### 1. Overview

This project demonstrates building artificial intelligence (AI) based outdoor camera . Solution is built upon android things OS, Tensorflow and google cloud service, Firebase. Although, I used Raspberry Pi 3 for building this project, any [hardware platform supporting Android Things](https://developer.android.com/things/hardware/developer-kits.html) could be used with having supported camera sensor and peripheral board, if required. This CCTV is remotely configurable using Firebase config and could OTA updated from Android Things Console.

Vision processing and Cloud upload is core of this project. Objective of this project is to build a camera device which could collect data and send those data to cloud for learning. Get update on the learning and then do further job based on those learning. Individual use case could easily be configured to the device over Firebase config interface and doesn't require OTA update for such tailoring.

### 2. Problem addressed with this project

When I was about to start working on project to submit on [Android Things: Rapid Prototypes to Real Products](https://www.hackster.io/contests/Google), I had started with different objective, to work on application based on OCR, apart from object detection. But soon I realised to address bigger objective which could help many of people who are working on AI and facing similar challenges as I am. This is actually the bottleneck of effective use of time while working for vision processing based AI, whether it is for embedded or with larger processors based device application.

* To have lots and lots of GPU processing power at the computer for training model during development, otherwise it is good to scare one off.
* To have good enough data for training. For image processing fare minimum image count is needed to get generous result. In deep neural network based training, in general, minimum of 1000 or even 10,000 image are needed to train for the object for practical use case.

With having this finding, I did set my objective to address the second bottleneck. And so this project right here is. To collect thousands of images straight from the ground where the device will be installed. And could be extend even more later on just by updating configuration from Firebase cloud. This allows one to have ever learning and fine tuning AI based product..

### 3. Providing solution

This camera works on AI. It works with initial graph to have tentative look of the object we want to get images of (if working for specifics of vehicle, one need vehicle images ). Minimal graph is needed while starting to use this camera, which acts as catalyst to start things up. And then let the aicamera look around for the object we want and train itself over the cloud. Which is ultimately the goal of this project.

At the other end of the cloud, dataset could be built out of these collected images and model could be trained upon to generate graph for AI which would be much easy with Google Cloud platform. Newly generated graph by training, based on these images. These images could be used for testing further for the model or graph. And upon having high confidence level seen while on testing, same graph could be deployed on field. This process could easily be used to fine tune the model or graph for any vision based AI.

Use same model for either testing further images or use it for having more precise images to collect. To simplify the topic, with this project use case, I am collecting images of car. I am using tensorflow inception graph, which does classify car as object. Google inception graph is catalyst for the project in this case. Based on used inception graph, plenty of car images will be captured by the camera which will be uploaded to the Firebase storage and will be used for training. Car as object to find is again set from Firebase config. Which could be change to any supported object by inception, such as TV, coffee mug, etc.

Who would need images of car as an example? Or in other sense, who will use this project to have car images? This could be the requirement for anyone who are working on model to find street coverage of certain type of car, or to train for front, side and rear view of car. Could be any specific preference, or to use these car images further for administrating license plate of those cars(which had been my initial object, and gone for now due to bottleneck. Which I will solve with this project). This exmaple is just to count few to justify the use case.

These data could be turned into services for one to generate dataset or graph. A business for you and so for you to use this project J.

### 4. Quick rundown of what we are going to do

Camera is made to provide visual, image, feed to cloud to accumulate training images.

* An IoT edge device having embedded processing to acquire visual data, [Android Things](https://developer.android.com/things/index.html) and [Tensorflow](https://www.tensorflow.org/)
* Captured object image are pushed to the cloud, [Firebase Storage](https://firebase.google.com/products/storage/).
* Object selection is configured at [Firebase remote config](https://firebase.google.com/docs/remote-config/). Which can easily be configured to change select the object, such as car, sports car, coffee mug, so on and so forth.
* Solution also provides way for training with these images for updating inception graph.
* Object detection could also be trained. Here darkflow is used for training object detection graph.

