Deep Learning at the Edge – Activity 2

Task 2: Image Classification

Introduction

In this lab, we will go through the steps of creating an image classification exercise using a Convolutional Neural Network (CNN). The same basic steps can be used to create a device that recognizes and classifies other images, too!

We will start with collecting sample images for our project. This has already been done, but feel free to play around with your own images.

In your Edge Impulse account, create a new project – **my-plant-picture-project**

Required Hardware

For collecting image data, you should have access to a recording device. This can be a smartphone, webcam, laptop, etc.

For deploying, we will use a smartphone (and an OpenMV H7 camera edge device) for this demo.

1. Collect Data

We will start with a pre-made image dataset. The dataset contains:

- Picture – 31 images of a framed picture
- Plant - 31 images of a house plant (Aloe vera)
- Unknown – 31 images of other random room objects

Importing this dataset

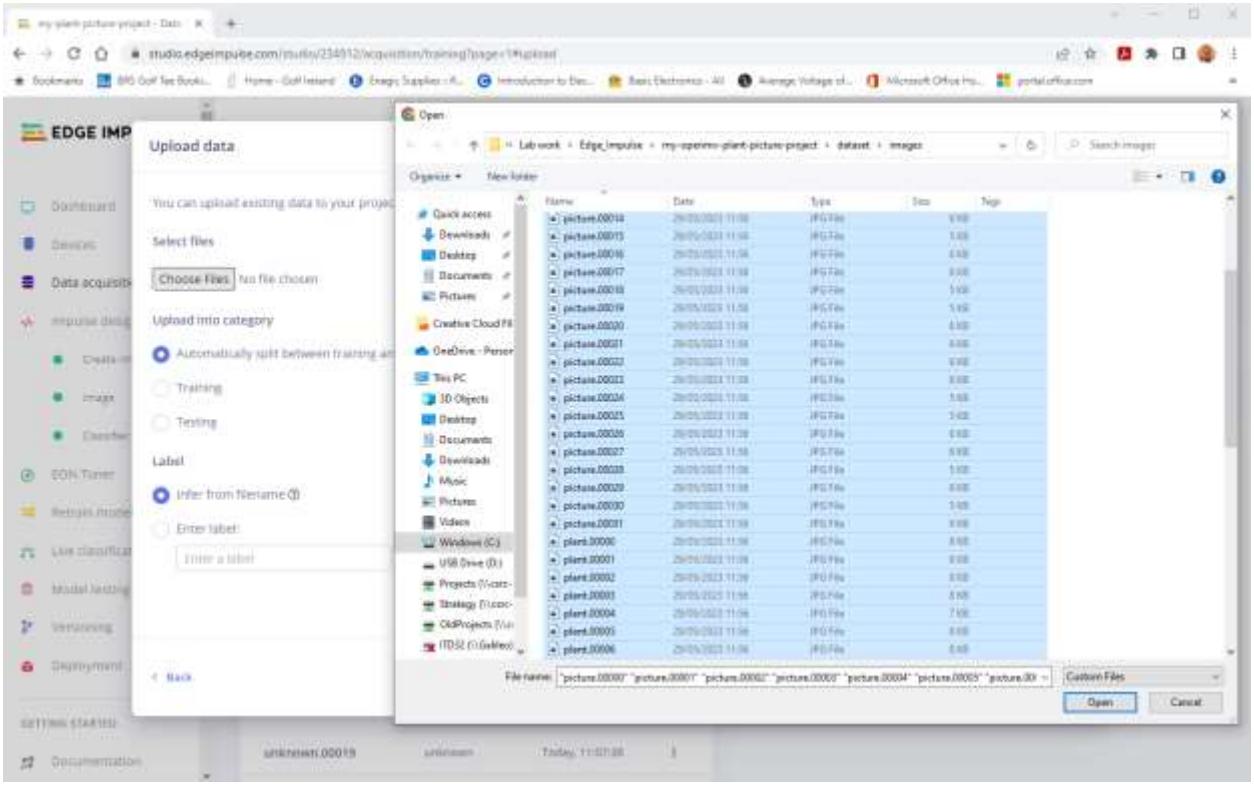
You can import this dataset to your Edge Impulse project by:

1. Downloading the "**images.zip"** dataset from https://github.com/brenmul/deep_learning_lab/tree/main/datasets
2. Unzip the file in a location of your choice.
3. In your project, go to **Data acquisition** and click on the 'Upload icon'. Follow the instructions on the screen.

2. Upload Data

In your project in Edge Impulse. Head to the **Data Acquisition** page and then **Go to the upload data** option.

On this new page, click **Choose files**. Select all the ‘image files from your unzipped images set.



