Technical White Paper

Machine Learning on the Edge with the mmWave Radar Device IWRL6432

Muhammet Yanik, Akshay Kumar Chandrasekaran, Sandeep Rao

Table of Contents

Trademarks

1 Introduction

The past few years have seen a significant increase in sensing applications in the 60 GHz band (an unlicensed band open for a wide variety of applications). This is primarily due to advances in mmWave technology that have reduced the size, cost and power consumption of these sensing devices. Typical applications include:

1. Building automation (indoor and outdoor surveillance, elderly care)
2. Personal electronics (gesture recognition, presence detection)
3. Automotive body and chassis (in-cabin sensing, kick-2-open sensor).

The smaller wavelength of approximately 5 mm for 60 GHz signals directly translates to higher velocity resolution and small form factor. Many sensing applications that earlier operated at lower frequency bands (3-10 GHz) are now moving to 60 GHz to leverage these advantages.

While mmWave radar has been traditionally employed in target detection and tracking, there has been a trend toward using radar signals for target classification [3]. Many of these classification algorithms use Machine Learning (ML) to leverage the radar's high sensitivity to motion. The examples include: motion classification for reducing false alarms in building automation, fall detection for elderly care, and gesture recognition.

TI's integrated radar-on-chip devices, such as IWRL6432 [1], are designed for the above applications. IWRL6432 has an integrated RF front end and a Hardware Accelerator (HWA@80 MHz) optimized for radar signal processing. The on-chip MCU, Arm® ***Cortex***®-***M4F*** (@160 MHz), provides sufficient computational capability for post-processing algorithms such as tracking and classification. The device is designed with various low-power modes to support applications where minimizing power consumption is critical.

2 Machine Learning in mmWave Sensing

Figure 2-1. A Representative Radar Based Classification Chain

Figure 2-1 depicts the typical processing flow of an algorithmic chain that incorporates an ML-based classifier. First basic physical layer processing is done on the raw ADC data received from the radar front end. This block involves a series of FFTs to separate the signal based on range, Doppler, and angle of arrival [2]. In some cases (e.g. Case-Study-1: Motion Classification), this can additionally be followed by detection and tracking algorithms.

The next step involves feature extraction, where the output of the previous block is processed to provide input to an ML-based classifier. The choice of feature extraction and the corresponding classifier that follows are closely coupled. Some popular options found in the literature [3] are listed below:

1. *Range-Doppler heat map*: A 2-dimensional Range-Doppler heat map is created for each frame of data that is sent to a Convolutional Neural Network (CNN). In some cases, the output of the CNN can be fed to another network (such as an Recursive Neural Network (RNN) or a Temporal Convolutional Network (TCN) to exploit temporal signatures in the data.
2. *Micro-Doppler vs. time:* A Micro-Doppler spectrum is created from each frame of data. This spectrum is then concatenated across frames to create a 2-dimensional Doppler vs. time image, which can then be processed through a CNN model for classification.
3. *Hand-Crafted Feature extraction:* The 2-dimensional images (such as the Range-Doppler in (a) or the Doppler vs. time in (b) can be pre-processed to create hand-crafted features (a small set of parameters that capture the essence of the image). A single frame yields a single value for each feature, while a sequence of frames generates a corresponding time series. These are then sent to a classifier such as ANN.

There is a tradeoff between the intelligence in the feature extraction block and the complexity of the classifier. Approach (3) typically results in the lighter-weight classifier. The selection of the feature extraction block and the classifier architecture depends on the complexity of the classification task as well as requirements such as classifier performance, frame rate, and available memory and computing power.

In a typical implementation on the IWRL6432, the radar physical layer (PHY) processing is handled by the HWA while the classifier runs on the MCU (M4F). Hand-crafted feature extraction typically runs on the M4F with some assistance from the HWA.