### 5. Android things application

Application is built with camera2 API to interface camera capture device and registers bitmap image on callback for processing the image for object classification with tensorflow.

final CaptureRequest.Builder previewBuilder =

mCameraDevice

.createCaptureRequest(CameraDevice.TEMPLATE\_STILL\_CAPTURE);

previewBuilder.addTarget(mImageReader.getSurface());

previewBuilder.set(CaptureRequest.CONTROL\_AE\_MODE,

CaptureRequest.CONTROL\_AE\_MODE\_ON);

Log.d(TAG, "Capture request created.");

mCaptureSession.capture(previewBuilder.build(),

mCaptureCallback, null);

Classified image is chosen for looking for specific object, configured at Firebase Remote Config and upon finding bitmap image is compressed into JPEG and uploaded to Firebase Storage for training against vision processing.

final List<Classifier.Recognition> results = mTensorFlowClassifier.doRecognize(bitmap);

Log.d(TAG, "Got the following results from Tensorflow: " + results);

ImageUploader imageUploader = ImageUploader.getInstance();

imageUploader.uploadImage(bitmap, results);

Tensorflow training is used for classification where as darkflow training for object detection and is explained in later section. As we need image in practical scenario, we are using images capture then process and get result and then go for next image. There is a pause added between consecutive images which is again configurable from Firebase remote config. Processing on preview image makes sence for use case but for training image collection, it is better to delay a bit to have verity of images. And also, continuous image processing adds load on processer which has to be compensated by some means which is not needed here. We are using TEMPLATE\_IMAGE\_CAPTURE. We can do at preview as well but I didn't find compulsion on why should I.

This is the criteria for checking if image upload is required. Which is configured from Firebase remote config.

if(!lock

&& objects.get(i).getTitle().equals(RemoteConfigs.getTrainingObject())

&& (objects.get(i).getConfidence() > RemoteConfigs.getConfidenceThreshold())

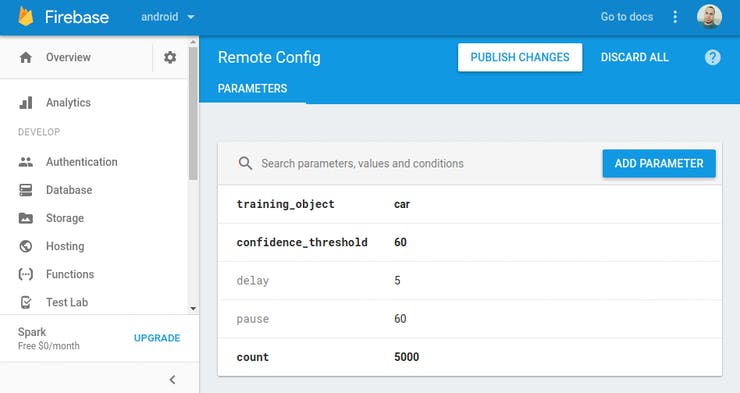
&& count < RemoteConfigs.getCount()) {

return true;

}

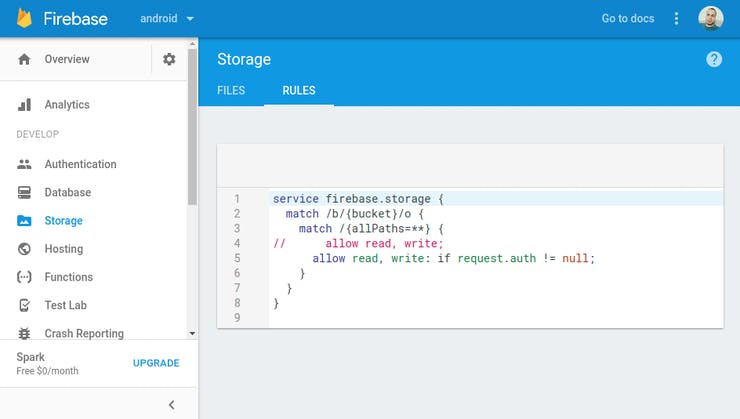
}

Criteria is configurable by Firebase remote configuration from console,



Firebase remote config

Here in configure I did change confidence threshold based, object and how many images to upload. I can update it based on analysis at cloud during training. Such as to include any other object, such as sports car apart from car, truck, etc. And then configuration is published. To be noted, I am using Spark plan of Firebase cloud which is a free service paid one has additional feature for corporate benefit but this is one is best for my stage. While experimenting, I did use storage rule to allow update without authentication during my experiment. Which is commented for now and enable it back during experiment.