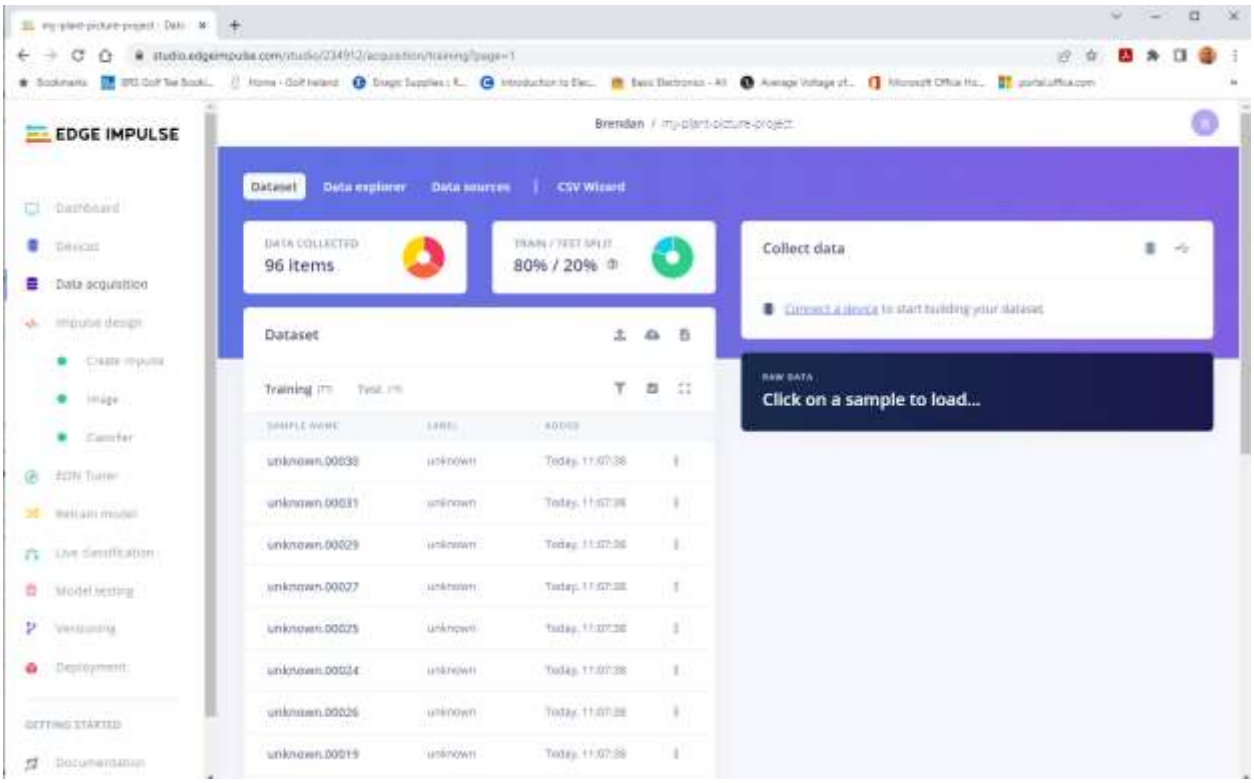
Click **Open**.

Leave *Automatically split between training and testing* selected. If the file naming scheme is as outlined above, leave *Infer from filename* option selected. Here the label can easily be inferred by the filename and is automatically applied to each image.

Click **Upload Data**.

Click on the **Data acquisition** link to go back to the Data Acquisition page. Here, make sure that all the images are present and that they are divided between the training and test sets (there should 20% of the image samples in the test set).

In this design, there are only 77 images in the training set and 19 images in the test set.

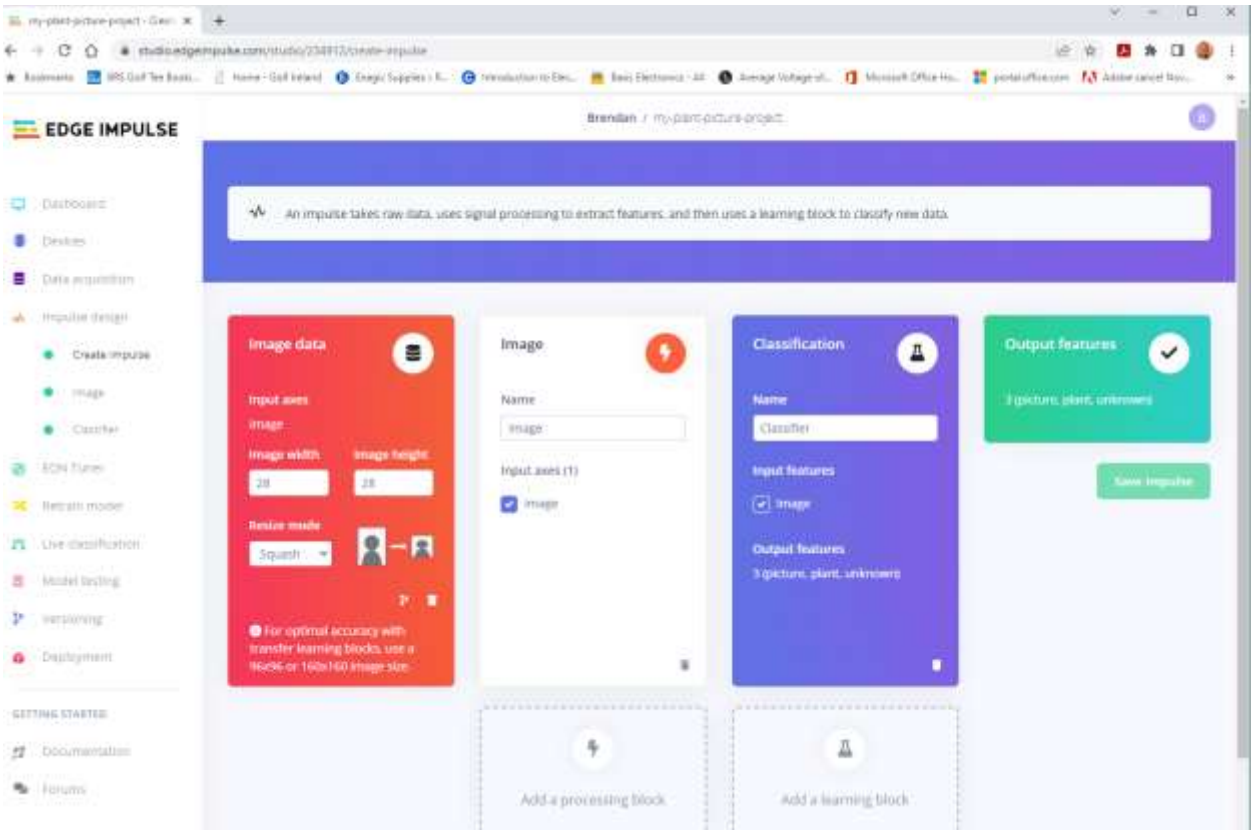


Alternate method: you can also collect image data straight from your Edge Impulse project! Go to *Data acquisition* in a new project and connect your smartphone to add your own images for this exercise.

