2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 29 Nov. – 2 Dec. 2021, Lloret de Mar, Spain

Completely Automated CNN Architecture Design Based on VGG Blocks for Fingerprinting Localisation

Shreya Sinha
Pervasive Systems Group
Faculty of EEMCS
University of Twente
Enschede, The Netherlands
s.sinha-2@student.utwente.nl

Duc V. Le
Pervasive Systems Group
Faculty of EEMCS
University of Twente
Enschede, The Netherlands
v.d.le@utwente.nl

***Abstract*—WiFi fingerprinting using Convolutional Neural Networks (CNN) is one of the most promising techniques for indoor localisation due to the extraordinary performance of CNN in image classification. However, the performance of CNN is architecture dependant, and thus an architecture that works well in one case may not work in another, especially for the WiFi-based localisation problems. Most of the solutions use an existing hand-crafted architecture or a semi-automated CNN design for fingerprinting, which requires significant CNN expertise and time. Therefore, a satisfactory solution may not be guaranteed as it is challenging to design numerous possible architectures. In this work, we address this challenge by developing a framework that completely automates the CNN architecture design process. Our automated architectures based on VGG blocks have shown superior performance compared to standard architectures such as VGG-16. We further explore three different heuristics for automation: Bayesian optimisation, Hyperband, and Random Search and demonstrate their importance towards the automated CNN architecture development for WiFi fingerprinting. Experiments are conducted on real-world datasets and, a comparative study between our automated architecture and other models is presented. This work would, therefore, facilitate the CNN design for indoor localisation.**

***Index Terms*—WiFi Fingerprinting, Automated Convolutional Neural Network, Bayesian optimisation, Hyperband, Random Search**

I. Introduction

Indoor localisation has always been important in various sectors such as healthcare, security, or location-based services [1]. Although the advent of Global Positioning System (GPS) has made outdoor localisation a part of our everyday lives, locating a person or device indoors remains an open challenge as GPS does not usually work indoors.

This work is supported by the InSecTT project, https://www.insectt.eu/, funded by the ECSEL Joint Undertaking (JU) under grant agreement No 876038. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey. The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

978-1-6654-0402-0/21/$31.00 ©2021 IEEE

Among many of the solutions studied, WiFi-based is one of the most attractive solutions for indoor localisation because: it is easy to deploy and maintain, wireless LANs (WLANs) are ubiquitous, and smartphones are nowadays WiFi-enabled [2]. Among the various techniques, fingerprinting is a popular one to identify the location of a user by characterising his radio signal environment [3]. It is completed in two phases: an offline phase and an online phase. In the offline or training phase, a site survey is performed and Received Signal Strength Indicator (RSSI) is calculated from all detected WiFi Access Points (APs) at each known reference point. A radio map is built consisting of vectors of RSSI values at their known locations. This is followed by the online phase, where the user collects RSSI measurements at their location. These measurements are then matched with the radio map to determine the user's position with the aid of position estimation algorithms.

The potential of Convolutional Neural Networks (CNN) to effectively model a radio map due to its ability to learn complex relationships between RSSI and coordinates was demonstrated in [4]. However, the performance of a CNN model depends heavily on its architecture, which is a possible combination of the number of layers, nodes, and associated parameters. Although previous works could enumerate each potential architecture and then train and validate it to choose the best among them, a satisfactory solution may not be optimal because it is difficult to counteract numerous possible combinations. This is because the computational complexity also increases greatly during runtime as the list of possible hyperparameters that need to be tuned grows.

Moreover, designing the architecture requires expertise in both its working and an understanding of the problem [5]. In practical scenarios such as WiFi-based fingerprint localisation, this is often more challenging. Even the popular hand-crafted architectures, including VGGNet [6], whose performance has been proven on many data types and applications [7], may still not guarantee good accuracy for fingerprint localisation.

To this end, the development of **completely automatic** CNN architectures for fingerprint localisation with promising

performance, constrained by limited computational resources, is still in its infancy. In this work, an automated framework based on the VGG block is developed, and the heuristic computational paradigms, such as Bayesian optimisation [8], Hyperband [9] and Random Search [10] are implemented for optimisation. To evaluate the framework, we use publicly available datasets such as Indoor Location, and Navigation competition series on Kaggle [11], as well as the UJIIndoorLoc dataset [12]. The Kaggle dataset embodies WiFi sensor data collected via crowdsourcing from a smartphone and the UJIIndoorLoc dataset was created in 2013 with the help of more than 20 different users collecting WiFi samples. The results show that CNN models automatically designed by our proposed framework, especially using the Bayesian optimisation algorithm, can achieve high localisation accuracy and outperform the standard VGG-16 [6]. The results also show the significant difference in localisation performance between the worst and the best models. A slight modification in the CNN architecture would lead to a massive change in terms of localisation accuracy. In other words, it is recommended to use our fully automated CNN design for indoor localisation.

