**Story**

**1. Introduction**

This project will present the architectural design behind a voice controlled AI camera. The main idea is to control a camera by a voice interface to take a photo and have it send you an email with the photo description of what it observes. The project encompasses multiple components starting from hardware modules like Matrix Voice, Jetson Nano and RPIU2 to software frameworks like SNIPS AI and of course custom Python AI scripts that comprise the neural backend engine.

So in a nutshell we will leverage the Snips AI platform and Matrix Voice for the voice interface and Keras/Tenorflow DNN running on a Jetson Nano SBC for the image inference AI.

The demo code attached is setup so that if a person is detected an email is sent to the user with the annotated image from the AI engine. The class can be changed however.

**2. Background**

The camera will leverage optimized deep convolutional neural networks. I'm using the RPI camera version 1.1. The RPI does not have the capability to run large CNN.

When deciding which CNN architecture to use I narrowed it down to 3 options:

* Inception V3 - Weights are around 92Mb, the network has 1000 classes
* Resnet50 - weights are a bit smaller, 90 mb
* Squeezenet - weights are ~ 5Mb and it provides AlexNet level accuracy

Even though Squeezenet has impressive characteristics whiche means you can run the whole thing on a RPI I opted to use InceptionV3 since practical tests gave better performance and the idea was to build a distributed system in the first place.

Since RPI3 performance is a bit anemic and in order to be able to scale the design to multiple cameras the program will be setup so that it transmits camera image data via TCP from RPI3 to a Jetson Nano. This design decision was made since due to the TegraX1 GPU, Jetson Nano is capable of running multiple neural networks in parallel. So in recipe format what we are going to build here is the following:

* 1. Raspberry Pi 3 captures camera photo
* 2. RPI3 resizes it to InceptionV3 resolution
* 3. RPI3 sends photo to Jetson Nano.
* 4. Jetson Nano runs inceptionV3 received photo.
* 5. Jetson Nano prints image description
* 5. Description is sent via email to user if class of interest matches

But first, will have to put all the HW together.

**3. Hardware setup**

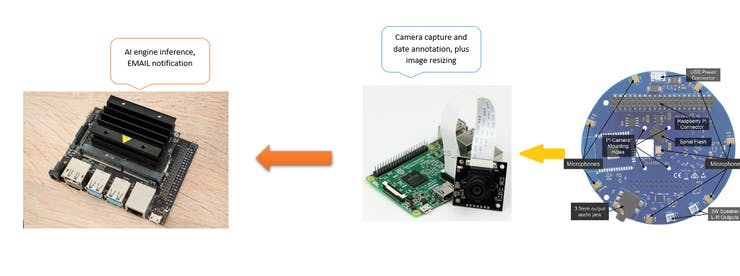
The Matrix Voice will be used as an audio input and output device. A Raspberry Pi V3 was used for interfacing with Matrix Voice.The same RPI3 was also used to connect RPI V1.1 camera.

The RPI V1.1 camera uses an OV5640 5MP pixel camera that can take 5MP resolution photos. This is more than enough.

Another option which was tested but not fished in time is to interface the OV7670 camera with the Matrix voice ESP32 micro as the frame capture device.

As mentioned the Jetson Nano SBC will be used to run the backend AI engine.

The system block diagram is as follows:



**5. Setting up the Jetson Nano.**

I used the following image as an OS for Jetson Nano.

https://developer.nvidia.com/jetson-nano-sd-card-image-r3221

The Jetson Nano is not compatible with RPI V1.1 of the camera. Nvidia uses a Sony camera which uses the same MIPI-CSI2 interface as RPI V2 camera. Incidentally, the community has released a custom kernel that implements the custom drivers for RPI V1.1 camera however the update procedure for the Nano is not as simple as burning a new image.

So anyway we have to install the following packages on Jetson nano.

pip install tensorflow = 1.14  
pip install keras = 2.2.1  
pip install imutils  
pip install pillow  
pip install opencv-python  
pip install opencv-contrib-python

**5. Setting up the RPI software**

On the RPI3 you need to install Keras, Scipy, OpenCV and Tensorflow.

