OpenVINO™ ツールキットとインテル® System Studio を使用する

この記事は、インテル® デベロッパー・ゾーンに掲載されている「Get Started Using the OpenVINO™ Toolkit with Intel® System Studio」の日本語参考訳です。

このガイドは、インテル® System Studio と Eclipse* ベースの統合開発環境 (IDE) および OpenVINO™ ツールキットを利用したコンピューター・ビジョンやディープラーニング最適化アプリケーションの開発に興味がある開発者向けに作成されたものです。

プロセスの概要

このガイドのセクションは、インストールの必要条件からプロジェクトの実行やデバッグまでのプロセスを順に説明します。

1. 必要条件
    - システム要件を確認してホストシステムにインテル® System Studio をインストールします。
    - インテル® System Studio を起動します。
2. インテル® System Studio での新しいプロジェクトの作成とビルド
    - インテル® System Studio で OpenVINO™ ツールキットを使用するアプリケーションをビルドできるようにするカスタム Docker* イメージを作成します。
    - サンプルを選択してプロジェクトをビルドします。
3. OpenVINO™ ターゲットシステムの準備
    - ターゲットシステムに OpenVINO™ ツールキットをインストールしてターゲットシステムへの接続を作成します。
    - サンプル・アプリケーションを実行してデバッグします。
        - セキュリティー・バリア・カメラのサンプル
        - インタラクティブ顔検出のデモ
        - 画像分類のサンプル
        - Hello 推論のサンプル
        - その他のサンプル
4. トラブルシューティング

定義

- **ホストシステム:** 開発ツールをインストールしたローカルシステム。グラフィカル・インターフェイスを備えています。
- **ターゲットシステム:** 開発およびテスト中にアプリケーションを実行するシステム。

ターゲットシステムは、ホストシステム (ローカル) と同じ、またはホストシステムからネットワークでアクセスできる別のシステム (リモート) にできます。

関連情報

OpenVINO™ ツールキットおよびインテル® System Studio の詳細は、次を参照してください。

- インテル® System Studio 製品ガイド (英語)
- ビジョン・コンピューティング (英語)
- インテル® System Studio

必要条件

動作環境

- **ホストシステム:** インテル® System Studio は開発システムとして Windows*、macOS*、または Linux* をサポートします。このガイドでは、Linux* システム向けの手順を説明します。
- **ターゲットシステム:** Linux*。このガイドの手順は、Ubuntu* 16.04 ターゲットシステムを使用して検証されました。手順では、Linux* システムのファイルとフォルダーパスを示します。

注
Windows* システムでは、ファイルとフォルダーパスをインテル® System Studio と OpenVINO™ ツールキットがインストールされている場所に変更してください。

ホストシステムへのインテル® System Studio のインストール

最初に、インテル® System Studio 2019 Update 1 以降をホストシステムにインストールする必要があります。

- インストール・パッケージを選択する前に、製品を登録し、アカウントを作成して、サインインします。
- このガイドの手順に従うには、Linux* ホストと Linux* ターゲットを選択します。
- ここ (英語) をクリックして開始します。

インテル® System Studio での新しいプロジェクトの作成とビルド

このセクションでは、インテル® System Studio の Eclipse* IDE で利用可能な OpenVINO™ ツールキットのサンプル・プロジェクトを基に新しいプロジェクトを作成してビルドする方法を示します。

最初に、ベース Docker* イメージとカスタム Docker* イメージを作成します。次に、サンプルを選択して、プロジェクトをビルドします。

ステップ 1: インテル® System Studio の起動

インテル® System Studio を起動するには、次の操作を行います。

1. 以下のスクリプトを実行します。

   ```
   /opt/intel/system_studio_2019/iss_ide_eclipse-launcher.sh
   ```

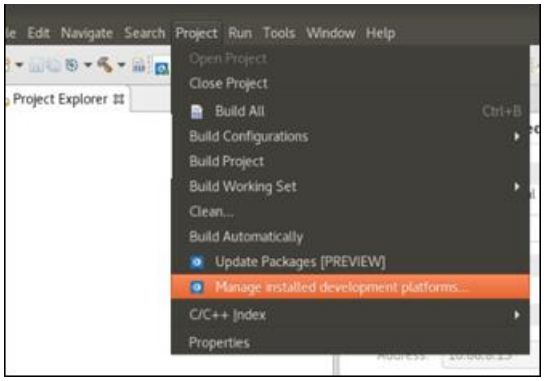
2. インストールで問題が見つかった場合は、インテル® System Studio コミュニティー・フォーラム (英語) やオンライン・サービス・センター (英語) を確認してください。

ステップ 2: ベース Docker* イメージの作成

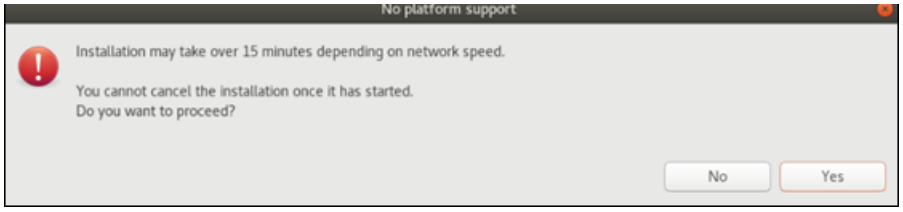
インテル® System Studio の Eclipse* IDE で OpenVINO™ ツールキットのサンプル・アプリケーションを実行またはデバッグする前に、ベース Docker* イメージとカスタム Docker* イメージを作成する必要があります。Docker* イメージは、画面上では「プラットフォーム」と表記される場合があります。

注
このステップは初回のみ実行します。実行済みの場合は、「サンプルの選択とプロジェクトのビルド」へ進んでください。

1. **[Project (プロジェクト)]** をクリックして、**[Manage installed development platforms (インストールされている開発プラットフォームの管理)]** を選択します。



2. デフォルトの Ubuntu* プラットフォーム (例: **iss-ubuntu-16.04:v14**) の横のチェックボックスをオンにして、**[Start (開始)]** をクリックします。
3. プラットフォームのインストールを確認するメッセージが表示されたら **[Yes (はい)]** をクリックします。



ステップ 3: カスタム Docker* イメージの作成

ベース Docker* イメージを作成したら、次にカスタム Docker* イメージを作成します。

注
このステップは初回のみ実行します。実行済みの場合は、「サンプルの選択とプロジェクトのビルド」へ進んでください。

カスタマイズされた Docker* ファイルのダウンロード

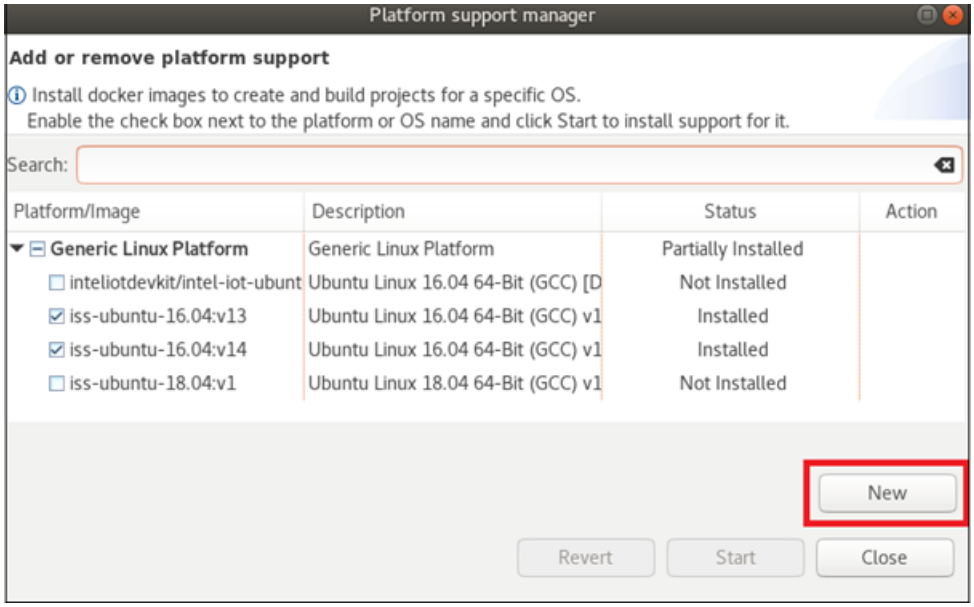
インテルでは、カスタム Docker* イメージの作成に使用可能な Docker* ファイルを提供しています。

