ML-based Image Classifier at IOT-Edge

Dr. Rajiv Misra

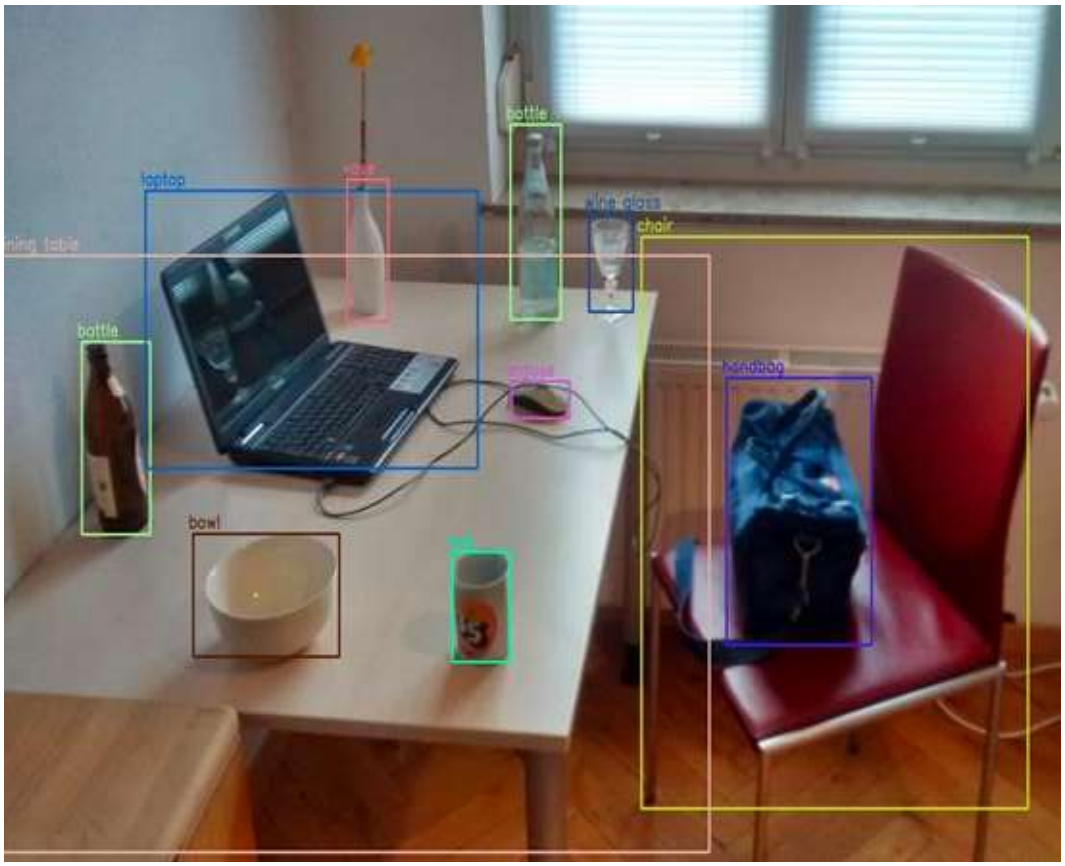
Professor, **Dept. of Computer Science & Engg. Indian Institute of Technology Patna** rajivm@iitp.ac.in

After Completion of this lecture you will knowing the following:

- Basics of computer vision in ML
- Different techniques of computer vision like image classification, detection, segmentation, etc
- Object detection models like RCNN, Fast RCNN, Faster RCNN, SDD, YOLO
- Azure compute vision as SaaS

Computer Vision: Introduction

Computer vision is a sub branch of machine learning which deals with giving computers the ability to see and interpret and extract information from images and videos, videos can be seen as collection of images.



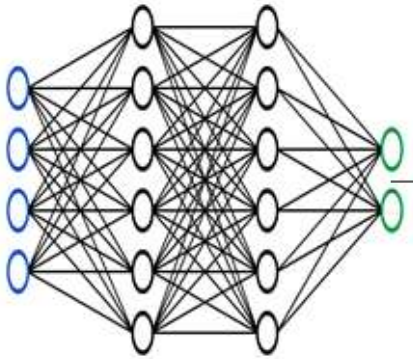
How Computer Vision Works?

To train a computer vision model, you essentially feed some thousands of images of cats and it's going to do some complex mathematics and feature extraction etc in the background.

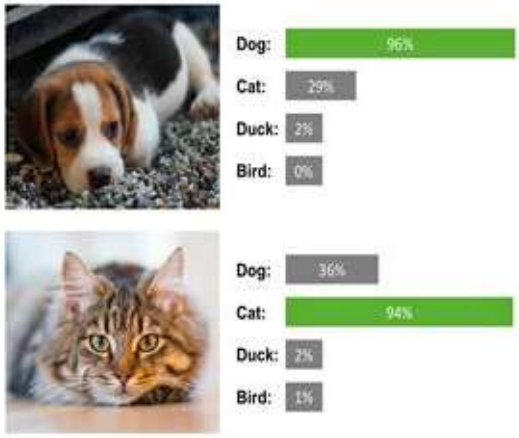
Based on that it learns some key understanding or properties that define cats.

Thousands of cats images

Input



Feature extraction



Output

Computer Vision: Data Analytics View

With every machine learning model, the model is not only the important part.

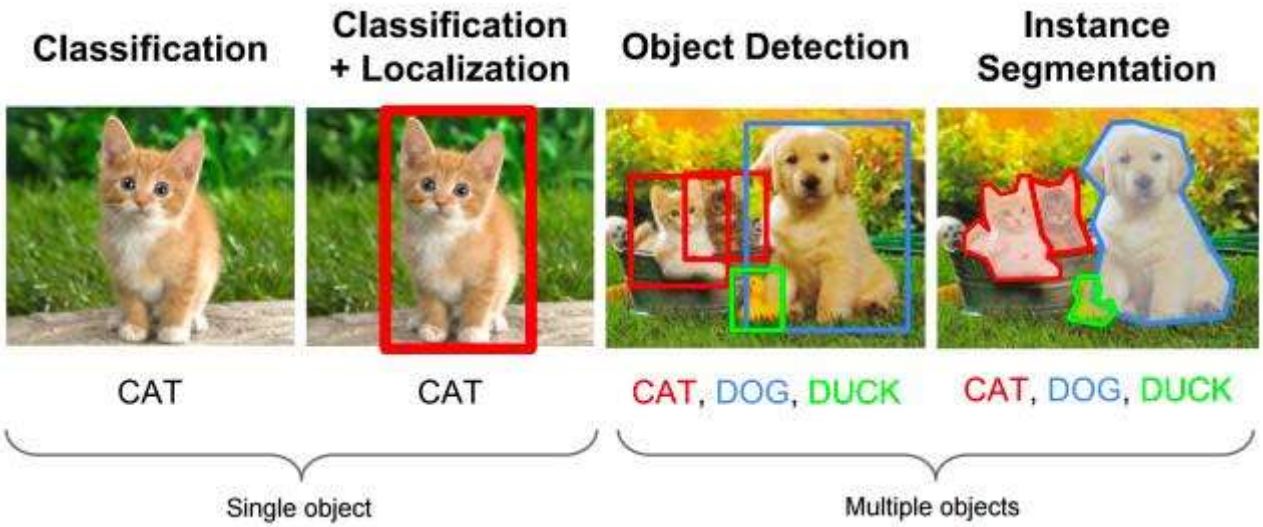
The fundamental fact that's going to determine how good your model is the data you feed it.

Today, this is another point that we want to focus on is the fact that your model is only as good as your data.

So one of the key things that we're going to focus on today's lecture is how to make sure that our data is good when we're building computer vision models.

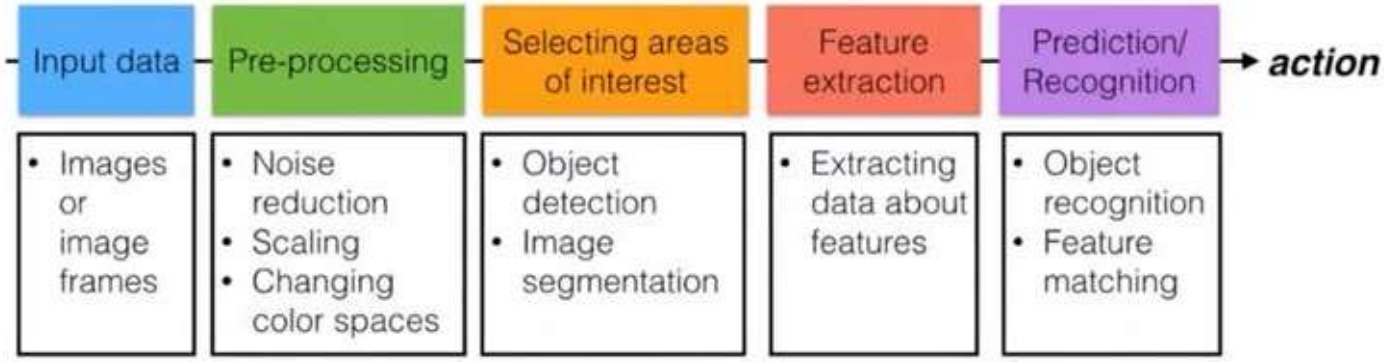
Computer Vision: Techniques

Computer vision deals with all the problems related to images and videos. There's a lot of techniques, fundamentals or problems that can be tackled with computer vision. A few of them includes image classification, image detection, image segmentation, pattern detection and object localisation. So object detection and image classification are the two things that we're going to talk about today and we're going to go into some of the predictive model build for an object detection model.