3. Feature Extraction

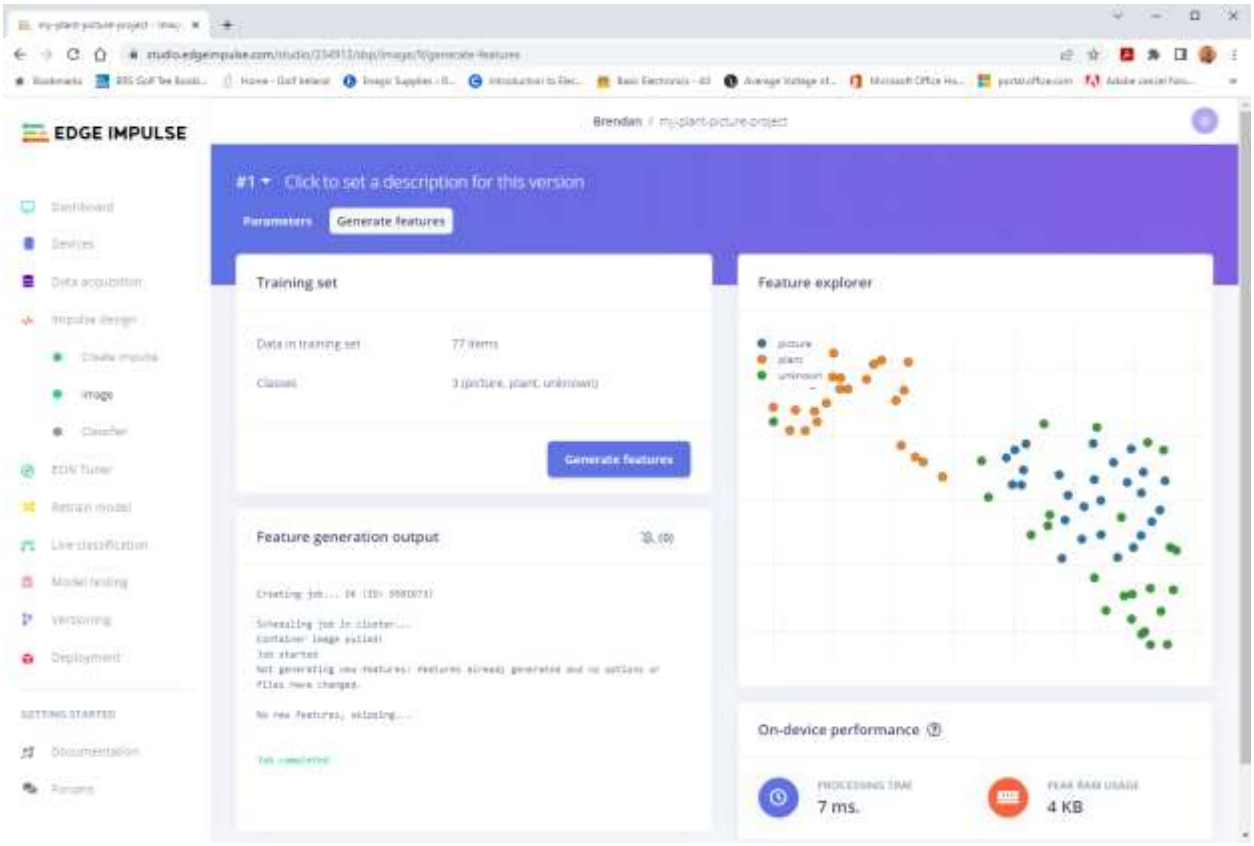
Navigate to the **Impulse design** page of your project. Add an **Image** processing block and a **Classification** learning block.

Deep Learning at the Edge – Activity 2



Click **Save Impulse.**

Go to the **Image** page and make sure the Image Parameters option is set to RGB, then click on the **Generate Features** tab and wait a moment while your image samples are analyzed. When it is done, look at the *Feature explorer* to see if you can identify separation among your classes.