1. 次の URL から Docker* ファイルをダウンロードします:
   https://software.intel.com/en-us/download/intel-system-studio-and-openvino-docker-file-for-r5
   https://software.intel.com/en-us/download/intel-system-studio-and-openvino-docker-file-latest-release.
2. アーカイブからホストシステム上の場所に Docker* ファイルを展開します。

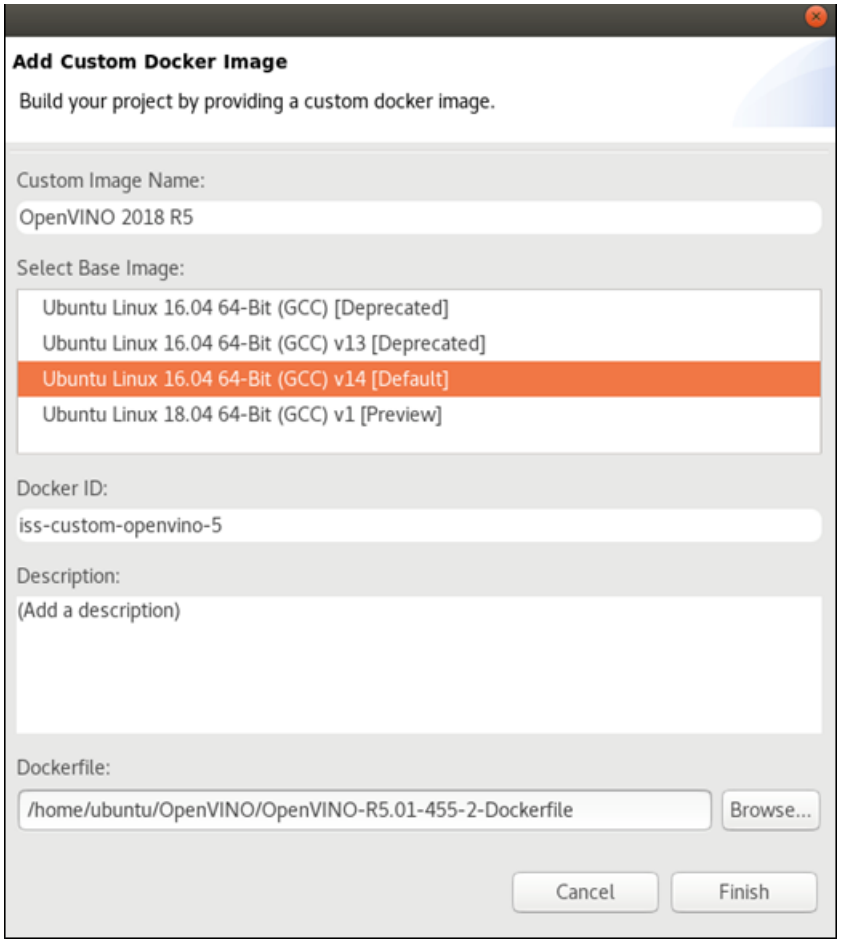
インテル® System Studio へのカスタム Docker* イメージの追加

インテル® System Studio の Eclipse* IDE で [Platform Support Manager (プラットフォーム・サポート・マネージャー)] を使用して、新しいカスタム Docker* イメージをインテル® System Studio に追加します。

1. IDE のツールバーから **[Project (プロジェクト)]** > **[Manage installed development platforms (インストールされている開発プラットフォームの管理)]** を選択して、[Platform Support Manager (プラットフォーム・サポート・マネージャー)] を開きます。



2. プラットフォーム･リストの下にある **[New (新規)]** ボタンをクリックして、**[Add Custom Docker Image (カスタム Docker* イメージの追加)]** ダイアログを開きます。

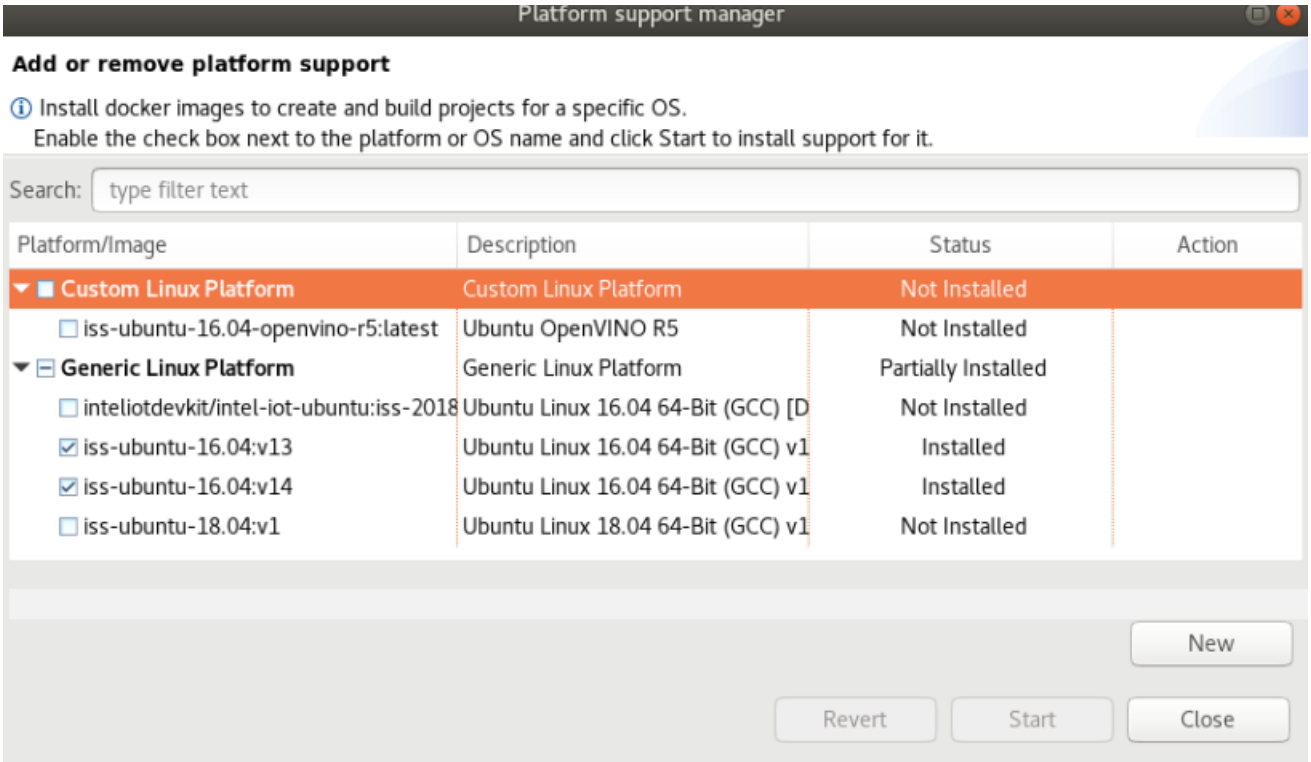


3. 次の情報を入力します。
    a. **[Custom Image Name (カスタムイメージ名)]:** カスタムイメージの名前を入力します。例: **OpenVINO 2018 R5**。

b. **[Select Base Image (ベースイメージの選択)]: Ubuntu* Linux* 16.04 64 bit (GCC) v14** 以降を選択します。
c. **[Docker ID]:** ID を入力します。すべて小文字でスペースを含めることはできません。例: **iss-custom-openvino-5**。ID がダッシュ (-) で終わっていないことを確認してください。これはエラーになります。
d. **[Description (説明)]:** 説明を入力します。このフィールドは空にできません。
e. **[Dockerfile]: [Browse (参照)]** をクリックして、ステップ 1 でダウンロードしたカスタム Docker* ファイルを選択します。

