EDGE IMPULSE

AI-Automated Hard-Hat Detection

Enhancing Health and Safety in Industrial Environments with Embedded Machine Learning

2021

The head is surely the most complex organ in the human body, but also the most delicate. The assessment and prevention of risks in the workplace remains the first priority approach to avoid accidents or reduce the number of serious injuries to the head. This is why wearing a hard hat in an industrial working environment is often required by law and helps to avoid serious accidents. This article will give you an overview of how to detect that the wearing of a helmet is well respected by all workers using a machine learning object detection model.

For this project, we have been using:

- Edge Impulse Studio to acquire some custom data, visualize the data, train the machine learning model and validate the inference results.
- Part of this public dataset from Roboflow, where the images containing the smallest bounding boxes has been removed.
- Part of the Flickr-Faces-HQ (FFHQ) (under Creative Commons BY 2.0 license) to rebalance the classes in our dataset.
- Google Colab to convert the Yolo v5 PyTorch format from the public dataset to Edge Impulse Ingestion format.
- A Raspberry Pi, NVIDIA Jetson Nano or with any Intel-based Macbooks to deploy the inference model.

Before we get started, here are some insights of the benefits / drawbacks of using a public dataset versus collecting your own.

Using a public dataset is a nice-to-have to start developing your application quickly, validate your idea and check the first results. But we often get disappointed with the results when testing on your own data and in real conditions. As such, for very specific applications, you might spend much more time trying to tweak an open dataset rather than collecting your own. Also, remember to always make sure that the license suits your needs when using a dataset you found online.

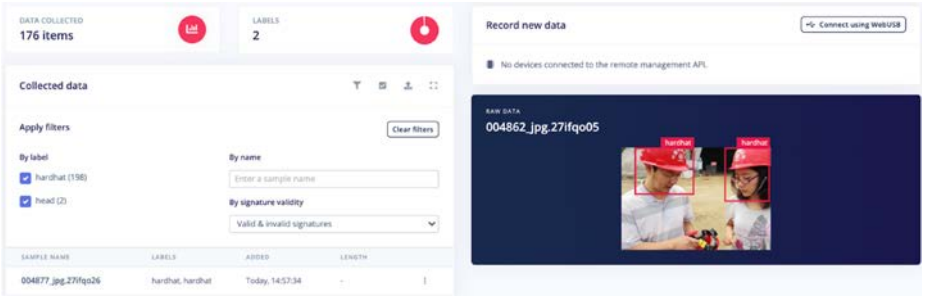
On the other hand, collecting your own dataset can take a lot of time, it is a repetitive task and most of the time annoying. But, it gives the possibility to collect data that will be as close as possible to your real life application, with the same lighting conditions, the same camera or the same angle for example. Therefore, your accuracy in your real conditions will be much higher.

Using only custom data can indeed work well in your environment but it might not give the same accuracy in another environment, thus generalization is harder.

The dataset which has been used for this project is a mix of open data, supplemented by custom data.

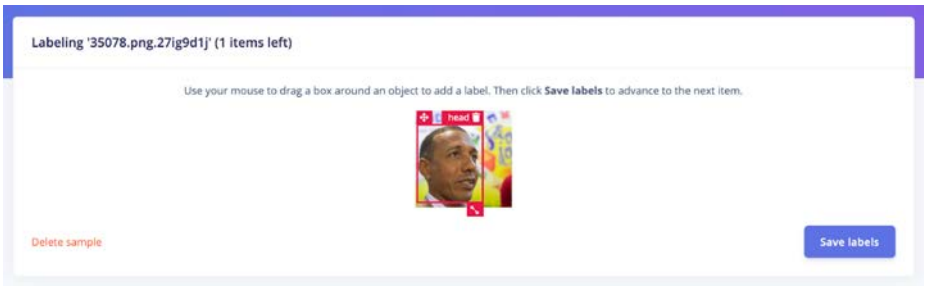
FIRST ITERATION, USING ONLY THE PUBLIC DATASETS

At first, we tried to train our model only using a small portion of this public dataset: 176 items in the training set and 57 items in the test set where we took only images containing a bounding box bigger than 130 pixels, we will see later why.