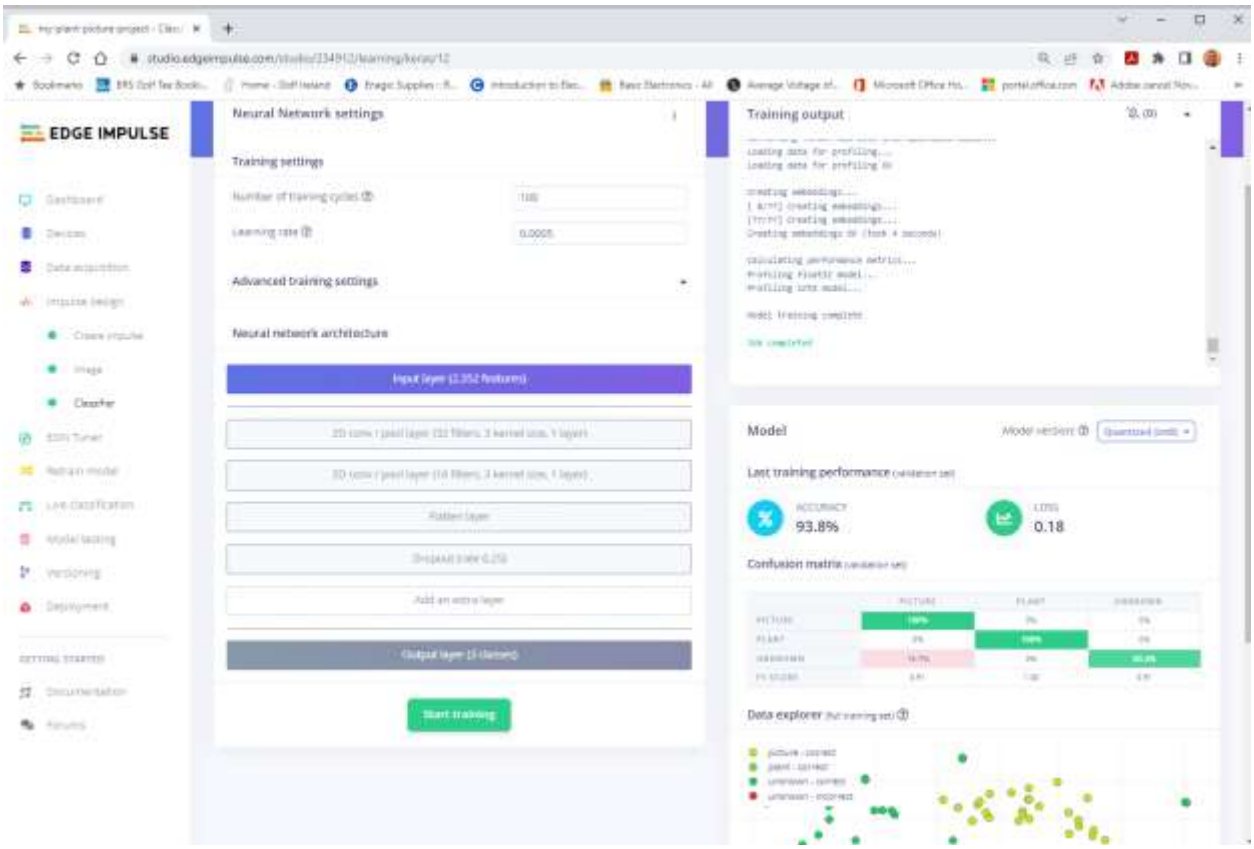


Some of these features do not look like they are separated very well. However, it is enough to get started for a project like this.

4. Model Training

Navigate to the **Classifier** page. Leave all the hyperparameters at their defaults, except for *the Number of training cycle* which is set at 100. Here we want to train the model over a larger number of epochs to improve performance. Then click **Start training**. When it's done, scroll down to view the *Confusion matrix* of the validation data.

Note the model that was chosen for this classifier – It has 2 CONV2D layers with 32 and 16 kernel filters, a Flatten layer, a Dense layer with a dropout rate of 0.25 and finally an output layer for the 3 object classes.

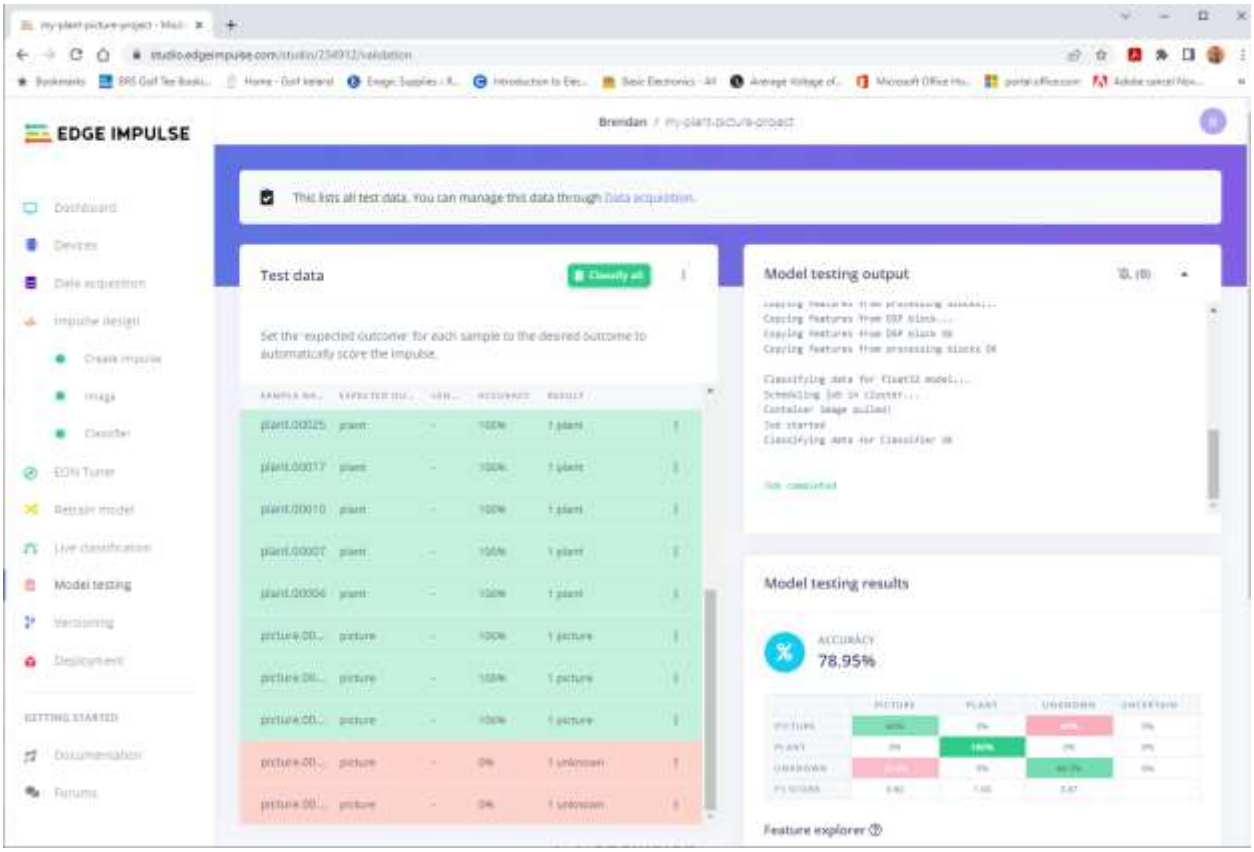


Note the *F1 scores* of each class and the *total accuracy*.

We have achieved a valid set accuracy of 93.8% which is very good. Feel free to try changing some of the hyperparameters and re-training your model to see if it improves the per-class accuracy. Note that there is no easy solution here: much of creating a better model is trial and error, and sometimes, you simply don't have enough (or the right kind of) data to train a good model. In those cases, it's back to the drawing board to gather data!