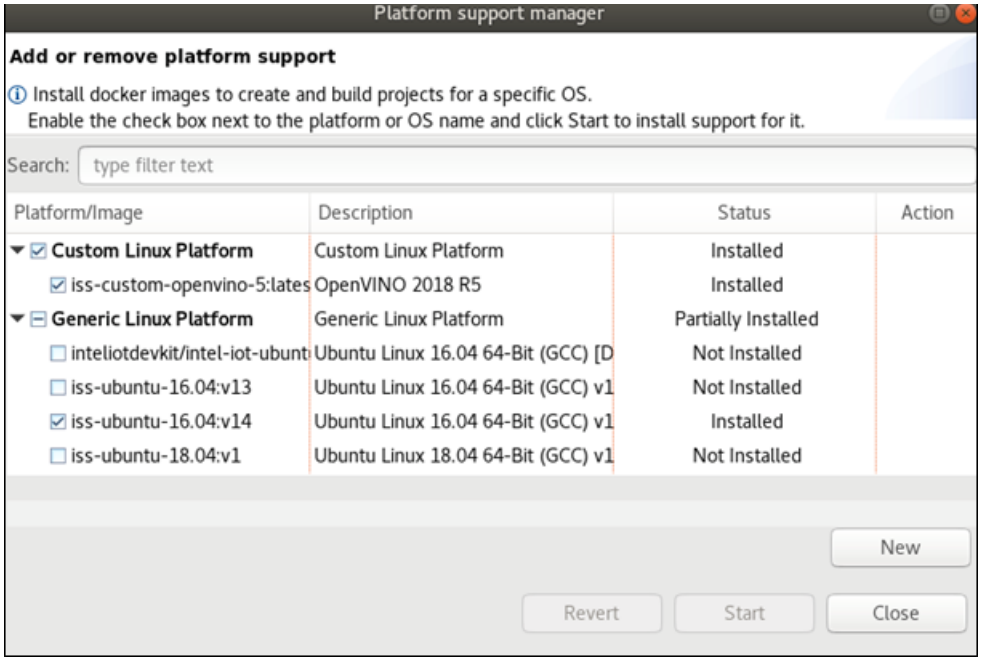
4. **[Finish (完了)]** をクリックします。

新しいカスタム Docker* イメージが **[Custom Linux Platform (カスタム Linux* プラットフォーム)]** 以下にリストされます。**[Status (ステータス)]** 列には **[Not Installed (インストールされていません)]** と表示されます。



新しいカスタム Docker* イメージのビルド

1. [Platform Support Manager (プラットフォーム・サポート・マネージャー)] ダイアログの **[Custom Linux Platform (カスタム Linux* プラットフォーム)]** 以下で、追加したカスタム Docker* イメージの横のチェックボックスをオンにします。



2. **[Start (スタート)]** をクリックします。
3. インストールに 15 分以上かかることを示すメッセージが表示されたら、**[Yes (はい)]** をクリックして続行します。

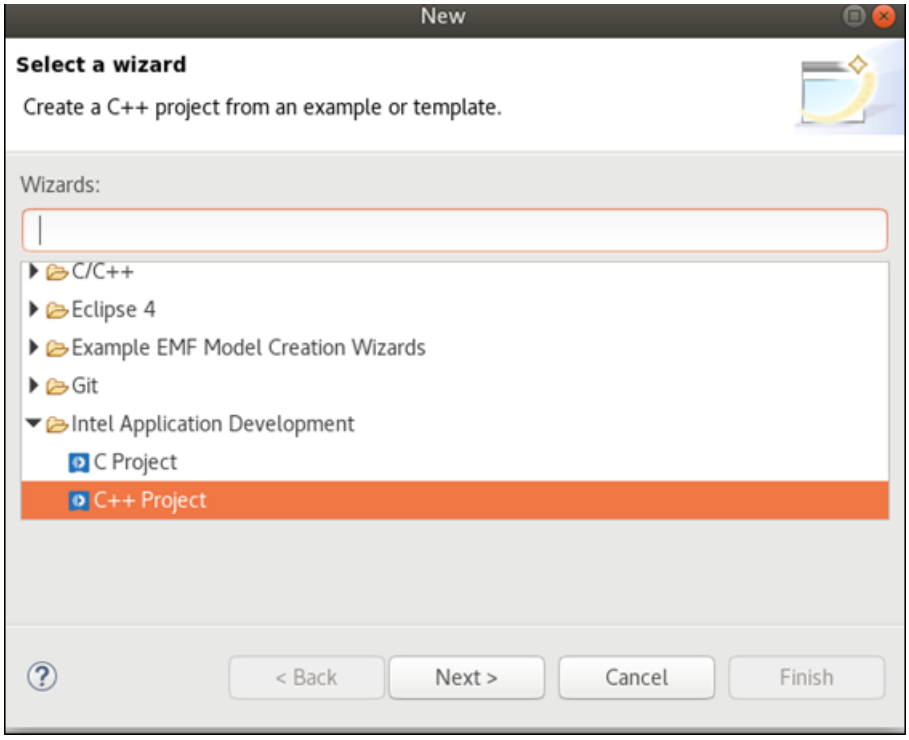
   プロセスの完了にかかる時間は状況によって異なり、途中コンソールに表示される進行状況バーとステータスメッセージが一定期間変わらないことがあります。

4. プロセスの完了を示す次のメッセージがコンソールに表示されるまで待ちます。
5. `Successfully built 123xyz`
   `Successfully tagged iss-ubuntu-16.04-some-id:latest`

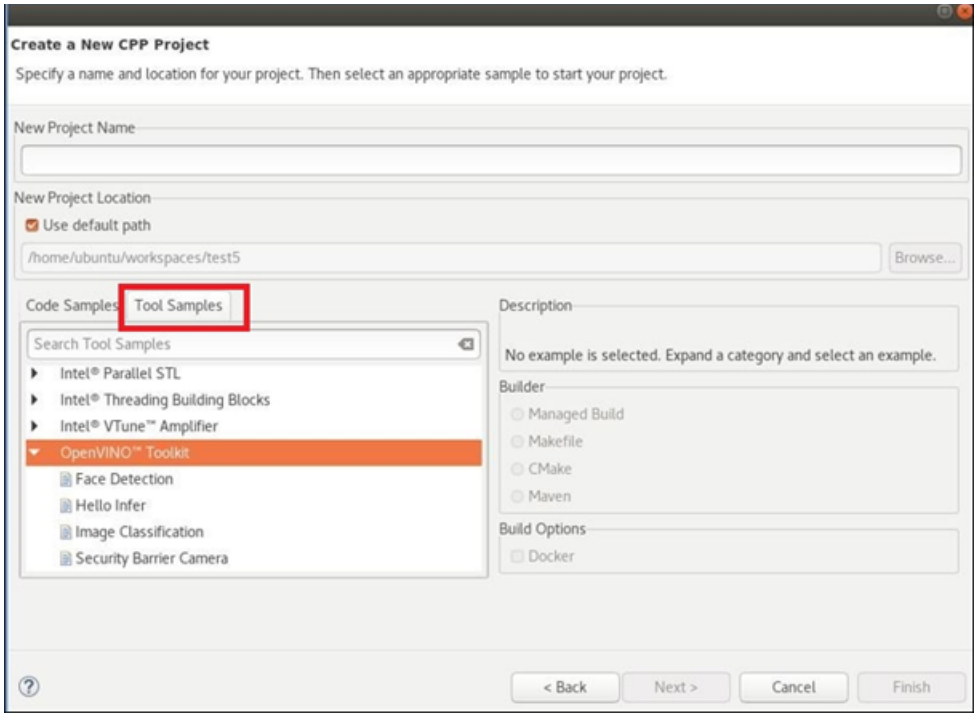
6. **[Close (閉じる)]** をクリックします。

ステップ 4: サンプルの選択とプロジェクトのビルド

1. メニューから **[File (ファイル)] > New (新規) > Project (プロジェクト)]** を選択して、新規プロジェクト・ウィザードを開始します。
2. **[Intel Application Development (インテル・アプリケーション開発)]** を展開して、**[C++ Project (C++ プロジェクト)]** を選択します。**[Next (次へ)]** をクリックします。