Computer Vision: Architecture

A typical end-to-end pipeline of computer vision architecture is shown below.



Input data, the images that you want to train your model on, but those images might be coming from a lot of different sources.

Computer Vision Architecture: Pre-processing

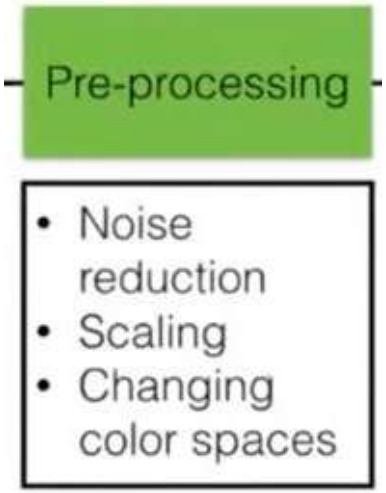
The second and important step is pre-processing of input data.

Machine learning depends on standardization, means you need to pre-process input images to make sure that they're all of the same size.

There might be some noise in the images, all of that needs to be dealt with before the image fed into the model.

If it's not done correctly, the model might learn noise or other features that are not good features or it might learn from those that actually mean that your model is going to be fundamentally flawed.

Therefore, pre-processing is essentially it's very important that need to be.

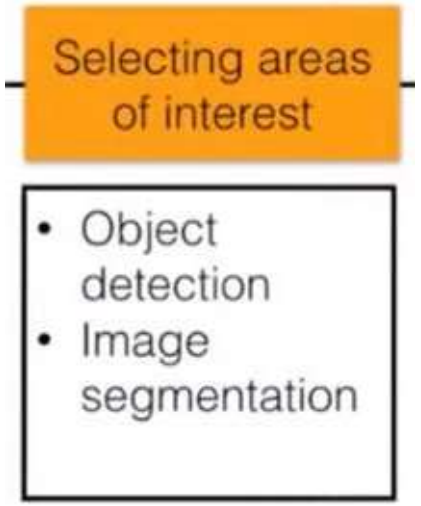


Computer Vision Architecture: Data labeling

The third step is labeling your data.

For an object detection problem, you have images with different objects, let's say if you have an image with a cat and a dog, you would need to label that specific part of the image where there's a cat and a dog.

This label or tags to that specific area where there's a dog or cat, so this is essentially labeling and this needs to be done as well.

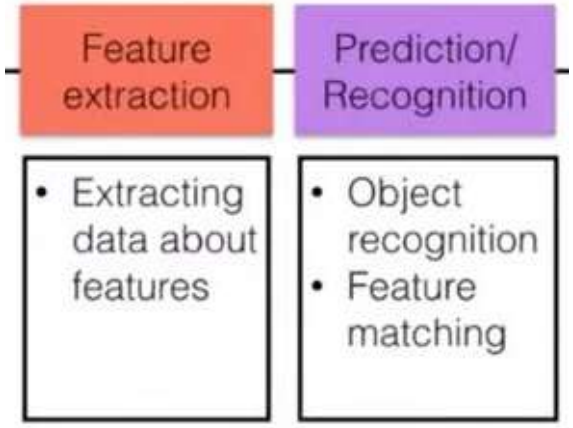


Computer Vision Architecture: Feature extraction and prediction

Then feature extraction and prediction part performed by a machine learning model.

These are part of the model training, the model learns about what features are present, it extracts the features.

And if features are relevant from the images, then those features are learned along with the patterns and later the model uses it to build a sort of rules for itself and these rules are used to predict the output.

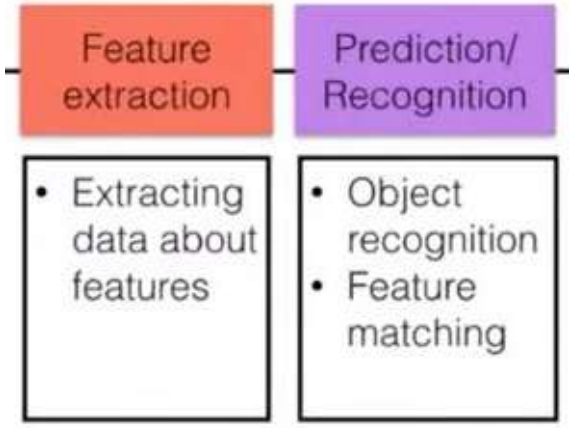


Computer Vision Architecture: Feature extraction and prediction

Then feature extraction and prediction part performed by a machine learning model.

These are part of the model training, the model learns about what features are present, it extracts the features.

And if features are relevant from the images, then those features are learned along with the patterns and later the model uses it to build a sort of rules for itself and these rules are used to predict the output.



Computer Vision: Object Detection Models

The field of object detection is not as new as it may seem. In fact, object detection has evolved over the past 20 years. Popular deep learning algorithm that achieved remarkable results in this domain are:

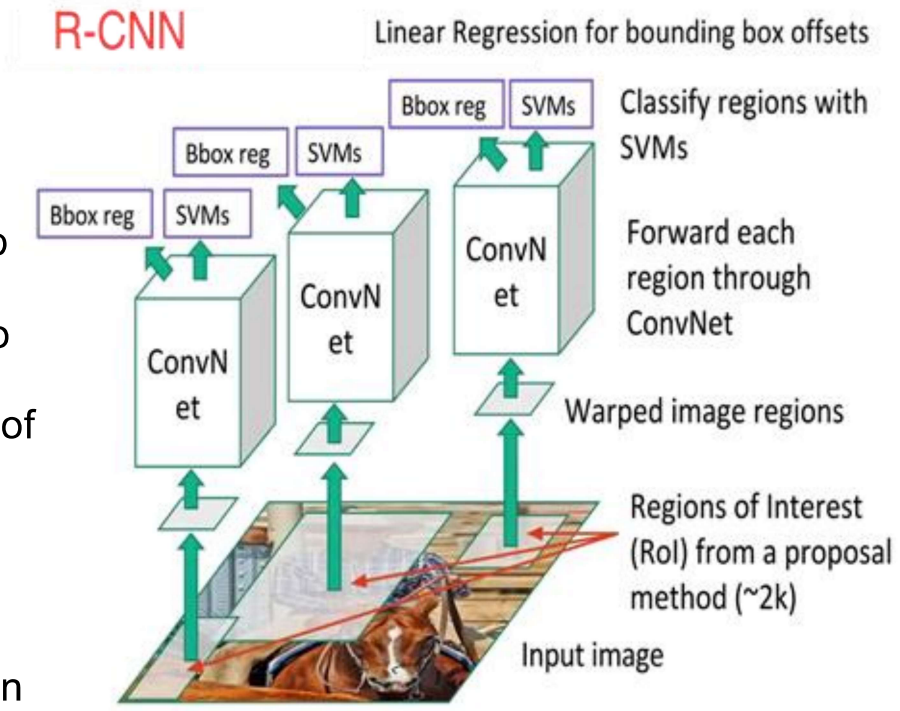
- RCNN
- Fast RCNN
- Faster RCNN
- YOLO
- SSD (Single Shot Detector)

Object Detection Model: RCNN

R-CNN, or Region-based Convolutional Neural Network, consisted of 3 simple steps:

1. Scan the input image for possible objects using an algorithm called Selective Search, generating ~2000 region proposals
2. Run a convolutional neural net (CNN) on top of each of these region proposals
3. Take the output of each CNN and feed it into a) an SVM to classify the region and b) a linear regressor to tighten the bounding box of the object, if such an object exists.

In other words, we first propose regions, then extract features, and then classify those regions based on their features. In essence, we have turned object detection into an image classification problem. R-CNN was very intuitive, but very slow.

