IoT Gets an ML Boost, From Edge to Cloud

Today, it’s easy to run Edge Impulse Machine Learning on any OS, like the Linux-powered Raspberry Pi, and on every cloud, like Microsoft’s Azure IoT.
Today, It’s Easy To Run Edge Impulse Machine Learning on Any OS, Like The Linux-Powered Raspberry Pi, And On Every Cloud, Like Microsoft’s Azure IoT

- Evan Rust

Building enterprise-grade IoT solutions takes a lot of practical effort and a healthy dose of imagination. As a foundation, you start with a highly secure and reliable communication between your IoT application and the devices it manages. We picked our favorite integration, the Microsoft Azure IoT Hub, that provides us with a cloud-hosted solution backend to connect virtually any device. For our hardware, we selected the ubiquitous Raspberry Pi 4, and of course Edge Impulse, which will connect to both platforms and extend our showcased solution from cloud to edge, including device authentication, out-of-box device management, and model provisioning.

FROM EDGE TO CLOUD - GETTING STARTED

Edge machine learning devices fall into two categories: some are able to run very simple models locally, and others have more advanced capabilities that allow them to be more powerful and have cloud connectivity. The second group is often expensive to develop and maintain, as training and deploying models can be an arduous process. That’s where Edge Impulse comes in to help to simplify the pipeline, as data can be gathered remotely, used effortlessly to train models, and then downloaded to the devices directly from the Azure IoT Hub, and run - fast.

This reference project will serve you as a guide for quickly getting started with Edge Impulse on a Raspberry Pi 4 and Azure IoT, to train a model that detects lug nuts on a wheel and sends alerts to the cloud.
SETTING UP THE HARDWARE

To begin, you’ll need a Raspberry Pi 4 with an up-to-date Raspberry Pi OS image that can be found [here](#). After flashing this image to an SD card and adding a file named `wpa_supplicant.conf`:

```
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
update_config=1
country=<Insert 2 letter ISO 3166-1 country code here>

network={
  ssid="<Name of your wireless LAN>"
psk="<Password for your wireless LAN>"
}
```

along with an empty file named `ssh` (both within the `/boot` directory), you can go ahead and power up the board. Once you’ve successfully SSH’d into the device with

```
$ ssh pi@<IP_ADDRESS>
```

and the password `raspberry`, it’s time to install the dependencies for the Edge Impulse Linux SDK. Simply run the next three commands to set up the NodeJS environment and everything else that’s required for the edge-impulse-linux wizard:

```
$ curl -sL https://deb.nodesource.com/setup_12.x | sudo bash -
$ sudo apt install -y gcc g++ make build-essential nodejs sox gstreamer1.0-tools gstreamer1.0-plugins-good gstreamer1.0-plugins-base gstreamer1.0-plugins-base-apps
$ npm config set user root && sudo npm install edge-impulse-linux -g --unsafe-perm
```

...
Training accurate machine learning models requires feeding plenty of varied data, which means a lot of images are required. For this use case, I captured around 50 images of a wheel that had lug nuts on it. After I was done, I headed to the Labeling queue in the Data Acquisition page and added bounding boxes around each lug nut within every image, along with every wheel.

To add some test data, I went back to the main Dashboard page and clicked the `Rebalance dataset` button that moves 20% of the training data to the test data bin.
TRAINING YOUR MODELS

So now that we have plenty of training data, it’s time to do something with it, namely train a model.

The first block in the impulse is an Image Data block, and it scales each image to a size of `320` by `320` pixels. Next, image data is fed to the Image processing block that takes the raw RGB data and derives features from it.

TESTING AND DEPLOYING YOUR MODELS

In order to verify that the model works correctly in the real world, we’ll need to deploy it to the Raspberry Pi 4. This is a simple task thanks to the Edge Impulse CLI, as all we have to do is run `$ edge-impulse-linux-runner` which downloads the model and creates a local webserver. From here, we can open a browser tab and visit the address listed after we run the command to see a live camera feed and any objects that are currently detected.
INTEGRATING YOUR MODELS WITH MICROSOFT AZURE IOT