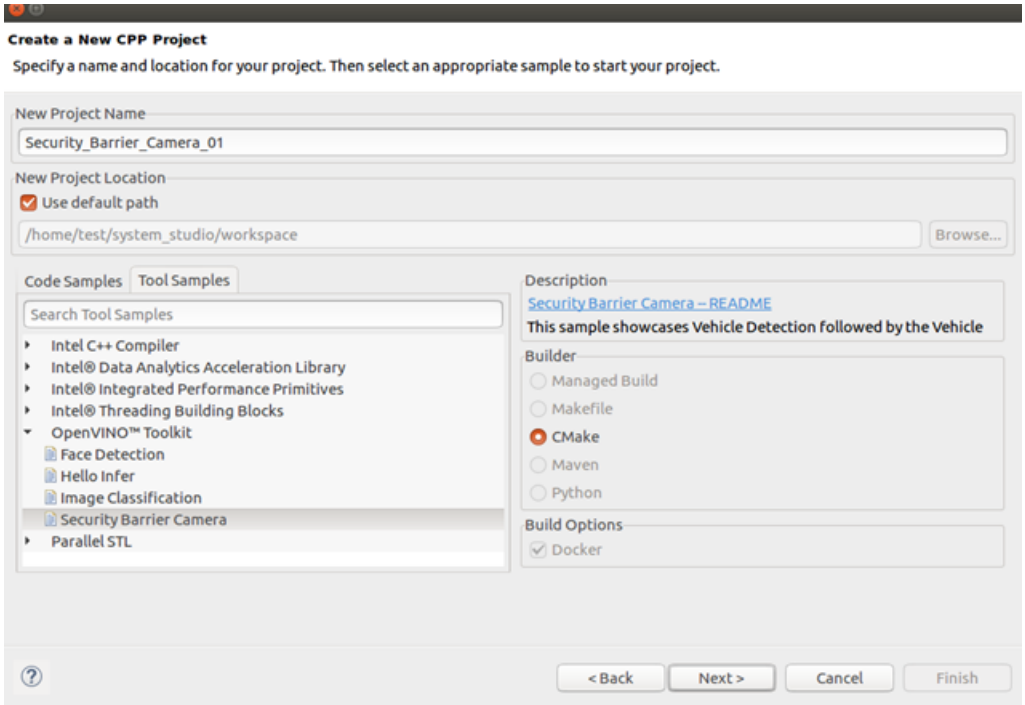
3. [Create a New CPP Project (新規 CPP プロジェクトの作成)] 画面で、**[Tool Samples (ツールサンプル)]** タブを開きます。



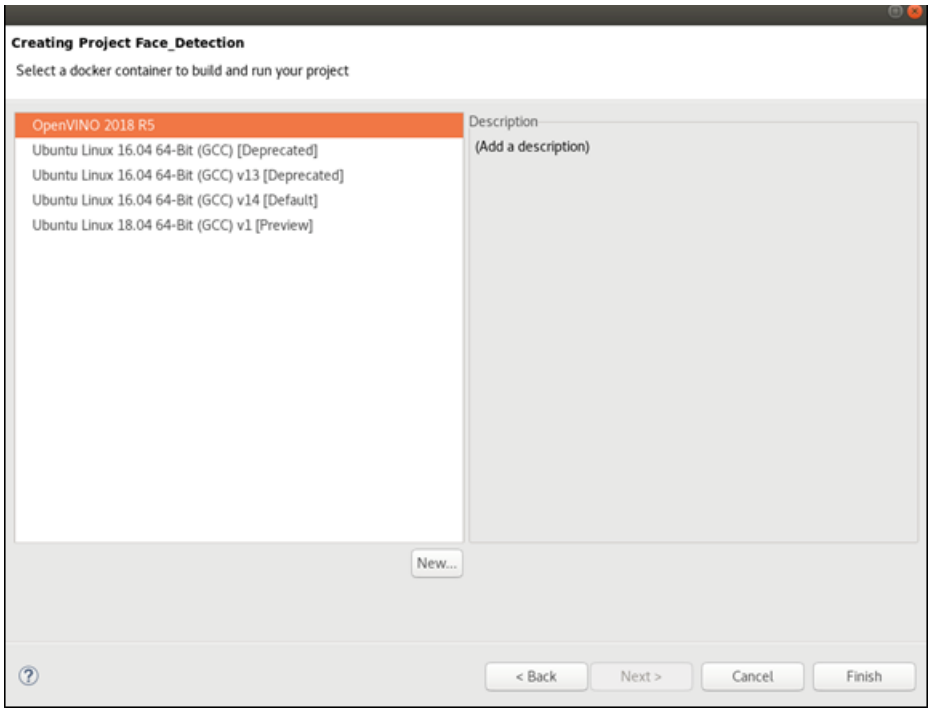
4. **[OpenVINO™ Toolkit (OpenVINO™ ツールキット)]** を展開して、サンプルを選択します。以下のスクリーンショットでは、**[Security Barrier Camera (セキュリティー・バリア・カメラ)]** サンプルが選択されています。**[Image Classification (画像分類)]** など、その他のサンプルも利用できます。

[New Project Name (新規プロジェクト名)] フィールドにサンプル･プロジェクト名が表示されます。必要に応じて変更します。

5. [Builder (ビルダー)] と [Build Options (ビルドオプション)] は、デフォルトの **[CMake]** と **[Docker]** のままにします。



6. **[Next (次へ)]** をクリックします。
7. 「カスタム Docker* イメージの作成」セクションで作成したカスタム Docker* イメージを選択します。



8. **[Finish (完了)]** をクリックしてビルドプロセスを開始します。

CMake はプロジェクトに必要な make ファイルを作成して、ビルドを開始します。ワークスペースの下部にある [CMakeconsole] と [CDT buildconsole] でビルドの進行状況を確認できます。

ヒント
コンソールを切り替えるには、**[Display Selected Console (選択したコンソールの表示)]** の横の下矢印をクリックして、ドロップダウン・リストからコンソールを選択します。

OpenVINO™ ターゲットシステムの準備

このセクションでは、ターゲットシステム上で実行またはデバッグするための準備について説明します。2 つのシナリオが考えられます。

- **ローカル開発:** ホストとターゲットシステムが同じです。インテル® System Studio と OpenVINO™ ツールキットは同じマシンにインストールされています。
- **リモート開発:** 別のターゲットシステムで実行してデバッグします。インテル® System Studio はホストシステムにインストールされており、OpenVINO™ ツールキットはターゲットシステムにインストールされています。ターゲットシステムは、ホストシステムからネットワークを介してアクセス可能でなければなりません。

ステップ 1: ターゲットシステムへの OpenVINO™ ツールキットのインストール

最初に、OpenVINO™ ツールキットをターゲットシステムにインストールする必要があります。これにより、OpenVINO™ ツールキットの共有ライブラリーがインストールされます。

- インストール・パッケージを選択する前に、製品を登録し、アカウントを作成して、サインインします。
- このガイドの手順に従うには、Linux* ホストと Linux* ターゲットを選択します。
- ここ (英語) をクリックして開始します。

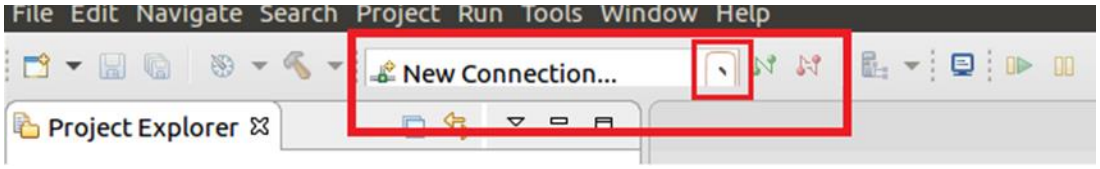
ステップ 2: ターゲットシステムへの接続の作成

注意
Eclipse* Target Communication Framework (TCF) を使用してターゲットに接続する場合、インテル® System Studio はローカル・ネットワーク上の任意のコンピューターからの接続をリスニングする TCF エージェントをターゲットにコピーします。IDE は認証資格情報を要求しますが、TCF エージェントには認証がありません。そのため、すべてのユーザーが接続して、エージェントを使用してターゲットデバイス上で任意のコードを実行できます。