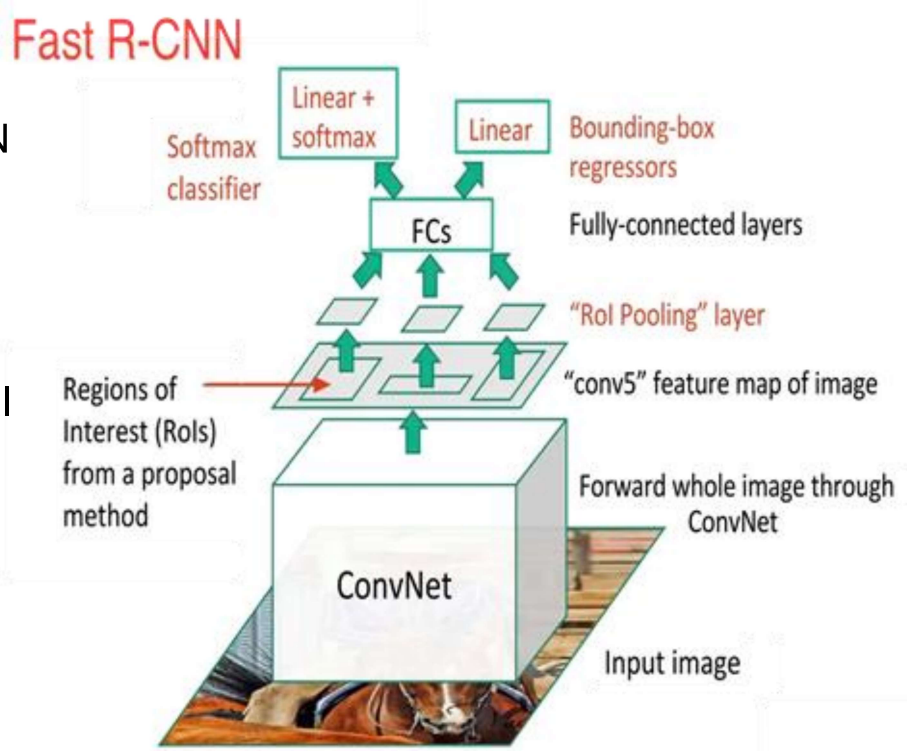


Object Detection Model: Fast RCNN

As we can see from the image, we are now generating region proposals based on the last feature map of the network, not from the original image itself. As a result, we can train just one CNN for the entire image.

In addition, instead of training many different SVM's to classify each object class, there is a single softmax layer that outputs the class probabilities directly. Now we only have one neural net to train, as opposed to one neural net and many SVM's.

Fast R-CNN performed much better in terms of speed. There was just one big bottleneck remaining: the selective search algorithm for generating region proposals.



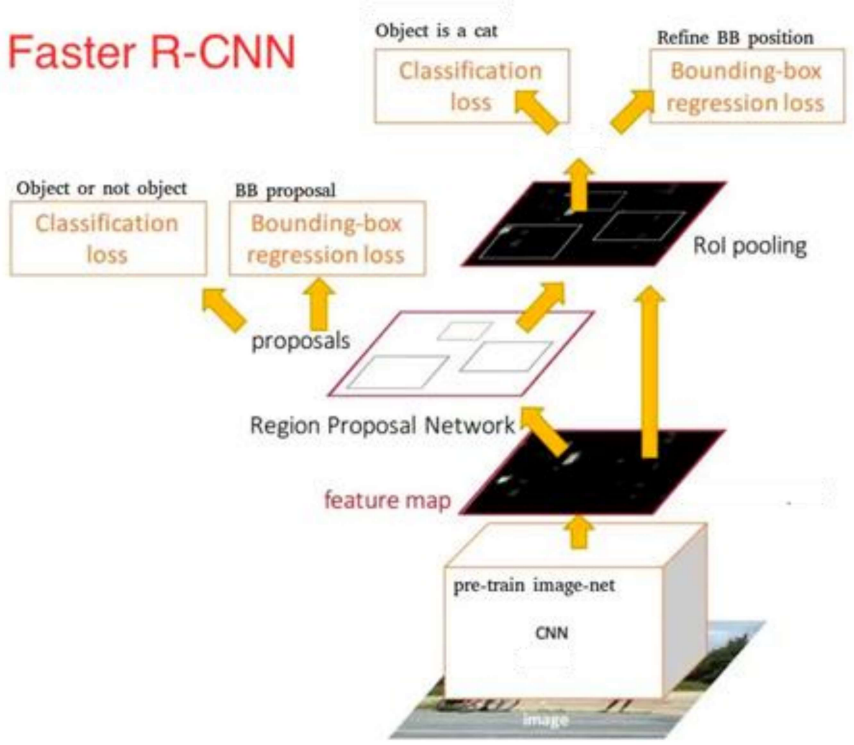
Object Detection Model: Faster RCNN

The main insight of Faster R-CNN was to replace the slow selective search algorithm with a fast neural net. Specifically, it introduced the region proposal network (RPN).

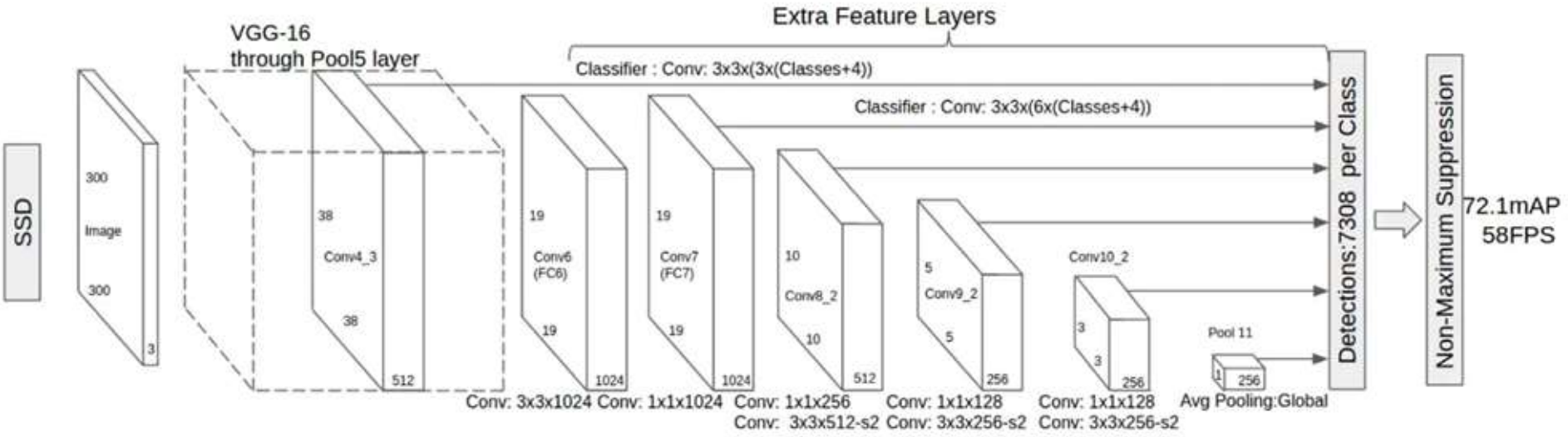
Here's how the RPN worked:

- At the last layer of an initial CNN, a 3x3 sliding window moves across the feature map and maps it to a lower dimension (e.g. 256-d)
- For each sliding-window location, it generates multiple possible regions based on k fixed-ratio anchor boxes (default bounding boxes)
- Each region proposal consists of a) an "objectness" score for that region and b) 4 coordinates representing the bounding box of the region

In other words, we look at each location in our last feature map and consider k different boxes centered around it: a tall box, a wide box, a large box, etc. For each of those boxes, we output whether or not we think it contains an object, and what the coordinates for that box are.



Object Detection Model: SSD



Object Detection Model: SSD

SSD stands for Single-Shot Detector. Like R-FCN, it provides enormous speed gains over Faster R-CNN, but does so in a markedly different manner.

Our first two models performed region proposals and region classifications in two separate steps. First, they used a region proposal network to generate regions of interest; next, they used either fully-connected layers or position-sensitive convolutional layers to classify those regions. SSD does the two in a "single shot," simultaneously predicting the bounding box and the class as it processes the image.

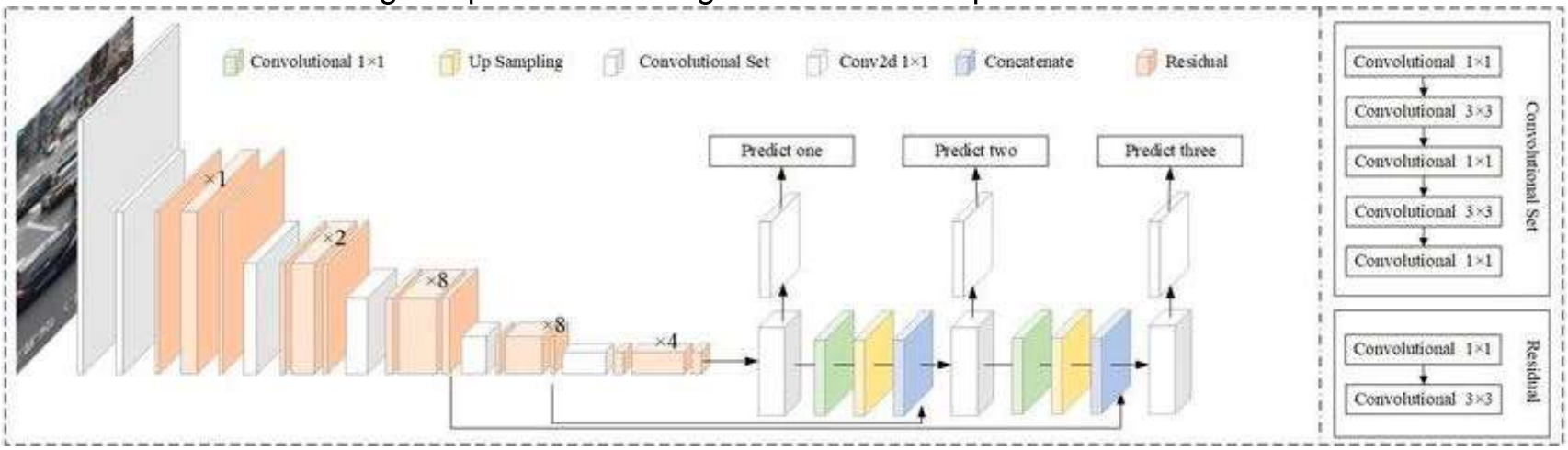
Given an input image and a set of ground truth labels, SSD does the following:

- Pass the image through a series of convolutional layers, yielding several sets of feature maps at different scales (e.g. 10x10, then 6x6, then 3x3, etc.)
- For each location in each of these feature maps, use a 3x3 convolutional filter to evaluate a small set of default bounding boxes. These default bounding boxes are essentially equivalent to Faster R-CNN's anchor boxes.
- For each box, simultaneously predict a) the bounding box offset and b) the class probabilities
- During training, match the ground truth box with these predicted boxes based on IoU. The best predicted box will be labeled a "positive," along with all other boxes that have an IoU with the truth >0.5.

Object Detection Model: YOLOv3

You Only Look Once or more popularly known as YOLO is one of the fastest real-time object detection algorithm (45 frames per second) as compared to the R-CNN family (R-CNN, Fast R-CNN, Faster R-CNN, etc.)

The R-CNN family of algorithms uses regions to localise the objects in images which means the model is applied to multiple regions and high scoring regions of the image are considered as object detected. But YOLO follows a completely different approach. Instead of selecting some regions, it applies a neural network to the entire image to predict bounding boxes and their probabilities.



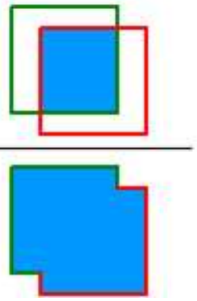
Object Detection Models: Performance Metric

An overview of the most popular metrics used to compare performance of different deep learning models:

Intersection Over Union (IOU)

Intersection Over Union (IOU) is a measure based on Jaccard Index that evaluates the overlap between two bounding boxes. IOU is given by the overlapping area between the predicted bounding box and the ground truth bounding box divided by the area of union between them:

$$\text{IOU} = \frac{\text{area}\,(B_p \cap B_{gt})}{\text{area}\,(B_p \cup B_{gt})} = \frac{\text{area of overlap}}{\text{area of union}} =$$



Object Detection Models: Performance Metric

Precision:

Precision is the ability of a model to identify only the relevant objects. It is the percentage of correct positive predictions and is given by:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{all detections}}$$

Recall:

Recall is the ability of a model to find all the relevant cases (all ground truth bounding boxes). It is the percentage of true positive detected among all relevant ground truths and is given by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{all ground truths}}$$

True Positive (TP): A correct detection. Detection with IOU ≥ threshold
False Positive (FP): A wrong detection. Detection with IOU < threshold
False Negative (FN): A ground truth not detected

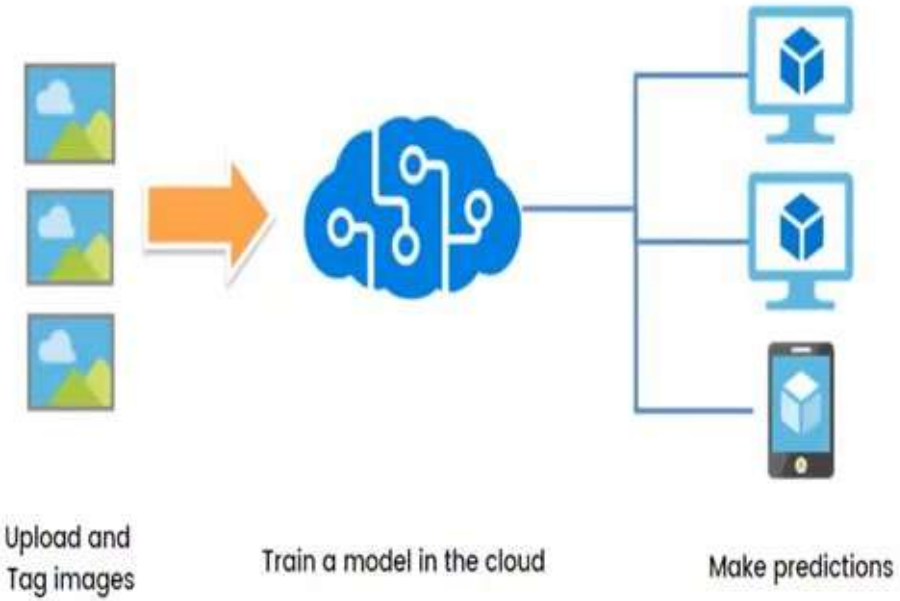
Computer Vision: SaaS Architecture

A computer vision architecture can easily be taken up by a cloud service that is running a computer vision model in the cloud.

A SaaS is a software as a service that is offered by all providers like azure, amazon aws, google cloud. All of them offers some variation of these for computer vision service.

In that architecture, all you need to do is you need to have your images and upload them and tag them. Tagging is vital because you as the domain expert know what information is present in the images.

Once you've uploaded them in the cloud, the model training and everything that is completely dependent on the cloud provider and fully managed by the cloud service provider.



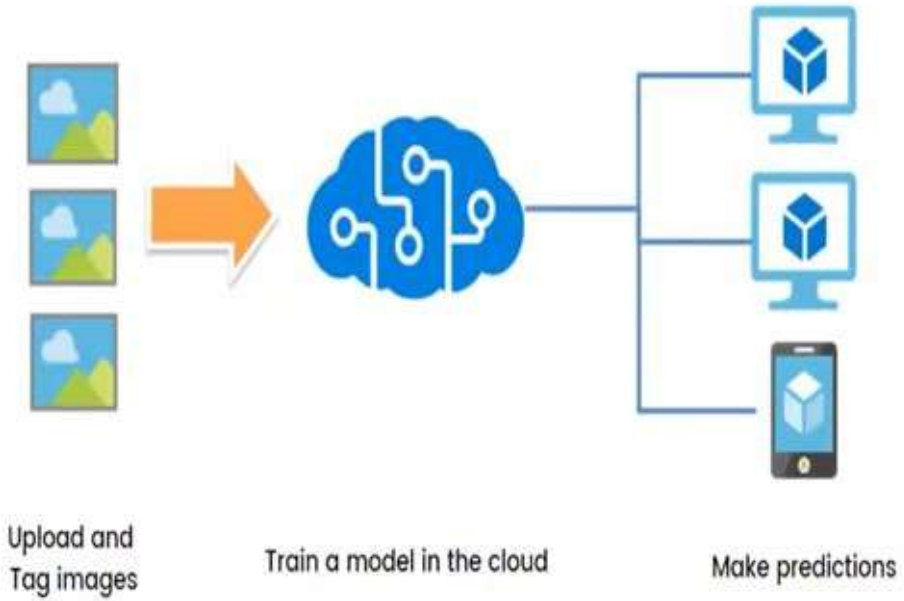
Computer Vision: SaaS Architecture

It's extremely easy to scale up your dataset and allow you to download the models that you've built that can later be used offline.

Once you've trained and download the model, simply use the rest API to query that model and get the predictions which is extremely reliable and simple.

The SaaS architecture provided by different cloud provider offers similar services.

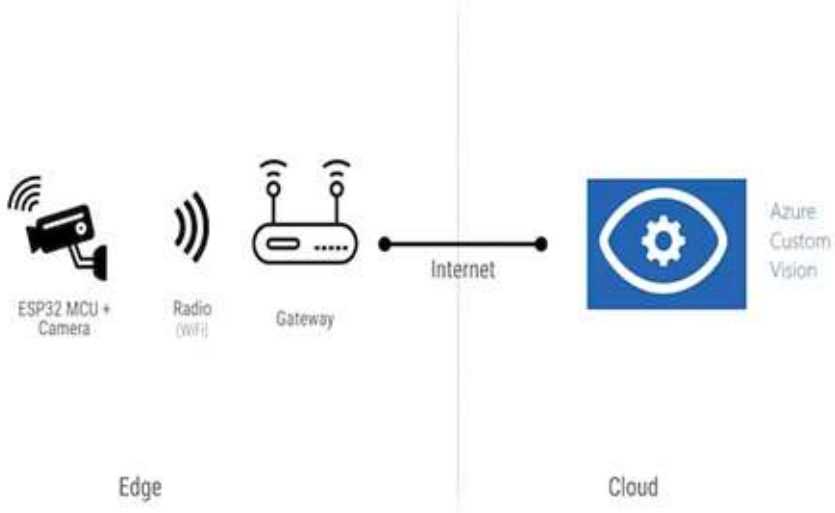
There might be fundamental differences in the ui or how you are uploading images or the api or how you're calling the services but under the hood they're doing the same thing.



