Elliot Blanford
Final Report 3/10/2021

Motivation:
The year is 2022. Quarantine is over. Seattleites have learned to enjoy human interaction once again. You find yourself at a social gathering where you are in conversation with a group of people who you don't know well. You somehow arrive at the topic of your involvement in the World Wildlife Fund project to save the Quokkas. You are passionately recounting the heroic efforts this organization is undertaking to preserve such a magical species when suddenly you notice the polite smiles in the group have turned to scowls and your new companions abruptly excuse themselves to refill their drinks.

The next day, you find out one of their close mutual friends has recently perished in a tragic quokka related accident. As is often the case when you mix alcohol and your passion for endangered marsupials, you missed all the warning signs that you were heading into dangerous territory with the quokka diatribe. Wouldn't it have been great if someone gave you a gentle notice before you lept off the deep end?

Fig 1: The majestic quokka
https://www.traveller.com.au/worlds-happiest-animal-the-quokka-becomes-the-most-popular-tourist-attraction-at-australias-rottnest-island-gunpvd

I see this device as a type of personal assistant for a problem I sometimes have: lack of social awareness. It will be able to assess the faces of your conversation partners and give you feedback on changes in their mood, presumably due to something you said or some event that is going on, *just in case you missed it.*

Objective:
To build and demonstrate an embedded device which can do the following:

1. Capture video stream with camera
2. Analyze faces in the frame and classify their emotions
3. Provide feedback to a user via buzzer when a change in emotional state is detected

The metrics used to characterize success will be:

1. Classification accuracy (F1 score)
2. Inference speed (frames processed per second)

System Description:
Hardware:
The system may be broken down into three modules.

1. Raspberry Pi 4B+
2. Raspberry Pi Camera V2.1
3. Custom feedback module

The first two components are standard off the shelf items. The custom feedback module consists of a vibrating motor (3 VDC, 85mA, 12000 rpm), transistor (P/N BC547), wiring harness, and 3D printed enclosure.

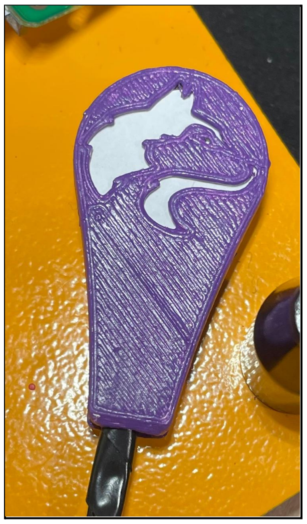
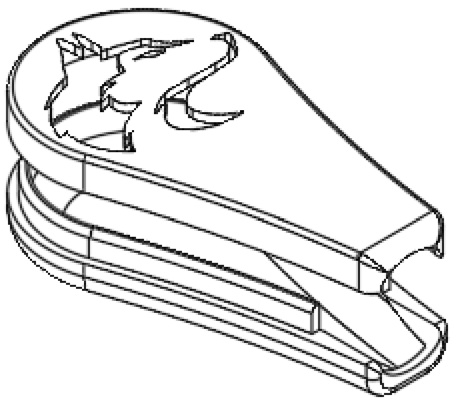



Figure 2: Left: Custom feedback module. Right: CAD Iso view

Software:
There are two unique pieces of software developed for this project: Both can be found on my github: https://github.com/Elliot-BI/Emotion_Detector
The first part is a Colab notebook that trains and evaluates the emotion detection neural network. The output of this notebook is a file called 'emotions.tflite' which contains all the weights of the neural net. The nature of colab notebooks lend themselves to embedded documentation, but I will summarize the process.

1. Extract the data and explore. Look at the tensor shape and distribution of classes.
2. Format the data for training. Create a validation set.
3. Define neural network architecture and set training parameters.
4. Train