5. Testing

Head to the **Model testing** page and classify all the test data.

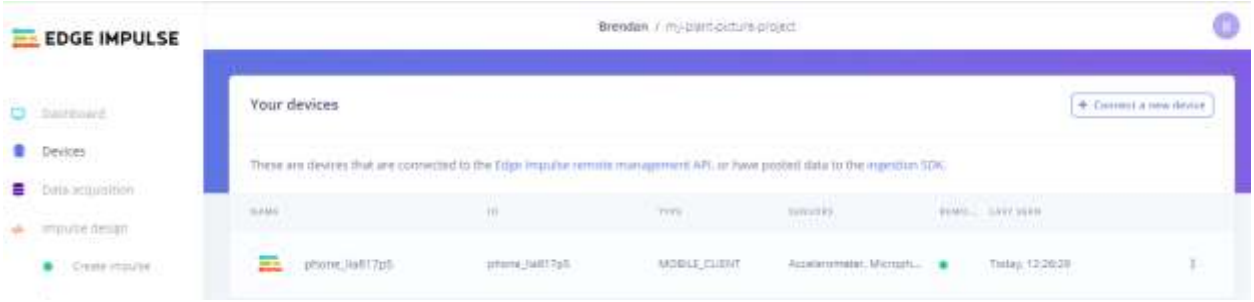


If you're happy with the test results, continue to the deployment step. 79% is good, but this performance indicates that we need to collect more data for this project.

6. Deployment – Smartphone and OpenMV H7 device

6.1 Smartphone:

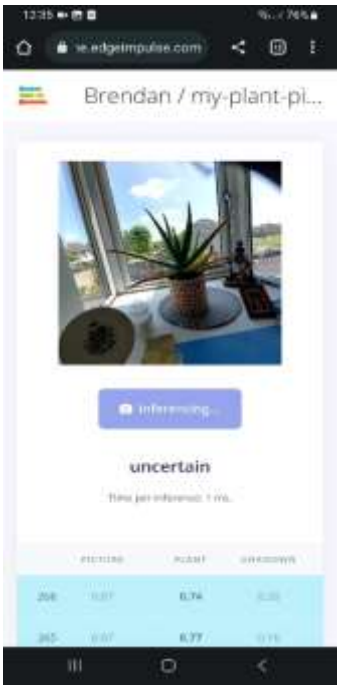
Connect your smartphone to your project (if you have not done so already). Go to the **Devices** page and scan the QR code to add your smartphone as a device.



Deep Learning at the Edge – Activity 2

Head to *smartphone.edgeimpulse.com* on your phone's browser. Scroll down and click **Switch to classification mode**. If asked, allow the program to access your phone's camera.

Hold your phone camera near to the actual picture or plant objects to see if it can identify them.



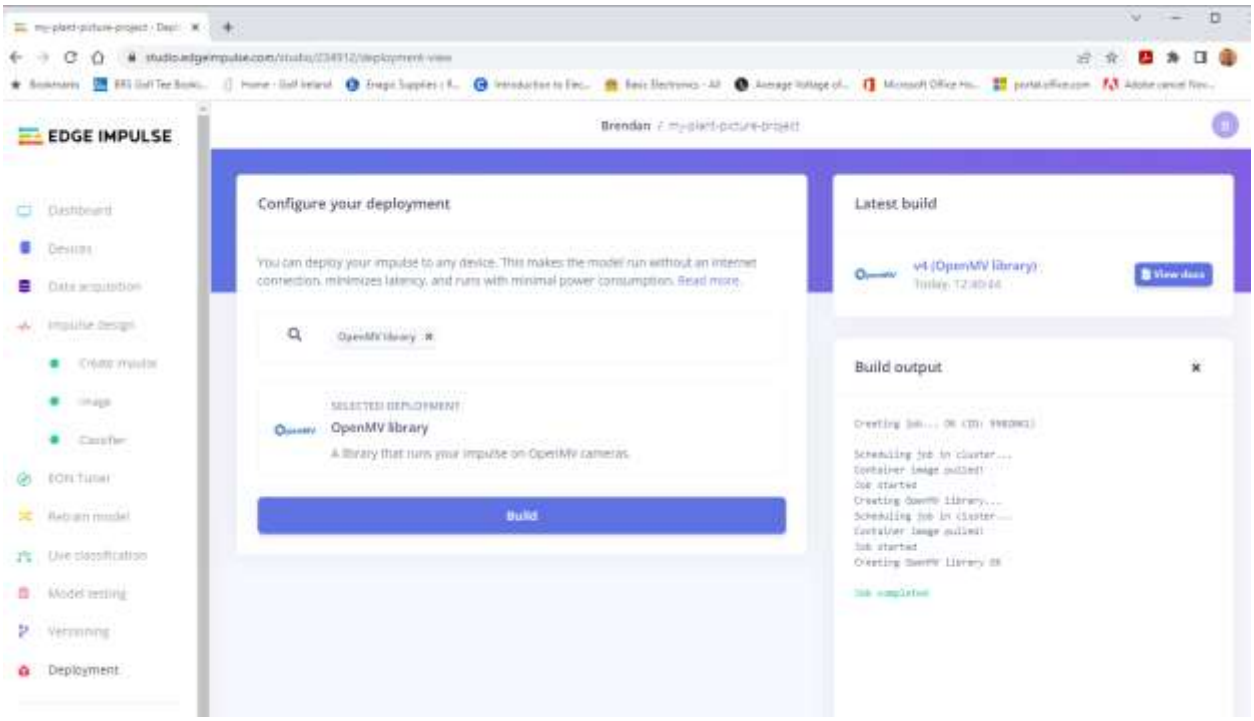
6.2 OpenMV H7 Camera Device:

The OpenMV H7 Cam is a small, low power, microcontroller board which allows you to easily implement applications using machine vision in the real-world. You program the OpenMV Cam in high level Python scripts (courtesy of the MicroPython Operating System) instead of C/C++.

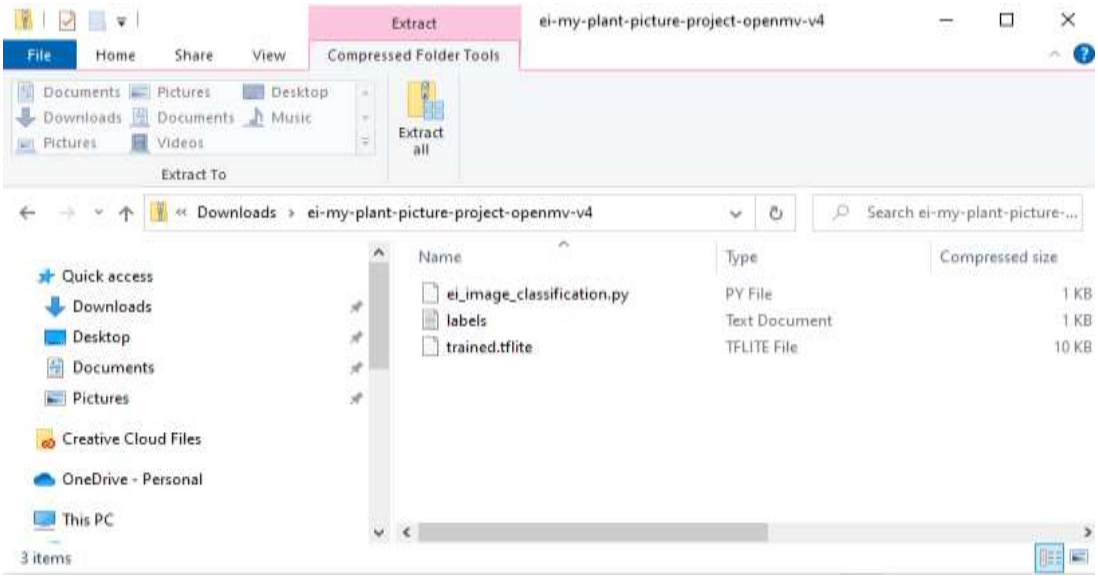
The device has a STM32H743VI ARM Cortex M7 processor running at 480 MHz with 1MB SRAM and 2MB of flash. The OpenMV Cam H7 comes with a MT9M114 image sensor capable of taking 640x480 8-bit Grayscale images or 640x480 8-bit BAYER images at 40 FPS when the resolution is above 320x240 and 80 FPS when it is below.

In your Edge Impulse project, head to the **Model Deployment** page and select the OpenMV Library within the *Configure Your Deployment* option list. Then click **Build** which starts the process of building the files you will need to deploy to the OpenMV camera device.

Deep Learning at the Edge – Activity 2



A zip file is produced as follows.

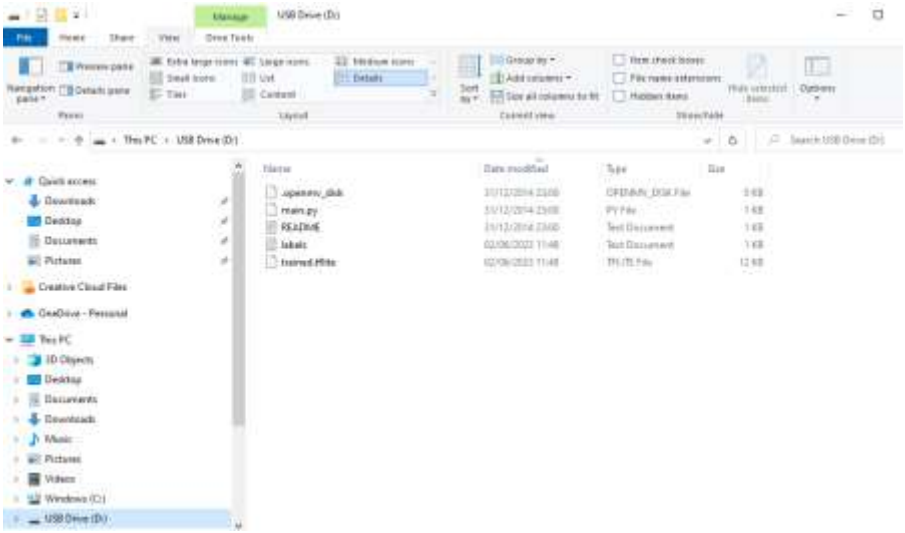


The zip file contents has 3 files:

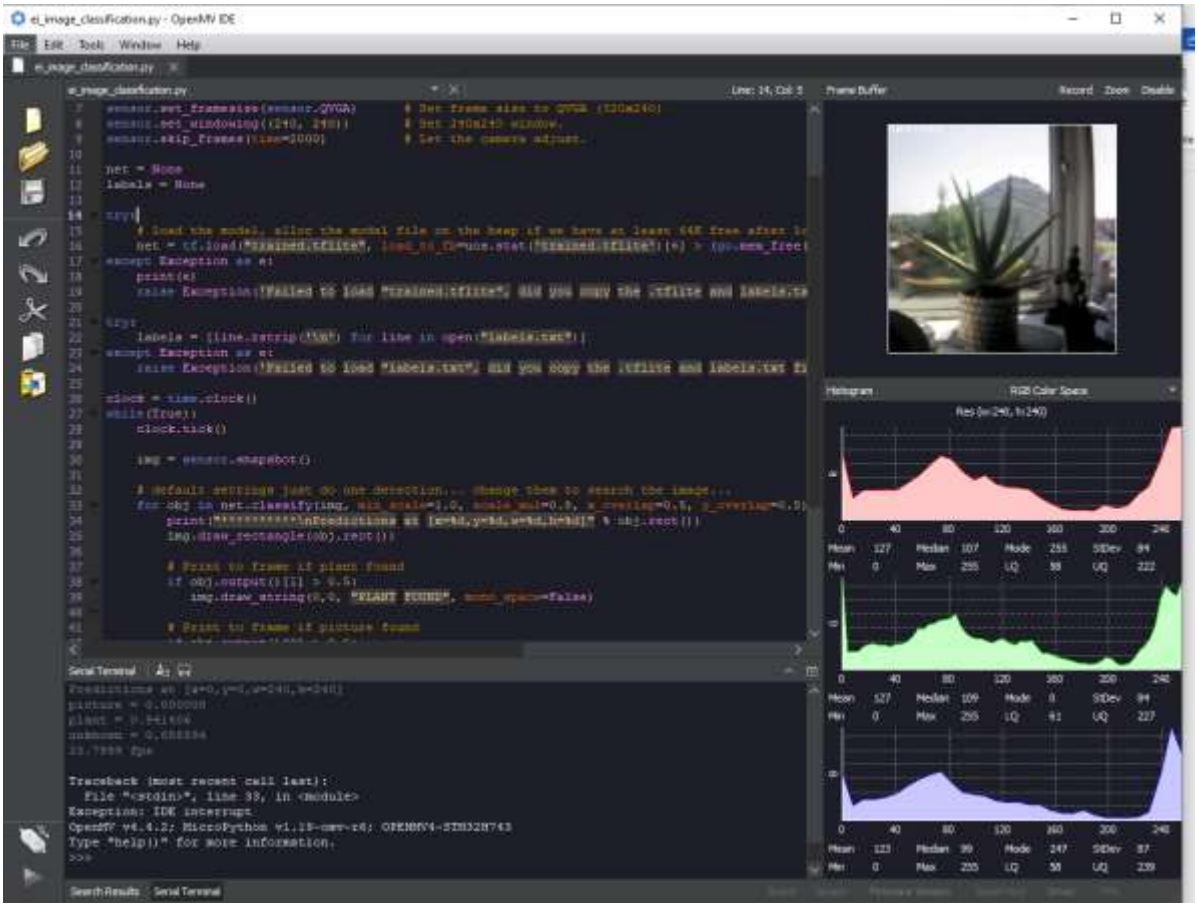
1. A *ei_image_classification.py* python script that the device will run.
2. A *labels.txt* file containing the 3 classes for this exercise.
3. A *trained.tflite* file– A unint8 quantized TensorFlow Lite file containing the model and weights/bias values for this image classification project. Note the size of this file at ~10KB. This OpenMV device only allows for a model size of ~256KB, so the generated model is well within this size limitation.

Deep Learning at the Edge – Activity 2

With the OpenMV H7 device connected to your PC, you will need to copy the *labels.txt* and *trained.tflite files* to the OpenMV H7 device home directory. This will appear as a USB Drive (D:) on your computer.

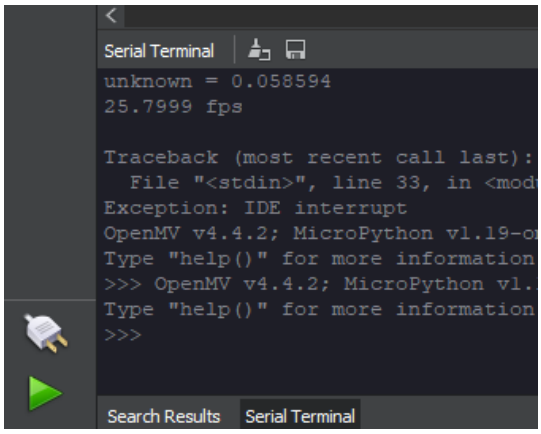


Launch the OpenMV IDE (The IDE is available here: https://openmv.io/pages/download) and open the classification python script as follows.

