Building products using TinyML on Arm MCUs

Stuart Feffer, co-founder + ceo, Reality AI
26 Jan 2021
<table>
<thead>
<tr>
<th>Date</th>
<th>Title</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 9th</td>
<td>Supercharge your development time with the new Arm NN 20.11 release</td>
<td>Arm</td>
</tr>
<tr>
<td>February 23rd</td>
<td>Hands-on with PyArmNN for object detection</td>
<td>Arm Workshop</td>
</tr>
<tr>
<td>March 9th</td>
<td>Automate tinyML Development &amp; Deployment with Qeexo AutoML</td>
<td>Qeexo</td>
</tr>
</tbody>
</table>

Visit: developer.arm.com/solutions/machine-learning-on-arm/ai-virtual-tech-talks
is for building products with machine learning and sensors
Signal data opens the door to a wide range of product solutions that go beyond voice and vision.

**Time Series**
Periodic readings with frequency up to a couple of times per second.

**Speech Recognition**
Human speech and language recognition, used for voice assistants.

**Imaging / Vision**
Imaging using cameras, video, or specialty sensors.

**Signals**
High sample rate inputs, including sound, vibration, acceleration, RF, electrical and other waveforms.

“Alexa, play ‘Shake it Off’”
AI-driven feature discovery based on advanced signal processing math

Features are mathematical descriptions of “things that matter” for purposes of telling the difference between classes, predicting a variable, or detecting anomalies.

Features searched by Reality AI Tools® include:

- Common transforms on raw data, including logs, powers, derivatives, sign, and more
- Parametric statistical features and peak analysis
- Spectral features, including power, phase, spectral shape, periodicity, cepstral, wavelet, etc.
- Linear and non-linear dimensionality reduction
- Time-Frequency sparse coding and time pattern analysis
- Binary pattern and texture analysis

Often, we do not use deep learning or neural networks.
<table>
<thead>
<tr>
<th>Reality AI Tools® 4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AI Explore™</strong></td>
</tr>
<tr>
<td>Automated Feature Exploration and Model Generation</td>
</tr>
<tr>
<td>- AutoML (no coding)</td>
</tr>
<tr>
<td>- Explainability</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Building products > Building models
Building products = Building models + Building hardware + Building confidence
**Case Study:** Self-monitoring features for a residential air conditioner

<table>
<thead>
<tr>
<th>Objective</th>
<th>Detect blockages, leaks and other conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1:</td>
<td>Prove feasibility</td>
</tr>
<tr>
<td>Phase 2:</td>
<td>Design product</td>
</tr>
<tr>
<td>Phase 3:</td>
<td>Complete product development and prepare for manufacturing / distribution</td>
</tr>
</tbody>
</table>

© 2021 Reality Analytics Inc. All Rights Reserved.
Case Study: Self-monitoring features for a residential air conditioner

Phase 1: Feasibility - PoC

1. Start with a single test unit + IoT dev kit in likely location

2. Collect small amount of data in multiple conditions:
   - Running normally
   - With a fan fault
   - With a compressor fault
   - With a blocked coil

3. Run data thru Reality AI Tools® to build and test PoC model

Quick. Low cost. Good result could justify further effort.
Reality AI Tools® 4.0 Demonstration
Part 1 - AI Explore

- AutoML
- Explainability
Case Study: Self-monitoring features for a residential air conditioner

Objective: Detect blockages, leaks and other conditions

Phase 1: Prove feasibility

Is it possible?  ✓
Do I understand it?  ✓
Is it worth proceeding to product development?  ✓

Our PoC project was successful! Now we need to develop a product.
Building products = Building models + Building hardware + Building confidence
<table>
<thead>
<tr>
<th>Building Models</th>
<th>Building Products</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instrumentation</strong></td>
<td>Dev board</td>
</tr>
<tr>
<td><strong>Processing</strong></td>
<td>Try to optimize</td>
</tr>
<tr>
<td><strong>Power Use, Size and Weight</strong></td>
<td>Not generally a concern</td>
</tr>
<tr>
<td><strong>Other stakeholders</strong></td>
<td>Data Scientists: “I built it, so I understand it.”</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Not a factor</td>
</tr>
</tbody>
</table>
Case Study: Self-monitoring features for a residential air conditioner

Objective: Detect blockages, leaks and other conditions

Phase 2: Design product

How many sensors do we need?