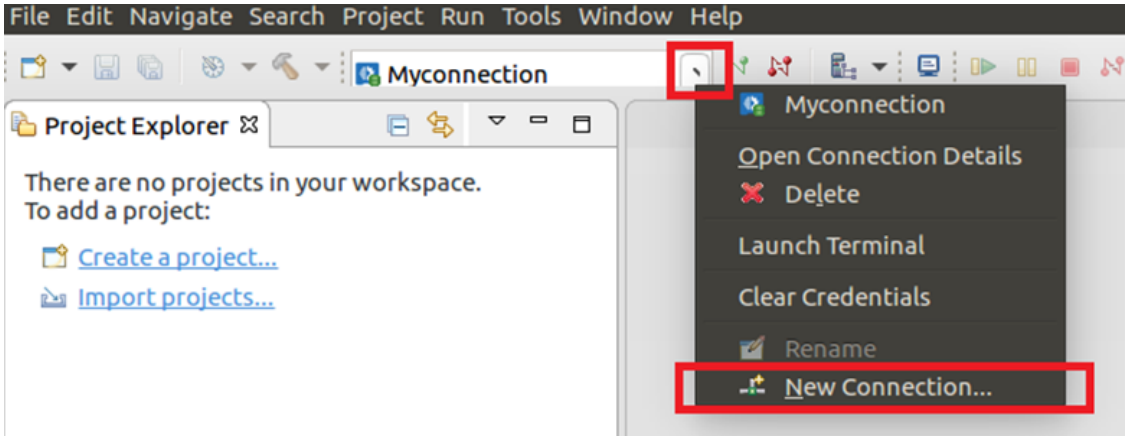
既存の接続を複数のプロジェクトで使用できます。

1. ターゲットデバイスが適切に機能しており、ホストシステムがネットワークを介してアクセスできることを確認します。
2. メインツールバーの [New Connection (新しい接続)] ボックス (既存の接続がない場合) または [Myconnection] ボックス (既存の接続がある場合) の横の下矢印をクリックして、ドロップダウン・リストから **[New Connection (新しい接続)]** を選択します。

   既存の接続がない場合:



既存の接続がある場合:



3. **[Connection for container based C/C++ applications or Java* applications (コンテナーベースの C/C++ アプリケーションまたは Java* アプリケーション向けの接続)]** を選択して、**[Next (次へ)]** をクリックします。
4. **[Connection Name (接続名)]** フィールドに名前を入力します。
5. **[Address (アドレス)]** フィールドにボード名または IP アドレスを入力して、**[Finish (完了)]** をクリックします。
6. アクティブな接続が切断されることを示す警告メッセージが表示されたら、**[Yes (はい)]** をクリックして続行します。
7. プロンプトが表示されたら、ターゲットデバイスにアクセスするための適切な資格情報を提供して、**[OK]** をクリックします。

ステップ 3: 資格情報の設定

ターゲットデバイスにアクセスするための資格情報を設定します。次のいずれかの認証方法を使用できます。

公開キーによるログイン

アプリケーションがハードウェア・センサーや LED にアクセスしたり、システム特権を必要とする場合は、より高いレベルのセキュリティーを確保するため公開キーに基づく認証を使用します。「SSH 接続: リモート Linux* ターゲットデバイスへのパスワードなしアクセス」(英語) の手順に従ってください。

パスワードによるログイン

アプリケーションがシステム特権を必要としない場合は、パスワードによるログインを使用できます。

注
ローカル開発 (ターゲットシステムとホストシステムが同じ) では、[New Connection (新しい接続)] ダイアログの [Address (アドレス)] フィールドにシステムの IP または localhost を指定できます。

セキュリティー・バリア・カメラのサンプル

このサンプルは、車両を検知して、車両属性とナンバープレート認識を適用します。

ここでは、サンプルの実行とデバッグの手順を説明します。

適切なトレーニング済みモデルの入手

セキュリティー・バリア・カメラのサンプルを使用するには、インテルが提供する適切な事前トレーニング済みモデルを入手する必要があります。事前トレーニング済みモデルは、インテル® デベロッパー・ゾーン から入手できます: https://software.intel.com/en-us/openvino-toolkit/documentation/pretrained-models (英語)。

ターゲットシステムに OpenVINO™ ツールキットがインストールされている場合は、ターゲットシステムで次のステップを実行します。そうでない場合は、ホストシステムでこのステップを実行して、出力ファイルをターゲットシステムのフォルダーにコピーします。

1. Python* 3 をインストールします。
2. ターミナルを開いて、必要なパッケージをインストールします。

   ```
   sudo -E pip3 install pyyaml requests
   ```

3. モデル・ダウンローダーを実行して、推論エンジン向けにモデルを準備します。手順は、インテル® デベロッパー・ゾーンから入手できます: https://software.intel.com/en-us/articles/model-downloader-essentials。

   ```
   python3
   'path_to_OpenVINO/deployment_tools/tools/model_downloader/downloader.py --
   name vehicle-license-plate-detection-barrier-0106,vehicle-attributes-
   recognition-barrier-0039,license-plate-recognition-barrier-0001,face-
   detection-adas-0001 -o /tmp/OpenVINO
   ```

   モデルは、/tmp/OpenVINO フォルダーにダウンロードされます。

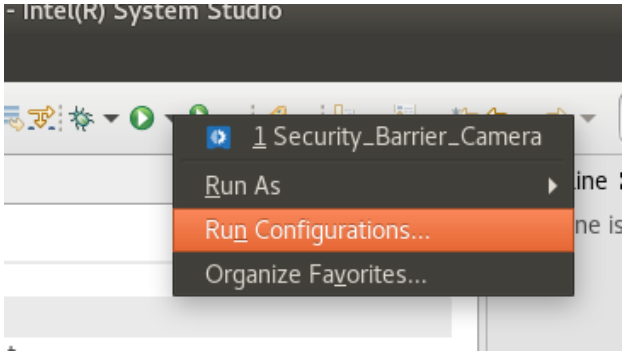
作成とビルド

「インテル® System Studio での新しいプロジェクトの作成とビルド」の手順を参照してください。

実行とデバッグ

ステップ 1: 起動設定の確認と更新

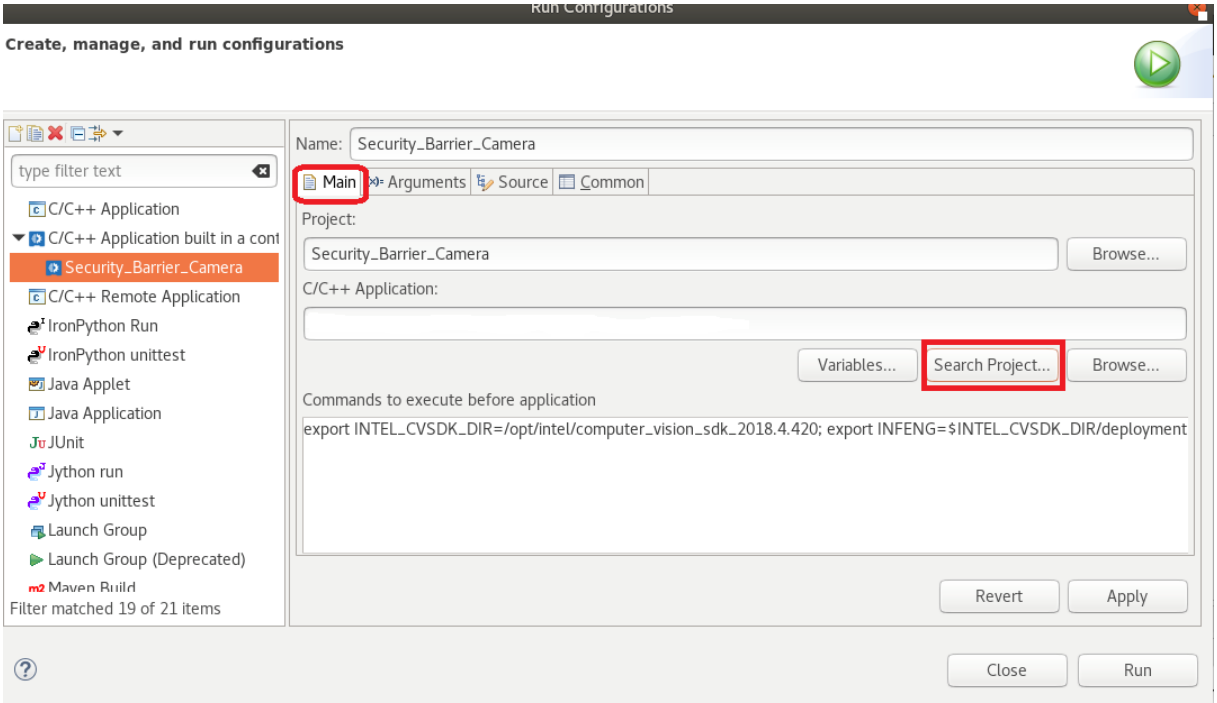
1. メインツールバーの **[Run (実行)]** ボタンの横の下矢印をクリックして、**[Run Configurations (実行設定)]** を選択します。