The installation of the OpenCv V4.0 is a bit involved since there is no python wheel package so it has to be installed via a custom script.

Then install MatrixIO :

curl https://apt.matrix.one/doc/apt-key.gpg | sudo apt-key add -  
echo "deb https://apt.matrix.one/raspbian $(lsb\_release -sc) main" | sudo tee /etc/apt/sources.list.d/matrixlabs.list  
sudo apt-get update  
sudo apt-get upgrade  
sudo reboot  
sudo apt install matrixio-kernel-modules  
sudo reboot

The next step is to install the SNIPS AI framework:

On a Host laptop install Node JS and then install SAM:

sudo npm install -g snips-sam

Then install SNIPS and connect to RPI3 and init the service:

sam connect raspberrypi.local  
sam init

Then, configure SNIPS and the configuration file by issuing:

sudo nano /etc/snips.toml  
mike = "MATRIXIO-SOUND: - (hw:2,0)"  
  
#also add  
portaudio\_playback = "default"

finally check that the sound configuration file has the correct Matrix Voice Codec:

nano /etc/asound.conf

Finally install Node Js on RPI3:

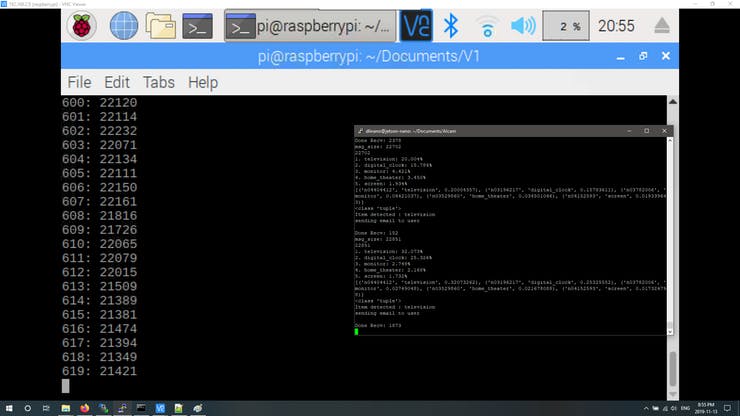
sudo apt-get update  
sudo apt-get upgrade  
curl -sL https://deb.nodesource.com/setup\_10.x | sudo -E bash -  
sudo apt-get install -y nodejs

The complete client software runs in its own python environment.

**6. Testing the network camera**

The first version of the software simply records video from the client side and sends it via TCP to the Jetson Nano which performs classification in it.

Basically this part of the Python script implements live video frame classification on the Jetson Nano using RPI camera as a satellite device. On the image below the camera is pointed to a monitor.



You can see the frames and the classification AI engine running on the Jetson nano on the right side.

The next component to test was the email notifications.

**7. Testing email notifications**

To test the email notification I wrote a Python script found below that takes an image from the current directory and sends it via email.

The idea behind this is that once the image is received via TCP it is saved on the Jetson Nano and then if the image belongs to a specific class, let's say person, the script will attach the frame as an attachment and send an email to the user.

This was tested with a Microsoft Outlook account and as you can see below it successfully works.

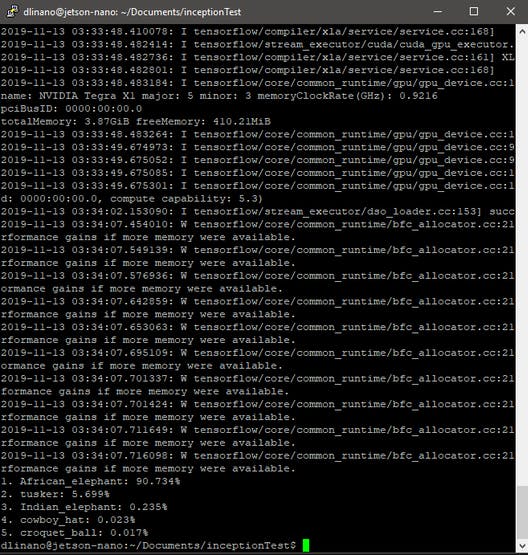
This script was then merged with the server side software.

