Indoor Location Sensing Using Smartphone Acoustic System

What kind of deep models could be used for indoor location recognition? How to deploy and evaluate the model on smartphones and make the inference run in real time?

Radoslav Sozonov[1]

Supervisor(s): Qun Song[2]

[1]EEMCS, Delft University of Technology, The Netherlands

Abstract

Indoor localization is a field in a development process. Different solutions have been introduced in recent years. Some of the solutions use beacons, WI-FI access points, different smartphone sensors, or acoustic sensing to make localizations. This paper is presenting an application that uses acoustic sensing data to perform localization with different deep models. The research aims to explore different models and evaluate their performance metrics in the classification of three different acoustic data sets and their overhead on the system. Two different architecture designs are implemented - a client-server one as the models are stored on the server and one only front-end oriented as in this case compressed models are used. The results show that the client-server approach outperforms the front-end only design as the former's models reach classification results of 98%, 90% and 90% tested on three different data sets, despite taking longer to fetch a prediction result from the server compared to the compressed models stored on a smartphone device.

1 Introduction

Indoor localization using the acoustic system of smartphones is a recently introduced topic that is in a development process. For an outdoor environment, the global positioning system (GPS) is suitable for sub-meter localization but for indoor locations, it is not reliable, because GPS service is often not available in such places or the localization is not accurate [1]. It is worth solving such a problem because it will revolutionize the localization for closed places which can be helpful for indoor navigation applications for big buildings where you can get lost easily. Additionally, for the development of such technology is important to evaluate different architectural application designs and performance metrics on different data sets to achieve the maximum.

The contributions of this paper include (i) in-depth research and experiments on what deep models with different architectures can be used for indoor localization sensing, Floating-point operations (FLOPS)[2], time to train a model, and time to fetch the prediction result using Convolutional neural network (CNN), Recurrent neural network (RNN) and Dense neural network (DFNN). (ii) the design of the system for successful deployment and evaluation and (iii) inference run and models' robustness in real-world environments such as a public building with collected different data sets.

For the successful completion of the research, a mobile device is used to emit a fixed number of 2ms 20kHZ inaudible chirps every 100ms by the loudspeaker and the same device microphone to capture the reflected frequencies. Then this data is converted to spectrograms and provided to the deep models to evaluate their performance. During the training of the models is computed the FLOPS taken for each model, training parameters and the required time to train it.

The results from the research show that the architecture design with a client and a server performs better than an application with a front-end part only because on the front-end should be used compressed models that deteriorate the performance on the data sets, despite taking more time to fetch the predicted result from the server, 60-100ms compared to 1ms for the latter. On the server, the models can achieve classification results of 98%, 90%, and 90% tested on three different data sets. Additionally, the models with more parameters take longer time to train and have more FLOPS except for the RNN models whose values for the last metric do not pass 1500 FLOPS.

The rest of this paper is organized as follows. In Section 2 is presented the related work and background of the deep models mentioned in this section, then in section 3 the measurements study to understand rooms' responses to inaudible chirps. In the next section, 4, is presented the methodology. In 5 and 6 are implementation setup and results from the research are discussed, in the next one, 7, is discussed the responsible research. Section 8 is the discussion and the last section, 9, is the paper's conclusion and future work on the problem.

2 Related work and Background

This section presents papers with research connected with the work in this one and introduces background information about the deep neural networks involved in the experiments.

2.1 Related work

Different Machine learning models have been used for solving problems about localization and positioning in closed environments. Batphone [3] uses the nearest-neighbour model to calculate the acoustic background distance to previously recorded rooms and this way to make localization. This approach was able to reach 69% accuracy in classifying 33 rooms in a quiet environment.

With the development of deep models and neural networks, they started to be used more often for acoustic sensing. The deep models in combination with spectrogram features outperform SVM models as stated in [4], they use the k-SpecNet model for localization and classification of overlapping sound events based on spectrogram-keypoint using acoustic-sensor-network data.

In [5] and [1] is presented a work with a client-server application similar to the one described in this paper. This application also emits 20kHZ chirps every 2ms on every 100ms and captures the reflected frequencies by the microphone, then the collected data is converted to spectrograms and used for training the models. For models, they use DNN and CNN models for making classifications as they achieve on average 97% accuracy with their CNN on different data sets from different places.

2.2 Background

A lot of different machine learning and deep learning models can be used for the classification of spectrograms but as always some have better performance in some metrics compared to others and vice-versa. Traditionally models like SVM, SGD, KNN and DTC can be used for the classification of spectrograms as used in [6], but with the development

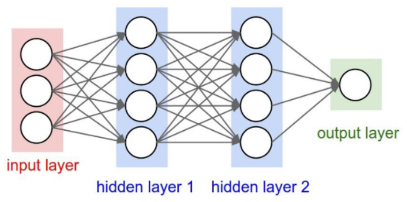

Figure 1: Dense Neural Network [8]

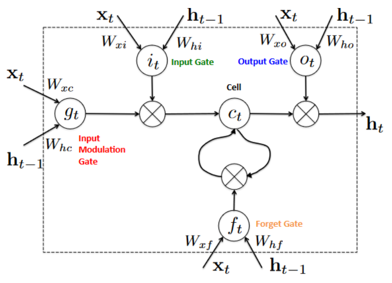

Figure 2: Arhitecture of LSTM unit [9]

of the deep models and when being fed up with a sufficient amount of data they can achieve higher accuracy compared to the traditional models. They can learn better on complicated data representations for classification. The evidence for this is stated in [7]. Although the deep models have better accuracy performance they are more time-consuming for training in comparison to the machine learning models because the latter are more simple models with fewer parameters.