2. **[C/C++ Application built in a container and running on Linux (コンテナーでビルドされた Linux* で実行する C/C++ アプリケーション)]** をクリックして開きます。
3. **[Security_Barrier_Camera]** を選択します。設定名はプロジェクト名と対応しています。

ステップ 2: (オプション) 実行ファイルの確認

[Main (メイン)] タブの **[C/C++ Application (C/C++ アプリケーション)]** に **security_barrier_camera_demo** が表示されていることを確認します。



ステップ 3: (オプション) コマンドの確認

次のように、**[Main (メイン)]** タブの **[Commands to execute before application (アプリケーションの前に実行するコマンド)]** にターゲットシステムの適切なパスが設定されていることを確認します。

注
以下は、OpenVINO™ ツールキットが root 権限のユーザーのデフォルトの場所にインストールされていることを前提としています。OpenVINO™ ツールキットを異なるフォルダーにインストールした場合は、最初の行の OpenVINO™ ツールキットのインストール・ディレクトリーを変更してください。

```
export OPENVINO_DIR=/opt/intel/openvino; export
INFENG=$OPENVINO_DIR/deployment_tools/inference_engine; export
IE_PLUGINS_PATH=$INFENG/lib/intel64; [ ! -d /tmp/OpenVINO ] && mkdir
/tmp/OpenVINO; cp $INFENG/lib/intel64/libcpu_extension_sse4.so
/tmp/OpenVINO/libcpu_extension.so; export
LD_LIBRARY_PATH=/tmp/OpenVINO:$OPENVINO_DIR/opencv/lib:/opt/intel/opencl:$INFENG/
external/gna/lib:$INFENG/external/mkltiny_lnx/lib:$INFENG/external/tbb/lib:$INFEN
G/lib/intel64:$LD_LIBRARY_PATH
```

ヒント
プロセッサーがインテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) 命令をサポートする場合、**libcpu_extension** の最適化バージョンを使用できます。

1. 上記のコマンドの次の部分を変更します。

   ```
   cp $INFENG/lib/ubuntu_16.04/intel64/libcpu_extension_sse4.so
   /tmp/OpenVINO/libcpu_extension.so;
   ```

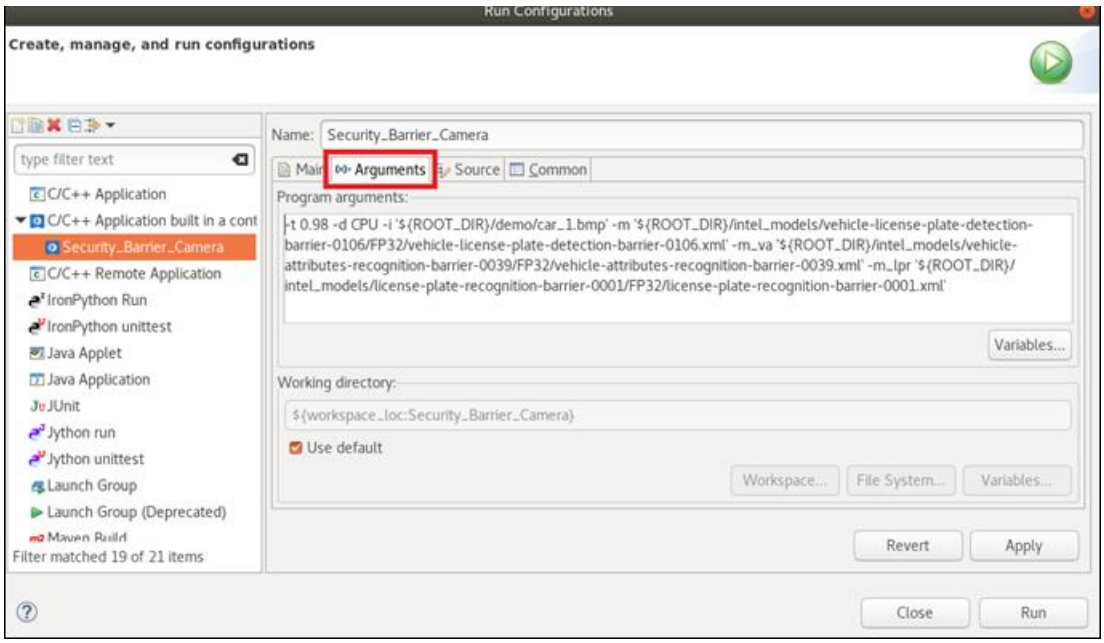
   変更後:

   ```
   cp $INFENG/lib/ubuntu_16.04/intel64/libcpu_extension_avx2.so
   /tmp/OpenVINO/libcpu_extension.so;
   ```

2. [Project Properties (プロジェクト・プロパティー)] を開きます。
3. **[C/C++ Build (C/C++ ビルド)]** > **[CMake]** > **[Host OS Overrides (ホスト OS のオーバーライド)]** で `ENABLE_SSE42` プロパティーを選択して削除します。
4. インテル® AVX 命令が使用されるようにプロジェクトをリビルドします。

ステップ 4: 引数の確認

1. **[Arguments (引数)]** タブを選択します。
2. **[Program arguments (プログラム引数)]** が適切に設定されていることを確認します。必要に応じて変更します。



3. (オプション) 単一の画像の代わりにビデオを実行します。**car-detection.mp4** ビデオを使用するには、次の操作を行います。
    a. GitHub* からビデオを入手します: intel-iot-devkit/sample-videos (英語)。
    b. ビデオをターゲットシステム上のフォルダーにダウンロードします。
    c. 上記のステップ 3 の引数でビデオファイルへのパスを指定します。
4. **[Apply (適用)]** をクリックしてから **[Close (閉じる)]** をクリックして [Run Configuration (実行設定)] ウィンドウを閉じます。

ステップ 5: 実行とデバッグ

- **実行**: **[Run (実行)]** ボタンをクリックします。
- **デバッグ**: **[Debug (デバッグ)]** ボタンをクリックします。

注
最近の起動がないことを示すエラーメッセージが表示された場合は、**[Run (実行)]** > **[Run Configuration (実行設定)]** から **[Launch (起動)]** を選択します。

結果

アプリケーションはいくつかのステップを経てから推論を開始します。ターゲットシステムのディスプレイでウィンドウを開いて、出力画像を表示しようとします。

アプリケーションのグラフィカル表示を開く

結果に次のエラーが出力されることがあります: **Gtk-WARNING**:cannot open display (Gtk-WARNING**: 表示を開くことができません)**。

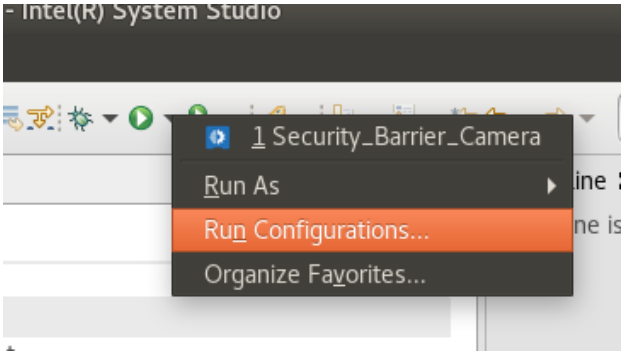
この問題を解決する推奨方法は、モニターをターゲットシステムに接続するか、リモート・デスクトップ・クライアントを使用してターゲットにリモートログインすることです。ターゲットシステムのデスクトップが表示されたら、次の操作を行います。