Storage rule

I did this way as at one point while debugging, my device keep flooding files to cloud so when I feel it is right, I enable auth at cloud which blocks further upload. This is only during experiment, at the cost of any one to upload data to my cloud space without authentication. So, not at all for public code/application. In later phase, once this module is good I will add auth at application by key signing.

And mirror of these configurations are at application source xml

<?xml version="1.0" encoding="utf-8"?>

<defaultsMap>

<!-- training object -->

<entry>

<key>training\_object</key>

<value>truck</value>

</entry>

<!-- train if above confidence threshold -->

<entry>

<key>confidence\_threshold</key>

<value>60</value>

</entry>

..

..

<entry>

<key>delay</key>

<value>5</value>

</entry>

..

..

<entry>

<key>count</key>

<value>5000</value>

</entry>

..

..

</defaultsMap>

Configuration is very light job to network and processor. So, using firebase remote config has good benefit at no cost.

Well, coming back to our code, we are uploading image at ImageUploader class.

delayNextUpload();

String uploadFileName = RemoteConfigs.getTrainingObject()

+ "/"+ RemoteConfigs.getTrainingObject()

+ "\_" + count + ".jpg";

Log.d(TAG, "Uploading Image:" + uploadFileName);

++count;

OutputStream stream;

try {

stream = new FileOutputStream(LOCAL\_TEMP\_IMAGE);

bitmap.compress(Bitmap.CompressFormat.JPEG, 100, stream);

}

..

..

Uri uri = Uri.fromFile(new File(LOCAL\_TEMP\_IMAGE));

StorageReference uploadRef = mStorageRef.child(uploadFileName);

uploadRef.putFile(uri)

.addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {

@Override

public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {

Log.d(TAG, "success");

}

})

.addOnFailureListener(new OnFailureListener() {

@Override

public void onFailure(@NonNull Exception e) {

Log.e(TAG, "Upload filed for reason:" + e.getCause());

}

});

Basically, here I next upload is locked for the duration as specified in firebase remote config delay parameter and bitmap is converted into .jpg and uploaded to server. Nothing to do if it succeed or fail exclusively based on current use case, we can add later when required, after all we have android things console privilege J.

One can build application based on this training and to verify with cloud vision request.

* Building Cloud Vision request:

..

ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

bitmap.compress(Bitmap.CompressFormat.JPEG, 90, byteArrayOutputStream);

byte[] imageBytes = byteArrayOutputStream.toByteArray();

// Base64 encode the JPEG

base64EncodedImage.encodeContent(imageBytes);

annotateImageRequest.setImage(base64EncodedImage);

// add the features we want

annotateImageRequest.setFeatures(new ArrayList<Feature>() {{

Feature labelDetection = new Feature();

labelDetection.setType("TEXT\_DETECTION");

labelDetection.setMaxResults(5);

add(labelDetection);

}});

..

* Building remote configuration:

<?xml version="1.0" encoding="utf-8"?>

<defaultsMap>

<!-- training object -->

<entry>

<key>training\_object</key>

<value>truck</value>

</entry>

<!-- train if above confidence threshold -->

<entry>

<key>confidence\_threshold</key>

<value>60</value>

</entry>

..

..

<entry>

<key>delay</key>

<value>5</value>

</entry>

..

..

<entry>

<key>count</key>

<value>5000</value>

</entry>

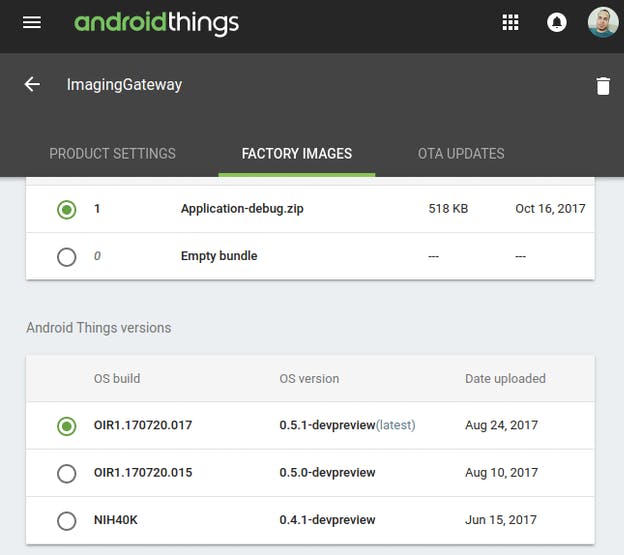
..