The main novel contributions of this work may be summarised as follows.

- A framework is designed to automatically deliver a promising CNN architecture for WiFi fingerprint localisation. The proposed framework assumes no prior knowledge of the user about the CNN design process, investigated dataset, and genetic algorithms.
- The CNN models, whose architecture is designed by the proposed framework, are extensively evaluated with several publicly available datasets.

The rest of this paper is organised as follows. Section II discusses the related work regarding CNN localisation and architecture optimisation. Section III provides background information. The prerequisites for WiFi localisation are discussed in the following Section IV. The experimental setup is discussed in Section V, followed by results in Section VI. The conclusion is discussed in Section VII.

II. Related Work

Numerous studies have addressed fingerprinting-based WiFi localisation. Initially, machine learning algorithms (ML) such as K-nearest neighbors (k-NN) [13], Support vector machine (SVM), Random forests and, Decision trees [14] were explored. In [15], nine teams undertook the "Ubiqum Challenge", nine teams investigated different combinations of machine learning algorithms to improve position accuracy. Their main goal was to classify the floor a user was situated on and perform regression techniques to estimate longitude and latitude. In [16], a comparative study is conducted between six different types of ML algorithms to measure their performance over a common dataset.

Since Deep Learning can solve a complex problem better than standard machine learning, Deep-Neural-Network (DNN)-based methods have also been studied [17]. The main drawback of these methods is the poor accuracy they achieve when the dataset is insufficient [18]. Reference [18] analyses WiFi localisation using CNN, where the authors convert RSSI vectors into 2-D images for multi-floor classification. Similarly, CNN-LOC [19] generates input images using RSSI from WiFi signals, which are then used to train a CNN model to classify the user at one of the reference points. Tianqi Qu et al. presented a similar approach using a modified CNN model and an analysis of its performance compared to a regular CNN architecture in [20].

Although the proposed CNN models could improve location estimation, they are technically challenging and time-consuming. Moreover, it requires expertise in both CNN theory and understanding fingerprint localisation [5]. Moreover, an exact CNN architecture such as VGG-16 [6] is unable or ineffective to solve some problems. As a result, architecture optimisation algorithms based on heuristic paradigms, such as Random Search [10], Bayesian optimisation [8], and Hyperband [9], have been proposed. However, to our knowledge, none of the previous works has proposed a fully automated CNN architectural design for fingerprint localisation.

To this end, in this paper, we propose a framework that can automate the CNN architectural design for fingerprinting localisation, which requires minimal effort and CNN knowledge.

III. Background

In this section, we highlight the VGG-16 default CNN architecture and alternatives for fine-tuning the default architecture.

A. VGG-16

VGG-16 was first presented in [6]. The authors investigated the impact of the depth of the CNN on its accuracy in large-scale image recognition. They passed the input image to a stack of convolutional layers (Conv2D), consisting of filters responsible for extracting features from the previous layer. Spatial pooling was performed by MaxPool layers, placed after some, but not all, Conv2D layers. At the end of the architecture, a stack of fully linked layers with different dimensions is placed, the value of which depends on the problem. A complete VGG-16 architecture is shown in Fig. 1.

Fig. 1: A complete VGG-16 architecture

B. Heuristic Algorithms

While CNNs are powerful tools, they still require further tuning of their hyperparameters to obtain the best possible architecture. Hyperparameter tuning is the process of finding the right combination of hyperparameters, to maximize the performance of the CNN. It can be computationally intensive and time-consuming [21] and becomes even more complicated as the network gets deeper. An example of a popular tuning algorithm is Grid Search [22], which tries all possible combinations of hyperparameters. However, this

method can become difficult for larger networks and models. As an alternative, several other algorithms such as Bayesian optimisation, Hyperband and, Random Search have recently come into play.