With the model working locally on the device, let’s add an integration with an Azure IoT Hub that will allow the Pi to send messages to the cloud. First, make sure you’ve installed the Azure CLI and have signed in using ‘az login’. Then get the name of the resource group you’ll be using for the project. If you don’t have one, you can follow this guide on how to create a new resource group. After that, return to the terminal and run the following commands to create a new IoT Hub:

```bash
$ az iot hub create --resource-group <your resource group> --name <your IoT Hub name>
$ az extension add --name azure-iot
$ az iot hub device-identity create --hub-name <your IoT Hub name> --device-id <your device id>
```

and register a new device ID:

```bash

```

```bash
$ az extension add --name azure-iot
$ az iot hub device-identity create --hub-name <your IoT Hub name> --device-id <your device id>
```

```bash

```
Retrieve the connection string with
`
$ az iot hub device-identity connection-string show
--device-id <your device id> --hub-name <your IoT Hub name>`

and set it as an environment variable with
`
$ export IOTHUB_DEVICE_CONNECTION_STRING="<your connection string here>"
` in the Pi’s SSH session, as well as
`$ pip install azure-iot-device`
to add the necessary libraries. (Note: if you do not set the environment variable or pass it in as an argument the program will not work!) The connection string contains the information required for the device to establish a connection with the IoT Hub service and communicate with it. You can then monitor output in the Hub with
`$ az iot hub monitor-events --hub-name <your IoT Hub name> --output table`
or in the Azure Portal.
To make sure it works, download and run this example to make sure you can see the test message. For the second half of deployment, we’ll need a way to customize how our model is used within the code. Thankfully, Edge Impulse provides a Python SDK for this purpose. Install it with

```
$ sudo apt-get install libatlas-base-dev libportaudio0 libportaudio2 libportaudiocpp0 portaudio19-dev
$ pip3 install edge_impulse_linux -i https://pypi.python.org/simple
```

There’s some simple code that can be found here on Github, and it works by setting up a connection to the Azure IoT Hub and then running the model.

```
1  #!/usr/bin/env python
2  import sys
3  import cv2
4  import numpy as np
5  import signal
6  import time
7  import rtsp
8  import videocapture
9  import threading
10
11  def run_capture_thread()
12      print('Running capture thread...')
13      cap = videocapture.VideoCapture(0)
14      while True:
15          frame = cap.read()
16          if frame is None:
17              print('Frame is None')
18              continue
19          cv2.imshow('frame', frame)
20          if cv2.waitKey(1) & 0xFF == ord('q'):
21              break
22  thread = threading.Thread(target=run_capture_thread)
23  thread.start()
24
25  def run_prediction_thread()
26      print('Running prediction thread...')
27      prediction = edge_impulse.predict_image(model_file, frame)
28      print('Prediction:', prediction)
29      # Do something with prediction...
30  thread = threading.Thread(target=run_prediction_thread)
31  thread.start()
32
33  if __name__ == '__main__':
34      run_capture_thread()
35      run_prediction_thread()
36```

Once you’ve either downloaded the zip file or cloned the repo into a folder, get the model file by running

```
$ edge-impulse-linux-runner --download modelfile.eim
```

inside of the folder you just created from the cloning process. This will download a file called `modelfile.eim`. Now, run the Python program with

```
$ python lug_nut_counter.py ./modelfile.eim -c <LUG_NUT_COUNT>
```

where `<LUG_NUT_COUNT>` is the correct number of lug nuts that should be attached to the wheel (you might have to use `python3` if both Python 2 and 3 are installed).
Now whenever a wheel is detected the number of lug nuts is calculated. If this number falls short of the target, a message is sent to the Azure IoT Hub.
And by only sending messages when there’s something wrong, we can prevent an excess amount of bandwidth from being taken due to empty payloads.

---

The Possibilities Are Endless

Imagine utilizing object detection for an industrial task such as quality control on an assembly line, or detecting ripe fruit amongst rows of crops, machinery malfunction and remote, battery-powered inferencing devices. Between Edge Impulse, hardware like Raspberry Pi, and the Microsoft Azure IoT Hub, you can design endless models and deploy them on every device, while authenticating each and every device with built-in security. You can set up individual identities and credentials for each of your connected devices to help retain the confidentiality of both cloud-to-device and device-to-cloud messages, revoke access rights for specific devices, transmit code and services between the cloud and the edge, and benefit from advanced analytics on devices running offline or with intermittent connectivity. And if you’re really looking to scale your operation and enjoy a complete dashboard view of the device fleets you manage, it is also possible to receive IoT alerts in Microsoft’s Connected Field Service from Azure IoT Central - directly.

Feel free to take the code for this project hosted here on GitHub and create a fork or add to it.
The complete project is available here. Let us know your thoughts at hello@edgeimpulse.com. There are no limits, just your imagination at work.