ファイルの設定内容の説明」の記載のとおり、Compose ファイルに"restart:on-failure"を指定します。

"restart:on-failure"の指定がない場合（デフォルト）は、コンテナの自動再起動を行いません。

"on-failure"には、リトライ回数の指定ができます。リトライ回数を設定するメリットとデメリットを、次の表に示します。

表 9-3 リトライ回数を設定するメリットとデメリット

項番	リトライ回数設定	メリット	デメリット
1	リトライ回数を多く設定した場合	障害発生時に、復旧できる可能性が高まります。	リトライによるコンテナ再起動のログによって、初回障害発生時のログが消えるおそれがあります。
2	リトライ回数を少なく設定した場合	初回障害発生時のログが消える可能性が低くなります。	<ul style="list-style-type: none"> 障害発生時に復旧できる可能性が低くなります。 リトライ回数はコンテナ起動からの累計回数であるため、一過性障害であっても、長期稼働するシステムでは、復旧できる可能性が低くなります。

9.4 稼働情報収集登録方法

CE50-10 で提供している保守情報一括収集コマンド (eclogsave) は、OS が記録・保存しているログデータ、ユーザーが定義ファイルに登録したログファイルおよび障害解析時に必要となるシステム情報を一括して収集するコマンドです。CE50-10A では、/var/log/docker.log についても、eclogsave によってデフォルトで収集対象となります。

また、ユーザーがコンテナ上で作成したログなどのファイルは、ホスト側から参照できるようにしておくことで、eclogsave の収集対象とすることができます。

コンテナ上のファイルをホスト側から参照するためには、Compose ファイルに volumes 指定した共有ディレクトリを活用することが、一例として挙げられます。

ユーザーファイルを eclogsave の収集対象とする場合は、ユーザー定義ファイル (/hitachi/etc/save_applog.def) に収集対象ファイルをホスト側の絶対パスで記述してください。

ユーザー定義ファイル (/hitachi/etc/save_applog.def) の記述例

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

10 アップデート機能

アップデート機能の概要、アップデート方法について説明します。

10.1 アップデート機能の概要

Docker では、コンテナに搭載したソフトウェアやファイルを、イメージとして他のホスト上にコピーして動作させることができます。

通常、プログラムを別のホスト上で動作させるためには、次の作業が必要となります。

1. 前提ライブラリのインストール
2. プログラムの実行に必要なファイルの配置

同じプログラムを複数のホスト上で動作させる場合、1 台ごとに上記の作業が必要となり、ホスト上で動作させるまでに時間が掛かります。

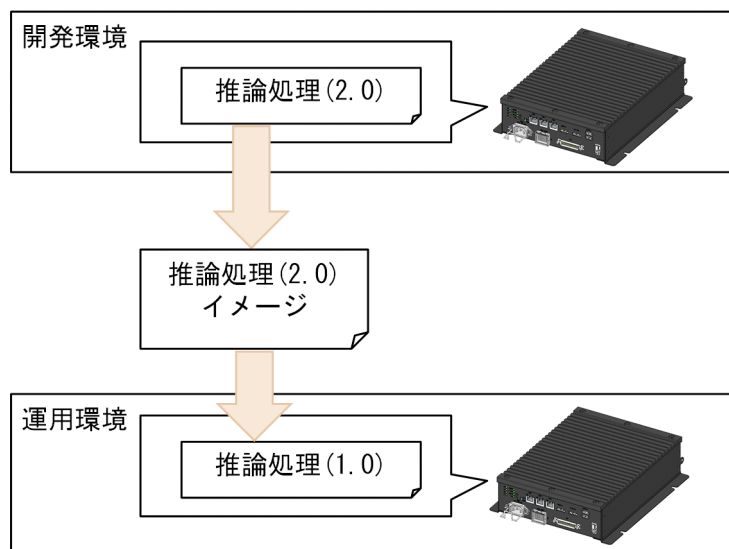
Docker イメージでは、前提ライブラリやプログラムの実行に必要なファイルを 1 個のイメージにまとめて取り扱うことができます。ホストにイメージを配置し、イメージからコンテナとして実行させるだけでプログラムを実行でき、上記 1.、2. に示すような作業の時間を減らすことができます。

また、プログラムのアップデートをする場合も、イメージファイルを置き換えるだけで済ませることができます。

10.2 アップデートの流れ

アップデート機能を利用する環境を、次の図に示します。

図 10-1 アップデート機能の利用例

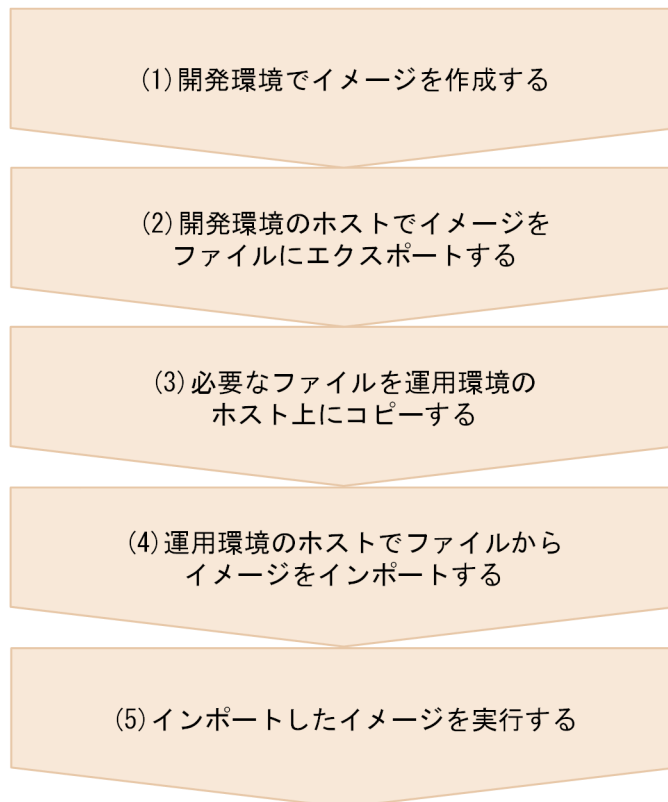


この章の説明では、運用中の CE50-10A では推論処理バージョン 1.0 が動作している状態、開発中の CE50-10A では推論処理バージョン 2.0 が動作している状態を想定します。

10.2.1 アップデート方法

旧バージョン (1.0) から新規バージョン (2.0) へアップデートする場合の流れを、次の図に示します。

図 10-2 アップデートする場合の流れ



推論実行機能コンテナが実行中であっても (1) ~ (4) の作業はできますが、事前に推論実行機能コンテナを停止することを推奨します。

(1) 開発環境でイメージを作成する

CE50-10A で提供しているイメージに、作成した設定ファイルや推論のプログラムを組み込み、新しいイメージとして Docker に登録します。

実施手順は「7.2.1(2) ファイルを組み込んだ Docker イメージの作成」を参照してください。

ここでは、推論実行機能コンテナのイメージを想定して記載しています。データ管理機能コンテナ、データ入力機能コンテナなどでも同じ手順でコンテナを作成できます。

(2) 開発環境のホストでイメージをファイルにエクスポートする

次に示すコマンドを実行して、イメージをファイルにエクスポートします。

コマンド実行例：

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

(3) 必要なファイルを運用環境のホスト上にコピーする

ネットワーク経由での scp や、USB メモリなどの記憶媒体を用いて必要なファイルをホスト上にコピーします。

ネットワーク経由でコピーする場合、開発環境のホスト上で次のコマンドを実行してください。

コマンド実行例：

```
$ scp ./people_count_2_0.tar edgeadm@<運用ホストIPアドレス>:<イメージファイル配置先パス>
```

(4) 運用環境のホストでファイルからイメージをインポートする

次に示すコマンドを実行して、イメージをファイルにインポートします。

コマンド実行例：

```
$ cd <イメージファイル配置先パス>  
$ sudo docker load < people_count_2_0.tar
```

(5) インポートしたイメージを実行する

イメージ名が以前から変わっている場合には、次に示す Compose ファイルの image: の個所を変更してください。

変更前：

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

変更後：

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

次のコマンドでイメージを更新したサービスを起動します（内部的にコンテナが停止され、新しいコンテナが起動します）。

```
$ cd <docker-compose.yamlを配置しているパス>  
$ sudo docker-compose up -d <サービス名>
```


11

トラブルシューティング

AI 映像アプリ機能が表示するエラーメッセージと、その対処方法について説明します。

11.1 docker-compose コマンド実行時のエラーメッセージ

Docker-compose コマンド実行時に表示されるエラーメッセージ、および対処方法について記載します。

表 11-1 docker-compose コマンド実行時のエラーメッセージ

項番	メッセージ	説明	対処方法
1	ERROR: Can't find a suitable configuration file in this directory or any parent. Are you in the right directory? Supported filenames: docker-compose.yml, docker-compose.yaml	Compose ファイルが見つかりません。	Compose ファイルを-f オプションで指定するか、カレントに docker-compose.yaml または docker-compose.yml のファイル名で配置してください。
2	Pulling <XXXX> (<YYYY>:<ZZZZ>).. ERROR: Get https://registry-1.docker.io/v2/: dial tcp: lookup registry-1.docker.io: Temporary failure in name resolution※	Compose ファイルに記述した Docker イメージが見つかりません。 <XXXX> : サービス名 <YYYY> : Docker イメージ名 <ZZZZ> : タグ名	Compose ファイル内に記述した Docker イメージ名を見直してください。
3	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※	コマンドが実行できません。 <XXXX> : サービス名 <YYYY> : コマンド	コマンド名、パスを見直してください。
4	ERROR: Service '<XXXX>' depends on service '<YYYY>' which is undefined.	<XXXX>サービスの depends_on に記述した<YYYY>サービスが見つかりません。	コンテナの依存関係を見直し、depends_on に正しく記述してください。
5	ERROR: No containers to start	開始 (start) するコンテナがありません。	docker-compose ps で開始すべきコンテナが存在していることを確認してください。存在していない場合は、docker-compose up [-d]でコンテナを生成して開始してください。

