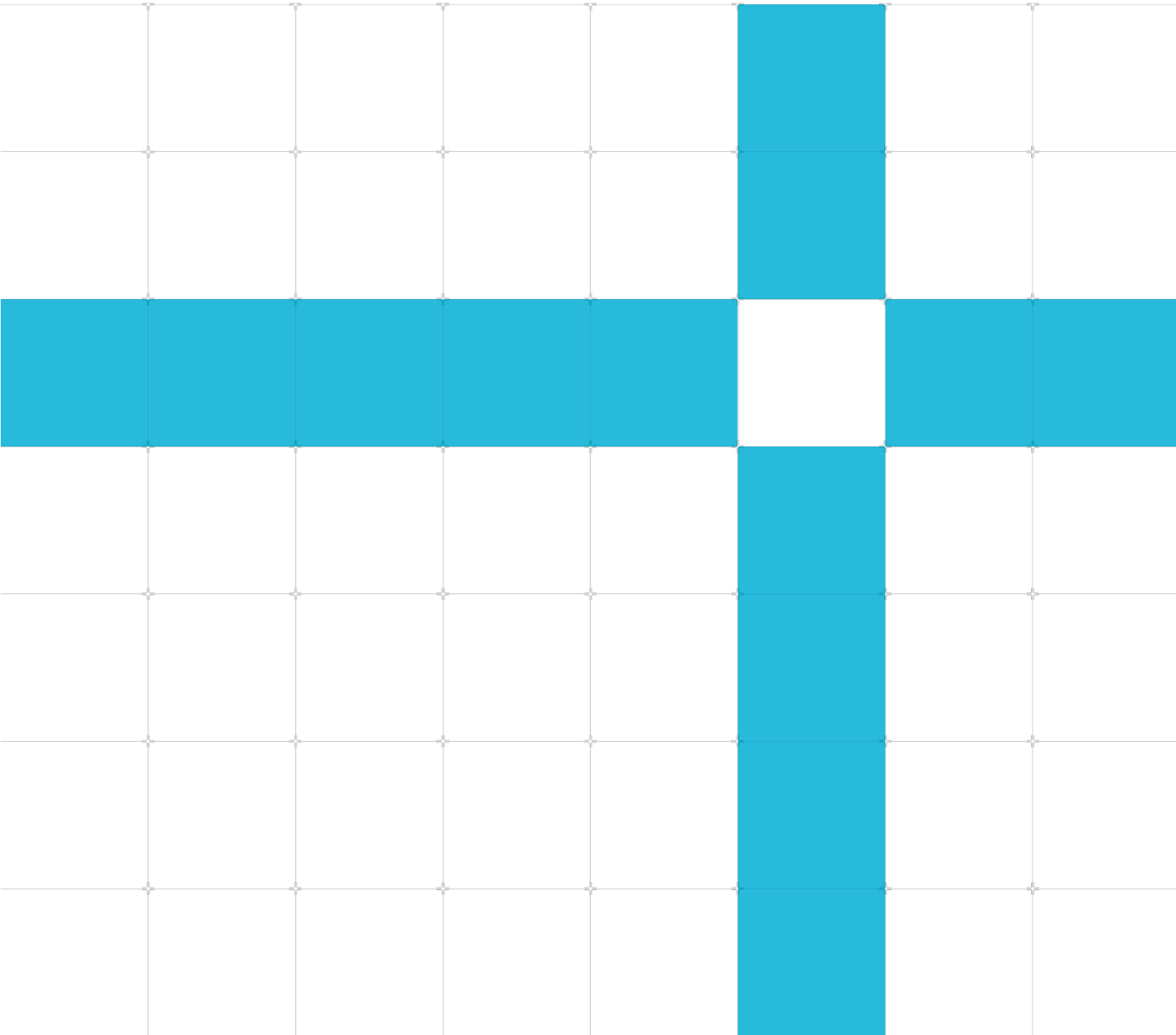
arm

# Arm NN accelerated object detection with Autoware autonomous driving stack

Non-Confidential



**Issue 0101**
102229



# Arm NN accelerated object detection with Autoware autonomous driving stack



### Release information

### Document history

| Issue | Date | Confidentiality | Change |
|---|---|---|---|
| 01 | 13 August 2020 | Non-Confidential | First release |

## Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

## Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

## Web Address

**www.arm.com**

# Contents

# 1 Overview

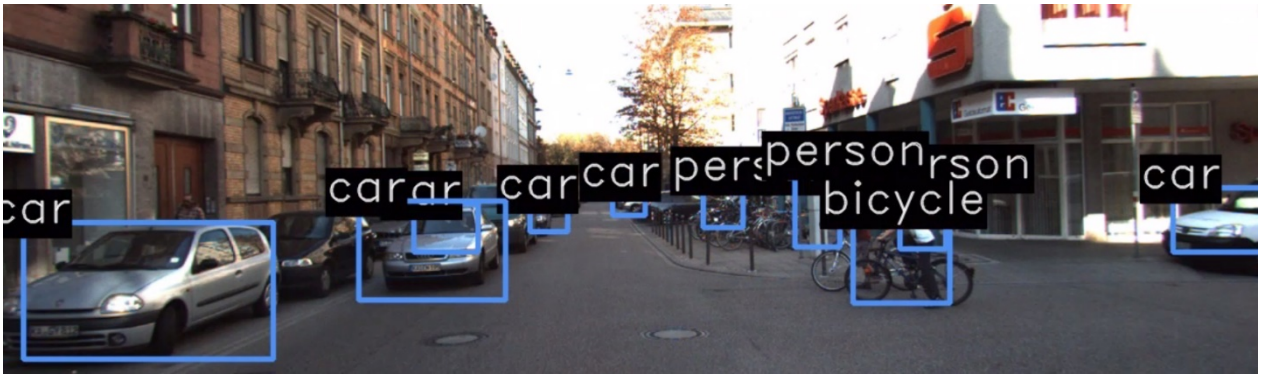
This guide shows you how to use Autoware.AI and Arm NN to build and run a real-time object detection system. Object detection is central to the development of automated vehicles. Quick and accurate detection and tracking of objects around an autonomous vehicle is crucial to operational safety.

The system runs on a HiKey 960 device using Arm Cortex-A73 and Cortex-A53 CPUs, and a Mali-G71 GPU.

This guide covers the following topics:

- How to build an object detection node for Autoware.AI
- How to run the object detection node on a HiKey 960 device
- How to use Arm NN to accelerate neural network performance

The following image shows the final object detection system that we will build:



# 2 Before you begin

To work through this guide, you will need two hardware devices and some installed software.

You will need these devices:

- A HiKey 960 development board

  The HiKey 960 runs the final demo, using Autoware and Arm NN to run a real-time object detection system.


Note

This guide has been developed and tested on the HiKey 960 device. If you choose to use another platform, you will need to modify the various scripts used in this tutorial.

- A Linux or Apple Mac host machine

  The host machine is used to prepare an Ubuntu filesystem and flash the base firmware and OS onto the Hikey960 device. The host machine is also used for some of the more computationally intensive tasks, like converting the neural network to the TensorFlow protobuf format.

Before starting the instructions in this guide, you must install Ubuntu Linux and build Arm NN on the HiKey 960. This installation process is described in our guide Running and profiling Arm NN on the HiKey 960 .

Specifically, you must complete the following steps in Running and profiling Arm NN on the HiKey 960:

- Before you begin
- Run Ubuntu Linux on the HiKey 960
- Build an Ubuntu filesystem
- Flash the base firmware and OS
- Flash the base firmware and OS - recovery mode
- Flash the base firmware and OS - fastboot mode
- Boot Linux
- Add more diskspace

You can stop after performing the Add more diskspace step, which includes building Arm NN.