SaaS: Azure Custom Vision

Azure Custom Vision is a cloud service used to build and deploy computer vision models.

Custom Vision uses a pretty interesting neural network technique called transfer learning, which applies knowledge gained from solving one problem to a different, but related situation. This can substantially decrease the time needed for creating the models.

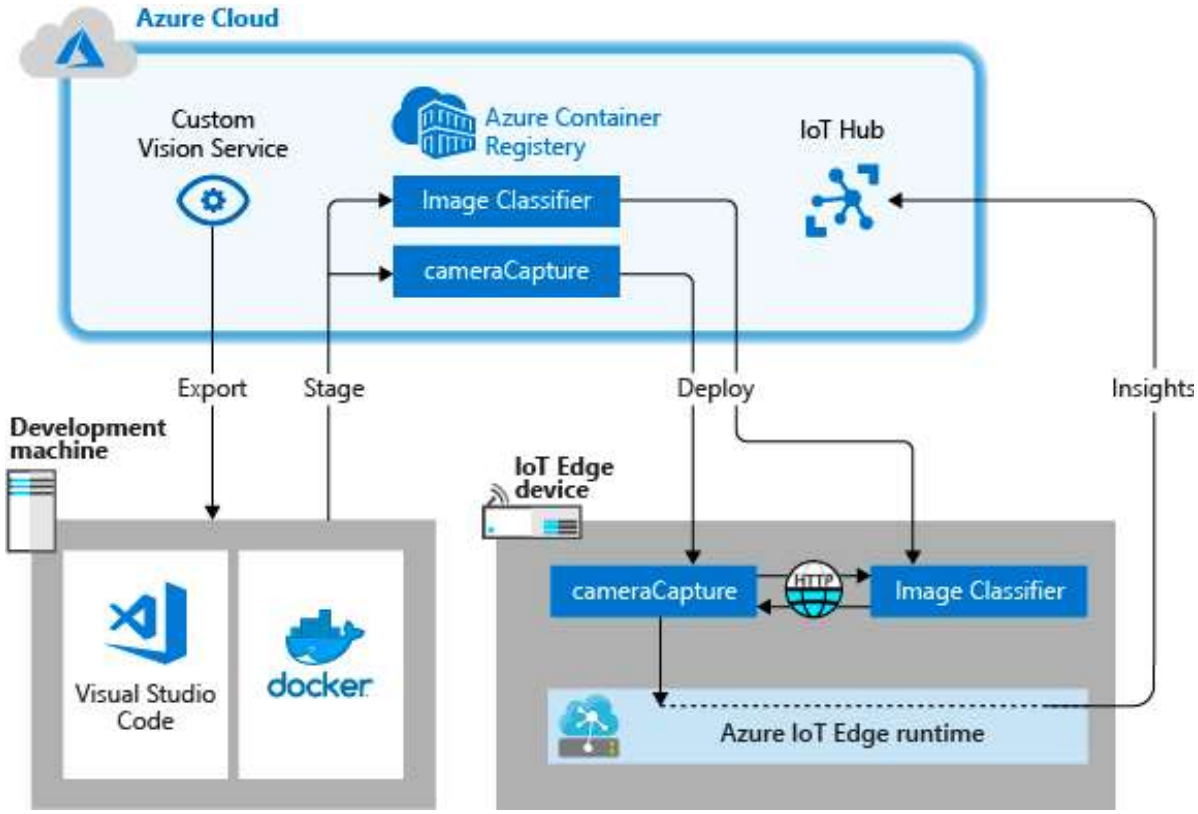


Features provided by Azure Custom Vision service:

- Train a computer vision model by simply uploading and labeling few images.
- Build image classifier model using code-free and code-first approach.
- Deploy the model in the cloud on-premise, or on edge devices.

Azure Custom Vision on an IoT Edge device

- Build an image classifier with Custom Vision.
- Develop an IoT Edge module that queries the Custom Vision web server on device.
- Send the results of the image classifier to IoT Hub.



Use Case: Creating an image recognition solution with Azure IoT Edge and Azure Cognitive Services

Although there are lots of applications for image recognition but we had chosen this application which is a solution for vision impaired people scanning fruit and vegetables at a self-service checkout.

Required Components

Raspberry Pi 3B or better, USB Camera, and a Speaker.

Note, the solution will run on a Raspberry Pi 3A+, it has enough processing power, but the device is limited to 512MB RAM. A Raspberry Pi 3B+ has 1GB of RAM and is faster than the older 3B model. Azure IoT Edge requires an ARM32v7 or better processor. It will not run on the ARM32v6 processor found in the Raspberry Pi Zero.

Desktop Linux - such as Ubuntu 18.0

This solution requires USB camera pass through into a Docker container as well as Azure IoT Edge support. So for now, that is Linux.

Guide for installing Raspberry Pi

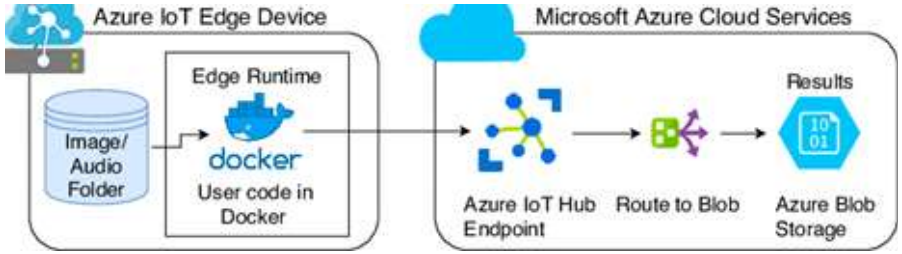
Set up **Raspbian Stretch Lite** on Raspberry Pi: Be sure to configure the correct Country Code in your wpa_supplicant.conf file.

Azure subscription: If you don't already have an Azure account then sign up for a free Azure account. If you are a student then sign up for an Azure for Students account, no credit card required.

Create an **Azure IoT Hub**, and an **Azure IoT Edge device**: Install Azure IoT Edge runtim on Raspberry Pi and download the deployment configuration file that describes the Azure IoT Edge Modules and Routes for this solution. Open the deployment.arm32v7.json link and save the deployment.arm32v7.json in a known location on your computer.

Install **Azure CLI** and **Azure CLI command line tools**: With CLI open a command line console/terminal and change directory to the location where you saved the deployment.arm32v7.json file.

Deploy edge Iot to device: The modules will now start to deploy to your Raspberry Pi, the Raspberry Pi green activity LED will flicker until the deployment completes. Approximately 1.5 GB of Dockers modules will be downloaded and decompressed on the Raspberry Pi. This is a one off operation.



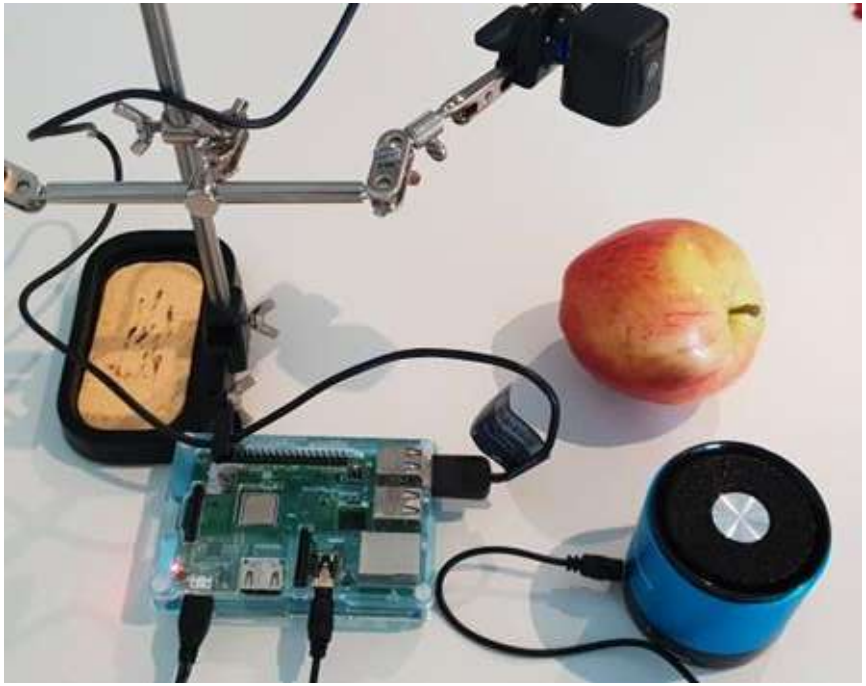
Considerations and constraints for the solution

The solution should scale from a Raspberry Pi (running Raspbian Linux) on ARM32v7, to my desktop development environment, to an industrial capable IoT Edge device such as those found in the Certified IoT Edge Catalog.