3 Development Process Flow

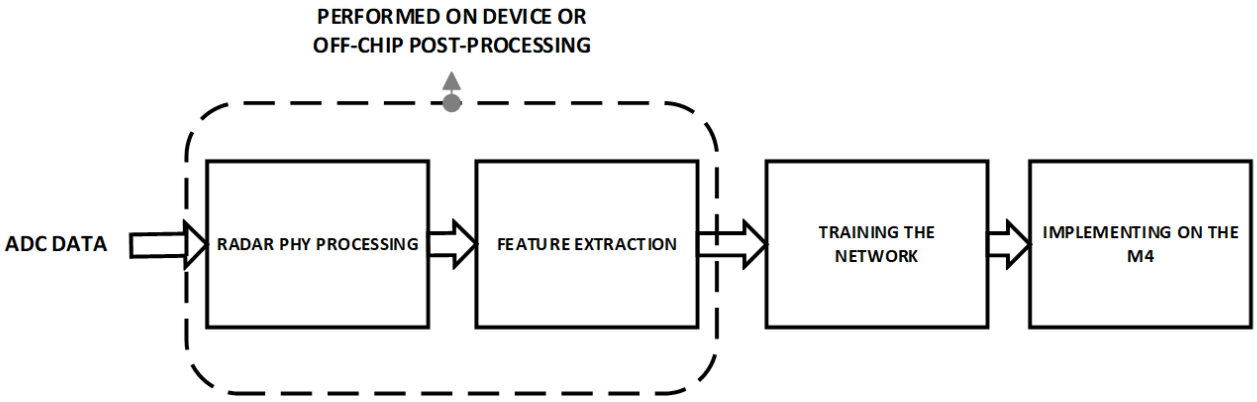


Figure 3-1. Development Flow for an ML-based Application with mmWave Radars

The development flow can be divided into 3 steps:

1. Data collection, feature extraction and annotation
2. Architecting and training the classifier
3. Implementing the classifier on the IWRL6432 (M4F)

1. *Data collection, feature extraction, and annotation:* Since ML-based classifiers are highly data-driven, collect a sufficient amount of data in scenarios that are representative of the use case being considered. There are two ways to conduct a data collection campaign:
   a. The Radar PHY processing and feature extraction are done in real-time on the IWRL6432 [1] , with the feature vectors being streamed to a laptop
   b. A data capture board (DCA1000) coupled to the IWRL6432 is used to directly stream raw ADC data to a laptop via Ethernet. The Radar PHY processing and feature extraction is done on the laptop (using for e.g. MATLAB® or Python®)

   Since features are a compact representation of the raw ADC data, the former method requires lesser storage on the laptop. The latter method (with access to raw ADC data) enables one to experiment with different radar pre-processing techniques for post data collection.

   Accurate annotation or labeling of the data (i.e., associating the features with the right label) is critical. While not always possible, the goal is to automate this process with minimal human oversight. In some cases, the data collection process includes a video stream to record ground truth for labeling purposes.

2. *Training a classifier:* There are various training options available. KERAS is an open-source library written in python that provides a plug-and-play framework to quickly build, train and evaluate classifiers. TensorFlow® and Pytorch® are two other popular open-source frameworks. For the kind of Edge-AI applications targeted for the IWRL6432, we have found KERAS to be a good choice providing the right tradeoff between ease of use and performance. MATLAB® also has a licensed Deep Learning Toolbox with an easy to use interface. For customers with no or limited inhouse ML experience, there is a rich ecosystem of 3rd parties that can support the development of ML applications.

   Two things must be kept in mind for robust performance of the classifier:

   a. The data set must incorporate variations that are expected to be seen in the actual use-case.
   b. The classifier must be tested with sufficient isolation between the training and test data set (in a motion classification use-case, the training and testing data set can have different users)

[1] Feature vector streaming is first supported in SDK5.2

3. *Implementing on the Arm® Cortex®-M4F:* Once the ML network (architecture, weights, and biases) has been chosen in step 2, the final step is implementing the network in the embedded system. Some of the options available are:
   a. Hand-written bare-metal C code (with weights and biases in floating point)
   b. CMSIS-NN is a software library with neural network kernels optimized for Arm® ***Cortex®-M4F*** processor cores. The library supports only quantized weights and biases (8-bit or 16-bit). The library also provides a recipe to quantize the weights and biases of a floating-point network.
   c. Using TFLite, which is an open-source library developed by Google for deploying machine learning models to embedded devices. This takes a floating-point network as an input and generates a quantized version of the network that can run on an embedded device.
   d. 3rd parties also provide support in porting classifiers to an embedded environment.

4 Case-Study-1: Motion Classification

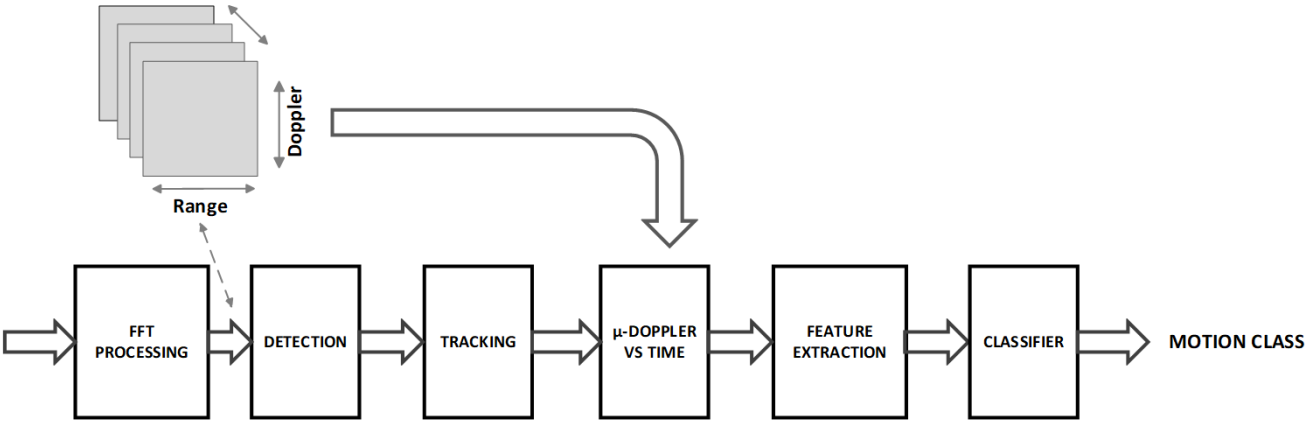


Figure 4-1. Processing Chain for Motion Classification

Radars natively have very good velocity resolution. A machine learning backend enables the radar to recognize motion signatures. This can be leveraged in many applications that can benefit from the classification of tracked objects.

Figure 4-1 is a representative signal processing chain for motion classification. The received ADC data is first processed using traditional FFT processing (Range-FFT, Doppler-FFT, Angle-FFT) to generate a radar cube. Next, a detection layer identifies peaks and generates a point cloud. The point cloud is then clustered into objects using a multi-target group tracker (perhaps based on an Extended Kalman Filter). μ-Doppler spectrograms are then generated for each tracked object. The centroid for each tracked object is mapped to the corresponding location in the radar-cube and a μ-Doppler spectrum is extracted from the neighborhood of this centroid. The μ-Doppler spectrum is concatenated across multiple consecutive frames to generate a μ-Doppler vs. time heatmap. This can be sent to a 2D-CNN for classification. Alternatively, a hand-crafted feature extraction block can be implemented to create a 1D time sequence data from the sequence of spectrograms, which can then be classified using a 1D-CNN. This has much less complexity than a 2D-CNN.

In our study, the above approach was used to discriminate human vs. non-human motion [4]. This problem is relevant in indoor and outdoor surveillance, where discrimination between motion from humans and other sources such as pets, trees, fans, etc. prevents false alarms. A total of six features (upper and lower envelopes, mean and median, normalized bandwidth, and normalized spectral entropy) are extracted from μ-Doppler to be used by the classifier. The feature extraction is repeated for consecutive frames within a sliding window, whose size is determined based on the tradeoff between the classification accuracy and the latency. In this study, a typical block length of 20-40 frames (2-4 sec @10 Hz) is used. We then build and train a 1D-CNN model to classify the target objects (human or non-human) given the extracted features as time series data.

The 1D-CNN architecture is shown in Figure 4-2. The input size is configured as the total number of features. Two blocks of 1D convolution, ReLU, and layer normalization layers are used, where the convolutional layer has a filter size of 3. To reduce the output of the convolutional layers to a single vector, a 1D global average pooling layer is added. To map the output to a vector of probabilities, a fully connected layer is used with an output size of two (matching the number of classes), followed by a softmax layer and a classification layer. As illustrated in

Figure 4-2, an optional block of 1D convolution with 64 filters, ReLU, and layer normalization layers (following the first two blocks) is also added to compare the performance of 2 vs. 3 layers CNN models

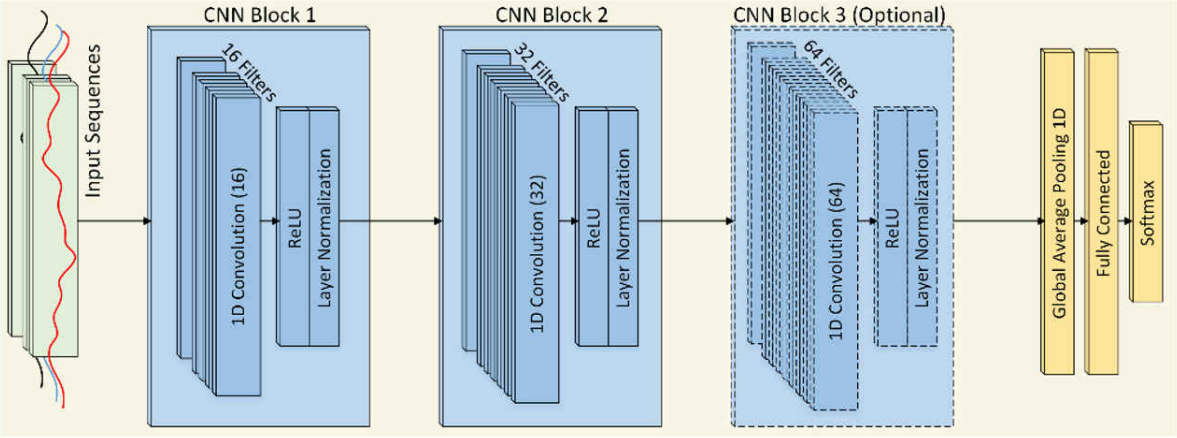
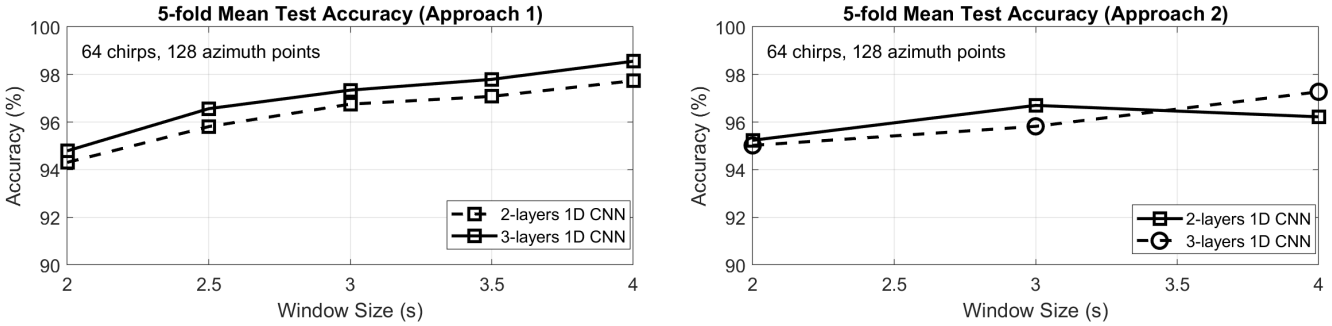


Figure 4-2. 1D CNN Architecture for Motion Classification

In the experimental results, a total of 125816 frames of data were captured at 10 Hz (corresponds to 3.5 hours in total) from different human and non-human targets. In the data capture campaign, 310 scenarios are created in distinct environments with around 20 different people and numerous non-human targets (e.g. fan, tree, dog, plant, drone, etc.). The data is captured with synchronized video to assist in labeling. In total, 31611 observations (14901 human, 16710 non-human) are generated when the time window size is configured to 20 frames (2 sec at 10 Hz), and 2 frames stride is applied to allow the changes in the statistics.

The performance of the classifier was characterized using two approaches. The first approach uses all the available data combined, shuffled, and split as follows: 40% for training, 10% for validation, and 50% for test. The second approach reserves some specific portion of the data set for testing, consisting of human and non-human subjects that were completely absent from the training set. With either approach, the overall classification error is calculated using 5-fold cross-validation. For each iteration (i.e., fold), data is reshuffled, and the accuracy is calculated with different random combinations of training, validation, and test data. The average of the test accuracies obtained from each iteration is then reported as the final accuracy metric. Figure 4-3 summarizes the performance of the first and second approaches, respectively. These results show that the proposed 1D-CNN classifier can achieve about 95% average accuracy in challenging test scenarios even with a 2 sec window and 2 layers.

Figure 4-3. Performance Analysis of the 1D-CNN



5 Case-Study-2: Gesture Recognition

The high range and velocity (or Doppler) resolution possible using FMCW radars is a good fit for a gesture-based touchless interface. Potential applications include controlling house hold electronics such as TV, light or thermostat, controlling the infotainment system or kick-based trunk opening for a car.

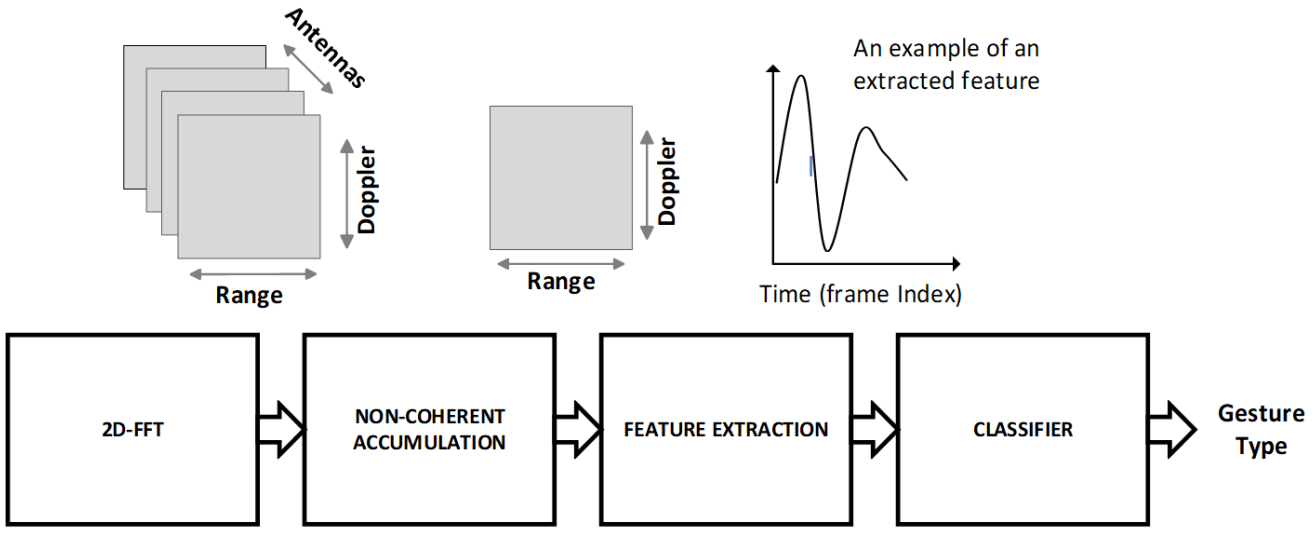


Figure 5-1. Processing Chain for Gesture Recognition

Figure 5-1 is a representative processing chain for a gesture-based recognition application [5]. First, the device performs a 2-D FFT on the ADC data collected across chirps in a frame. This resolves the scene in range and Doppler. The device then computes a 2-D FFT matrix for each RX antenna (or each virtual antenna if the radar is operating in MIMO mode). The non-coherent accumulation of the 2-D FFT matrix across antennas creates a range-Doppler heat map. The next step involves the extraction of multiple hand-crafted features from the range-Doppler heat map. We have identified a comprehensive set of 10 features that are useful in discriminating a variety of gestures. The hand-crafted features include basic statistics such as the average of range, Doppler, azimuth, and elevation angle, with the averages being weighted by the signal level in the corresponding range-Doppler cell. Note that the angle statistics cannot be computed solely from the range-Doppler heatmap but requires access to the complex radar cube data. Additional feature vectors are created from the correlation of the basic feature vectors (such as the correlation of Doppler and angle).

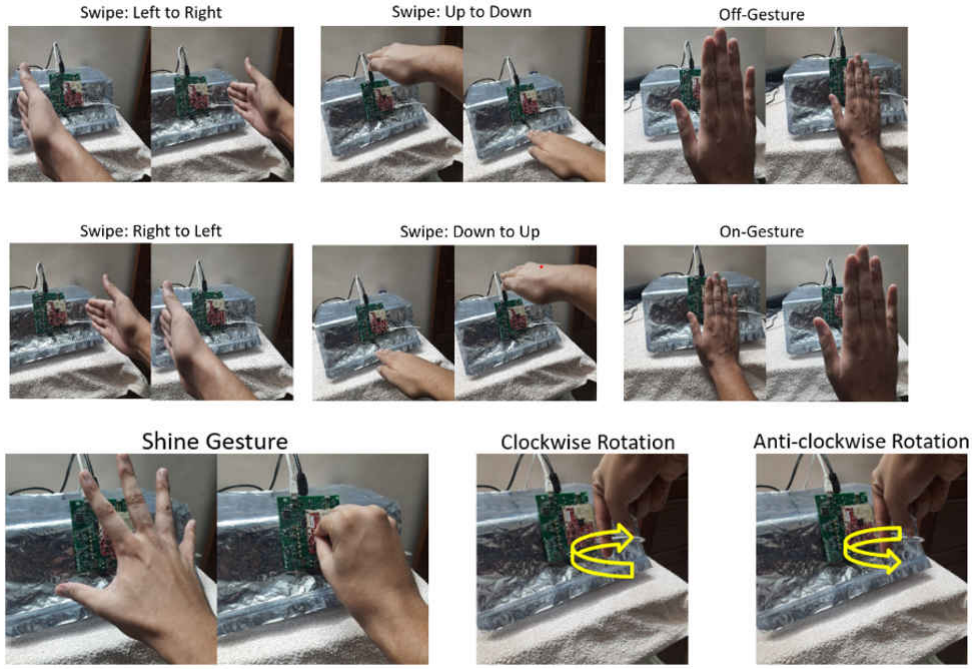


Figure 5-2. Example Gestures

The features extracted from the range-Doppler heatmap across a set of consecutive frames is sent as input to an Artificial Neural Networks (ANN) for the purpose of classification.

Figure 5-3 shows a representative implementation of an ANN for gesture recognition. The input to the ANN is a vector of length 60 consisting of 6 features across 15-frames. The first two layers of the ANN have 30 neurons and 60 neurons, respectively. The final softmax layer outputs 10 classes (nine classes corresponding to nine gestures shown in Figure 5-2 and one class indicating the absence of any valid gesture). This ANN was benchmarked on the M4 and found to take about 250us per invocation.

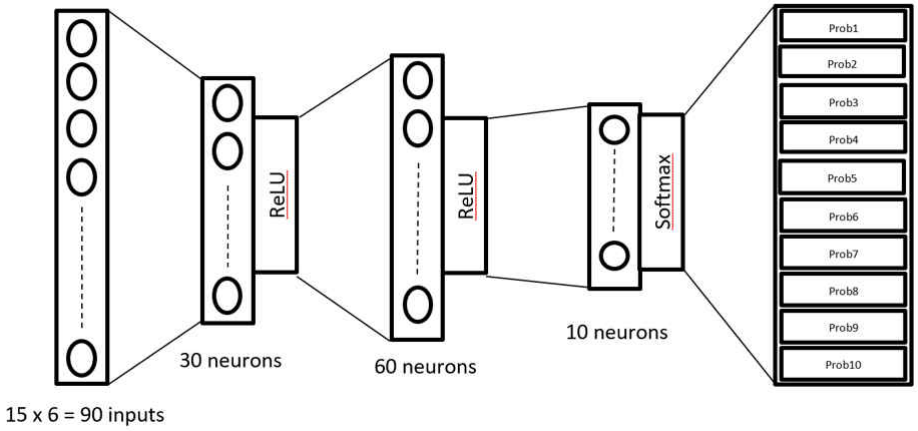


Figure 5-3. ANN Architecture for Gesture Recognition

Data collected from five subjects were used for testing and validation. The classifier was then tested on 10 subjects (different from the subjects in the training set). A testing accuracy of around 93% was achieved.

6 References

1. IWRL6432 data sheet
2. Fundamentals of mmWave Sensing, mmWave Training Series
3. S. Z. Gurbuz and M. G. Amin, "Radar-Based Human-Motion Recognition With Deep Learning: Promising Applications for Indoor Monitoring," in IEEE Signal Processing Magazine, vol. 36, no. 4, pp. 16-28, July 2019.
4. M. E. Yanik and S. Rao, “Radar-Based Multiple Target Classification in Complex Environments Using 1D-CNN Models,” to appear in 2023 IEEE Radar Conference (RadarConf'23), San Antonio, USA, May 2023.
5. P. Goswami, S. Rao, S. Bharadwaj and A. Nguyen, "Real-Time Multi-Gesture Recognition using 77 GHz FMCW MIMO Single Chip Radar," 2019 IEEE International Conference on Consumer Electronics (ICCE), 2019, pp. 1-4.75

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265