In the deployed server, the full potential of the models is used while for smartphone deployment the deep models are compressed, meaning that they do not operate at their full potential. For both architecture design types are used DNN, RNN and CNN models.

The DNN models consist of fully connected layers with nodes in them. The nodes in a layer are not connected, they are connected with all nodes from the previous step and next step layers as in figure 1. The training algorithm determines the weights and the biases for each edge between two nodes in the network to best fit it to the input data. The input for the first layer is flattened 1D spectrogram data, then the input data for each hidden layer is the result of the activation function in the previous one. The data goes through multiple hidden layers until it reaches the output one which consists of the same number of nodes as the different classes in the training data. This layer returns probabilities of how probable is the input data to be a certain class.

Unlike the DNN model, the RNN and the CNN models can take as input multidimensional data. The RNN also consists of input, hidden and output layers. The difference is in the hidden layers. The hidden layers consist of Long-Short Term Memory (LSTM) units. This network design has been first invented by Hochreiter and Schmidhuber in 1997 [10]. These units are trying to solve the problem of the vanishing gradient passing through the layers. The core of the units is the so called ”memory cells” that can learn tasks from events that happened 1000 or million-time steps earlier in the network

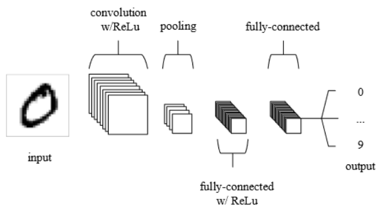

Figure 3: Simple architecture of a CNN model [12]

[11]. The LSTM consists of 4 gates - input, output, input modelation and forget gates, as the last one controls what information to drop and what to keep [9]. The architecture of the unit can be seen in figure 2.

The CNN model consists of convolutional, pooling, and fully connected (or dense) layers that search for patterns in the data. The convolutional layers aim to apply an ’element-wise’ activation function to the outputs of each neuron, while the purpose of the pooling layers is to downsample on the data, reducing the parameters [12]. The parameters of this network are set up during the training process, except for the pooling layers which do not learn. Figure 3 presents a simple CNN architecture of 5 layers.

3 Measurements study

In this section is presented the measurement study and showed why our architecture design and approach are suitable to solve the mentioned problems in section 1. Acoustic data has been collected from 5 different location points in a public building. The sketch of the building floor is shown in figure 4.

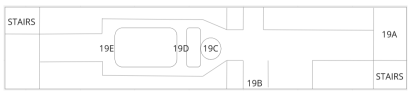

Figure 4: Floor plan of the 19th floor of EWI - faculty building in the TU Delft campus. The letters show the spots from where data is collected

3.1 Measurement Setup

The front-end part of the application has been deployed on a Samsung Galaxy A51 because the smartphone is able to emit and capture high frequencies of 20kHZ. This is shown in figure 5. There is presented the 2ms chirp at 20kHZ emitted and captured after that by the same device. The developed front-end app emits 2ms inaudible chirps every 100ms using the phone’s loudspeaker. Alongside this process, the app is continuously using the microphone of the same device to capture the frequency reflected responses at a rate of 44.1ksps. The collected data is sent to the server where it is stored on a database for future use. Before being stored, the first 3ms of each 100ms recording is removed. The first 2ms data is directly propagated from the phone’s loudspeaker to its microphone and the remaining 1ms is removed just because of a safeguard because the first data reflected is from the person who is holding the device so it is not data reflected from the

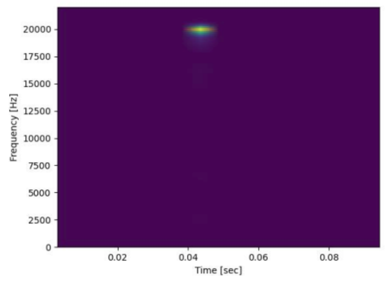

Figure 5: Spectrogram showing 2ms chirp at 20KHZ in the span of 100ms

surrounding walls. For each data point, 1004 chirps are emitted as the first four and the last two are removed because of data corruption. This amount of data is sufficient to understand the places' acoustic responses.

A few different studies on active acoustic sensing use sweep chirp [13], multi-tone chirp [14] and ets. These chirps cover a wide acoustic spectrum but they can be annoying for the others. So for this application is used 20kHZ following the approach in [1] where an experiment with frequencies between 20kHZ and 21.6kHZ has been conducted. The results there show that despite 20kHZ being the lowest inaudible frequency, it is at the same time the one with the highest signal power, so for the remainder of this paper 20kHZ is the chosen lowest inaudible frequency played by the developed application. Because of the mechanical dynamics of either the loudspeaker or the microphone a frequency above the halve of the sample rate (44.1ksps) can not be used.

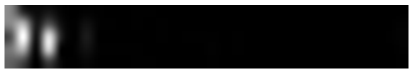

Figure 6: Spectrogram for spot A from DS1

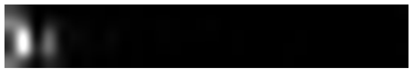

Figure 7: Spectrogram for spot A from DS2

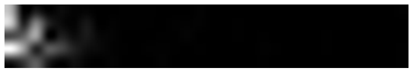

Figure 8: Spectrogram for spot C from DS1

For the classification of the data, RNN, DNN and CNN models have been chosen with API provided by TensorFlow [1]. All of the models are capable of classifying 2D and 1D data and these models are often used for image classification and pattern recognition. For the design with the back end, the full potential of the models is used while for the front-end only application, these models have been compressed again using Tensorflow APIs to make them work on an Android application.

For the evaluation of the models has been used their accuracy in classifying the data sets, confusion matrices showing which places are classified correctly and incorrectly, number of FLOPS and time taken during training. The training of the models is performed on laptop Lenovo Legion Y520 with 16GB Ram and 4-cores.

[1]https://www.tensorflow.org

3.2 Time-Frequency Analysis

The input data for the models are spectrograms, a time-frequency representation of the collected data. To generate the spectrograms, 256-point Hann windows with 128 points of overlap between two neighbour windows are applied to generate 32 data blocks from 4278 out of the 4410 data points in the collected data intervals. The used spectrogram is a grey-scale image with frequencies between 19.5kHZ and 20.5kHZ. The size of the spectrogram is 32(time) x 5(frequency).

The spectrograms in figures 6 and 7 can be noticed that they have the same outlook. The reason for this is that they are from the same spot 19A but from different data sets. This shows that the data collected from different sets at the same location spot has the same spectrogram out-look. In figure 8 is presented a spectrogram for spot 19C. This spectrogram is different from the ones for spot 19A because the reason is that it is another spot location. This shows that the different spots have different spectrograms out-look.

4 Methodology

This section starts with presenting the Problem statement, followed by System architecture design, Neural network models and Models real-time classification evaluation.

4.1 Problem statement

The key question is what models can be used for classification and where is more suitable to deploy the models - on a separate server or on a smartphone and what are the advantages and the disadvantages of the two approaches.

All of the chosen models are frequently used in image classification, pattern recognition and speech recognition as to some extent all of these fields take part in our research as the input data is a collection of spectrograms that represents grey-scale images of acoustic data with different patterns for the different locations.

Both architecture designs are possible for implementation. However, this paper presents what are the strong and weak sides of both of them based on results from experiments.

4.2 System Architecture design

Two different architectural designs have been implemented. One with front-end and back-end parts and one only with front-end. In both cases the front-end part is responsible for data collection as on the implementation with the back end the data is sent to it, where it is pre-processed and stored on a database while for the front-end only part the data is pre-processed on it and stored there. For the model creation, the architecture with two parts trains the models on the back end using the data from the database, while for the implementation with only one part, the models are trained in a separate Python environment and after that included on the front end. Extended information about the implementation details is provided in section 5.

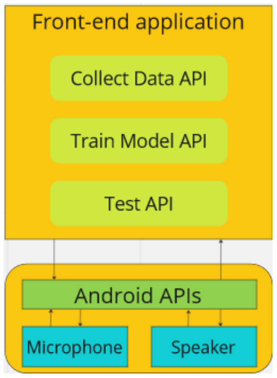

Figure 9: Frontend design

4.3 Neural network models

All models are trained using TensorFlow API. At the current moment, it is only possible to train the model in a Python environment, so the models created at the server are also used on the front-end only application with the difference that these models on the front end are compressed and used only for predictions.

For each neural network type, 6 models with different architecture designs are used. For each type, there are 2 models with 300K parameters, 2 models with 700K parameters and 2 models with 1500K parameters. During the training of each model is computed the time and the FLOPS it takes to train it. The results from the training for each model are presented in section 6.

4.4 Models real-time classification evaluation

Once the models are ready and compressed and work properly, then their accuracy is tested on three data sets created at three different times as the models have been trained on the first collected data set. Additionally is computed the time it takes for making a prediction using the back-end stored and compressed models on the front end.

5 Implementation

This section presents the Data collection, the Similarities and the Different parts of the two design approaches, the Models training process and finishes with Hyperparameter settings

5.1 Data collection

For each location, 1004 chirps are emitted every 100ms and collected the reflected data from them by a smartphone. The collected data will be divided into 1004 intervals of 4410 data points and 4278 of them will be converted into spectrograms with 32x5 dimensions. Three data sets have been collected using the same device.

5.2 Common parts between the two design choices

The common part of both designs is the Collecting Data APIs and Test API. Figure 9 presents the common design part of the front end as only adding the training API for the approach with the back end.

The front-end application is responsible for emitting and collecting data. This part emits a fixed number of 20kHZ chirps every 100ms as every chirp is 2ms. In our case, the chirps are 1004. The chirp audio data for 100ms are initially set up and then reproduced 1004 times. This way the played audio track plays the chirps on the same interval. Otherwise, if every time the audio data for the chirps is generated then the interval will not be exactly 100ms between every two chirps because of system delays. An array of shorts is used to represent the generated audio data. One 100ms interval consists of 4410 data points as only the first 88 data points represent the chirp signal values while the rest data points are zero. The reason is that we need exactly 2ms chirp and the rest of the 100ms interval has to be quiet. Alongside this thread of operations, there is a second one.

