Person detection using FWCW radar and Long-Short Term Memory Convolutional Neural Network

About this document

This document contains algorithm description and implementational details behind pipeline for detecting persons on Range Doppler images generated by FMCW radar.

Scope and purpose

Scope of this document contains experiment and data acquisition description, data analysis and preprocessing algorithms and finally architecture of CNN LSTM used for classification as well as reasoning for choosing this approach.

Document's purpose is to highlight problems encountered during algorithm development and reasoning behind decisions that were made to combat these problems. After reading this document reader should be able to recreate results that are going to be presented.

Intended audience

Main audience intended to read this document are engineers at Infineon who will be judging algorithm as part of final round of EESTec Challenge Hackathon 2022. Besides these engineers this document can be helpful to every individual trying to solve some similar problem using FMCW radar.

Table of contents

1 Data acquisition

The goal of competition was to create model that will be able to recognize people based on Range Doppler images created using *Infineon BGT60TR13C* FMCW radar. In total there were 4 classes: zero, one, two and three persons.

1.1 Experiment

For the experiment, radar was placed on an elevated surface 1.5m above the ground. Raw data was collected at two different locations. Common for both locations is that radar was facing corner were there were no passersby with little foreign objects.

Recording process was done in several phases. In first phase only locations with no people were recorded. Second phase consisted of one person moving nonstop while the recording was taking place. After that more recordings were made with only one person in frame, only this time person was allowed to stop, stay still and that continue walking. This represented class with one person in frame. Same was done for recordings with two and three persons. Also, to add some variance, different objects were brought into scene (chairs, blackboard…).

None of the videos contained persons leaving or entering the frame. This was done for two reasons. First one was that doing it this way was much easier for labeling as all frames of one experiment were one class. In testing phase, entering and leaving the frame was a way to test algorithms robustness and generalization.

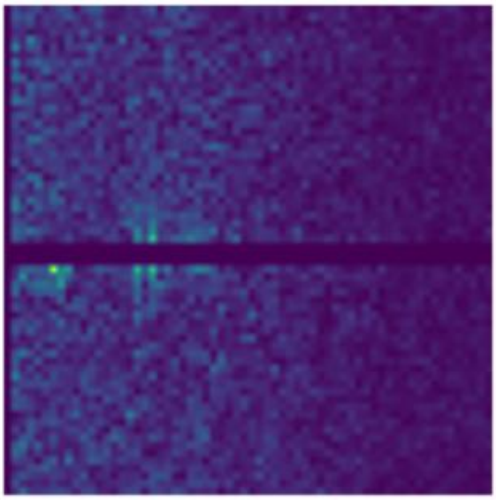
In total 23 recordings were made, lasting between 750 and 1500 frames and around 24000 frames in total.

2 Data preprocessing

Data preprocessing was done on Range Doppler maps. These maps themselves were created by applying fast Fourier transform twice on raw data collected from the sensor.

Looking at this data, DC component stands out compared to the rest of the range. DC component is an artefact of stationary obstacles bouncing signal back to the radar to be detected. As stationary obstacles are not of interest, they are removed by resetting pixel values in three rows on the center of an image. Resetting pixel values to zero has shown to be the least computationally expensive procedure allowing whole pipeline to work faster than sensors framerate.

Besides DC removal, before training each channel was normalized to range [0,1]. This allowed better and faster network training.



(Image 3) One channel after preprocessing

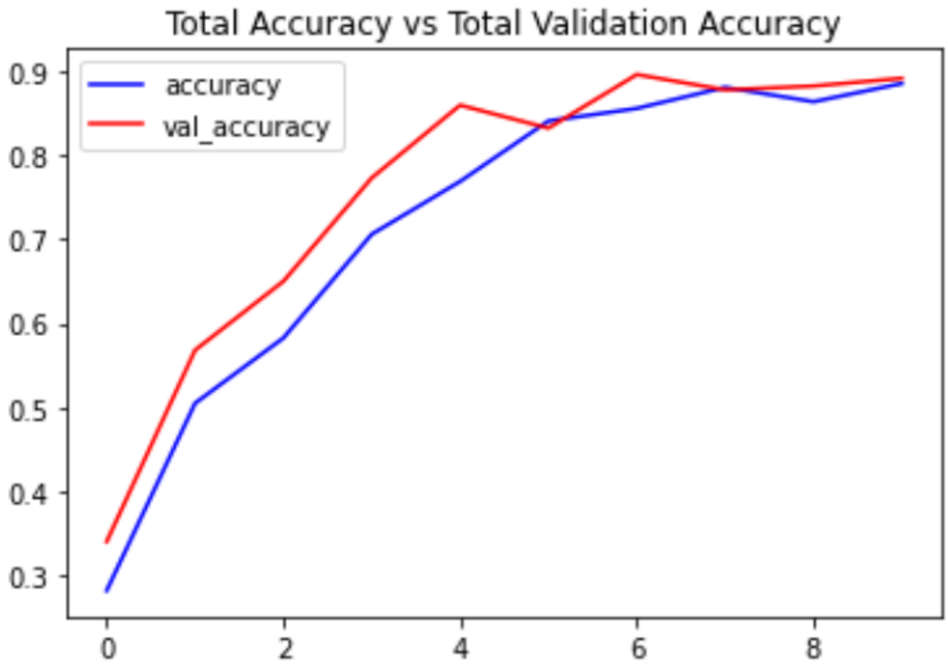
3 Model

Model used for this algorithm was CNN LSTM. As people are invisible when they stand still in front of the radar, there had to be a way to make decision not only based on current frame. One way to do that is to use Long-Short Term Memory as a way to take into account time. This model is fed with time series that is used to constantly modify its parameters so that it can make better decision. This approach has allowed us to differentiate between cases when people are standing in frame and should be counted and cases where people went out of radars range.
As LSTM network take a lot of computational power to train, feeding them images would be too time consuming. In order to reduce input complexity, before LSTM there are several convolutional layers used for feature extraction. These CNN layers enable us to condense images into more manageable data before feeding them to LSTM.
The architecture itself looks like:

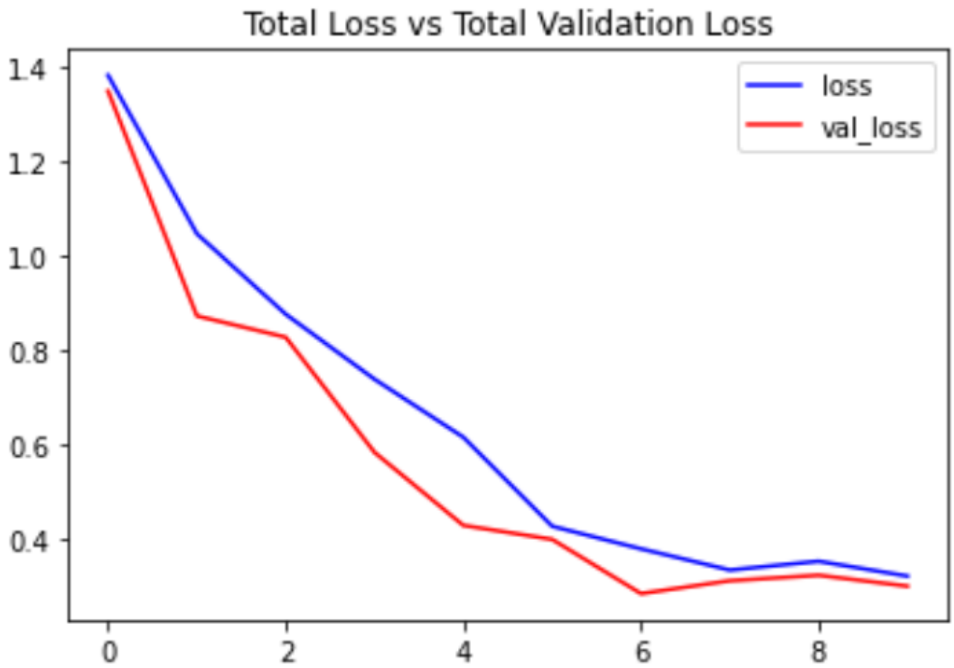
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_13 (TimeDis (None, 10, 64, 64, 16)    448
_________________________________________________________________
batch_normalization_4 (Batch (None, 10, 64, 64, 16)    64
_________________________________________________________________
time_distributed_14 (TimeDis (None, 10, 16, 16, 16)    0
_________________________________________________________________
time_distributed_15 (TimeDis (None, 10, 16, 16, 16)    0
_________________________________________________________________
time_distributed_16 (TimeDis (None, 10, 16, 16, 32)    4640
_________________________________________________________________
batch_normalization_5 (Batch (None, 10, 16, 16, 32)    128
_________________________________________________________________
time_distributed_17 (TimeDis (None, 10, 4, 4, 32)      0
_________________________________________________________________
time_distributed_18 (TimeDis (None, 10, 4, 4, 32)      0
_________________________________________________________________
time_distributed_19 (TimeDis (None, 10, 4, 4, 64)      18496
_________________________________________________________________
batch_normalization_6 (Batch (None, 10, 4, 4, 64)      256
_________________________________________________________________
time_distributed_20 (TimeDis (None, 10, 2, 2, 64)      0
_________________________________________________________________
time_distributed_21 (TimeDis (None, 10, 2, 2, 64)      0
_________________________________________________________________
time_distributed_22 (TimeDis (None, 10, 2, 2, 64)      36928
_________________________________________________________________
batch_normalization_7 (Batch (None, 10, 2, 2, 64)      256
_________________________________________________________________
time_distributed_23 (TimeDis (None, 10, 1, 1, 64)      0
_________________________________________________________________
time_distributed_24 (TimeDis (None, 10, 1, 1, 64)      0
_________________________________________________________________
time_distributed_25 (TimeDis (None, 10, 64)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                12416
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 132
=================================================================
Total params: 73,764
Trainable params: 73,412
Non-trainable params: 352
```

4 Training

Before training three different datasets were created. 10% was saved as test data, while 20% of the remaining data was saved as validation. During training metrics that were followed were training accuracy and loss as well as validation accuracy. Training process was stopped after three epochs in which validation accuracy has dropped. At the end of the training and after the testing, results are:



(Image 1) Training and validation accuracy



(Image 2) Training and validation Loss

test accuracy = 95%

Because of specificity of the problem and time restrictions, hyperparameter fitting was done manually with most common values used for DNNs instead of basic cross-validation methods. Values of these parameters are:

$$\textit{batch size} = 8$$
$$\textit{epochs} = 20$$
$$\textit{iterations} =$$
$$\textit{sequence} = 10$$
$$\textit{overlapping} = 40\%$$

Training of LSTM neural networks is somewhat different than training of a regular CNN. Instead of feeding data one sample at a time, LSTM require data series as its input. In our case we opted for sequence with the length of 8. Neighboring sequences have overlapping of 40%.

4.1 Complexity

In the end, all that was left was to test model on live data. With one thread implementation we achieved average framerate of 9 fps. That showed to be to slow for this use case thus giving poor results. After dividing data acquisition in one thread and data processing in another, average framerate achieved was 19fps. This showed to be more than enough to work on real time data, achieving good precision on all classes.

5 Future work

One of the things we would like to do in future would be to change preprocessing algorithm. In our case, we are destroying DC component completely. By inducing some calibration procedure, we could model the environment and use that for removing stationary object. Doing thing this way could possibly allow us to make algorithm even more robust for persons entering the frame and standing still for longer periods of time.

Expanding dataset could also possibly increase performance. This expansion could be done either by filming at different locations or by generating synthetical data similar to the data already present in dataset. Data augmentation could also be used for this purpose.

And lastly, hyperparameters could be better fitter by using some cross-validation method.

Revision history

Document version	Date of release	Description of changes
V1.0	26.05.2022.	Creation of the document

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition yyyy-mm-dd

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AppNote Number