**8. Testing the AI engine**

On the RPI install Tensorflow V1.14. This is needed just for test:

$ sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev zip libjpeg8-dev  
$ sudo apt-get install python3-pip  
$ sudo pip3 install -U pip  
$ sudo pip3 install -U numpy grpcio absl-py py-cpuinfo psutil portpicker six mock requests gast h5py astor termcolor protobuf keras-applications keras-preprocessing wrapt google-pasta  
$ sudo pip3 install --pre --extra-index-url https://developer.download.nvidia.com/compute/redist/jp/v42 tensorflow-gpu==1.14.0+nv19.9

The AI engine runs on the Jetson Nano.



1 / 2

**9. Matrix Voice setup**

After plugin in the Matrix Voice on the RPI3, the first step was to install the packages:

curl https://apt.matrix.one/doc/apt-key.gpg | sudo apt-key add -  
echo "deb https://apt.matrix.one/raspbian $(lsb\_release -sc) main" | sudo tee /etc/apt/sources.list.d/matrixlabs.list  
sudo apt-get update  
sudo apt-get upgrade  
sudo reboot  
sudo apt install matrixio-kernel-modules  
sudo reboot

Next go to Snips AI and open an account. Then, install NodeJs on the development host.

Install Node Js on the RPI also by issuing the following:

sudo apt-get update  
sudo apt-get upgrade  
curl -sL https://deb.nodesource.com/setup\_10.x | sudo -E bash -  
sudo apt-get install -y nodejs

Once Node is installed, install SAM.

sudo npm install -g snips-sam

Then connect with the RPI3 on the local netwok and install SNIPS on it.

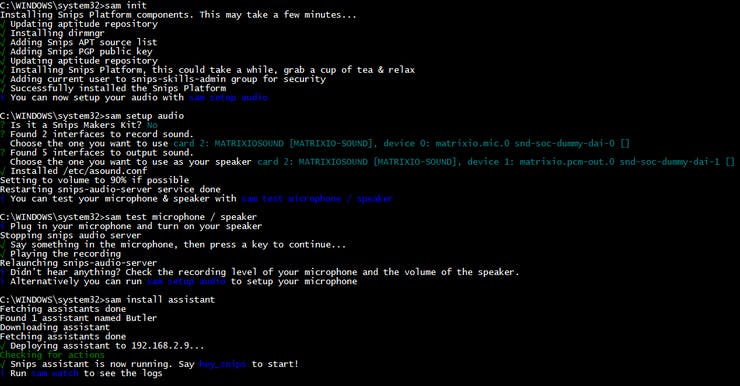
sam connect raspberrypi.local  
sam init

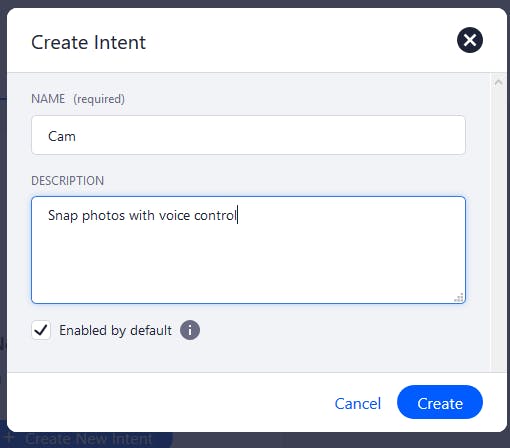
The next step is to setup the microphone and audio. I connected a pair of headphones to the audio jack of the Matrix Voice.