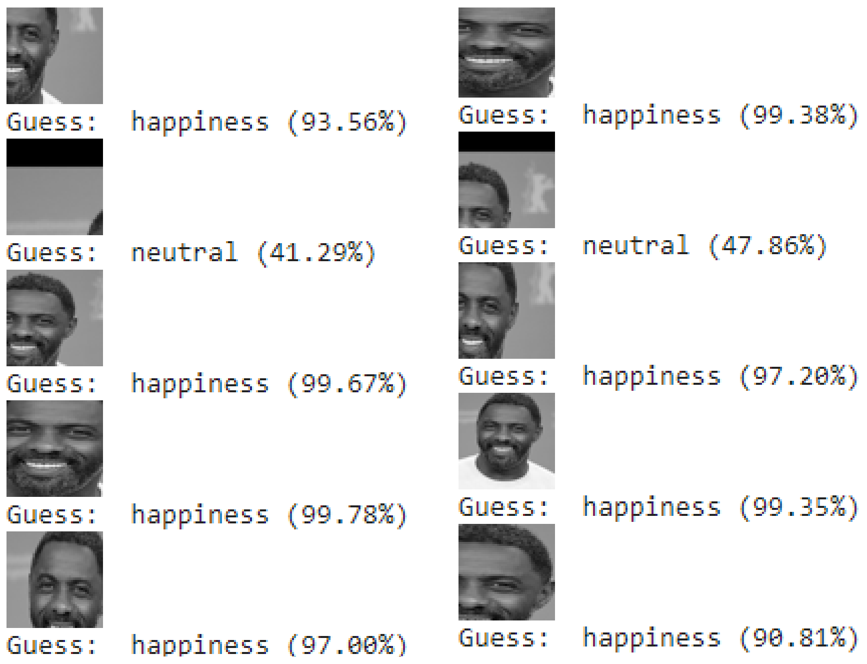
5. Check for convergence and overtraining
6. Check validation set accuracy
7. Look at examples of failed classification. Are these images unusual? Are there easy improvements to be made?
8. Convert model to TFLite model and export

The second piece of software is a python program which runs continuously on the Raspberry Pi. This program interfaces with the camera to capture an image, then runs the face detector cascade classifier to find a region of interest (ROI), then runs inference on that ROI using the TFLite model and returns an emotion prediction and confidence level.

Improvements to emotion detection accuracy

The initial accuracy of the emotion detection device was very poor. This was because the images from the training set were very close cropped faces with no background. Since a neural network is only really good at classifying things which are similar to the training data, trying to pick out the emotion on a small face in a scene was nearly impossible. The only way to get reasonable results was to position the camera so that the face fills more than 80% of the frame. This is not practical, so the next best option is to locate the face and digitally crop the image before sending it through the neural network. This requires another learning algorithm so it would undoubtedly have an impact on the inference speed.

I designed a test to get an idea of how accurate the cropping algorithm needed to be. It turns out that the emotion detector will work well with as little as 40% of the face in frame. This is good news. It means I can prioritize speed over accuracy when selecting a face detection algorithm.



```
Guess:  happiness (93.56%)      Guess:  happiness (99.38%)
Guess:  neutral (41.29%)        Guess:  neutral (47.86%)
Guess:  happiness (99.67%)      Guess:  happiness (97.20%)
Guess:  happiness (99.78%)      Guess:  happiness (99.35%)
Guess:  happiness (97.00%)      Guess:  happiness (90.81%)
```

Figure 3: Inference results for poorly cropped faces

The algorithm I selected is a boosted cascade classifier invented by Viola and Jones. This algorithm is extremely fast because it transforms the image to an “integral image” to speed up calculations and it is able to quickly determine regions of non-interest and ignore them. It is also built in to openCV, so it was very easy to implement. It takes an image as the input and returns the bounding box shown in green below.

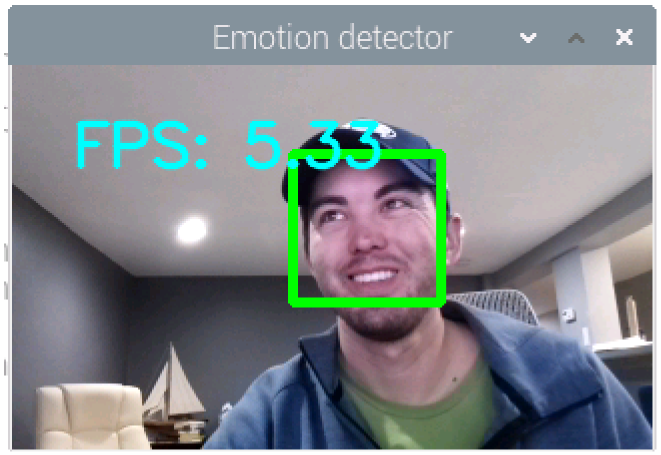


Figure 4: Example of Pi Camera preview with bounding box around the face

The inference accuracy (F1 score: 0.70) was good enough right off the bat, so I focused my efforts on speeding it up. However, I think there are still some low hanging fruits I did not attempt such as data augmentation. There may even be a good publicly available emotion detection model that I could convert into a TFLite model.

Improvements to inference speed:

The first effort, converting the model to TFLite, was not really an improvement. It’s more of a necessity for running on an embedded device. This is a slightly complicated process, but in the end you have an object called an Interpreter which takes the properly formatted tensor as input and returns the prediction tensor.

The original implementation of cascade classifier to put a bounding box on faces brought the frame rate down from 5.17 to 1.92. The problem is, it is searching for faces of all sizes. If I can put a bound on this, it will greatly reduce the amount of computation needed. Since I have a good idea of how big a face will be, I set the minimum size parameter to 200x200 pixels and the frame rate went up to 4.35! When I sent only the face box to the neural network, the frame rate jumped to 4.60.