注※

ネットワーク環境や実行するコマンドなどによって、下線部分の内容は異なる場合があります。

11.2 データ入力機能のエラーメッセージ

データ入力機能の使用時に表示されるエラーメッセージ、および対処方法について記載します。

表 11-2 データ入力機能のエラーメッセージ

項番	メッセージ	説明	対処方法
1	Failed to parse <data_input.json へのパス>	data_input.json の書式が誤っています。	data_input.json の記載内容を見直してください。
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. または Error: Could not open resource for reading and writing. または Error: Unauthorized	カメラに接続できません。	カメラが正しく接続されているか確認してください。また、docker-compose.yaml の設定を見直してください。
3	Error received from element ximagesink0: Could not initialise X output Debugging information: ximagesink.c(860): gst_x_image_sink_xcontent_get (): /GstPipeline:pipe0/GstXImageSink:ximagesink0: Could not open display	プレビュー利用時、画面出力に失敗しました。	docker-compose.yaml に X 転送の設定が正しく記載されているか確認してください。
4	Error: Internal data stream error.	USB カメラ使用時、V4L ドライバでエラーが発生しました。	USB カメラが正しく接続されているか、適切な解像度・FPS を設定しているか確認してください。

11.3 データ管理機能のエラーメッセージ

データ管理機能の使用時に表示されるエラーメッセージ、および対処方法について記載します。

表 11-3 データ管理機能のエラーメッセージ

項番	メッセージ	説明	対処方法
1	Failed to parse <data_manager.jsonへのパス>	data_manager.json の書式が誤っています。	data_manager.json の記載内容を見直してください。
2	Failed to parse <dataman.jsonへのパス>	dataman.json が読み込めませんでした。または、書式が誤っています。	dataman.json の記載内容を見直してください。
3	The value is over usable volume size.	unfinished_limit、finished_limit の値が、ボリュームの空き容量を超えています。	ボリュームの空き容量より小さい値に設定し直してください。
4	Failed to connect Data Manager API.	API にアクセスできません。	dataman.json に記載した URL に誤りがないか確認してください。また、docker-compose.yaml 上のデータ管理機能コンテナのポート設定が適切か確認してください。

11.4 推論実行機能のエラーメッセージ

推論実行機能の使用時に表示されるエラーメッセージ、および対処方法について記載します。

表 11-4 推論実行機能のエラーメッセージ

項番	メッセージ	説明	対処方法
1	Failed to connect Data Manager API.	推論実行機能コンテナを起動したあと、データ管理機能コンテナから映像ファイルが取得できません。	データ管理機能コンテナの状態を確認し、コンテナがうまく起動していない場合は、「4.3.1 Docker コンテナの起動方法」を参照してコンテナを起動してください。
2	Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?	docker デーモンが起動していません。	「3.1.2 Docker サービスの起動」を参照して、docker デーモンを起動してください。
3	RuntimeError: Failed to create plugin /opt/intel/opencv/deployment_tools/inference_engine/lib/intel64/libcLDNNPlugin.so for device GPU	コンテナ上で GPU が起動できません。	「4.1 Compose ファイルを利用した使用方法」を参照して、GPU デバイスをコンテナに割り当てる設定を docker-compose.yaml に記載してください。

11.5 推論開発機能のエラーメッセージ

推論開発機能の使用時に表示されるエラーメッセージ、および対処方法について記載します。

表 11-5 推論開発機能のエラーメッセージ

項番	メッセージ	説明	対処方法
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>	CE50-10A とコンテナのポートが接続できません。	docker デーモンを再起動してください。

付録

付録 A CE50-10A のインタフェース

CE50-10A がサポートするインタフェースについて説明します。

付録 A.1 サポート仕様

AI 映像アプリ機能がサポートする仕様を、次の表に示します。

表 A-1 サポート仕様

項番	項目	仕様
1	カメラサポート	<ul style="list-style-type: none"> • USB2.0/3.0 接続の USB カメラ (USB Video Class 1.0/1.1 対応) • IP カメラ (RTSP) コーデック <ul style="list-style-type: none"> • USB カメラ：非圧縮、Motion-JPEG • IP カメラ：Motion-JPEG、H.264
2	入力ファイル形式	JPEG、PNG、MPEG-2、MPEG-4 (H.264)
3	最大解像度	3840×2160
4	推論処理 開発言語	Python
5	学習済モデルサポート	OpenVINO IR (拡張子.xml、.bin) 精度：FP32、FP16、INT8
6	推論実行デバイス	CPU、GPU
7	開発環境	OpenVINO、Jupyter Notebook、Python
8	ネットワーク経由操作時に使用する PC	<ul style="list-style-type: none"> • OS：Windows 10 1803 以降 • Web ブラウザ：Google Chrome