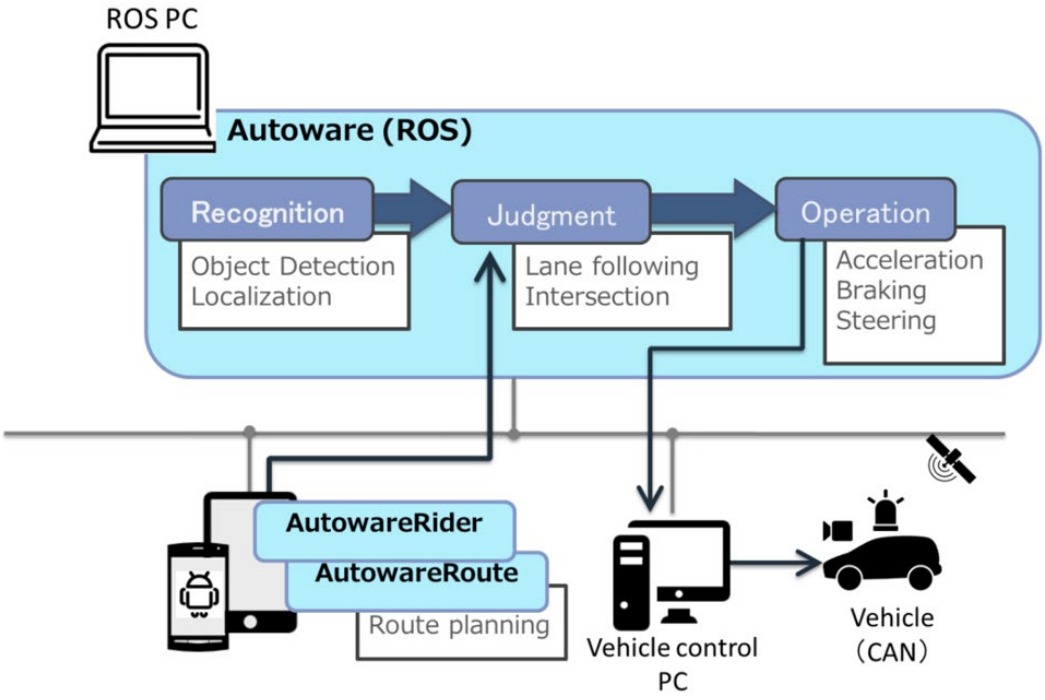
# 3 What is Autoware.AI?

Autoware.AI is the first all-in-one open-source software for autonomous driving technology. Autoware.AI is based on ROS 1, and Autoware.AI source code is available under the Apache 2.0 license.

The Robot Operating System (ROS) is a flexible framework for robotic software development. ROS provides a collection of tools and libraries that simplify the task of developing robust, general-purpose robot software. ROS includes the following:

- An original build system called Catkin
- An OpenCV image processing library
- A ROSBAG-based simulation environment for people who do not own real autonomous vehicles

The following diagram shows a high-level overview of an example system that is based on Autoware:

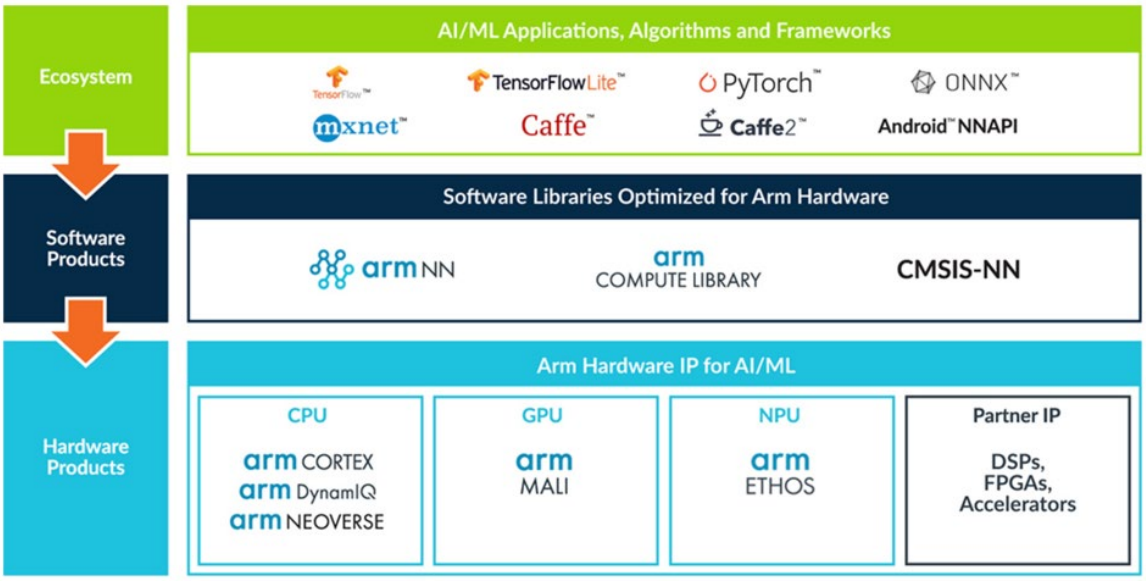



[Image credit: The Autoware Foundation]

# 4 What is Arm NN?

Arm NN is an inference engine for Arm CPUs, GPUs, and NPUs. Arm NN executes machine learning models on Arm-based devices to make predictions based on input data.

Arm NN bridges the gap between existing NN frameworks and the underlying IP, as shown in the following diagram:




Arm NN enables efficient translation of existing neural network frameworks, like TensorFlow Lite, TensorFlow, ONNX, and Caffe. This translation process allows neural networks to run efficiently, without modification, across Arm Cortex-A CPUs, Arm Mali GPUs, and Arm Ethos NPUs.