In the second thread, the collection of the raw data is happening. The collection thread should start just before the thread that emits the chirps. The collected data is stored in an array with size 4410 * 1004 of shorts. After that, the collected data is sent to the back end or stored as wave files in the device memory. For the emission and data collecting are used the Android APIs AudioTrack[2] and AudioRecord[3].

5.3 Different parts between the two design choices

The difference between the two implementations is that one has a separate server application, while the other one has not and the data stays in the same application and is not sent.

Front-end and Back-end design

Once the data is collected, it is sent through an HTTP request to the back end with a specified label and a building name. There the data is prepossessed. The start of the first chirp is determined because the data collecting and chirp emission can not be started exactly at the same time with no delay and the start of the chirp should be found. The data points present an array with shorts showing the amplitude at each data point. So to find the start of the chirp the implementation iterates over the array and looks for a data point with a value above 10000. Once found, it shows the chirp beginning. Then is returned the index of the found value above 10000. This is the offset we apply for each chirp in the array to find its beginning. Additionally, 50 index positions are returned to get more exactly the start of the chirp. Figures 10 and 11 present a wave plot with no offset applied and a wave plot with the applied offset for this chirp.

When the chirp offset is found, a spectrogram is created from each 4410 data points interval. The first 132 points are removed as a safeguard. The generated spectrogram shows all frequencies between 0 and 22050 kHz in the span of 100ms. Only the part between 19.5kHZ and 20.5kHZ is taken. Then this part of the data is stored on a non-relational database. This process is applied to all 998 out of 1004 intervals of data except for the first four and last two which are removed as a safeguard. To find the start of the chirp and apply the offset value, only the first chirp is used after the first 4 have been removed.

[2]https://developer.android.com/reference/android/media/AudioTrack
[3]https://developer.android.com/reference/android/media/AudioRecord

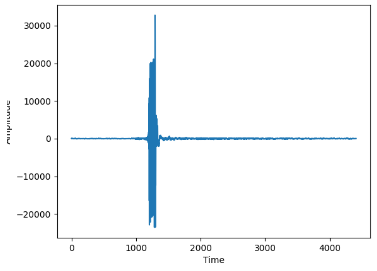

Figure 10: Wave plot with no offset

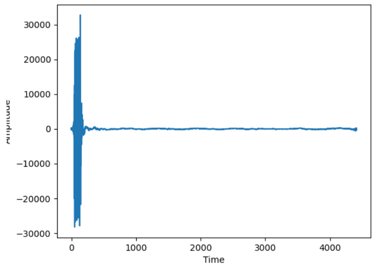

Figure 11: Wave plot with offset

For the training part again an HTTP request is sent from the front end with the name of the model, model type, the hyperparameters setup and the building name. Then a model is initialized with this setup and the data for the specific building. When the training of the algorithm is ready, its weights are stored on the backend for further use.

For the test part, an HTTP request is sent from the front-end with the name of the model and an array with 13230 data points for classification. The first 4410 data points are omitted and to the rest are applied the pre-processing steps from the first 2 paragraphs with the difference that the data is not saved in the database, but it is classified by the specified algorithm which returns the localized label which is returned as a response on the front end. Figure 12 presents the back-end design.

Front-end only design

With this design approach, the data is pre-processed and saved on the in-memory database of the device. Unfortunately, it is not possible to train TensorFlow models on Android applications, so a separate Python environment is used, the models are trained with the collected data and compressed. After the compression, they can be used for predictions on the front-end application. The architecture of this design approach can be seen in figure 13.

5.4 Model training

The sample data is divided into 70% training samples and 30% validation samples for each model. The training process of all algorithms is the same. It consists of 100 epochs, in each epoch a mini-batch of randomly selected 32 training samples is used. The only difference is the training strategy

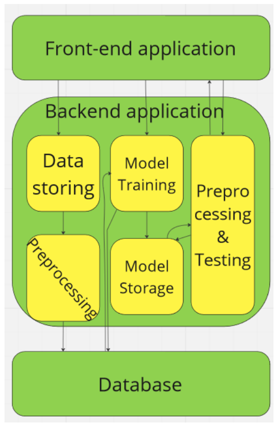

Figure 12: Backend design

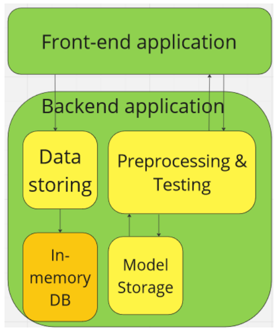

Figure 13: Front-end only part design

each model uses. Each of the neural networks' output layer is a dense layer with K units, the number of classes in the training set, and a softmax activation function. Also, just before the output layer, there is a dropout regulation layer with a 0.4 rate. It aims to avoid overfitting and improve the models' performance.

5.5 Hyperparameter Settings

Convolutional neural network layers

	conv layer filters	**dense layers**
C1	16 x 32	1024 x K units
C2	32 x 32	1024 x K units
C3	128 x 64 x 64	1024 x K units
C4	32 x 64 x 128	512 x K units
C5	256 x 128	1024 x K units
C6	512 x 128	512 x K units

Table 1: Number of filters in each convolutional layer and units in dense layer

Each model has 2 max-pooling layers. Each convolutional layer applies an 'elementwise' operation with 4x4 filter, 0 paddings, a stride of 1 and a 'relu' activation function, as it preserves the dimensions of its input data. On the other side, the max-pooling layer uses 2x2 filter and a stride of two.