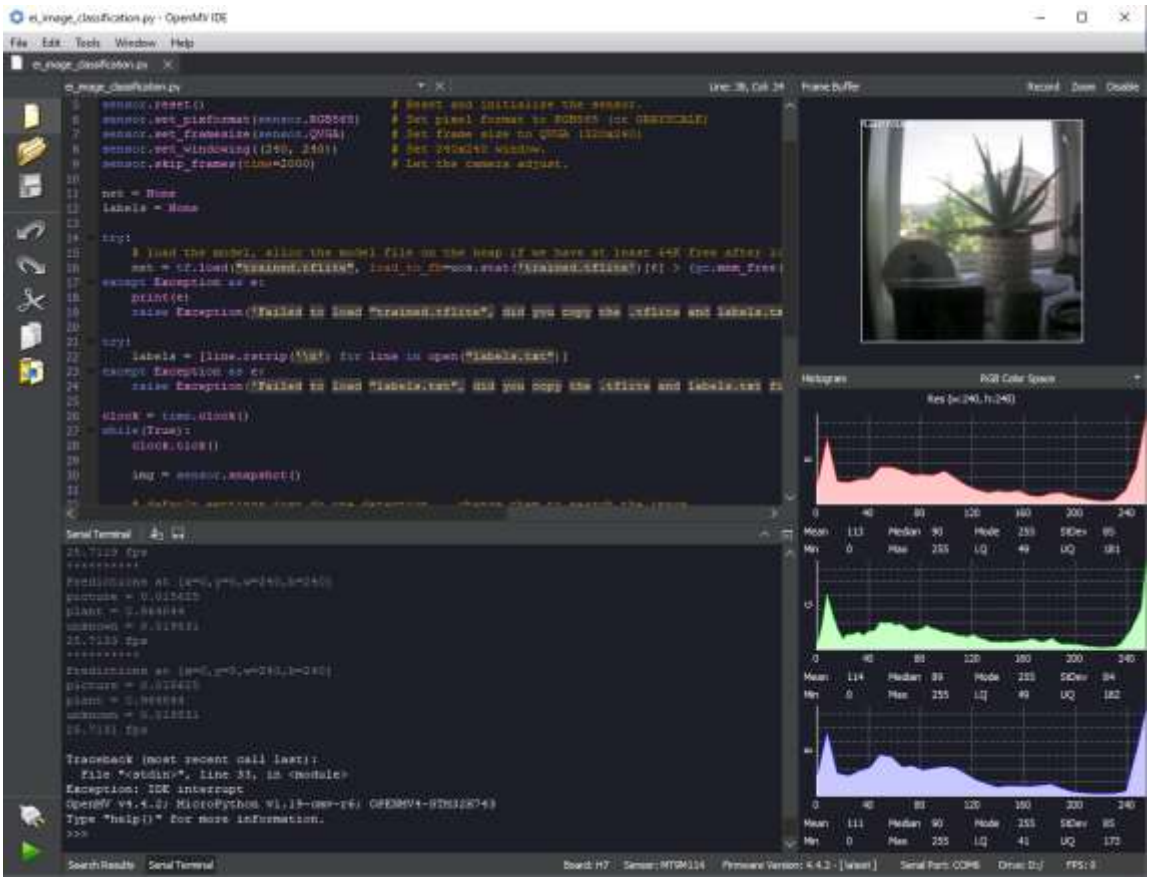


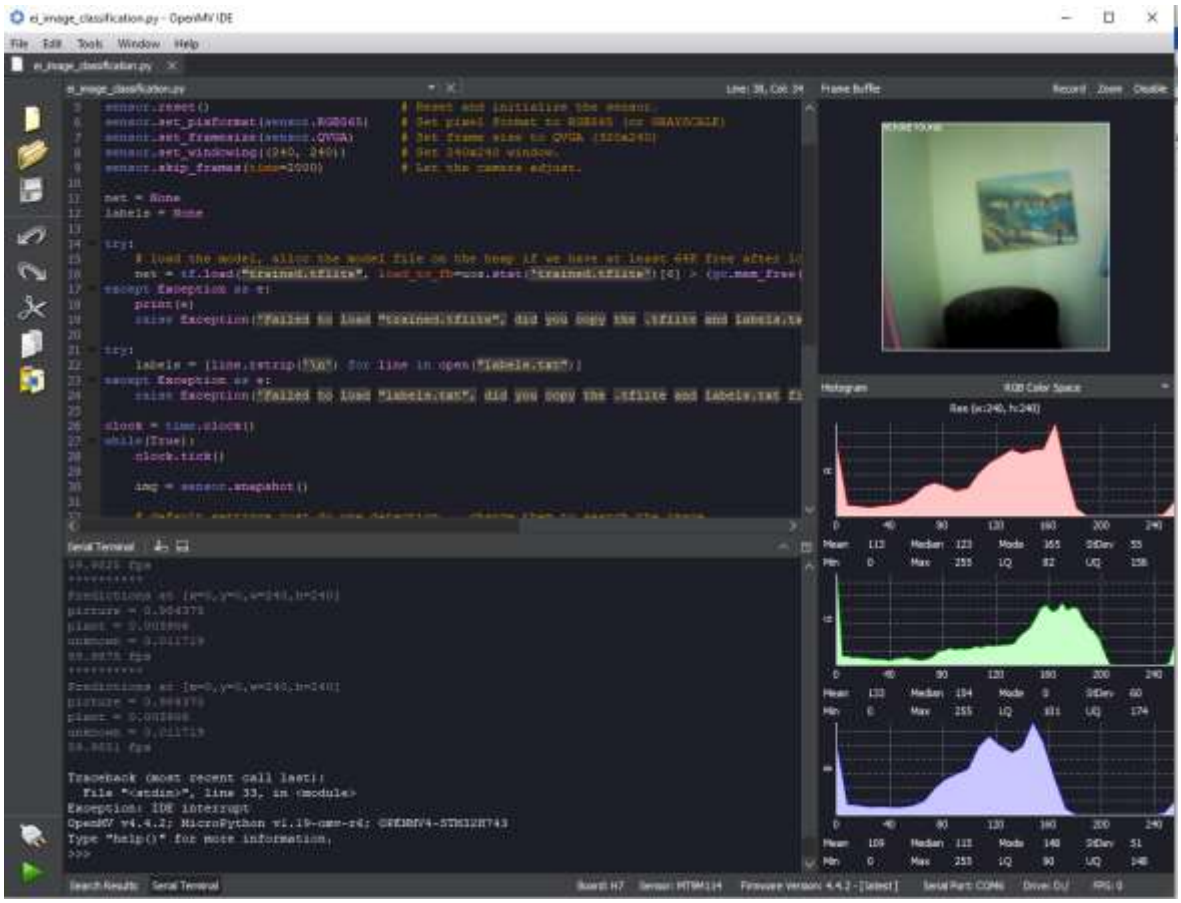
Deep Learning at the Edge – Activity 2

Connect to the device by pressing the connect logo located at the bottom LHS of the IDE. The arrow button turns Green when connected properly. Also set the Serial Terminal on.



Press the Green button to begin running the image classification model in real-time on this device. You should see text appearing in the Serial Terminal as below. As this programme is running, place the camera near to the *plant* or *picture* object to see if this model is working as expected.





Here, we see that when the device camera is located near to the *plant* or *Picture*– the model classifies the object correctly with a probability of >0.9. Note that this device operates in real-time at 25~60 Frames per Second (FPS) for this programming exercise which is impressive. We would expect that with a harder classification task such as cifar10 classification (10 objects) and a larger CNN model, the design would not be able to achieve this same frame rate!

7. Conclusion

Congratulations, I hope this has helped you get familiar with image data classification and developing a suitable model deployed to an actual edge device! Please note that this project is just a start as we only used 96 images for this exercise. It will require a lot more data and effort to get a working model ready for production!