Arm NN includes the TensorFlow parser `armnnTfParser`. This parser is a library for loading neural networks that are defined by TensorFlow into the Arm NN runtime. This guide uses the TensorFlow parser to parse our object detection model.

# 5 Run the object detection demo

This section of the guide shows how to install, configure, and run the components that are required by the object detection demonstration.


Note

Please ensure you have completed the instructions in Before you begin in advance of following the steps in this section of the guide.

The following steps show how to run the object detection demo:

1. Download the Arm ML examples on the HiKey 960.

```
cd $HOME/armnn-devenv
git clone https://github.com/ARM-software/ML-examples.git
```

2. Make space on the HiKey960 root partition.

   The instructions in this guide install several packages. We must ensure there is enough space on the HiKey device root partition for these packages. To create space, move some of the larger system files from the root partition to the user partition.

   Run the `make_space.sh` script to move `/var/cache` and `/usr/lib/aarch64-linux-gnu` from the root partition to the user partition:

```
cd $HOME/armnn-devenv/ML-examples/autoware-vision-detector
./scripts/make_space.sh
```

3. Install the Robot Operating System (ROS) on the Hikey960.

   Autoware.AI is based on the Robot Operating System, a flexible framework for writing robot software. Before we build Autoware.AI, we must install ROS Kinetic.

   The `autoware-vision-detector` example in the Arm ML examples repository includes a helper script to install ROS. Run the helper script as follows:

```
cd $HOME/armnn-devenv/ML-examples/autoware-vision-detector
./scripts/ros_setup.sh
```

   The `ros_setup.sh` script does the following:

   - Downloads the Autoware.AI source code
   - Removes unrequired Autoware.AI modules to simplify the source and dependency tree
   - Installs all Autoware dependencies
   - Copies Arm NN include files and libraries into `/opt/armnn`. The object detection module expects these files to be in this location.


Note

The ROS.org installation instructions include more information about how to install ROS Kinetic on Ubuntu.

4. Build Autoware.AI on the HiKey 960.

   The `autoware-vision-detector` example in the Arm ML examples repository includes a helper script to install both ROS and Autoware.AI.

   Run the helper script as follows:

```
cd $HOME/armnn-devenv
./ML-examples/autoware-vision-detector/scripts/autoware_setup.sh
mkdir $HOME/armnn-devenv/autoware/src/arm/vision_detector
cd $HOME/armnn-devenv/ML-examples/autoware-vision-detector
cp -r * $HOME/armnn-devenv/autoware/src/arm/vision_detector/
```


Note

The Autoware.AI source build instructions include more information about how to install Autoware.AI.

5. Prepare the Tiny YOLO v2 neural network on the host machine.

   This guide uses the Tiny YOLO v2 neural network.

   You Only Look Once (YOLO) is a network for objection detection. Object detection identifies the presence and location of certain objects in an image and classifies those objects. Tiny YOLO is a variation of YOLO which offers a smaller model size and faster inference speed. Tiny YOLO is naturally suited for embedded computer vision and deep learning devices.

   Before Arm NN can parse the network, we must convert the network from its original darknet format to the TensorFlow protobuf format. To conserve disk space on the HiKey device, we perform this conversion on the host machine.

   The `autoware-vision-detector` example in the Arm ML examples repository includes a helper script to download and convert the Tiny YOLO v2 neural network. Run the helper script as follows:

```
cd $HOME
git clone https://github.com/ARM-software/ML-examples.git
cd $HOME/ML-examples/autoware-vision-detector
./scripts/get_yolo_tiny_v2.sh
```

   The script creates a file called `yolo_v2_tiny.pb` in the current directory.


Note

The helper script installs TensorFlow 1.15 if it is not already available on the host machine. For information about which versions of Python 3 are compatible with TensorFlow 1.15, see Install TensorFlow with pip. We tested this guide using Python 3.6.

6. Copy the Tiny YOLO v2 neural network from the host machine to the HiKey 960.

   Use the `scp` command to copy the `yolo_v2_tiny.pb` file created by the previous step to the HiKey 960 device:

```
cd $HOME/ML-examples/autoware-vision-detector
scp yolo_v2_tiny.pb \
   arm01@<hikey_ip_address>:~/armnn-devenv/autoware/src/arm/vision_detector/models
```


Note

For more information about how to convert YOLO graphs to the TensorFlow protobuf format, see the darkflow documentation.

