Application Notes

EZTech - EESTech Challenge

Our Solution

# Scope and purpose

In this project we had to do a Machine Learning Algorithm that detects the number of people moving in an indoor room using data provided by Frequency-Modulated Continuous Wave (FCMW) Radar.

# Intended audience

We hope with this project to implement a simple algorithm that could count the number of people inside an indoor room. We did a algorithm that could easily be run in any recent , so anyone with the radar and a computer can run this program.

Table of contents

[Our Solution 1](#_Toc104475172)

[Table of contents 1](#_Toc104475173)

[1 Project 2](#_Toc104475174)

[1.1 Gathering and handling the data 3](#_Toc104475175)

[1.2 Convolutional Neural Network 6](#_Toc104475176)

[1.2.1.1 CNN Structure 6](#_Toc104475177)

[1.2.1.2 CNN Saving 6](#_Toc104475178)

[1.2.1.3 CNN Loading 6](#_Toc104475179)

[Revision history 8](#_Toc104475180)

# Project

# For this project we decided to use a Convolutional Neural Network (CNN) to classify the data we gather from the FCMW Radar. We decided on a CNN due to its simplicity and great compatibility with 2D data. Also, the CNN can be easily run in almost every recent computer/smartphone due to its low computational effort, we can see that by the time that the prediction takes, 24ms.

# During training the data, we achieved an accuracy of 77,25%, we know that this doesn’t correspond to the real-world accuracy due to many factors, like overfitting, different places that we gather the data, the small amount of data that we gather compared to what is needed (we had to gather our own data, so we didn’t had time to build a huge dataset), etc.

Also, we decided to use some out of the box ideas and implementation, we will talk more about these ideas in the following points.

## Gathering and handling the data

A big part of this challenge was the data gathering, since none was provided to the participants to train and test the models. The conditions where the data was gathered were not ideal but good enough to see promising results.

So, as to gather the data from the sensor, we used the pre-processing function provided by Infineon. In addition, we added some lines of code for data processing, as seen in the following table:

| Code Listing 1: |
| --- |
| 1. for k in range(0,data\_epoc): 2. for j in range(0,4): 3. display\_text("STOP - NEXT: " + text[j]) 4. time.sleep(1) 5. display\_text(text[j]) 6. for i in range(0,f2r): 7. raw\_data.clear() 8. hit = 0 9. for i\_frame, frame in enumerate(device): # Loop through the frames coming from the radar 11. raw\_data.append(np.squeeze(frame['radar'].data/(4095.0))) # Dividing by 4095.0 to scale the data 12. if(i\_frame == number\_of\_frames-1): 13. data = np.asarray(raw\_data).astype('complex128') 14. range\_data=processing.processing\_rangeDopplerData(data,True) 15. #range\_data=np.abs(range\_data) 17. for n1 in range(3): 18. for n2 in range(number\_of\_frames): 19. aux = aux + np.abs(range\_data)[n2,n1,:,:]; #average of the 10 frames 21. data\_dump.append((aux/number\_of\_frames)) 22. #print(np.shape(data\_dump)) 23. break 24. ts=calendar.timegm(time.gmtime()) 25. hit=hit+1 27. print("finished frames") 28. print(np.shape(Y\_train)) 29. print(np.shape(np.ones((f2r), dtype=float)\*j)) 30. if np.size(Y\_train) > 1 : 31. Y\_train = np.append(Y\_train, np.ones((f2r), dtype=float)\*j) 32. else: 33. Y\_train = np.ones((f2r), dtype=float)\*j 34. print("training") 35. #Neural Network 37. X\_train=data\_dump |

In theses lines of code, we basically gather the data from the sensor and store the Doppler Data in the *X\_train* variable, with this data we can train our model. We decided to use the data that we gather directly in the model, training it in real-time.

We decided on this real time approach due to the great size of the data needed to be used, it would be simpler and more effective since there would be no need to store this data and load it repeatedly. As such, we train the model in real time for quicker and less memory demanding computing. This is similar to a calibration and can be used to better adapt for each case.

Also, we store all models so we could use in another time without the need to train the model again, we have a program for that purpose (**load\_model.py**).

For the data that we gathered to be useful, we had to label it, so we decided to do a program that presented the number of moving people that had to be in front of the sensor, as can be seen by the following picture:

Uma imagem com texto, interior, computador

Descrição gerada automaticamente

Figure 1 – Display showing how many people should

be in front of the sensor during data acquisition

In addition, we had to introduce a timeout so the next person could come in on time and the data wouldn’t be badly labeled. So, for this purpose we present the following message on the screen:

Uma imagem com texto, interior, computador, portátil

Descrição gerada automaticamente

Figure 2 – Display showing how many people will need to

be in front of the sensor in the next stage of data acquisition

Also, like everyone else we had to be our on-test subjects so when the program told that we had to have one person moving in front of the sensor, one of us had to be there, like we can see in the following image:

Uma imagem com interior, teto

Descrição gerada automaticamente

Figure 3 – Data acquisition process

## Convolutional Neural Network

For the Machine learning model, we used a CNN, as priorly mentioned. As said before it is very simple and effective to use. It is also very not very resource demanding compared to other options, as we can see by the 24ms that took to predict the number of people in front of the radar.

The input data to this model was a 64 by 64 by 1 ***numpy*** array. As we were using this type and dimension of data. We decided on using a 2D convolutional layer.

### CNN Structure

The CNN was developed using the ***tensorflow*** library, with special use of ***keras***. We decided on this due to its simple implementation, versatility, and the team’s experience with the library.

The CNN created is sequential and composed of 2 convolutional layers of 128 with kernel sizes of (4,4), with hyperbolic tangent (*tanh*) activation before an average 2D polling layer. It is then followed by 2 more convolutional with 64 with the same kernel sizes and activation.

After this, its followed by a max polling and flatten layer, and then by a 2 fully connected 1028 layers and finally a 4-neuron layer with *softmax* activation for classification.

The code of this CNN is as followed:

| Code Listing 2: |
| --- |
| 1. model = models.Sequential() 2. model.add(layers.Conv2D(128, (4, 4), activation='tanh', input\_shape=(64, 64,1))) 3. model.add(layers.Conv2D(128, (4, 4), activation='tanh')) 4. model.add(layers.AveragePooling2D((4, 4))) 5. model.add(layers.Conv2D(64, (4, 4), activation='tanh')) 6. model.add(layers.Conv2D(64, (4, 4), activation='tanh')) 7. model.add(layers.MaxPooling2D((4, 4))) 8. model.add(layers.Flatten()) 9. model.add(layers.Dense(1024, activation='tanh')) 10. model.add(layers.Dense(1024, activation='tanh')) 11. model.add(layers.Dense(4, activation='softmax')) |

### Model Saving and Loading

Every CNN developed is saved for posterior use and testbenching.

We save the CNN by using the following code.

| Code Listing 3: |
| --- |
| 1. model.save(filename) |

We also developed a program solely for loading and running a model to see its “true” performance (**load\_model.py**).

This is used to test a model not by the accuracy obtained with a dataset but for a real-world applicability.

## Results

### Data Results

For the models developed, most obtained good results for the data acquired during the data acquisition process, with the highest accuracy obtained during validation of 77,25%.

There is, however, a great difference between the capability of obtaining great results with virtual stored data and obtaining results with real world and real time captured data. The “virtual” results do not reflect the real-world applicability.

There are a lot of factors that explain this difference such as poor conditions when acquiring data: people walking behind the defined area and other sensors being used close by. Also, the constraint of not being able to obtain large amounts of data make it difficult to generalize to most cases.

### Real World Applicability

Even with the constraints, the results show that a solution based on the work done during this hackathon is possible, since there were models capable of distinguishing fairly well the number of people.

Revision history

| Document version | Date of release | Description of changes |
| --- | --- | --- |
| 1.0 | 26/05/2022 | Added Solution and Project Chapters |
|  |  |  |
|  |  |  |