1. ホストシステムでターミナルを開きます。
2. 次のコマンドで DISPLAY の定義を確認します。
    - `ssh username@IP_address`
    - `echo $DISPLAY`

    次のような結果が表示されます: **:0**。

    結果が表示されない場合は、ターゲットにリモートログインして、リモート・デスクトップ・クライアントを介してターゲットのデスクトップを表示できることを確認します。上記のコマンドを再度実行して結果を取得します。

3. 結果を記録して、ターミナルは開いたままにします。
4. 起動設定を更新するため、次の操作を行います。
    a. メインツールバーの **[Run (実行)]** ボタンの横の下矢印をクリックして、**[Run Configurations (実行設定)]** を選択します。



   b. **[C/C++ Application built in a container and running on Linux (コンテナーでビルドされた Linux* で実行する C/C++ アプリケーション)]** をクリックして開きます。
   c. **[Security_Barrier_Camera]** を選択します。

5. [Main (メイン)] タブの **[Commands to execute before application (アプリケーションの前に実行するコマンド)]** に `export DISPLAY=:0;` を追加します。ここで、:0 は先程ターミナルに表示された結果です。コマンドはセミコロン (;) で区切ります。
6. プロジェクトを再度実行します。結果に **Gtk-WARNING**:cannot open display (Gtk-WARNING**: 表示を開くことができません)** エラーが表示されなくなります。ターゲットのディスプレイ (モニターまたはリモート・デスクトップ接続) に画像検出結果が表示されます。

注
ターゲットシステムがホストシステムと同じ場合も、これらのステップを実行する必要があります。その場合、リモートシステムに SSH 接続する代わりに、ターミナルを開いて、上記のステップ 2 で示したように `echo $DISPLAY` と入力します。

インタラクティブ顔検出のデモ

このデモは、一連のニューラル・ネットワークを使用して、顔認識にオブジェクト検出タスクを適用します。

非同期 API は、アプリケーションの全体的なフレームレートを向上できます。推論の完了を待機するのではなく、アプリケーションは、アクセラレーターがビジーな間、ホストで処理を続行できます。

このデモは、年齢/性別認識、頭部姿勢推定、感情認識、顔のランドマーク推定の 4 つの並列推論要求を同時に実行します。

このデモのほとんどのステップは、前出の「セキュリティー・バリア・カメラのサンプル」と同じです。

ステップ	説明
作成とビルド	「セキュリティー・バリア・カメラのサンプル」のステップに従います。
実行とデバッグ	**security_barrier_camera_demo** の代わりに **interactive_face_detection_demo** を選択することを除いて、「セキュリティー・バリア・カメラのサンプル」のステップと同じです。

画像分類のサンプル

このサンプル・アプリケーションは、AlexNet や GoogLeNet などの画像分類ネットワークを使用して推論を実行します。サンプル・アプリケーションはコマンドライン・パラメーターを読み取り、ネットワークとイメージを推論エンジンプラグインにロードします。推論が完了すると、アプリケーションは出力イメージを作成して、データを標準出力ストリームに出力します。

このセクションでは、画像分類アプリケーションを準備して実行する方法を示します。

適切なトレーニング済みモデルの入手

このサンプルを使用するには、適切なトレーニング済みモデルを入手する必要があります。多くの画像分類用のトレーニング済みモデルがあります。このサンプルでは、SqueezeNet v1.1 (英語) を使用します。

注
ターゲットシステムに OpenVINO™ ツールキットがインストールされている場合は、ターゲットシステムで次のステップを実行します。ターゲットシステムに OpenVINO™ ツールキットがインストールされている場合は、ターゲットシステムで次のステップを実行します。

このサンプルを実行すると、サンプルはモデルを実行して画像を分類し、(Imagenet 1000 の) 上位 5 つの一致ラベルとその確率を出力します。ラベルに使用されているテキストは次のサイトで確認できます: https://gist.github.com/maraoz/388eddec39d60c6d52d4 (英語)。

1. 次のファイルを squeezenet model (英語) からダウンロードします。
   - `deploy.prototxt` (html バージョンではなく、Raw バージョンをダウンロードします)
   - `squeezenet_v1.1.caffemodel`
2. OpenVINO™ ツールキットのモデル・オプティマイザー (英語) を実行して、推論エンジン向けにモデルを準備します。

   ```
   python3 'path_to_OpenVINO/deployment_tools/model_optimizer/mo.py' --
   input_model
   'path_to_downloaded_squeezenet/squeezenet/1.1/caffe/squeezenet1.1.caffemode
   l' --input_proto 'path_to_squeezenet/squeezenet/1.1/caffe/deploy.prototxt'
   -o <path to an optimized model output folder>
   ```

   ここで、*optimized model output folder* は、システム上の選択したフォルダーです。

作成とビルド

「インテル® System Studio での新しいプロジェクトの作成とビルド」の手順を参照してください。

実行とデバッグ

ステップ 1: ターゲットへの共有ライブラリーのコピー

OpenVINO™ ツールキットのサンプルをビルドすると、実行ファイルとともにサンプルの実行に必要ないくつかの共有ライブラリーが生成されます。これらのライブラリーは、Docker* コンテナー内で作成されるため、ターゲットシステムにコピーする必要があります。

1. ホストシステムで実行しているコンテナーをリストするには、ターミナルで `'docker ps'` と入力します。
2. 表示されたリストで「カスタム Docker* イメージの作成」で作成したカスタムイメージに対応するコンテナーを見つけます。
3. コンテナー ID の最初の数文字を記録します。

   ヒント
   カスタムイメージに対応する Docker* コンテナーが見つからない場合は、次の操作を行います。

   - IDE で [Build (ビルド)] アイコンをクリックして別のプロジェクトのビルドを開始します。
   - CDT Build コンソールで次のようなコマンドを見つけます: `'docker exec -i xyz_id /bin/bash ...'`。
4. コンテナー ID をコマンドにコピーします。
5. 次のフォルダーをホストシステム上に作成します: `/tmp/OpenVINO`。
6. ターミナルで次のコマンドを入力します。

```
docker cp
xyz_id:/workspace/Image_Classification/build/Debug/intel64/Debug/lib/libfor
mat_reader.so /tmp/OpenVINO
```

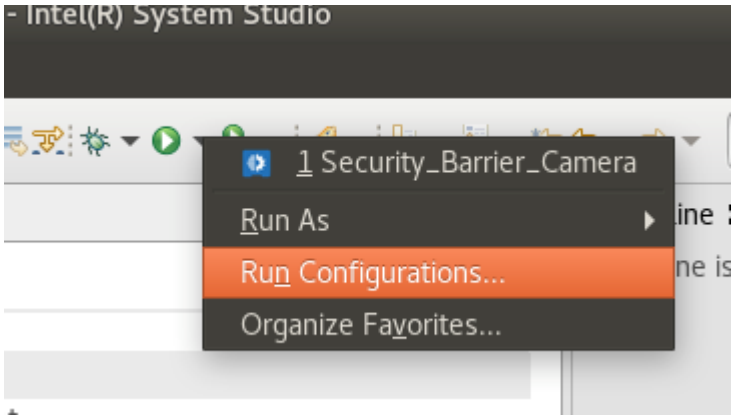
   注
   上記のパスで *xyz_id* を実行中のコンテナーの ID に置き換え、実際のプロジェクト名を使用します。

7. プロジェクトをリモートで実行またはデバッグする場合は、次のコマンドを使用してファイルをローカルフォルダーからホストシステム上のフォルダーへコピーします (上記のステップ 6 を参照)。

```
scp /tmp/OpenVINO/libformat_reader.so user@targetip:/tmp/OpenVINO
```

ステップ 2: 起動設定の確認と更新

1. メインツールバーの **[Run (実行)]** ボタンの横の下矢印をクリックして、**[Run Configurations (実行設定)]** を選択します。



2. **[C/C++ Application built in a container and running on Linux (コンテナーでビルドされた Linux* で実行する C/C++ アプリケーション)]** をクリックして開きます。
3. **classification_sample** を選択します。