Where are the best places to mount them (most accurate and cheapest to manufacture)?

What are the minimum specs for each component (so we can source them)?
Case Study: Self-monitoring features for a residential air conditioner

Phase 2: Design product

1. Multiple test units
2. Mount lab-grade sensors of different types in different candidate locations (accelerometers, thermocouples)
3. Collect data on each test unit in wider range of conditions, recording data from all sensors / placements
4. Run data thru Reality AI Tools® to optimize BoM
Case Study: Self-monitoring features for a residential air conditioner

Mounting locations for lab-based data collection:

**Accelerometers:**
- Chassis
- Fan motor
- Compressor shell

**Thermocouples:**
- Intake air
- Outflow air
- Coil

*In the final product we want to use as few channels as possible. But which ones?*
Reality AI Tools® 4.0 Demonstration

Part 2 - BoM Optimization

- Cost-optimized specifications
- Minimum sensor set
Case Study: Self-monitoring features for a residential air conditioner

Objective: Detect blockages, leaks and other conditions

Phase 2: Design product

Can we build it? ✔️
Is cost within limits? ✔️
Will it work? ✔️

Time to build test units and collect data for a production model!
Building products

Must work *everywhere* customers expect it to work.
In *every configuration*.
With *accuracy* that customers view as acceptable.

> Building models

Delivers good statistics on my test data.
Case Study: Self-monitoring features for a residential air conditioner

Objective: Detect blockages, leaks and other conditions

**Phase 3:** Large Scale Data Collection / Testing

1. Build and deploy test units with production instrumentation.
2. Collect a lot more data.
   - Types of customer implementations
   - Operating environments & environmental conditions
   - Different SKUs of similar construction
   - Multiple units (to account for small manufacturing differences)
4. Iteratively improve prediction models in Reality AI Tools®.
5. Export final model for firmware integration.
What’s the best way to control the cost of data collection?
What’s the best way to control the cost of data collection?

Only do it once.
Reality AI Tools® 4.0 Demonstration
Part 3 - Data Readiness

- Automated
- Consistency
- Quality
- Coverage
Reality AI Tools® 4.0 Demonstration

Part 4 - Edge Deployment / Integration

Edge AI / TinyML
Super-compact, efficient code for the smallest MCUs

- Embedded code generation
- Ease of deployment
/*
 * @brief ACConditionCheck1
 * Performs classification on the input int16 data array vector of 4 x 4096 and
 * returns one of the enumerated classes. Returns -1 if any detectable error occurs.
 *
 * This function is used to classify a single instance of data vectors of fixed window
 * length into an output class, with the customer providing sampling and data handling.
 *
 * @param X(data vector) signed 16-bit integers.
 * @return uint8.
 */

/* ARM CMSIS LIBRARY REQUIRED: Please ensure you have included the appropriate library for
 * your hardware in your project, see arm_math.h for details.
 * Required library headers - comment out if included in your own code
 */
#include "arm_const_structs.h"
#include "arm_math.h"

/*RealityAI Classifier Definitions*/
#define realityAI_NUM_CLASS 3 // number of classes in the current classifier
#define realityAI_WINDOW_SIZE 512 // the length of data input window to classify
#define realityAI_DATA_COLUMN 4 // the number of channels of data input window

/*List of possible classes returned by the classifier mapped to user defined named constants*/
enum realityAI_classes{
    REALITYAI_CLASSIFICATIONERROR = -1,
    REALITYAI_NORESULT = 0,
    REALITYAI_OK,
    REALITYAI_COIL_BLOCKAGE,
    REALITYAI_COMPRESSOR_FAULT
};