The solution needs camera input, uses a USB Webcam for image capture as it was supported across all target devices.

The camera capture module needed Docker USB device pass-through (not supported by Docker on Windows) so that plus targeting Raspberry Pi meant that need to target Azure IoT Edge on Linux.

To mirror the devices plus targeting, ir requires Docker support for the USB webcam, so develop the solution on Ubuntu 18.04 developer desktop.

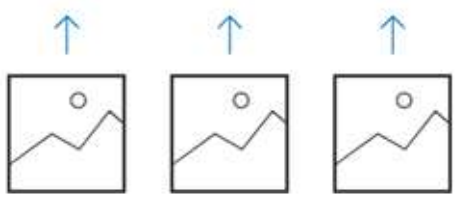


Creating the Fruit Classification Model

The **Azure Custom Vision** service is a simple way to create an image classification machine learning model without having to be a data science or machine learning expert.

You simply upload multiple collections of labelled images. For example, you could upload a collection of banana images and label them as ‘banana’.

It is important to have a good variety of labelled images so be sure to improve your classifier.



Upload Images

Bring your own labeled images, or use Custom Vision to quickly add tags to any unlabeled images.

Train

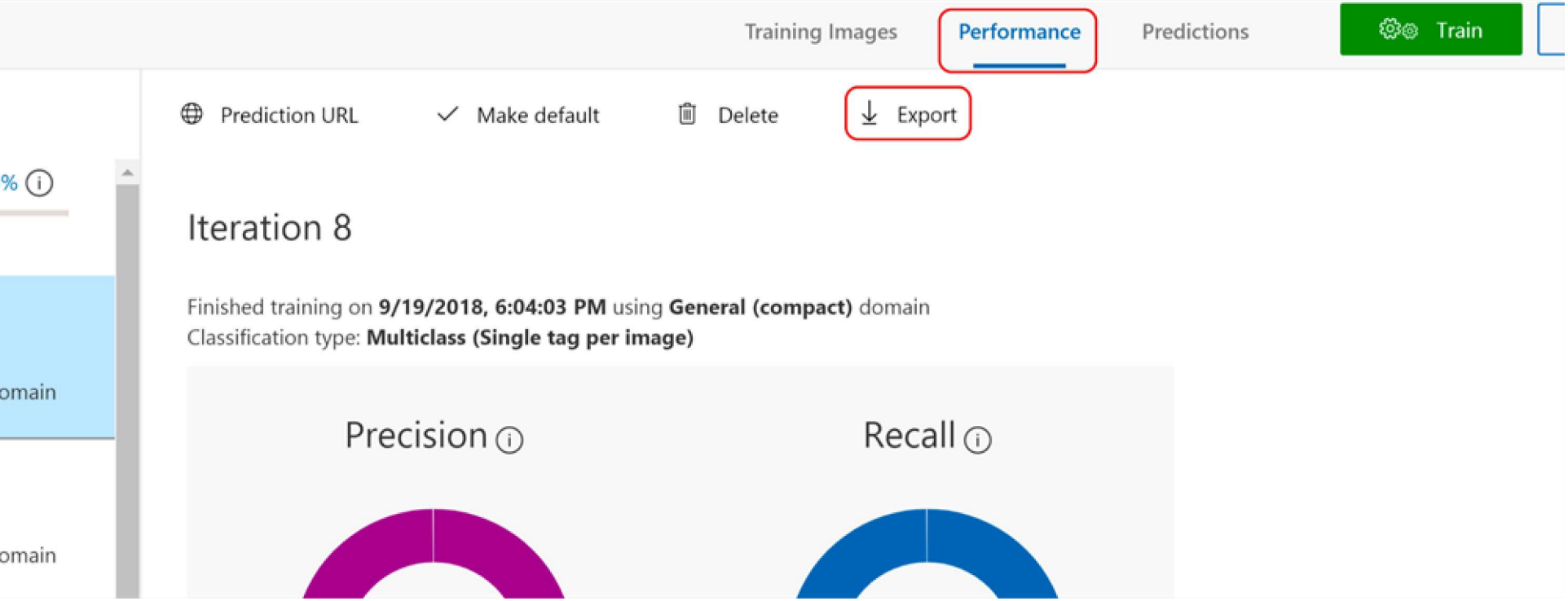
Use your labeled images to teach Custom Vision the concepts you care about.

Evaluate

Use simple REST API calls to quickly tag images with your new custom computer vision model.

Steps to export your Custom Vision project model.

Step 1: From the Performance tab of your Custom Vision project click Export.

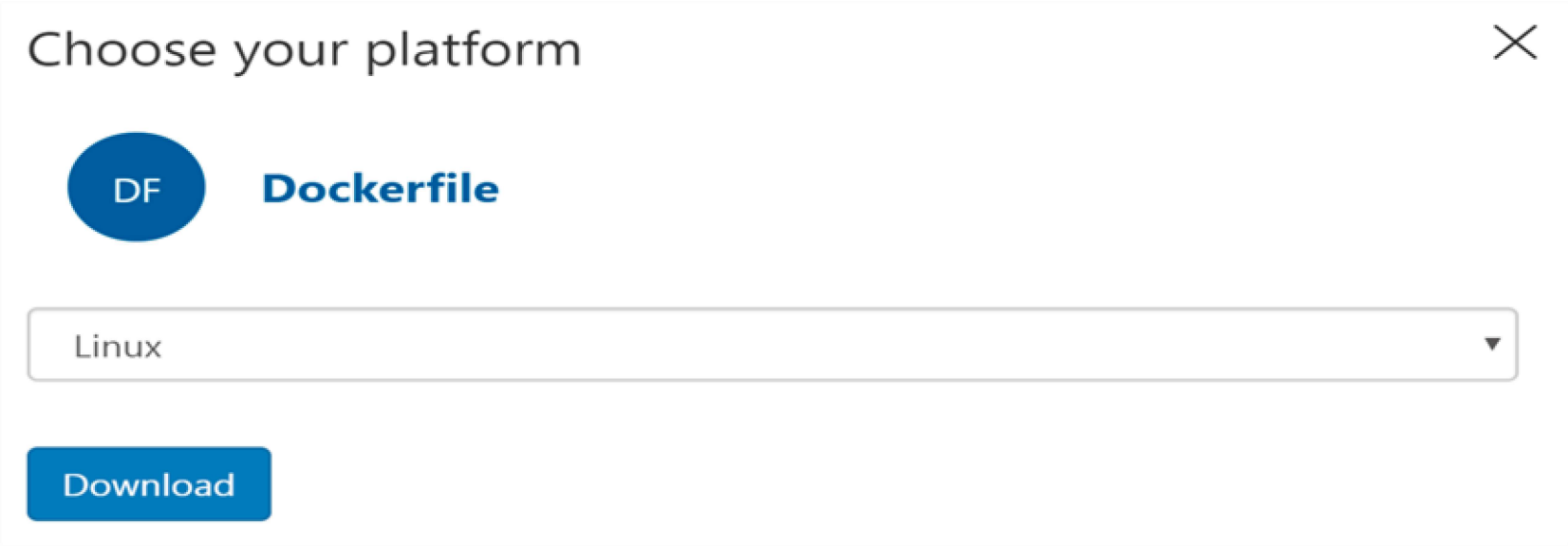


Steps to export your Custom Vision project model.

Step 2: Select Dockerfile from the list of available options

Steps to export your Custom Vision project model.

Step 3: Then select the Linux version of the Dockerfile.



Step 4: Download the docker file and unzip and you have a ready-made Docker solution with a Python Flask REST API. This was how the Azure IoT Edge Image Classification module is created in this solution.

Installing the solution

Step 1: Clone the repository for creating an image recognition solution with Azure IoT Edge and Azure Cognitive Services.

Step 2: Install the Azure IoT Edge runtime on your Linux desktop or device (eg Raspberry Pi).

Step 3: Install the following software development tools.

