Smartphone Sensing 2023
App 2, Option 2
group 9
Samsung Galaxy A13, Android 13

1 Load balance

1.1 Tony Yang, 5651794

- Android Method
  - Plot spectrogram, and save as image
  - Automated functions for data gathering
- Off-line Method
  - Exploring RNN + CNN for novelty points
- Tasks and Deliverable
  - Gather part of training/testing data

1.2 Giacomo Mazzola, 4872673

- Android Method
  - Deploy CNN model online
- Off-line Method
  - CNN implementation for spectrogram
- Tasks and Deliverable
  - Gather part of training/testing data

2 Methods

2.1 Signal Optimization

The chirp signal, varying from `12kHz` to `13kHz`, is deliberately omitted for a period of `10ms`. Initial analysis of environmental noise established the frequency range above 10kHz as noise-free. However, a correlation was later detected: an increase in chirp signal frequency corresponds to a decline in power. A frequency range between 12 and 13kHz has been determined to yield optimal results. Consequently, the omission duration has been aligned with this pattern, being extended to 10ms.

2.2 Real-Time Spectrogram

For on-the-fly integration of the neural network, real-time plotting of the spectrogram is required. Leveraging the `Apache Commons Mathematics Library` [1], Fast Fourier Transform (FFT) is performed, facilitating the execution of Short-Time Fourier Transform (STFT). The related values for FFT and STFT are set as `FFT size = Window size = 256`, `Window type = Hann`, `Overlap = 128`, `Sampling frequency = 44.1kHz`, and with `Zero-padding` excluded. `Bitmap` is employed to plot the spectrogram matrix, using a `grayscale` theme to maximize information-to-pixel efficiency. Noteworthy considerations include: (1) the relevant frequency range, specifically 12kHz to 13kHz, along with an additional frequency bin above and below, is encompassed in the plotted spectrogram; (2) the extracted spectrogram matrix's size is `10×50`. For enhanced visibility on the bitmap, it is upscaled to `20×100` for display and archiving, and the training process operates on this upscaled image.

2.3 GUI

The GUI of the App is as depicted in Figure 1.

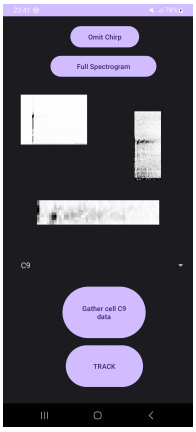

Figure 1: Current GUI (Spectrograms displayed with buttons to gather each corresponding cell)

2.4 Machine learning model

Given the two dimensional nature of the spectogram extracted as main feature from the recording of the chirp's echo, we opted to implemented a CNN using python based on PyTorch as opposed to a DNN. The model is trained on a laptop Dell XPS15 equipped with a Nvidia GeForce RTX 3050 Mobile and 32 GB of RAM, requiring a training time of about 1 minute.

2.5 Description of CNN

The CNN consists of the following layers: `conv1, maxpool1, conv2, maxpool2, linear1, linear2`. A short description of each layer is given below:

- `conv1`: it applies 16 3×3 convolutional filters to the spectogram with a stride of 1 and a padding of 1. In addition, this layer applies to the 16 generated output images the ReLU function to introduce some non linearity in the process. The ReLU

function was chosen instead of other non-linear activation functions, such as Sigmoid, because it allows to reach convergence faster.

- `pool1`: it applies a max pooling function with a size of 2 and a stride of 2 to each output image of conv1 layer. the pooling layer is important because, by reducing the size of the images, it makes the training process less computational intensive and prevents overfitting.

- `conv2 & pool2`: these layers are very similar to `conv1` and `pool1`. At the end of this layer, the images are flattened and concatenated into a vector of size 4000 (5×25×32) to be fed to the fully connected network.

- `linear1`: it is the first layer of the fully connected network, which applies a linear function to an input array of size 4000 (5×25×32) and reduces it to an output array of length 64. At the end of this layer a `ReLU` is again applied to the output to introduce some non linearity in the network.

- `linear2`: this is the last layer of the fully connected network which takes as input an array of length 64 and outputs and array of 7, which represents the classes involved in the classification problem.

The hyperparameters of the network were fine tuned experimentally to `batch_size`= 32, `epoches`= 150 and `learning_rate`= 0.0001.

Finally, the loss function used to train the network is `CrossEntropy` and the optimizer we chose is `Adam`.

2.6 CNN on Android

The trained CNN model, saved as a `.ptl` file using the `optimize_for_mobile()` function in PyTorch, can be loaded into an Android application using the `LiteModuleLoader` provided by the PyTorch library. This allows the model to be deployed and used for location detection on the Android platform.

To perform location detection, a sample input is provided to the loaded model in the Android application. The input is processed by the model, and the class with the highest score or probability is selected as the predicted location. This predicted location can then be displayed on the screen of the Android device.

2.7 Training & Testing process

To create a usable dataset to train the network, we collected hundreds of data samples for each cell from C1 to C7. Specifically, we gathered 400 samples for C1, C2, and C3, and 200 samples for the remaining cells. However, the data collection process is still ongoing, and we plan to gather a total of 1000 data samples for each cell under different days and ambient conditions.

To ensure the validity of our study, we randomly selected 20 samples for each cell to form the testing set. The remaining samples are utilized for training your model.

3 Evaluation setup

Each cell from C1 to C7 underwent real-time online testing `20 times`. The comprehensive outcomes of this evaluation for all four cells can be found in the results shown in Section 4.

4 Results

4.1 Accuracy

The evaluation phase reveals an approximate accuracy of 80%. The Confusion matrix in Figure 2 indicates that the accuracy of cells C1, C2, C3, and C7 is nearly optimal. However, the model's performance is relatively poorer on cells C4, C5, and C6. This observation can be attributed to the significant similarity between these cells, as they consist of three identical spots situated on the stairs across three floors.

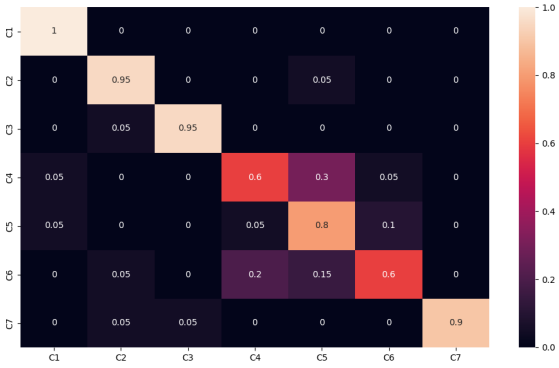

Figure 2: Confusion matrix of the evaluation results

4.2 Robustness

The model's robustness is currently weak due to the low ambient noise conditions during sample collection and evaluation. To improve the model's performance in varying environments, a two-week training phase is scheduled. By exposing the model to different settings during training, it will become more resilient and better able to handle real-world scenarios.

5 Conclusion

This report highlights the progress made in the Smart Phone Sensing project during assignment 2, comparing it to the previous report. Currently, the model has been trained on 7 out of the total 16 cells present in building 28.

The goal for assignment 3 is twofold:

- Improve network accuracy and robustness and extend it on all the available cells.

- Implement a novel technique (CNN+RNN) to improve the performance of the network.

References

[1] Apache. *Commons Math: The Apache Commons Mathematics Library*. `https://commons.apache.org/proper/commons-math/`. Accessed: 2023-05-21. 2023.