This layer controls the over-fitting and improves the robustness to small distortions in the input image. The output of the first convolutional layer is 5x32, then the max-pooling layer downsized the data dimensions to 2x16. The second convolutional layer keeps the same dimensions, 2x16, while the last max-pooling layer downsized the data to a 1x8 image. Then the following applied convolutional layers preserve the same dimensions of 1x8.

The input data for the first dense layer is the number of filters in the last convolutional layer multiplied by 1x8 flattened. For the hidden dense layers 'relu' activation function is used.

Dense neural network layers

	dense layers
D1	512 x 256 x 256 x K units
D2	256 x 512 x 256 x K units
D3	1024 x 512 x K units
D4	256 x 512 x 1024 x K units
D5	2048 x 512 x 256 x K units
D6	512 x 128 x 2048 x 512 x K units

Table 2: Number of units in dense layer

For the dense layers 'relu' activation function is used. The input data for the first layer is a flattened 1D array.

Reurrent neural network layers

	LSTM layers	**dense layers**
R1	64 x 64	256 x K units
R2	128	128 x K units
R3	64	1024 x K units
R4	32 x 64	1024 x K units
R5	64 x 128	1024 x K units
R6	256 x 128	256 x K units

Table 3: Number of units in LSTM and dense layer

Each of the LSTM layers is Bidirectional, which means the input data goes in both directions. The input of the first layer is 5x32, then the output of each LSTM layer size is 5x twice the number of units.

Each RNN model has a dense layer after the LSTM ones with 'relu' activation function. The input dimension of the first dense layer is 5x twice the number of units of the last LSTM layer flattened.

6 Results

In the following section are presented results from the experiments and is made a comparison between the metrics specified in section 4 - accuracy, time to train, parameters, prediction times, FLOPS and accuracy on three data sets for the models on the back end and the compressed models on the front end.

6.1 Evaluation of the training of the models

The results from tables 4, 5 and 6 show that the CNN and DNN models are more suitable for the classification of such data. All models have above 93% accuracy on their validation data sets, except for the RNN models which have accuracy between 72% and 90%.

M	**Acc val**	**Params**	**Time**	**FLOPS**
C1	94%	280k	4.17 m	37M
C2	94%	290k	4.72 m	59M
C3	94%	730k	11.63 m	360M
C4	94%	700k	7.27 m	173M
C5	94%	1500k	36.28 m	1186M
C6	93%	1500k	57 m	2270M

Table 4: This table presents the training results for each CNN model - M - model, Acc vall - accuracy on validation data, Params - Parameters, T - Training time in minutes, and Floating point operations.

From the tables presenting the results from the training is observed a connection between the parameters, time to train and FLOPS. With the increase of the parameters, the rest of the values also increase. However, the accuracy of the validation data is not connected with the parameter increase.

M	**Acc val**	**Params**	**Time**	**FLOPS**
D1	94%	280k	2.23 m	19M
D2	94%	290k	2.32 m	19.5K
D3	95%	700k	5 m	44M
D4	94.3%	700k	5.3 m	45M
D5	94%	1500k	8.85 m	96M
D6	94.5%	1500k	9 m	93M

Table 5: This table presents the training results for each CNN model - M - model, Acc vall - accuracy on validation data, Params - Parameters, T - Training time in minutes, and Floating point operations.

The flops metric shows that the CNN and DNN models take more FLOPS depending on the number of parameters in comparison to the RNN models which take around 0.8K and 1.5K for training.

M	**Acc val**	**Params**	**ime**	**FLOPS**
R1	74%	315k	4.75 m	0.8K
R2	90%	330k	3.87 m	1.5K
R3	90%	711k	4.82 m	0.8K
R4	84%	740k	5.7 m	0.8K
R5	82%	1600k	9.23 m	1.5K
R6	72%	1500k	20.57 m	1.5K

Table 6: This table presents the training results for each CNN model - M - model, Acc vall - accuracy on validation data, Params - Parameters, T - Training time in minutes, and Floating point operations.

6.2 Accuracy on different data sets

The models have been evaluated on three different data sets - DS1, DS2 and DS3. DS1 has been used for training the models. The three best and worst performing models are presented in tables 7 and 8 with their inference run times. The results for all models are in Appendix A.

For the models in the back end, the results in table 7 show that models C5, D2 and R2 have the highest accuracy on the three data sets. C5 and D2 have higher accuracy performance on DS1 - 98% - than R2 which has an accuracy of 90%. For data sets 2 and 3 the three best-performing models have similar results of 86% - 89%. In table 8 for the worst performing models is shown that the accuracy performance of the C1 and D6 is not much lower than the best performing C5 and D2 models in table 7. However, this is not the case for the RNN model R6 as the average accuracy on the three data sets for the worst performing RNN model is 18% below the best performing one R2.

From the confusion matrices in figures 14, 15, 16, 17, 18 and 19 can be observed how the accuracy is changing for the best and worst performing models on data set 2. For the CNN models, there is an increase in the false positive for 19D being incorrectly classified as 19A and 19E classified as 19B. For the DNN models, there is a slight increase in the misclassifications of 19A with 19D, 19B with 19C and 19C with 19B. The biggest increase in misclassified locations is for the RNN models. The confusion matrix shows an increase of false positive classifications of 19D being classified as 19A and false positives of 19E being classified as 19B and 19E being classified as 19C. The reason for that can be noise in the data. These plots show that the CNN and DNN models are more robust to data noise. The results for the matrices on DS3 for the same models are similar to the results for DS2. The results for DS3 can be seen in Appendix B.