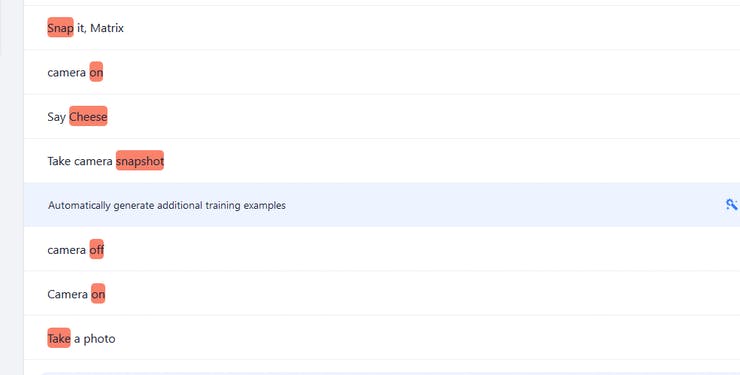
sam setup audio

Finally test the audio and microphone by issuing, say something and then press a key.

sam test microphone / speaker







1 / 2

I encountered a lot of issues with the SNIPS configuration where it would break the day after even though everything was working correctly the day before.

One solution was to remove all the SNIPS packages and re-install them again:

sudo apt-get purge snips-platform-voice  
sudo apt-get purge snips-platform-demo  
sudo apt-get purge snips-tts  
sudo apt-get purge snips-maker-tts  
sudo apt-get purge snips-watch  
sudo apt-get purge snips-template  
sudo apt-get purge snips-skill-server  
sudo rm -rf /usr/share/snips  
sudo rm -rf /var/lib/snips  
sudo userdel \_snips  
sudo userdel \_snips-skills

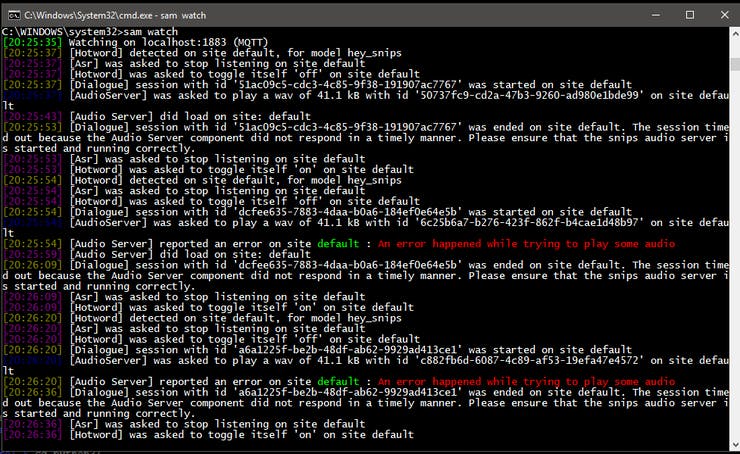
Then install the snips Camera assistant. this basically recognizes Hey snips Hotword and the intent camera on and camera off.

sam install assistant -i proj\_de2aOplEwD

SNIPS ASR running on the RPI performs the word to text transcribing. When the hotword and intent are detected it gets then routed though a MQTT call to the on\_connect() function to execute the respective tasks.

When issuing a voice command, I see that it gets detected, however there are problems with the Snips playback and intent detection.

I also observed that while the intent was accurately transcribed a couple of times it stops working at a later date.



The mosquitto server was then not running so I had to search around for a fix.

Looking on the forum I was able to get it back running by issuing the following:

sudo chown mosquitto /var/log/mosquitto  
sudo chown mosquitto:mosquitto /var/log/mosquitto/mosquitto.log  
  
  
sudo systemctl restart mosquitto

To test the SNIPS voice interaction I embedded the camera control under the onMessage MQTT function call:

# Process a message as it arrives  
def on\_message(client, userdata, msg):  
if not msg.topic == HOTWORD\_DETECTED:  
return  
payload = json.loads(msg.payload)  
model\_id = payload["modelId"]  
if model\_id in HOTWORDS\_ON:  
 print("camera\_on")  
elif model\_id in HOTWORDS\_OFF:  
 print("camera\_off")  
else:  
print("Camera snapshot: un-mapped hotword ")

This is the part that I ended up spending the most amount of time due to SNIPS constantly breaking down. Specifically the services would stop working when the app was updated. After un-installing and re-installing a couple of times I ended up calling the client Python script from the javascript program used to turn on and off the lights. This worked a couple of times until Snips decided not to run any of the services.