4. **[Main (メイン)]** タブの **[C/C++ Application (C/C++ アプリケーション)]** で **[Search Project (プロジェクトの検索)]** をクリックします。

ステップ 3: (オプション) 実行ファイルの確認

[Main (メイン)] タブの **[C/C++ Application (C/C++ アプリケーション)]** に **classification_sample** が表示されていることを確認します。

ステップ 4: (オプション) コマンドの確認

次のように、**[Main (メイン)]** タブの **[Commands to execute before application (アプリケーションの前に実行するコマンド)]** にターゲットシステムの適切なパスが設定されていることを確認します。

注
以下は、OpenVINO™ ツールキットが root 権限のユーザーのデフォルトの場所にインストールされていることを前提としています。OpenVINO™ ツールキットを異なるフォルダーにインストールした場合は、最初の行の OpenVINO™ ツールキットのインストール・ディレクトリーを変更してください。

```
export OPENVINO_DIR=/opt/intel/openvino; export
INFENG=$OPENVINO_DIR/deployment_tools/inference_engine; export
IE_PLUGINS_PATH=$INFENG/lib/intel64; [ ! -d /tmp/OpenVINO ] && mkdir
/tmp/OpenVINO; cp $INFENG/lib/intel64/libcpu_extension_sse4.so
/tmp/OpenVINO/libcpu_extension.so; export
LD_LIBRARY_PATH=/tmp/OpenVINO:$OPENVINO_DIR/opencv/lib:/opt/intel/opencl:$INFENG/
external/gna/lib:$INFENG/external/mkltiny_lnx/lib:$INFENG/external/tbb/lib:$INFEN
G/lib/intel64:$LD_LIBRARY_PATH
```

ステップ 5: 引数の確認

1. **[Arguments (引数)]** タブを選択します。
2. [Arguments (引数)] が適切に設定されていることを確認します。次の例は、ターゲットシステムの指定した場所に **horse.jpg** という名前の画像があると仮定します。任意の画像を使用できますが、ファイル名はこの設定と一致していなければなりません。

   ```
   -d CPU -i /home/ubuntu/OpenVINO/horse.jpg -m
   /home/ubuntu/OpenVINO/Squeezenet/output/squeezenet_v1.1.xml -nt 5
   ```

3. 必要に応じて引数を変更します。
4. **[Apply (適用)]** をクリックしてから **[Close (閉じる)]** をクリックして [Run Configuration (実行設定)] ウィンドウを閉じます。

ステップ 6: 実行とデバッグ

- **実行**: **[Run (実行)]** ボタンをクリックします。
- **デバッグ**: **[Debug (デバッグ)]** ボタンをクリックします。

結果

サンプルはモデルを実行して画像を分類し、(Imagenet 1000 (英語) の) 上位 5 つの一致ラベルとその確率を出力します。ラベルに使用されているテキストは、こちら (英語) を参照してください。

Hello 推論のサンプル

このセクションでは、アプリケーションで推論エンジンの新しい推論要求 API を使用する方法を示します。詳細は、「推論エンジンをアプリケーションに統合する方法」 (英語) を参照してください。このサンプル・アプリケーションは、前出の「画像分類のサンプル」の簡易版です。以下に示すとおり、ほとんどのステップは「画像分類のサンプル」と同じです。

ステップ	説明
適切なトレーニング済みモデルの入手 作成とビルド	「画像分類のサンプル」のステップに従います。
実行とデバッグ	**classification_sample** の代わりに **hello_infer** を選択することを除いて、「画像分類のサンプル」のステップと同じです。

その他のサンプル

その他の OpenVINO™ ツールキットのサンプルの詳細およびプログラム引数と説明の一覧は、https://software.intel.com/en-us/articles/OpenVINO-IE-Samples (英語) を参照してください。

トラブルシューティング

問題	解決方法
実行またはデバッグを開始した直後にアプリケーションが終了する	必要な共有ライブラリーが不足している可能性がありますが、ターミナルデータが上書きされるため特定が困難です。 この問題を回避するには、ターミナルで問題を表示できるように、**[Run (実行)]** の代わりに **[Debug (デバッグ)]** を選択します。
リモートデバッグ中ターミナルに **cannot run gdbserver (gdbserver を実行できません)** メッセージが表示される	[Launch Configuration (起動設定)] を開いて、[Commands to execute before application (アプリケーションの前に実行するコマンド)] を確認します。コマンドリストにアプリケーションの実行を妨げるエラーがある可能性があります。
プロジェクトの実行時に「Illegal instruction (不正な命令)」などのメッセージが表示される これは、ホストとターゲットシステムのプロセッサー機能が異なることを示しています。 例えば、一方はインテル® ストリーミング SIMD 拡張命令 (インテル® SSE) またはインテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) をサポートしていますが、もう一方はサポートしていない場合などです。	1. プロジェクトを右クリックして **[Properties (プロパティー)]** ウィンドウを開きます。 2. **[C/C++ Build (C/C++ ビルド)]** > **[CMake]** から **[Symbols (シンボル)]** タブを選択します。 3. **ENABLE_SSE42** という名前の **STRING** 型のシンボルを追加して、その値を **ON** に設定して、インテル® Celeron® プロセッサー、インテル® Pentium® プロセッサー、または Intel Atom® プロセッサー・ベースのターゲット向けにサンプルをビルドします。  SSE42 値を明示的に ON に設定すると、AVX2 と AVX512 拡張が無効になり、不正な命令が発生します。 OpenVINO™ ツールキットのサンプルをビルドする際のインテル® SSE とインテル® AVX サポートの切り替えに関する詳細は、`/opt/intel/computer_vision_sdk/inference_engine/src/extension/` にある *CMakeLists.txt* ファイルを参照してください。

問題	解決方法
プロジェクトを実行すると「GTK-WARNING**: cannot open display (GTK-WARNING**: 表示を開くことができません)」メッセージが表示され、出力が表示されません	アプリケーションのグラフィック出力を表示するには、DISPLAY 環境変数を定義して適切な値を設定する必要があります。次の操作を行います。 1. 出力を表示するには、モニターまたは VNC クライアントをターゲットシステムに接続します。 2. ターゲットシステムでターミナルを開いて次のコマンドを入力します: `echo $DISPLAY`。次のような出力が表示されます: **:0**。これを記録しておきます。 3. ホストシステムで **[Run (実行)]** > **[Run Configurations (実行設定)]** からサンプルの実行設定を選択します。 4. **[Main (メイン)]** タブの **[Commands to execute before application (アプリケーションの前に実行するコマンド)]** に次のコマンドを追加します。 5. `export DISPLAY=:0; export XAUTHORITY=/home/your_actual_username/.Xauthority;` 6. `DISPLAY=:` の値は、ターミナルで `echo $DISPLAY` を実行したときに表示される値 (上記のステップ 2) と同じでなければなりません。 ほとんどのサンプルでは、`-no_show` プログラム引数で出力を無効にできます。 サンプルの引数に関する追加の情報は、https://software.intel.com/en-us/articles/OpenVINO-IE-Samples (英語) を参照してください。
アプリケーションは正しくビルドされますが、インテル® System Studio IDE のエディターウィンドウでいくつかのインデックス・エラーが表示される	プロジェクト・プロパティーを次のように変更します。 1. **[Project (プロジェクト)]** > **[Properties (プロパティー)]** > **[C/C++ Build (C/C++ ビルド)]** > **[Settings (設定)]** を選択します。 2. **[GCC C++ compiler (GCC C++ コンパイラー)]** を展開して、**[Dialect (方言)]** を選択します。 3. **[Language standard (言語標準)]** ドロップダウン・リストから **[Language Standard (言語標準)]** > **[ISO C++ 11]** を選択します。 4. **[Apply (適用)]** をクリックします。 5. **[Includes (インクルード)]** を選択して、**[Include paths (インクルード・パス)]** エリアの **[Add (追加)]** ボタンをクリックします。 6. [Directory (ディレクトリー)] に `/opt/intel/computer_vision_sdk/inference_engine/include` と入力して、**[OK]** をクリックします。 7. **[OK]** をクリックしてから、**[Apply (適用)]** そして **[Close (閉じる)]** をクリックします。

著作権と商標について

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。