..

</defaultsMap>

* Console to build image

I implemented minimal feature and with hardware ready to use. As having console for OTA update is a privileged to work with android things. It is more of setup once and take one software then on.



Things Console

**6. CCTV hardware setup**

I did purchase CCTV toy camera to make use of its case for enclosing my setup. Well, it had been much of worth.



1 / 2 • CCTV toy camera

Did unscrew the casing to make room for fitting electronic modules beneath it.

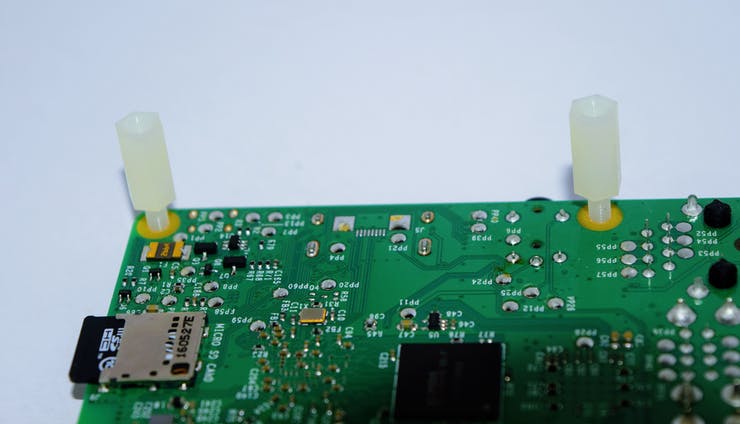


CCTV base

CCTV base was used for mounting Raspberry Pi3 board.



Raspberry Pi 3 and supported camera module was used for hosing CCTV heart within the camera casing.



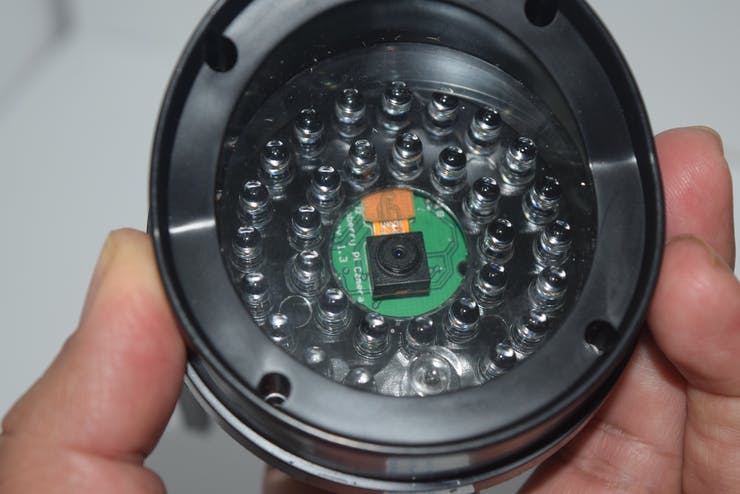
Two plastic screws were used for holding board in place. Used only at micro-usb end as to keep some space for usb head to run within the casing. Other edge was left without screw.



Board was placed in case as shown.







1 / 2 • camera module at its place

Camera module was placed with corrected posting.

In the end things ware assembled back by adding power bank left floating within.



1 / 3 • CCTV assembled

Before assembling everything within the casing, MMC card was flashed with things OS and application which is explained in software section.

### 7.1 Re-train the model with Tensorflow

$ cd tensorflow

$ python tensorflow/examples/image\_retraining/retrain.py \

--architecture mobilenet\_0.25\_128\_quantized

It is much easier way to retrain inception model to use. But if there are ready images in VOC format available with annotations, then darkflow would be worth to tryout.