7. Build the vision detector node on the HiKey 960.

   To build the vision detector node, run the following commands:

```
cd $HOME/armnn-devenv/autoware
source /opt/ros/kinetic/setup.sh
colcon build --packages-up-to vision_detector
```

   You can test the build by running unit tests, as follows:

```
colcon test --packages-select vision_detector --event-handlers console_cohesion+
```

8. Download the demonstration data on the HiKey 960.

   The vision detector node takes images from the `/image_raw` topic and outputs detections on the `/vision_detector/objects` topic.

   ROS uses message passing with topics for inter-process communication. In ROS, each node runs independently. One node writes, or publishes, messages into a topic while another node reads, or subscribes to, the messages of the same topic.

   To provide input for the demo, we download some images from the KITTI dataset and send them to the `/image_raw` topic, using the `images_to_rosbag.py` helper script. The KITTI data is in the form of a series of PNG image files. The `images_to_rosbag.py` script packages the images into a rosbag for easy integration into ROS. In ROS, all subscribed messages of ROS topics are logged and time-stamped to a `.bag` file structure called a rosbag. Developers can use the official rosbag tool to read, filter, and extract message data from rosbag files.

   Download the images and run the `images_to_rosbag.py` helper script as follows:

```
cd $HOME/armnn-devenv
mkdir images
cd images
wget https://s3.eu-central-1.amazonaws.com/ \
   avg-kitti/raw_data/2011_09_26_drive_0106/2011_09_26_drive_0106_sync.zip
unzip 2011_09_26_drive_0106_sync.zip
../ML-examples/autoware-vision-detector/scripts/images_to_rosbag.py \
   2011_09_26/2011_09_26_drive_0106_sync/image_02/data/ demo.rosbag
cp demo.rosbag $HOME/armnn-devenv/autoware/
```

Note

The preceding code downloads `2011_09_26_drive_0106_sync.zip` as an example, but you can choose any sequence from the KITTI dataset.

9. Run the demo on the HiKey 960.

   To run the demo, we must do the following:

   - Launch the object detection node
   - Launch `web_video_server` to view the output
   - Play the newly created rosbag in a video loop

   A helper script, `run_demo.sh`, performs the tasks that are described in this step.

   Run the script from the root of your Autoware source tree, which is the same location where you copied the rosbag file, as follows:

```
cd ~/armnn-devenv/autoware
~/armnn-devenv/ML-examples/autoware-vision-detector/scripts/run_demo.sh
```

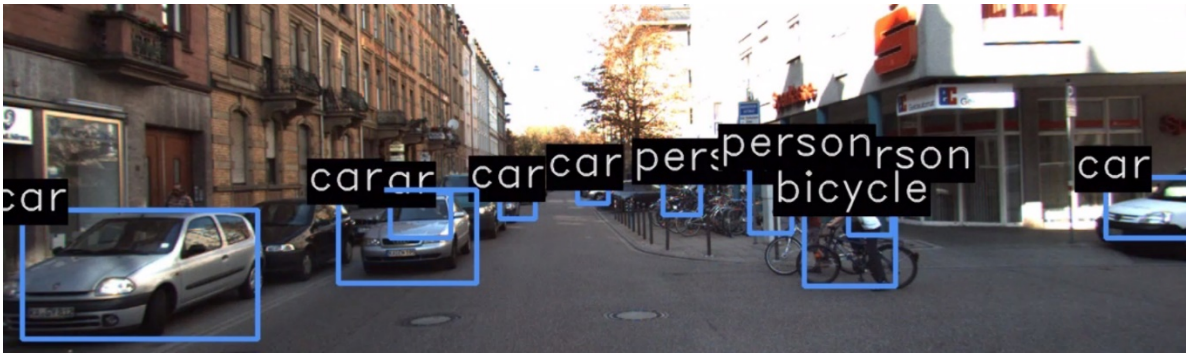
   You will see continuous output on the command-line console to indicate that the demo is running.

10. View the results in a web browser.

    Open the following URL in your favorite web browser:

```
http://<ip address of the HiKey>:8080/stream?topic=/vision_detector/image_rects
```

    The web browser shows the looping video with real-time object detection, as you can see here:



# 6 Examining the Arm NN code

In this section of the guide, we examine the demonstration code and highlight some of the key functional points for integrating with the Arm NN C++ API.