/*Input Data - example, comment out this line if your input data variable is
defined in your own code*/
int16_t adata[realityAI_DATA_COLUMN*realityAI_WINDOW_SIZE];

/*Classifier function prototype*/
int8_t ACCOnditionCheck1(int16_t* adata);

© 2021 Reality Analytics Inc. All Rights Reserved.
/* Example code provided by Reality AI */
/*
Generated by Reality AI Tools
Date:        2020-04-12
Account:     EngineerCorpAdmin
Project:     AC -- Live Data
Classifier:  AC Live Chassis 400 Hz 512 win
Version:     1.1
Target Proc: ARM Cortex M4F
(c) Reality Analytics, Inc. All rights reserved. Use and distribution are subject to applicable contract restrictions.
*/

/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
* ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

/* Standard Includes - these are intended as typical but generic examples only, edit to
suit your project. */
#include <stdio.h>
#include <stdlib.h>
#include "ACConditionCheck1.h"

int8_t ret = 0;
int r = 0;

int main(void) {
    /* Init Your board and hardware here. */

    /* Example code loop
    * Enter an infinite loop, simply generates a random vector of input and then attempt to
    * classify it,
    * printing the result. Replace this with your actual data read and windowing process.
    */
    while(1) {
        /* Fetch next data input window from your device */
        /* Simulated with a random array of signed 16 bit ints */
        /* REPLACE THIS BLOCK WITH INPUT FROM YOUR DEVICE */
        for(int u = 0; u < realityAI_DATA_COLUMN; u++){
            for(int v = 0; v < realityAI_WINDOW_SIZE; v++){
                r = (rand() % 64000) - 32000;
                adata[u*realityAI_WINDOW_SIZE+v] = (int16_t)r;
            }
        }

        /* Classify the input data */
        ret = ACConditionCheck1(adata);
        /* Act on the output using a switch statement against the enumerated, named classes. */
        switch(ret){
            case REALITYAI_CLASSIFICATIONERROR:
                printf("Classification Error\n");
                break;
            case REALITYAI_NORESULT:
                printf("No valid class detected\n");
                break;
            case REALITYAI_OK:
                printf("Class Ok Detected\n");
                break;
            case REALITYAI_COIL_BLOCKAGE:
                printf("Coil Blockage Detected\n");
                break;
            case REALITYAI_COMPRESSOR_FAULT:
                printf("Compressor Fault Detected\n");
                break;
            default:
                printf("Invalid return value\n");
                break;
        }
    }
    return 0;
}

1) Pull in data from your sensors.
2) Call the Reality AI classifier
3) Act on results

© 2021  Reality Analytics Inc.  All Rights Reserved.
Case Study: Self-monitoring features for a residential air conditioner

Objective: Detect blockages and other conditions

Phase 1: Prove feasibility ✓
Phase 2: Design product ✓
Phase 3: Complete product development ✓

Product development is complete! Ready for manufacturing & distribution.
Reality AI Tools® 4.0

**AI Explore™**
Automated Feature Exploration and Model Generation
- AutoML (no coding)
- Explainability

**BOM Optimization**
Use AI to find the most cost-effective components
- Cost-optimized specifications
- Minimum sensor set

**Data Readiness**
Understand the state of training and testing data
- Automated
- Consistency
- Quality
- Coverage

**Edge AI / TinyML**
Super-compact, efficient code for the smallest MCUs
- Embedded code generation
- Ease of deployment

© 2021 Reality Analytics Inc. All Rights Reserved.

[info@reality.ai](mailto:info@reality.ai)  
[@SensorAI](https://twitter.com/SensorAI)
Thank you!

Tweet us: @ArmSoftwareDev

Check out our Arm YouTube channel and our Arm Software Developers YouTube channel

Signup now for our next AI Virtual Tech Talk here

Attendees: don’t forget to fill out the survey to be in with a chance of winning an Arduino Nano 33 BLE board
Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה