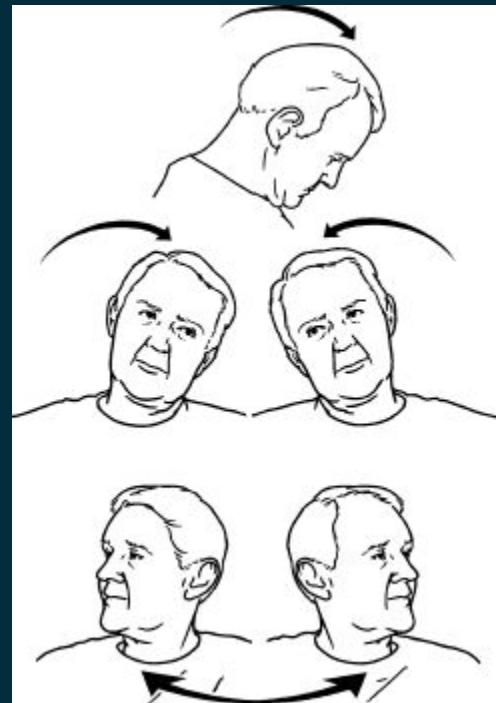
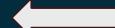
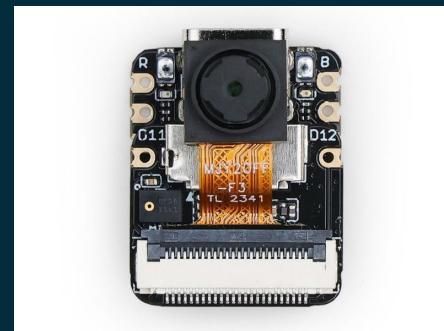
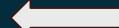


Head-Motion Detection for Computer Interfacing

Christian Cherry

18-444C Embedded Machine Learning

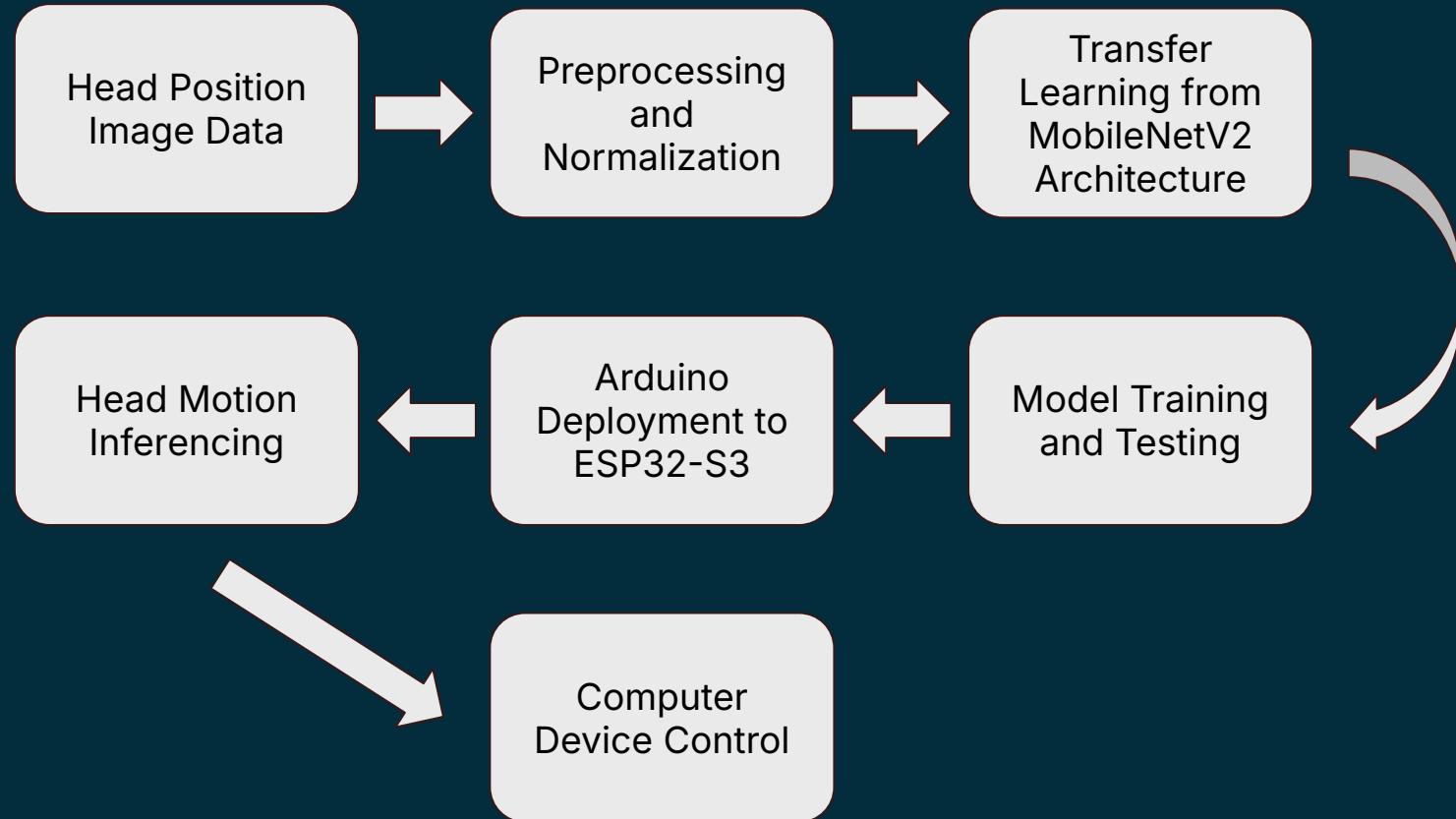
- ❖ General use computers and keyboards are commonly made to be used with hands
- ❖ Disabled individuals who can't extensively use their hands may have trouble using computers
- ❖ **GOAL:** Use Edge AI to detect different head motions of a user to interface with various devices, making for easier use of a computer or application for those who have trouble using their hands

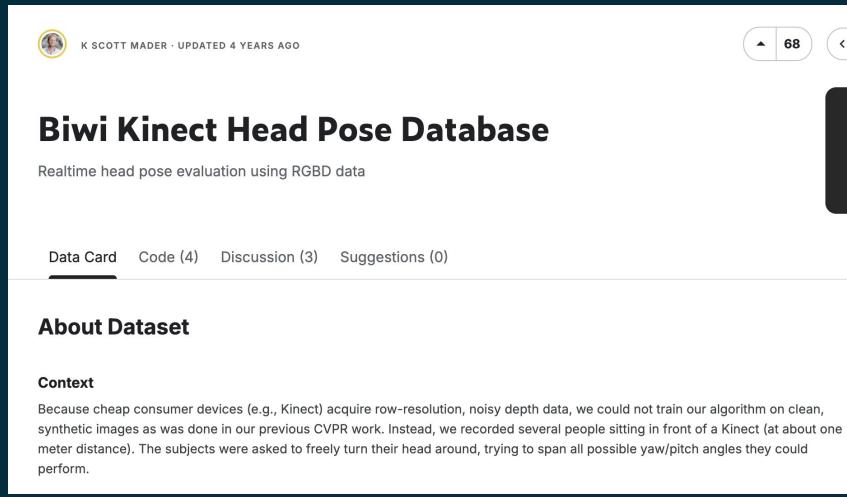


Using Embedded ML for head motion detection allows for an accessible and modular way to control a computer using head gestures.

- ❖ *Bandwidth:* A lot less data to process with camera feature extraction versus processing large video data
- ❖ *Latency:* Collecting data from a computer camera increases latency versus running an separate on-device model
- ❖ *Economics:* The cost of buying an Embedded ML device is more accessible than having to buy an entire computer to use the same feature
- ❖ *Reliability:* No reliance on network connection, can be portable and modular to other devices without relying on a specific computer's camera system
- ❖ *Privacy:* Having on-device compute without transferring facial data gives users a guarantee that their information is secure

Block Diagram





K SCOTT MADER · UPDATED 4 YEARS AGO

68

Biwi Kinect Head Pose Database

Realtime head pose evaluation using RGBD data

Data Card · Code (4) · Discussion (3) · Suggestions (0)