**`vision_detector_node.cpp`** contains the entry point to the ROS node. The key tasks performed by this code include the following:

1. Initialize and register the ROS node:
```
ros::init(argc, argv, "vision_detector");
```
2. Create an Arm NN runtime object:
```
// Create Arm NN Runtime
armnn::IRuntime::CreationOptions options;
options.m_EnableGpuProfiling = false;
armnn::IRuntimePtr runtime = armnn::IRuntime::Create(options);
```
3. Specify the list of backends that can be used to optimize the network:
```
// Enumerate Compute Device backends
std::vector<armnn::BackendId> computeDevices;
computeDevices.push_back(armnn::Compute::GpuAcc);
computeDevices.push_back(armnn::Compute::CpuAcc);
computeDevices.push_back(armnn::Compute::CpuRef);

detector_armnn::Yolo2TinyDetector<DataType> yolo(runtime);
yolo.load_network(pretrained_model_file, computeDevices);
```

**`armnn_yolo2tiny.hpp`** contains the definition of `Yolo2TinyDetector`. In `Yolo2TinyDetector` we call Arm NN to create a parser object that loads the network file. Arm NN has parsers for several model file types, including TF, TFLite, ONNX, and Caffe. Parsers create the underlying Arm NN graph, so you do not need to construct your model graph by hand.

The following example creates a TensorFlow parser to load our TensorFlow protobuf file from the specified path:

```
// Setup Arm NN Network
// Parse pre-trained model from TensorFlow protobuf format
using ParserType = armnnTfParser::ITfParser;
auto parser(ParserType::Create());
armnn::INetworkPtr network{nullptr, [](armnn::INetwork *) {}};
network = parser->CreateNetworkFromBinaryFile(model_path.c_str(),
    inputShapes,requestedOutputs);
```

This network is then optimized and loaded into the Arm NN runtime as follows:

```
// Set optimisation options
armnn::OptimizerOptions options;
options.m_ReduceFp32ToFp16 = false;

// Optimize network
armnn::IOptimizedNetworkPtr optNet{nullptr, [](armnn::IOptimizedNetwork *){}};
optNet = armnn::Optimize(*network, compute_devices, runtime->GetDeviceSpec(), options);

if (!optNet) {
  throw armnn::Exception("armnn::Optimize failed");
}

// Load network into runtime
armnn::Status ret = this->runtime->LoadNetwork(this->networkID, std::move(optNet));
```

```
if (ret == armnn::Status::Failure) {
  throw armnn::Exception("IRuntime::LoadNetwork failed");
}
```

Every time an image is published to the ROS topic `/image_raw`, the callback function `DetectorNode<T>::callback_image` is invoked. To run inference, the callback function calls `Yolo2TinyDetector<T>::run_inference`, which in turn calls Arm NN to execute the inference.

The Arm NN parser extracts the input and output information for the network. We obtain the input and output tensors, then retrieve the binding information. This binding information contains all the essential information about the layer. The binding information is a tuple containing integer identifiers for bindable input and output layers and the tensor information. Tensor information consists of data type, quantization information, number of dimensions, and total number of elements.

The `EnqueueWorkload()` function of the runtime context executes the inference for the network loaded as follows:

```
// Allocate output container
size_t output_size = this->output_tensor_shape.GetNumElements();
std::vector<T> output(output_size);

// Create input and output tensors and their bindings
armnn::InputTensors inputTensors{
  {0,
  armnn::ConstTensor(this->runtime->GetInputTensorInfo(this->networkID, 0),
                     input_tensor.data())}};

armnn::OutputTensors outputTensors{
  {0,
  armnn::ConstTensor(this->runtime->GetOutputTensorInfo(this->networkID, 0),
                    output.data())}};

// Run inference
this->runtime->EnqueueWorkload(this->networkID,inputTensors,outputTensors);
```

The output of the inference is decoded in the `Yolo2TinyDetector<T>::process_output` function. A score threshold is applied, and the `non_maximum_suppression` algorithm is applied to remove spurious detections. The final detection is output in `autoware_msgs::DetectedObjectArray` format.

# 7 Related information

Here are some resources related to material in this guide:

- **Arm NN**
- Arm Software Developer Kit (SDK)
- Autoware.AI
- ROS Kinetic
- Source code repositories:
  - https://gitlab.com/autowarefoundation/autoware.ai
  - https://github.com/ARM-software/ML-examples
  - https://github.com/ARM-software/Tool-Solutions

# 8 Next steps

In this guide, we used the Arm NN C++ APIs and Tiny YOLO v2 to create a real-time object detection system.

Arm NN lets developers focus on creating innovative AI-powered applications, rather than performance tuning for inference deployment on Arm Cortex-A hardware.