For the models on the front end, the best performing CNN model - C5 is able to reach accuracies of 75%, 64% and 68% on the three data sets which are on average 9% higher than the results of the worst performing CNN model C1 on the same data sets. For the RNN and the DNN models, the best and worst performing models have the same accuracies on the three data sets - 23% for the DNN models and 36% for the RNN models, except for R6 on DS3 which can reach an accuracy of 46%. The reason for this difference in the performance between the models on the back end and the compressed one is that the compressed ones' values of the weights are not as precise as those on the back end. The reason for that is during compression these weights values are shortened to make the model size smaller because it is assumed that smartphone devices have smaller memory storage compared to the servers. The compressed models are around 3 fold smaller than the uncompressed ones.

6.3 Prediction times

From the conducted experiments, it takes on average 70ms for classification on the back end and roughly 30ms more for the request to reach the back end and then return the result, so the whole round trip is on average 100ms as for the RNN models take a bit more to make a prediction. On the other hand, the predictions with the compressed models take no more than 1 ms on average for all models, however, the accuracy score is low. The results of these experiments are presented on average in tables 7 and 8.

The results from these experiments show that the best-performing compressed models are CNN. However, their performance is nowhere near the performance of the models on the back end which are able to reach an accuracy of 90%. The models there are also showing minimal overfitting of 8% better accuracy on the data set the models have been trained in comparison to the data sets they are tested. Based on the results the implementation of an additional back end has higher accuracy despite taking more time for interference run which would be hardly noticed.

Model	DS1	DS2	DS3	Time
C5	98% 75%	89% 64%	88% 68%	60ms 1ms
D2	98% 23%	89% 22%	89% 22%	60ms 1ms
R2	90% 37%	88% 38%	86% 38%	80ms 1ms

Table 7: The best three performing CNN, DNN, RNN models - the first number refers to the model on the backend and the second one to the compressed model.

Model	DS1	DS2	DS3	Time
C1	97% 64%	88% 58%	82% 59%	60ms 1ms
D6	98% 23 %	89% 21%	86% 22%	70ms 1ms
R6	73% 36%	67% 37%	64% 46%	97ms 1ms

Table 8: The worst three performing CNN, DNN, RNN models - the first number refers to the model on the backend and the second one to the compressed model.

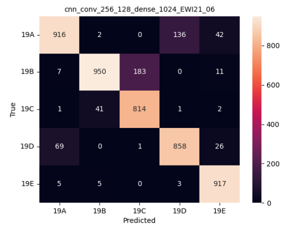

Figure 14: Confusion matrix for C5 on DS2

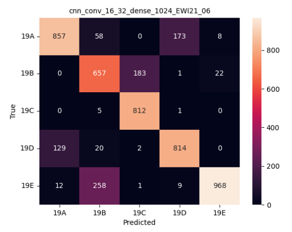

Figure 15: Confusion matrix for C1 on DS2

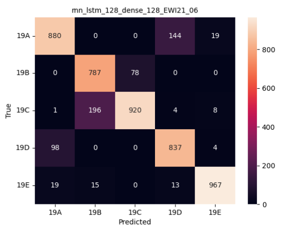

Figure 16: Confusion matrix for R2 on DS2

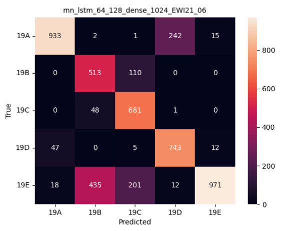

Figure 17: Confusion matrix for R6 on DS2

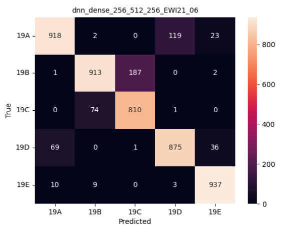

Figure 18: Confusion matrix for D2 on DS2

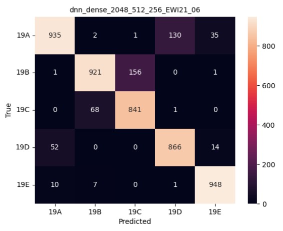

Figure 19: Confusion matrix for D6 on DS2

7 Responsible Research

Privacy concerns can appear with the collected audio data. This data consists of the frequencies collected at some period of time. Then this data can be easily converted back to human audible sound. In order to tackle this problem, the research application only works with frequencies in the inaudible range and immediately after the data is collected, it is pre-processed - converted to spectrograms that have a frequency range between 19.5kHZ and 20.5kHZ - this is the inaudible range for the people. Additionally, the Fast Fourier Transform (FFT) algorithm can be applied to the audio data to remove the human audible frequencies from it.

The used frequency for the experiments is inaudible for the people but it can be annoying for the animals. During the data collection, there were no animals in the building or on the floors of it.

8 Discussion

The conducted experiments show that simple DNN and CNN models are more suitable for classifying acoustic data than RNN. The best-performing models are the DNN models, they have the lowest time for training and FLOPS and at the same time the highest accuracy and data robustness. Overall the DNN and CNN models show better accuracy and robustness to different data sets compared to the RNN. The reason for that can be found in the applications for RNN models, they are more suitable for natural language processing [15] while the DNN and the CNN models are more suitable for image classification and pattern recognition [16].