About Dataset

Context

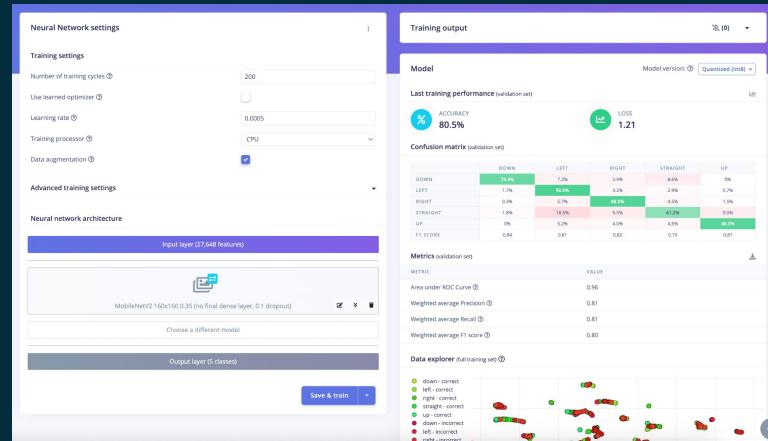
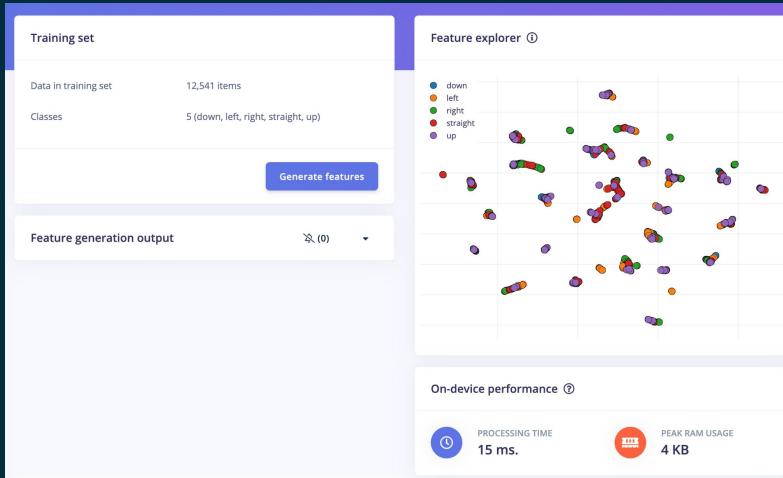
Because cheap consumer devices (e.g., Kinect) acquire low-resolution, noisy depth data, we could not train our algorithm on clean, synthetic images as was done in our previous CVPR work. Instead, we recorded several people sitting in front of a Kinect (at about one meter distance). The subjects were asked to freely turn their head around, trying to span all possible yaw/pitch angles they could perform.



frame	frame	frame	frame	frame	frame
frame_00143.rgb	frame_00225.rgb	frame_00153.rgb	frame_00235.rgb	frame_00092.rgb	frame_00188.rgb
ID: 2390314202 LABEL: up	ID: 2390314198 LABEL: straight	ID: 2390314197 LABEL: up	ID: 2390314196 LABEL: straight	ID: 2390314194 LABEL: left	ID: 2390314193 LABEL: straight
frame_00386.rgb	frame_00247.rgb	frame_00121.rgb	frame_00076.rgb	frame_00300.rgb	frame_00402.rgb
ID: 2390314192 LABEL: left	ID: 2390314184 LABEL: up	ID: 2390314183 LABEL: straight	ID: 2390314182 LABEL: left	ID: 2390314181 LABEL: up	ID: 2390314179 LABEL: up
frame_00066.rgb	frame_00181.rgb	frame_00310.rgb	frame_00362.rgb	frame_00004.rgb	frame_00326.rgb
ID: 2390314176 LABEL: left	ID: 2390314174 LABEL: straight	ID: 2390314173 LABEL: up	ID: 2390314172 LABEL: up	ID: 2390314171 LABEL: up	ID: 2390314169 LABEL: down

- ❖ Head pose estimation dataset from Kaggle.
- ❖ Used AI-labeling in Edge Impulse for "up", "down", "left", "right", and "straight" labels.
- ❖ *Challenge:* Manually cleaning up dataset for more accurate labels took quite a while.

Preprocessing an NN Architecture



- ❖ Utilized Edge Impulse's built in Image pre-processing
- ❖ *Challenge:* I initially used a pre-trained pose estimation block, which had way better accuracy, but this turned out to be too big to fit on my embedded camera
- ❖ Utilized the MobileNetV2 160×160 0.35 NN architecture to train model
- ❖ *Challenge:* Since only int8 quantization fits on my embedded camera, finding the right balance of efficiency and accuracy with the MobileNet architectures took much trial and error.

Training and Testing Results

Test data

Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.