So I had to re-install again and redo the configuration from scratch. While Matrix Voice works correctly as a microphone and speaker the SNIPS service has a lot of issues as of November 2019. Python 3 scripts were only added recently.

The plan is to revise this in a future update.

**10. Client software**

The software is partitioned into a client side and a server side. The complete system makes uses of multiple stacks such as Keras, Matrix IO API, python sockets and OpenCV.

The client software runs on the RPI3. It's purpose is to get a camera snapshot every time the user issues a voice command. Once the utterance is decoded to instruct the camera to take a photo, the client then encapsulates this photo and sends it via a TCP socket to the Jetson NANO server which runs the AI engine.

The server program on the Jetson nano must be started prior to the client.

The client software starts by turning on the camera and establishing a connection with the Jetson Nano server.

Then when the utterance is decoded from the MQTT server running a camera snapshot is taken. The Pickle library is used to encapsulate the frame into a packet and send it to the sever. The message size is also appended to the packet so that the server has an idea what to expect.

To use SNIPS and Matrix Voice as a voice control interface one has to implement mqtt binding for the intents and the hot word that has been selected.

<https://docs.snips.ai/articles/console/actions/>

**11.Sever software**

The server script does the following:

a)received the image frames from the TCP client side

b) decodes the TCP frames into images and saves it locally

c) passes the images to the CNN inception V3 network to obtain an object classification category

d) If the class belongs to a person it sends an email to the user with the locally saved frame

Once the server software is initialized the first step is the initialization of the AI engine. The first time the script will download the Inception V3 weights. The next step is to configure the email. I have used the Outlook client with the smtplib Python library but any email provider should do. Once the program is started it asks the user for :

a) the email address

b) the email password

c) the recepient address (for this you can enter the same email as the sender address)

Once the program obtains the data to configure email and then it proceeds to start the main TCP server.and place it in listening mode.

The next step is to define the AI CNN model. This start the Tensorflow engine on the background.

The server then waits for image packets from the client. Once it receives the image it decapsulates it from the Pickle frame, builds an image from them and passes that image to the AI engine.

The AI engine, obtains the image and decodes it into a class by giving a short description.

img = image.load*\_img("sframe.jpg", target\_*size=(299, 299))  
x = image.img*\_to\_*array(img)  
x = np.expand*\_dims(x, axis=0)*  
*x = preprocess\_*input(x)  
y = model.predict(x)  
# cnt = cnt + 1  
for index, res in enumerate(decode*\_predictions(y)[0]):*  
*print('{}. {}: {:.3f}%'.format(index + 1, res[1], 100 \* res[2]))*  
*print(decode\_*predictions(y)[0]) #this is a list  
print(type(decode*\_predictions(y)[0][1])) #this is a tuple*  
*print("Item detected : " + (decode\_*predictions(y)[0][0][1])) # this picks the class description from the first tuple that shows the class with the highest probabililty  
decobj = (decode*\_predictions(y)[0][0][1])*  
*print("sending email to user\r\n")*  
*sendmail(decobj)*

This description then is used as the EMAIL subject together with the photo.

The advantage behind using Jetson Nano as opposed to running the complete application on the RPI3 is that it can use multiple neural networks in parallel due to the use of the GPU. In addition by decentralizing the application it becomes more robust and easy to scale.

**12. Demo time.**

To test the design I initially setup the server side. Make sure to use python3.

After launching the server side python script move over to the RPI and launch the client script.

You will see a lot of frames being transported via the network. If the AI feature extractor determines that a person is present on the image frame, it sends an email to the user.

On RPI activate the python environment with the OpenCV and start the client:

source .profile  
workon cv  
python clientVideo.py

On Jetson Nano start the server simply use python3 to run the script/

python3 server.py

There is a multitude of augmentations that you can build upon this program.

First and foremost you can improve the FPS by modifying the AI engine to use TensorRT inference model which can speed things up.

**Conclusion**

This project implemented a voice controlled AI camera using a distributed architecture which allows it to scale to multiple cameras. The main aim of the design was to leverage the Matrix Voice for voice interaction and the Keras/Tensorflow stack for AI inference. Hopefully SNIPS will iron out the stability issues the near future.