The implementation of the two architectural designs presents that the implementation of a system with a front end and a back end is better because the accuracies of the models on the back end are way higher than the compressed one on the front end despite the time taking for predictions on the former is a bit bigger than this on the latter but the 100ms delay is hardly noticed by the users of the application for who will be far more important the accuracy in comparison to the insignificant delay. The results show better performance for the models on the back end compared to the compressed ones. The reason for this can be the compression of the models themselves. During the compression, the model is lightweight, to make it works on Android devices, which decreases its performance.

9 Conclusions and Future Work

This paper gives answers to the questions stated at the beginning of it - what deep models are suitable to use for indoor location recognition, deployment and evaluations of the system and inference run in real-time. The results show that the DNN and the CNN models perform better than the RNN models, as the DNN models have the shortest testing time and FLOPS. Additionally, the DNN and the CNN models show better robustness on different data sets, as the models on the back end significantly outperform the compressed models and in this order of statements shows it is better to implement a system with a front end and a back end with models trained and stored on the back end, despite the prediction time being a bit higher but 100ms of delay are hardly noticeable.

In the future, experiments can be conducted on which model is most robust on data collected by devices different from the one used for training. Additionally, another unexplored topic is what is the models' performance when there are emitted more than one frequency of 20kHZ at the same time and what is the influence on the model

References

[1] Q. Song, C. Gu, and R. Tan, "Deep room recognition using inaudible echos," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, pp. 1–28, 2018.

[2] E. Goubault, “Static analyses of the precision of floating-point operations,” in *Static Analysis: 8th International Symposium, SAS 2001 Paris, France, July 16–18, 2001 Proceedings*, pp. 234–259, Springer, 2001.

[3] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, “Indoor localization without infrastructure using the acoustic background spectrum,” in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pp. 155–168, 2011.

[4] W. Wang, F. Seraj, N. Meratnia, and P. J. Havinga, “Localization and classification of overlapping sound events based on spectrogram-keypoint using acoustic-sensor-network data,” in *2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, pp. 49–55, IEEE, 2019.

[5] D. Guo, W. Luo, C. Gu, Y. Wu, Q. Song, Z. Yan, and R. Tan, “Demo abstract: Infrastructure-free smartphone indoor localization using room acoustic responses,” 2021.

[6] J. Dennis, H. D. Tran, and H. Li, “Spectrogram image feature for sound event classification in mismatched conditions,” *IEEE signal processing letters*, vol. 18, no. 2, pp. 130–133, 2010.

[7] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] F. Amherd and E. Rodriguez, “Heatmap-based object detection and tracking with a fully convolutional neural network,” *arXiv preprint arXiv:2101.03541*, 2021.

[9] G. Chen, “A gentle tutorial of recurrent neural network with error backpropagation,” *arXiv preprint arXiv:1610.02583*, 2016.

[10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks*, vol. 61, pp. 85–117, 2015.

[12] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.

[13] M. Fan, A. T. Adams, and K. N. Truong, “Public restroom detection on mobile phone via active probing,” in *Proceedings of the 2014 ACM International Symposium on Wearable Computers*, pp. 27–34, 2014.

[14] K. Kunze and P. Lukowicz, “Symbolic object localization through active sampling of acceleration and sound signatures: (nominated for the best paper award),” in *UbiComp 2007: Ubiquitous Computing: 9th International Conference, UbiComp 2007, Innsbruck, Austria, September 16-19, 2007. Proceedings 9*, pp. 163–180, Springer, 2007.

[15] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative study of cnn and rnn for natural language processing,” *arXiv preprint arXiv:1702.01923*, 2017.

[16] N. Jmour, S. Zayen, and A. Abdelkrim, “Convolutional neural networks for image classification,” in *2018 international conference on advanced systems and electric technologies (IC_ASET)*, pp. 397–402, IEEE, 2018.

Appendix

A Results

The tables in this section present the models' results on data sets DS1, 2 and 3 for the server and compressed models. Additionally is added a column for the interference run times for making one prediction. The best performing models are C5, D2 and R5 with an accuracy on average of 87.5% for data sets 2 and 3. On the other side are models C1, D6 and R6 with the lowest accuracy. The difference in the performance for C5, D2 and C1, D6 is not so big as between R2 and R6 as the difference there is on average 18%. The same models are also the best and worst performing compressed models on the DS2 and DS3 except for R6 which has the highest accuracy on DS3 out of RNN models. The prediction time for the DNN and CNN models is similar - 60 ms for the models on the server and 1 ms for the compressed models. The RNN models on the server take around 30ms more for classification while their compressed forms no more than 1 ms.

Model	**DS1**	**DS2**	**DS3**	**Time**
C1	97% 64%	88% 58%	82% 59%	60ms 1ms
C2	97% 68%	89% 65%	85% 61%	60ms 1ms
C3	98% 72%	89% 63%	86% 64%	60ms 1ms
C4	98% 76%	89% 66%	84% 64%	60ms 1ms
C5	98% 75%	89% 64%	88% 68%	60ms 1ms
C6	97% 70%	88% 61%	86% 66%	60ms 1ms

Table 9: This table presents the accuracy results on 3 different data sets for each CNN model - the first number refers to the model on the backend and the second one to the compressed model.