SAMPLE NAME	EXPECTED OUTCOME	ACCURACY	RESULT	⋮
frame_00082_rgb	left	100%	1 left	⋮
frame_00198_rgb	down	100%	1 down	⋮
frame_00412_rgb	up	0%	1 uncertain	⋮
frame_00257_rgb	up	0%	1 left	⋮
frame_00131_rgb	up	100%	1 up	⋮
frame_00372_rgb	up	0%	1 uncertain	⋮
frame_00615_rgb	up	100%	1 up	⋮
frame_00032_rgb	right	0%	1 uncertain	⋮
frame_00597_rgb	up	100%	1 up	⋮
frame_00203_rgb	down	100%	1 down	⋮
frame_00604_rgb	up	100%	1 up	⋮
frame_00164_rgb	up	100%	1 up	⋮
frame_00256_rgb	up	0%	1 left	⋮

Classify all **T** **⚙**

Model testing output

Model version: **Quantized (int8)**

Results

ACCURACY **76.89%**

Metrics for Transfer learning

METRIC	VALUE
Area under ROC Curve ?	0.96
Weighted average Precision ?	0.82
Weighted average Recall ?	0.81
Weighted average F1 score ?	0.80

Confusion matrix

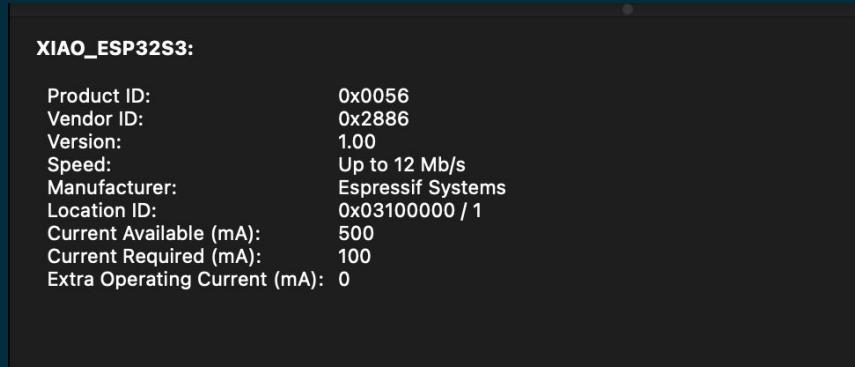
	DOWN	LEFT	RIGHT	STRAIGHT	UP	UNCERTAIN
DOWN	74.1%	5.5%	4.3%	4.6%	0.6%	11.0%
LEFT	0.8%	89.2%	2.6%	3.0%	0.6%	3.8%
RIGHT	0.8%	4.3%	85.3%	3.5%	0.8%	5.3%
STRAIGHT	1.5%	13.8%	5.9%	56.7%	6.5%	15.6%
UP	0%	6.5%	2.9%	3.6%	81.5%	5.5%
F1 SCORE	0.83	0.82	0.83	0.67	0.86	

Feature explorer [?](#)

Legend:

- down - correct
- left - correct
- right - correct
- straight - correct
- up - correct

The model has the most trouble classifying when your head is "straight".



```
26 // Includes
27 #include "head_Motion_Detection_inferencing.h"
28 #include "edge-impulse-sdk/dsp/image/image.hpp"
29
30 #include "esp_camera.h"
31
32 // --
33 // TINYUSB CONSUMER CONTROL HID
34 // --
35 #include "Adafruit_TinyUSB.h"
36
37 Adafruit_USBD_HID usb_hid;
38
39 // Raw HID report descriptor for Consumer Control (16-bit usage)
40 const uint8_t hid_report_descriptor_consumer[] = {
41     0x05, 0x0C, // Usage Page (Consumer)
42     0x09, 0x01, // Usage (Consumer Control)
43     0xA1, 0x01, // Collection (Application)
44     0x05, 0x01, // REPORT_ID (1)
45     0x15, 0x00, // Logical Minimum (0)
46     0x26, 0xFF, 0x03, // Logical Maximum (0x3FF)
47     0x19, 0x00, // Usage Minimum (0)
48     0x2A, 0xFF, 0x03, // Usage Maximum (0x3FF)
49     0x05, 0x01, // Report Size (16)
50     0x05, 0x01, // Report Count (1)
```

Output Serial Monitor <-->

Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem3101')

Predictions (DSP: 4 ms., Classification: 200)

Detected: down (value: 0.984948)

VOLUME DOWN

Predictions (DSP: 4 ms., Classification: 200)

Detected: down (value: 0.705729)

VOLUME DOWN

- ❖ Once the model is deployed, we connect the XAO ESP32-S3 to our computer via a TinyUSB Adafruit library.
- ❖ Real-time deployment yields a classification time of 200ms.
- ❖ We can then develop an embedded ML application to detect head motions to control the computer interface.
- ❖ *Challenge:* The accuracy of the real-time model is heavily biased towards "up" and "down", which is comes from combination of physical position sensitivity and model accuracy <80%. In addition, accuracy decreases when there are multiple people in the camera frame.