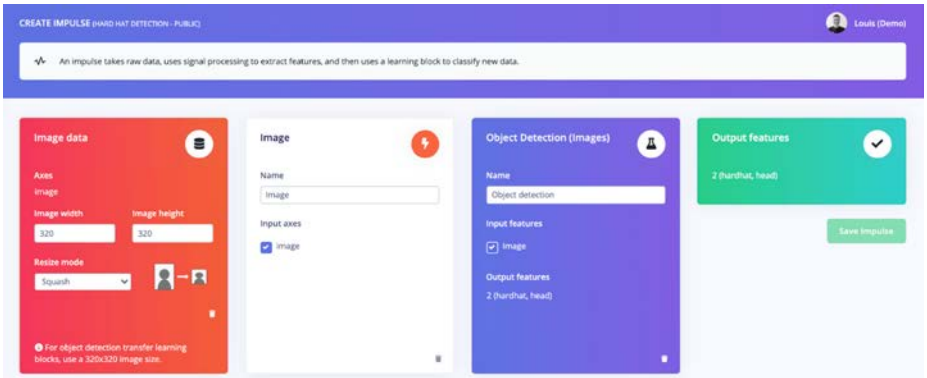


If you go through the public dataset, you can see that the entire dataset is strongly missing some “head” data samples. The dataset is therefore considered as imbalanced.

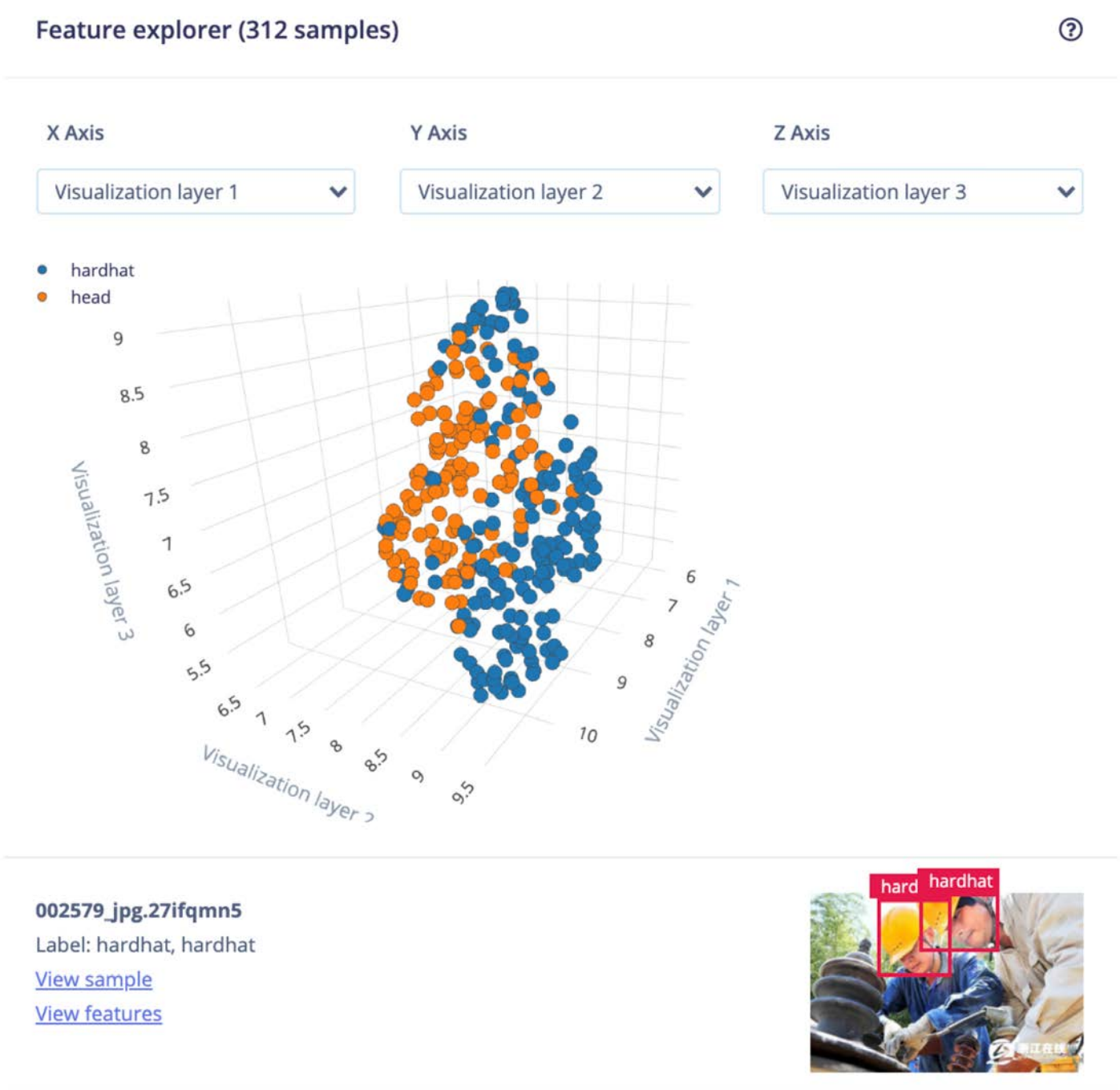
Several techniques exist to rebalance a dataset, here, we will add new images from Flickr-Faces-HQ (FFHQ). These images do not have bounding boxes but drawing them can be done easily in the Edge Impulse Studio. You can directly import them using the uploader portal. Once your data has been uploaded, just draw boxes around the heads and give it a label as below:



Now that the dataset is more balanced, with both images and bounding boxes of hard hats and heads, we can create an impulse, which is a mix of digital signal processing (DSP) blocks and training blocks:



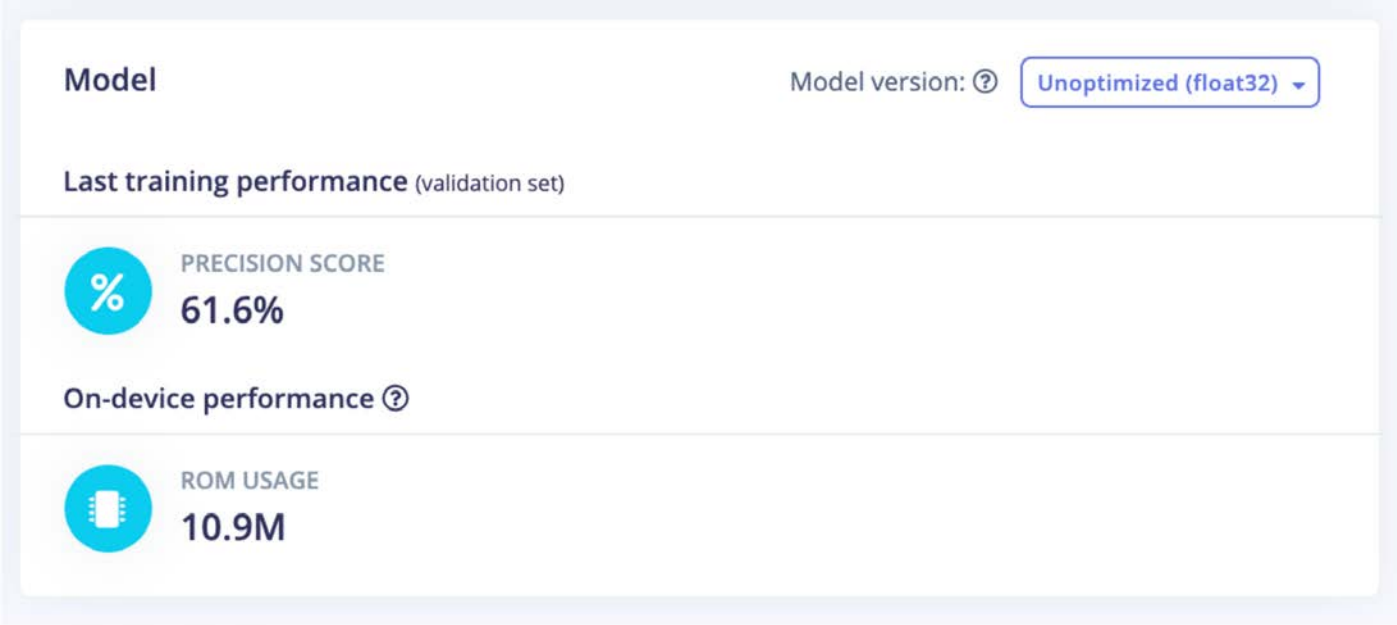
In this particular object detection use case, the DSP block will resize an image to fit the 320x320 pixels needed for the training block and extract meaningful features for the Neural Network. Although the extracted features don't show a clear separation between the classes, we can start distinguishing some clusters:



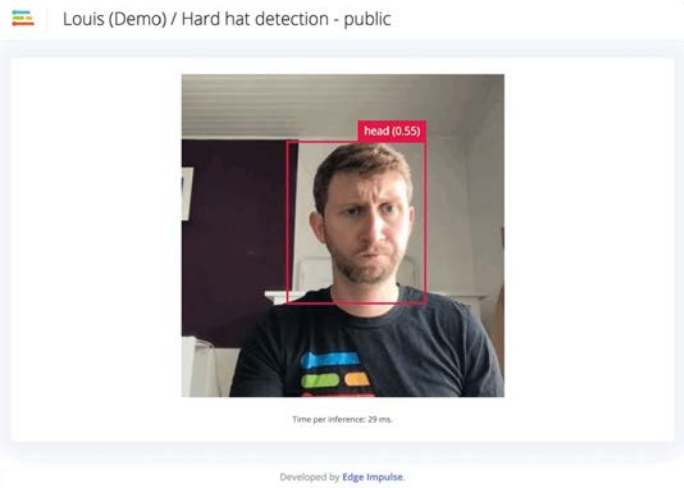
To train the model, we selected the Object Detection training block, which fine tunes a pre-trained object detection model on your data. It gives a good performance even with relatively small image datasets. This object detection learning block relies on MobileNetV2 SSD FPN-Lite 320x320.

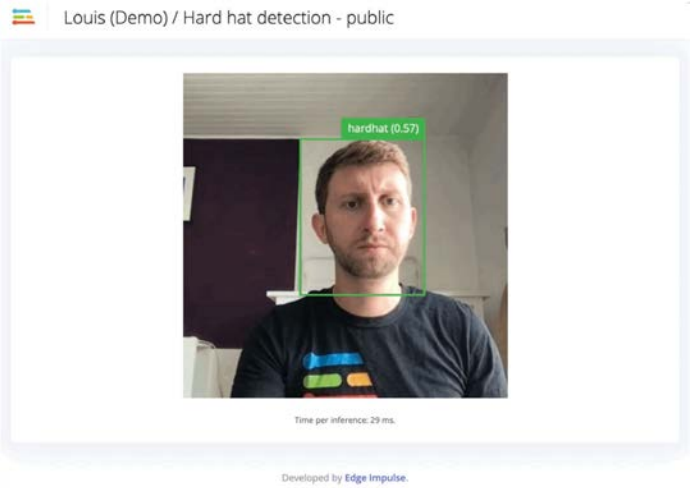
According to Daniel Situnayake, co-author of the TinyML book and founding TinyML engineer at Edge Impulse, this model "works much better for larger objects—if the object takes up more space in the frame it's more likely to be correctly classified." This has been one of the reason why we got rid of the images containing the smallest bounding boxes in our import script.

After training the model, we obtained a 61.6% accuracy on the training set and 57% accuracy on the testing set. You also might note a huge accuracy difference between the quantized version and the float32 version. However, during the linux deployment, the default model uses the unoptimized version. We will then focus on the float32 version only in this article.



This accuracy is not satisfying, and it tends to have trouble detecting the right objects in real conditions:

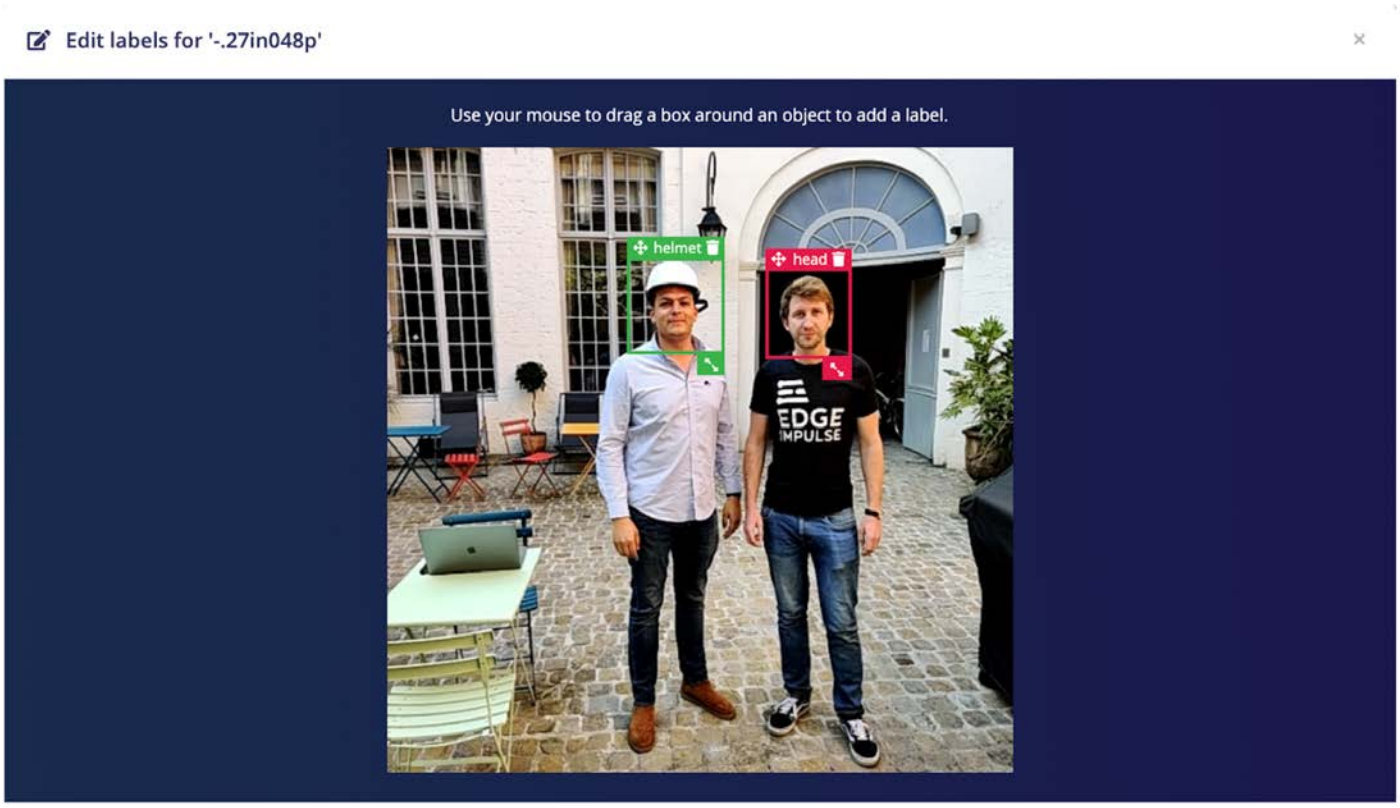




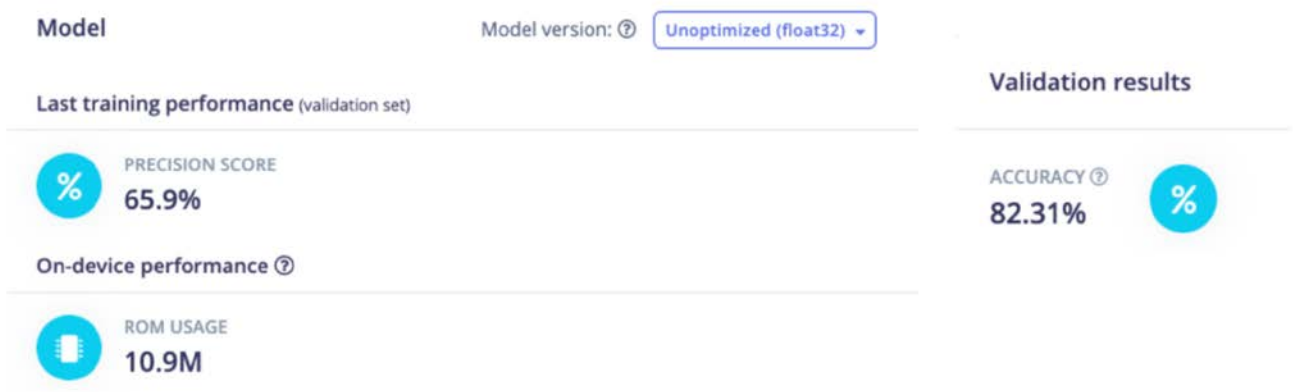
SECOND ITERATION, ADDING CUSTOM DATA

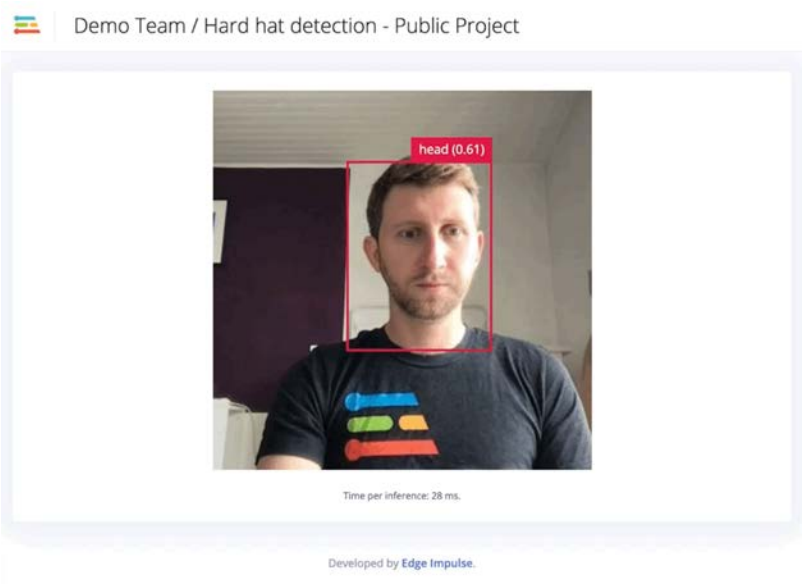
On the second iteration of this project, we have gone through the process of collecting some of our own data. A very useful and handy way to collect some custom data is using our mobile phone. You can also perform this step with the same camera you will be using in your factory or your construction site, this will be even closer to the real condition and therefore work best with your use case. In our case, we have been using a white hard hat when collecting data. For example, if your company uses yellow ones, consider collecting your data with the same hard hats.

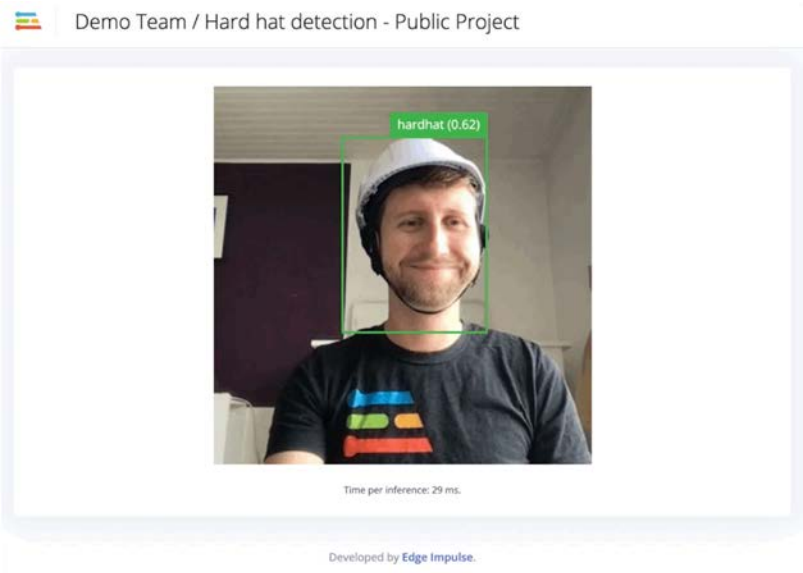
Once the data has been acquired, go through the labeling process again and retrain your model.



We obtain a model that is slightly more accurate when looking at the training performances. However, in real conditions, the model works far better than the previous one.







Finally, to deploy your model on your Raspberry Pi, NVIDIA Jetson Nano or your Intel-based Macbook, just follow the instructions provided in the links.

The command line interface `edge-impulse-linux-runner` will create a lightweight web interface where you can see the results. Note that the inference is run locally and you do not need any internet connection to detect your objects. Last but not least, the trained models and the inference SDK are open source. You can use it, modify it and integrate it to a broader application matching specifically to your needs such as stopping a machine when a head is detected for more than 10 seconds.

This project has been publicly released, feel free to have a look at it on Edge Impulse studio, clone the project and go through every steps to get a better understanding: https://studio.edgeimpulse.com/public/34898/latest

The essence of this use case is, Edge Impulse allows with very little effort to develop industry grade solutions in the health and safety context. Now this can be embedded in bigger industrial control and automation systems with a consistent and stringent focus on machine operations linked to H&S complaint measures. pre-trained models, which later can be easily retrained in the final industrial context as a step of "calibration," makes this a customizable solution for your next project.

EDGE IMPULSE

hello@edgeimpulse.com

Edge Impulse
3031 Tisch Way
110 Plaza West
San Jose, CA 95128
USA