Model	**DS1**	**DS2**	**DS3**	**Time**
D1	98% 23%	90% 21%	85% 22.6%	60ms 1ms
D2	98% 23%	89% 22%	89% 22%	60ms 1ms
D3	98% 27%	91% 24%	86% 26%	60ms 1ms
D4	98% 26%	89% 24%	86% 26%	80ms 1ms
D5	98% 25%	90% 22%	86% 24%	60ms 1ms
D6	98% 23 %	89% 21%	86% 22%	70ms 1ms

Table 10: This table presents the accuracy results on 3 different data sets for each DNN model - the first number refers to the model on the backend and the second one to the compressed model.

Model	**DS1**	**DS2**	**DS3**	**Time**
R1	75% 38%	76% 39%	74% 42%	90ms 1ms
R2	90% 37%	88% 38%	86% 38%	80ms 1ms
R3	91% 35%	85% 36%	82% 40%	75ms 1ms
R4	84% 36%	83% 38%	77% 44%	92ms 1ms
R5	82% 37%	77% 39%	72% 41%	95ms 1ms
R6	73% 36%	67% 37%	64% 46%	97ms 1ms

Table 11: This table presents the accuracy results on 3 different data sets for each RNN model - the first number refers to the model on the backend and the second one to the compressed model.

B Confission Matrices

In this section are presented confusion matrices of the best and the worst performing models on data sets DS2 and 3. The confusion matrices for the same models have almost similar results on the different data sets for correct and incorrect classification.

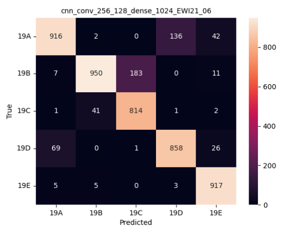

Figure 20: Confusion matrix for C5 on DS2

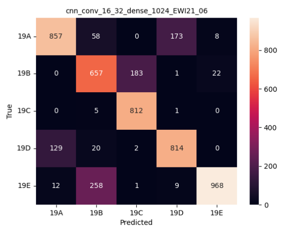

Figure 21: Confusion matrix for C1 on DS2

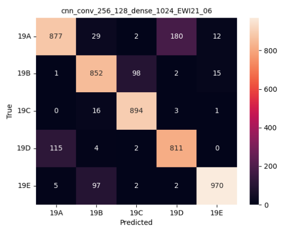

Figure 22: Confusion matrix for C5 on DS3

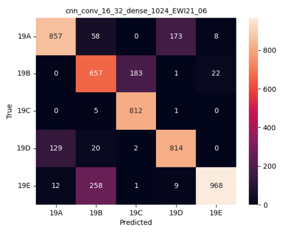

Figure 23: Confusion matrix for C1 on DS3

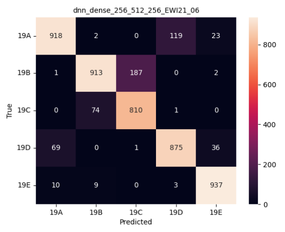

Figure 24: Confusion matrix for D2 on DS2

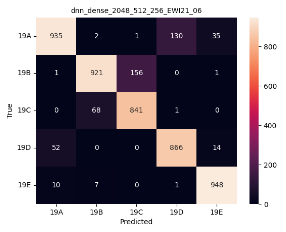

Figure 25: Confusion matrix for D6 on DS2

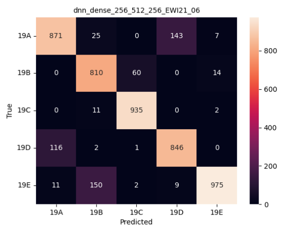

Figure 26: Confusion matrix for D2 on DS3

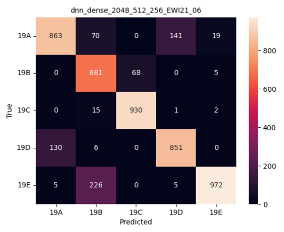

Figure 27: Confusion matrix for D6 on DS3

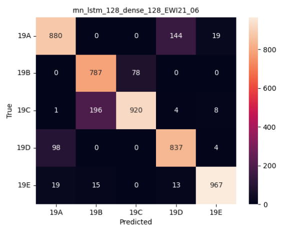

Figure 28: Confusion matrix for R2 on DS2

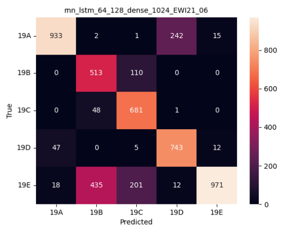

Figure 29: Confusion matrix for R6 on DS2

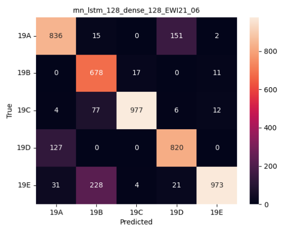

Figure 30: Confusion matrix for R2 on DS3

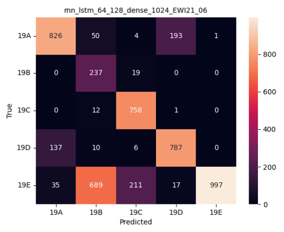

Figure 31: Confusion matrix for R6 on DS3