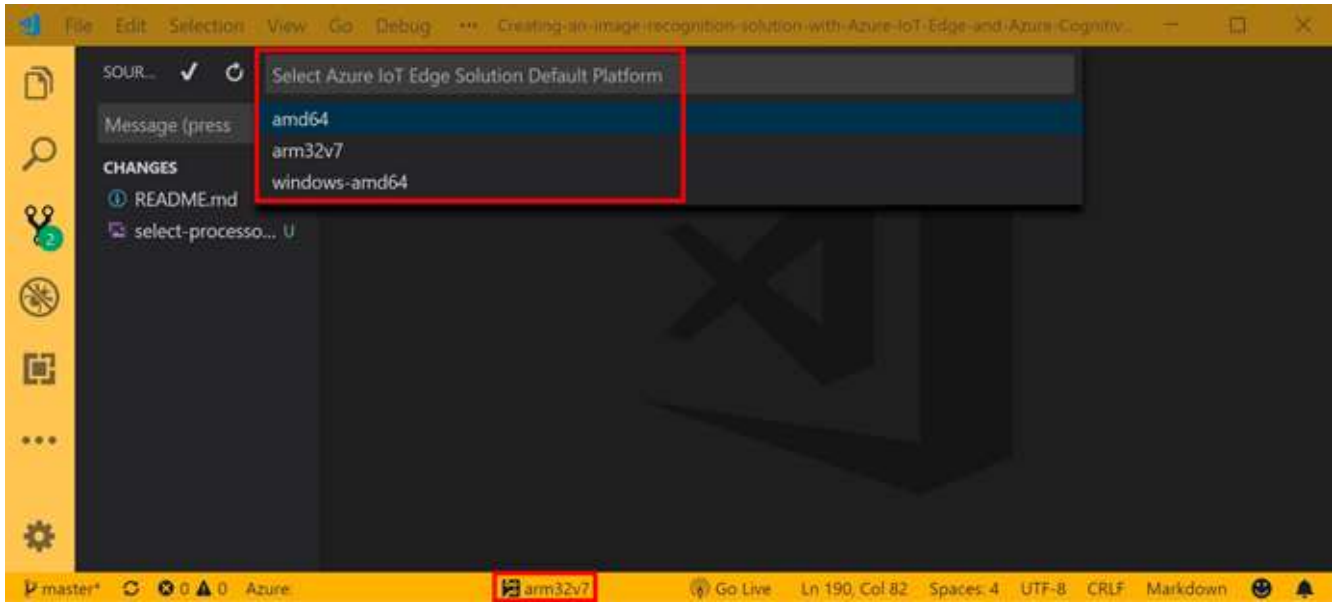
- Visual Studio Code
- Plus, the following Visual Studio Code Extensions
  - Azure IoT Edge
  - JSON Tools useful for changing the “Create Options” for a module.
- Docker Community Edition on your development machine

Step 4: With Visual Studio Code, open the IoT Edge solution you cloned to your developer desktop.

Building the Solution

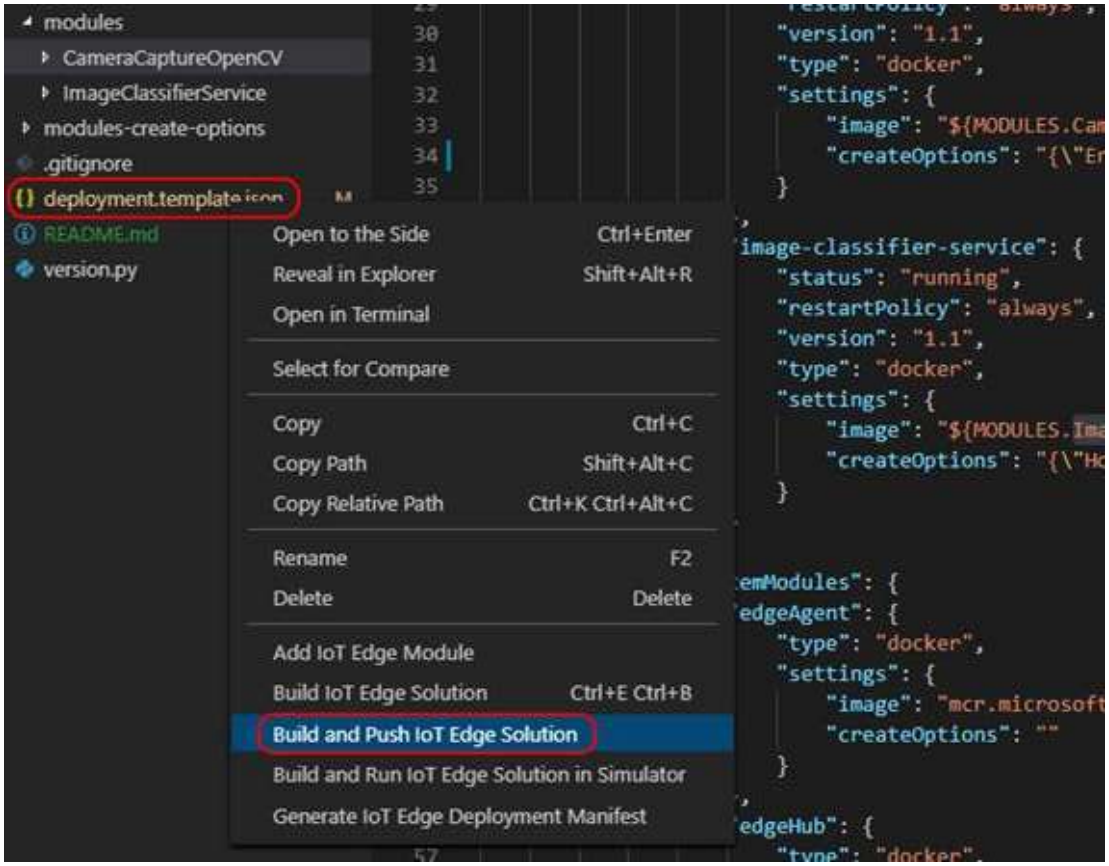
Step 1: Pushing the image to a local Docker repository with specifying the localhost.

Step 2: Confirm processor architecture using the Visual Studio Code



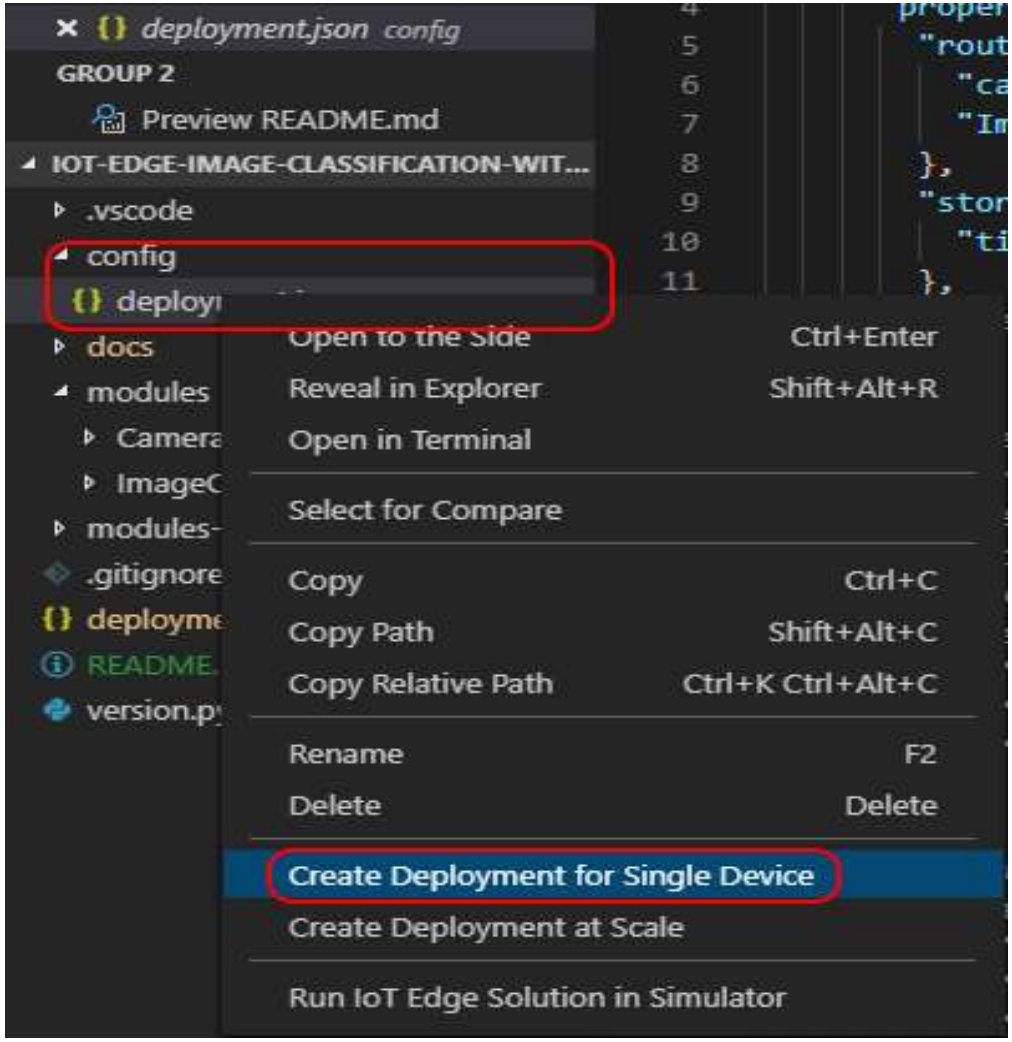
Building the Solution

Step 3: Build and Push the solution to Docker by right mouse clicking the deployment.template.json file and select “Build and Push IoT Edge Solution”.