The next question becomes, what is the optimal value for the minimum face size parameter? A smaller minimum can detect people further away, but that also means it has to scan the image many more times. In a 1280x720 image, there are 733,401 possible 100x100 boxes, 731,600 possible 101x101 boxes, etc. Viola-Jones does not check all of them, but the point is, there is a huge benefit if you have a tight minimum bound on the face size.

The field of view (FOV) of the RP4 camera V2 is 62.2x48.8 degrees (1.0856x0.8517 radians) and the resolution is 1280x720 pixels. One meter can be considered a comfortable conversation distance in a post-COVID world. So the width and height of a frame at one meter can be calculated by N = tan(Θ/2)*2*1m

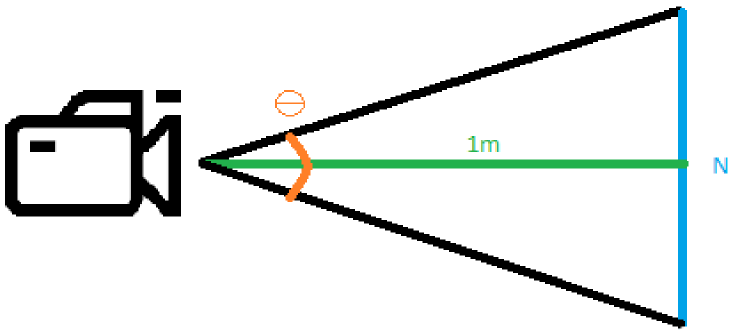


Figure 5: Pi Camera field of view

I estimated the dimensions of a face to be 0.2x0.25 m based on a rough measurement of my own head. This means the bounding box for a head at one meter should be 212x198 pixels. Since the neural net input is a 48x48 pixel image, this is more resolution than needed to get good results. In fact the image can be compressed by a factor of 4 in each dimension before running the cascade classifier. This downscaling resulted in a huge bump in frame rate, from 5.0 to 5.9.

FOV (degrees)	FOV in radians	View area at 1m	# pixels for a face	Compression
62.2	1.0856	1.2065	212.2	4.4
48.8	0.8517	0.9072	198.4	4.1

Custom Feedback Device:
The feedback device is intended to mimic a vibrating wristband or cell phone like device which can give the user a notification without distracting others. Implementation is fairly straightforward. It is a vibrating motor that runs off of 3V DC and 85 mA. However, the only pins which can provide this much current are the 5V power rails and those are always on. Since the 5V power rail cannot be controlled by software, a transistor is installed in series with the motor with the gate attached to a GPIO pin.

There is a slight problem with the circuit. The motor draws 85 mA at 3V, but since the rail is pushing 5V, it actually draws around 120 mA. The transistor is rated for only 100 mA. So far it is fine for short buzzes, but I think it will probably burn out sooner rather than later. Ideally, I'd use a better matched transistor or a small (~2V/85mA=24ohm) resistor to drop the voltage. The only reason I chose this transistor is because I had one on hand and nobody would sell me just one transistor.

Conclusion:
This project demonstrates an embedded device built on Raspberry Pi 4B which can locate a human face in a scene, classify it's emotional state, and alert the user when there is a change in state. All of the inference is done on the device so there is no WiFi requirement or bandwidth limitation. The device, in its current state of optimization, analyzes 5.9 frames per second and has an average F1 score of 0.70 over 7 classes of emotion.

References

https://github.com/Elliot-Bl/TensorFlow-Object-Detection-on-the-Raspberry-Pi/blob/0a4f408cd5389e66b212e2550a6a00a79b8e7cf9/Object_detection_picamera.py
C Language - A Modern Approach to Embedded Programming Andrew N. Sloss December 31, 2020
https://picamera.readthedocs.io/en/release-1.10/api_camera.html
https://github.com/tensorflow/models/tree/master/research/object_detection/models
https://github.com/lhelontra/tensorflow-on-arm/releases
https://github.com/protocolbuffers/protobuf/releases
https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/4
https://www.tensorflow.org/guide/saved_model
https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNet
Max power from GPIO pins:
https://raspberrypi.stackexchange.com/questions/51615/raspberry-pi-power-limitations
TFLite quick start: https://www.tensorflow.org/lite/guide/python
Guide to set up ML on RP4
https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi
A guide to Face Detection in Python (With Code)
https://towardsdatascience.com/a-guide-to-face-detection-in-python-3eab0f6b9fc1
Rapid Object Detection using a Boosted Cascade of Simple Features
https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
Install openVino for neural compute stick
https://docs.openvinotoolkit.org/2021.2/openvino_docs_install_guides_installing_openvino_linux.html#install-openvino