### 7.2 Training model with Darkflow

We will create our own dataset for our customised object. Dataset are processed data files (here image files) with annotation usually done manually image by image. Data set is parsed through series of neurons rigorously. These series of neurons are collectively called model. So, here dataset will be screened by the model to let model generate learning files. And this learning files will be used in Android Things application to predict object with tensorflow. By the way, neurons are mathematically complex algorithm which are made for specific feature extraction.

[Creating dataset](https://github.com/abhishekkumardwivedi/android-things-AI-Camera/wiki/1.-Creating-dataset)link explains bit of explanation it creating dataset. And [training the model](https://github.com/abhishekkumardwivedi/android-things-AI-Camera/wiki/2.-Train-the-model)provides details on training. Snip of command for training with VOC dataset is here. Training with tensorflow, which is mentioned next to it, is much easier but try with it if specific to one object to reduce processing time.

* Annotate images

$ sudo apt-get install pyqt5-dev-tools

$ sudo pip3 install lxml

$ git clone https://github.com/tzutalin/labelImg.git

$ cd labelImg

$ make qt5py3

$ python3 labelImg.py

// set default annotation directory and label in beginning to speedup

* Install darkflow

$ git clone https://github.com/thtrieu/darkflow.git # get darkflow

$ cd darkflow

$ pip install -e . # Install it globally in dev mode

# Installing another way should also be fine.

$ sudo pip \

install opencv-python # Required by darkflow flow script

# sudo permission might be needed.

$ flow -h # this should not give an error if all is good.

* Train the model

$ echo "licenseplate" > ./labels.txt

$ cd bin && \

wget https://pjreddie.com/media/files/tiny-yolo-voc.weights \

&& cd -

$ mkdir -p train/Annotations && mkdir train/Images

$ flow \

--model cfg/tiny-yolo-lpc.cfg \

--load bin/tiny-yolo-voc.weights \

--train \

--annotation ../rawImages/indialicenseplateAnnotation \

--dataset ../rawImages/indialicenseplate

--epoch 2000

//--gpu

I did purchase CCTV toy camera to make use of its case for enclosing my setup. Well, it had been much of worth.

* Get (.pb) graph

// Saving the latest checkpoint to protobuf file

$ flow --model cfg/tiny-yolo-lpc.cfg --load -1 --savepb

All of these are explained in [wiki](https://github.com/abhishekkumardwivedi/android-things-AI-Camera/wiki)page.

To save implementation cost, I tried not to go in areas which is not in my expertise. I didn't spend time in mechanical part making. Instead I did, bit of, re-engineer existing CCTV case to make my needed stuff to fit in (android things development board, camera and power bank). At software part, I did lot of effort by spreading my R&D into wide horizon of neural network. I stared with understanding DNN, looking around all different popular frameworks, training model, building application. And after having all these experiments I did refine my objective and this project objective. I did try to limit the things to make things complete, as it is very easy to get offtrack by lot of tempting stuff around. Anyways, by summarising for doable at one shot is summarised.

### 8. Possible use cases

Same camera could also be used for,

* Motion tracker
* Capturing vehicle license plate
* Sending wireless message for door or security gate open/close
* Looking after specific subject, such as specific vehicle or animal too
* Parking management
* Toll gate
* Building and apartment
* Smart home

### 9. Common Issues I did go through

* Training had not been easy if there are few images. Even hundreds of images might be very less to get result.
* Not having powerful GPU based PC was setback.
* Started getting crazy while getting into work. Lot of AI framework but non was fully documented.
* No well formed benchmark to find out comparisons beween AI frameworks.
* Number of AI frameworks, number of graph formats.

### 10. Conclusion

There are wide range of use case which could be implemented with this Android things, tensorflow for AI at ground, and Firebase for cloud. Having an easy tool like device, aicamera, would make things much easier for AI engineers life.

[While I was adding few more feature to this project, OCR reading, vision processing, cloud vision, I did realise objective started fading and thing started turning mess for any second user. And started undoing thing by removing big part of code and to keep it simple for anyone to use. And keep it aligned with existing example to take minimal time to understand and for me also to address issues if anyone raises. I will possibly build separate project with these interfaces, Google cloud vision and vision processing would be for my another project]