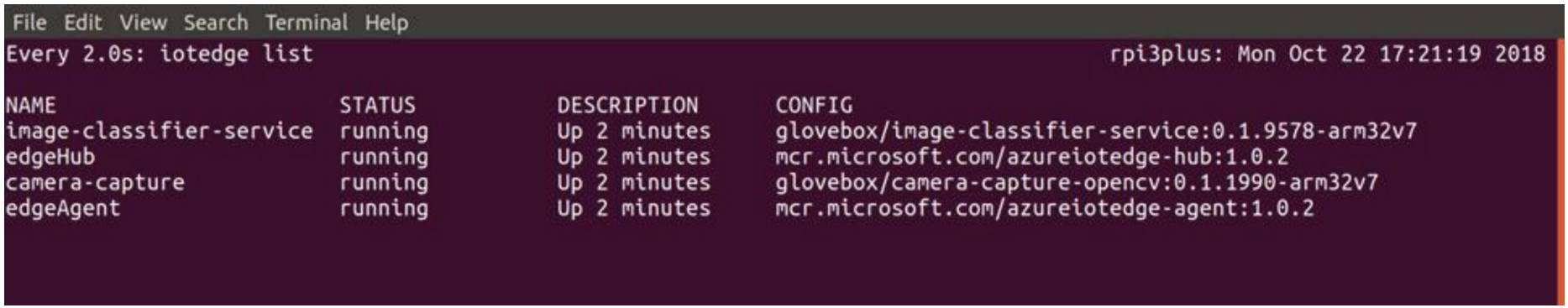
Deploying the Solution

When the Docker Build and Push process has completed select the Azure IoT Hub device you want to deploy the solution to. Right mouse click the deployment.json file found in the config folder and select the target device from the drop-down list.



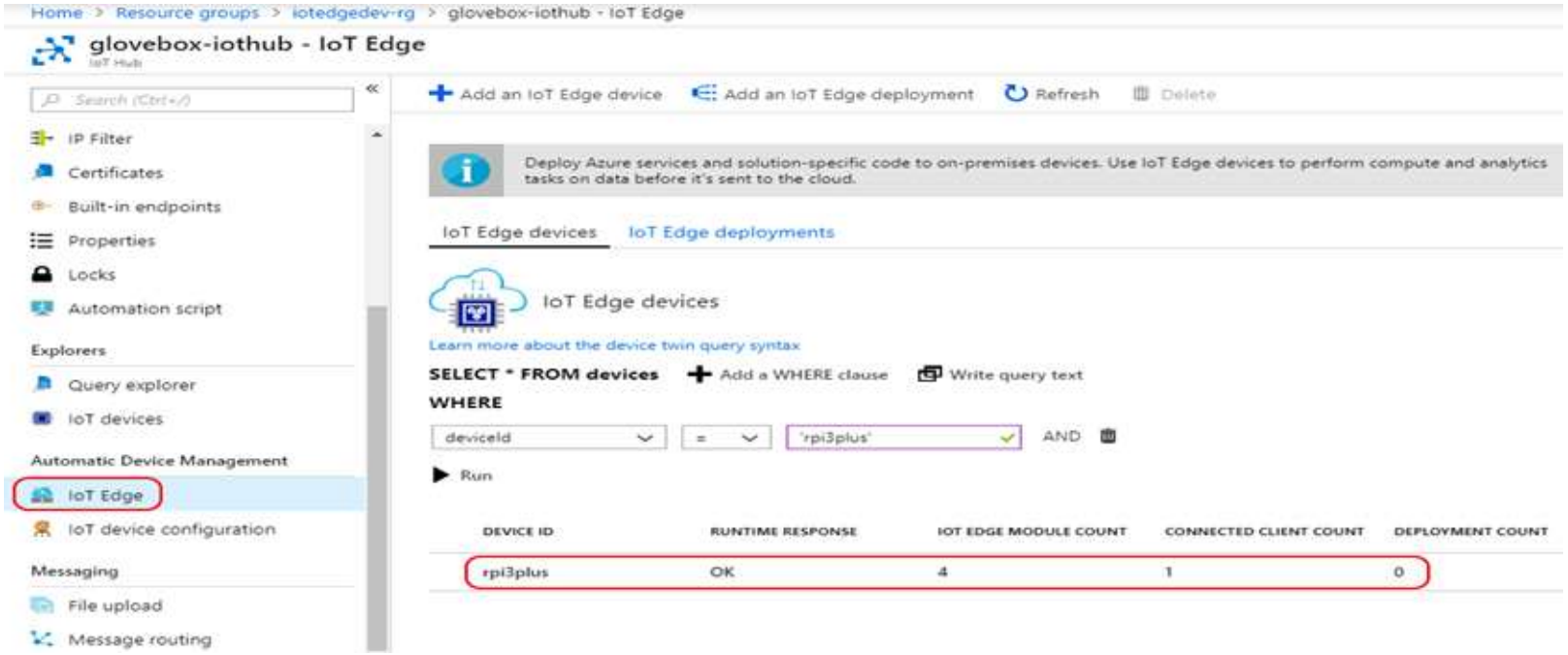
Monitoring the Solution on the IoT Edge Device

Once the solution has been deployed you can monitor it on the IoT Edge device itself using the *iotedge list* command.



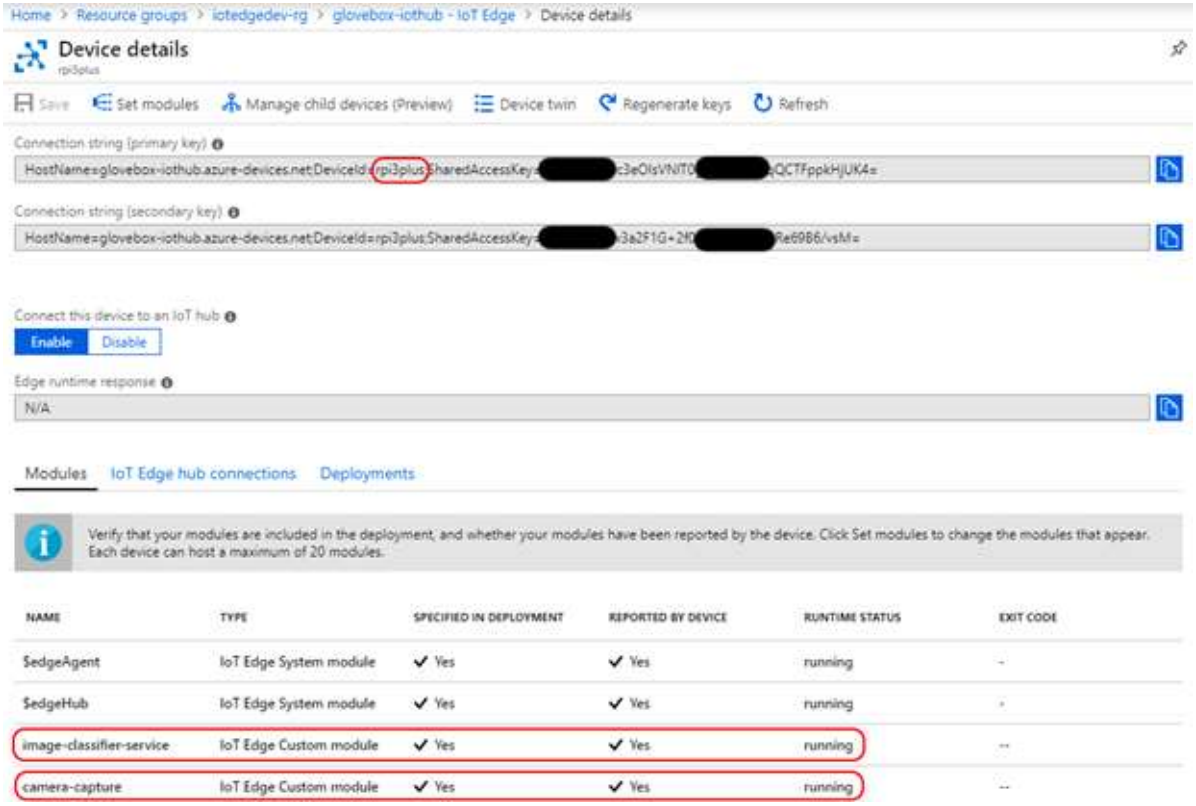
Monitoring the Solution on the IoT Edge Device

You can also monitor the state of the Azure IoT Edge module from the Azure IoT Hub blade on the Azure Portal.



Monitoring the Solution on the IoT Edge Device

Click on the device from the Azure IoT Edge blade to view more details about the modules running on the device.



Lecture Summary

- Computer Vision
  - Introduction
  - How it works?
  - Techniques
  - Architecture
- Objection detection models
  - RCNN
  - Fast-RCNN
  - Faster-RCNN
  - SSD
  - YOLO
- Azure compute vision as SaaS
- Usecase


THANK YOU!