The Bayesian optimisation [8] approach uses Bayes theorem to guide the search for detecting the minimum or maximum of an objective function. It consists of two main components: The objective function, which is modeled using a probabilistic Bayesian probabilistic model, and an acquisition function, which determines the samples to search next [8]. The probabilistic model of the objective function, known as the surrogate model, is a model trained on (hyperparameter, true objective function value) pairs. After certain trials, the next choice of hyperparameter is kept from the pair where the acquisition function was maximized. These corresponding hyperparameters are further used to test the surrogate model and update the true objective function. This process of updating the surrogate model continues until the maximum time, maximum iteration is reached.

Random Search [10] bases its optimisation strategy on a stochastic process. Therefore, unlike Bayesian optimisation, it does not consider past evaluations. It is often compared to Grid Search, but has been shown to outperform it [10]. Due to its random nature, the performance of this technique often depends on the number of trials performed.

Hyperband [9] is based on the principle of successive halving, which works with N different configurations and a budget ß. In each iteration, successive halving keeps the best half of the configurations and discards the other half. The process continues until a configuration is obtained. One limitation of this process is deciding the number of samples at the beginning. Hyperband solves this problem of 'N v/s ß' by considering several possible values of N for a fixed ß, essentially performing a grid search over practical values of N [9].

IV. Completely Automated CNN design for Fingerprinting localisation

In this section, we give an overview of our proposed framework. The framework helps to understand the automated end-to-end process of developing the CNN architecture for WiFi localisation. For a deeper understanding, we present the readers with the process of restructuring the data to create the input images for the CNN model, our proposed CNN building blocks, and finally the optimisation using the hyperparameter tuning technique.

A. Overview

The framework for developing an automated CNN architecture is shown in Fig. 2. The initial data collection phase is performed by recording WiFi scans using a smartphone. The recorded data is preprocessed to create a radio map, that serves as an input fingerprint database for WiFi localisation. The third stage of our framework involves developing a workflow for automating the CNN design process by finding the best selection of hyperparameters. This stage consists of two components, model build and model evaluation. The model build function selects the number of VGG blocks to design a new model each time it is called. All VGG blocks are based on the specified CNN model structure. This new model is passed to the heuristic algorithm tuner to compute the specified objective. The algorithm measures the performance of the model against a validation dataset. This process repeats until the specified number of trials is reached or, in the case of Hyperband, a single model remains. Each time during the model creation and evaluation, a new model and set of hyperparameters are selected. The model with the best performance for the specified target is retrieved and used as the final model for location prediction.

B. Data Preprocessing

In case of Kaggle dataset [11], the location information for each WiFi scan in the form of 'x' and 'y' coordinates is computed. A radio map is thus fabricated combining all the required information. Suppose that during each scan, the user receives RSSI values from neighbouring N APs, namely, the data $\mathbf{w} = (r_1, r_2...., r_N)$ is a 1-D vector of length N, where r_i denotes the RSSI value obtained from the AP i. Each data has labels x $\in \{1,..,M\}$ and y $\in \{1,..,M\}$. Consequently, for each data value w_j, we have the labels x_j, y_j. Note, that for training, we know both the N RSSI values and the corresponding label as (w_1,x_1,y_1), but for prediction, we know the RSSI values for w_2 and want to estimate the corresponding labels. We insert a value of -100 dBm for the APs if the corresponding AP was not detected at a particular location. Also, if a label occurred multiple times, we calculated the average RSSI values for each AP.

Since CNN requires a 2-D array, the number of APs in the 1-D data should be a perfect square. We thus add some dummy APs with a value of -100 dBm for all labels.

Similarly, UJIIndoorLoc dataset [12] also consists of vectors of RSSI values with their corresponding latitude (x) and longitude (y) information. We changed the value of 100 dBm, which indicates that an AP was not detected at that particular location, to -105 dBm. This was done to make it consistent with rest of the RSSI readings. We further scale both datasets by applying a standard scalar.

C. Model Backbone

In our study, we do not implement the entire original VGG-16 architecture, but instead, try to find the optimal number of VGG blocks needed. In VGG-16 [6], a VGG block consists up of two Convolutional layers and one MaxPool layer. In our proposed model backbone, we also add Batch Normalization [23] after each convolutional layer to standardize the inputs of each layer. This reduces the computation time for training a model. The backbone of our framework also contains two fully connected layers and an output layer, similar to VGG-16. However, an additional dropout layer is added between two fully connected dense layers to prevent overfitting. Overfitting is a concept where the model is trained so well that it is no longer able to generalise from training data to unseen data.

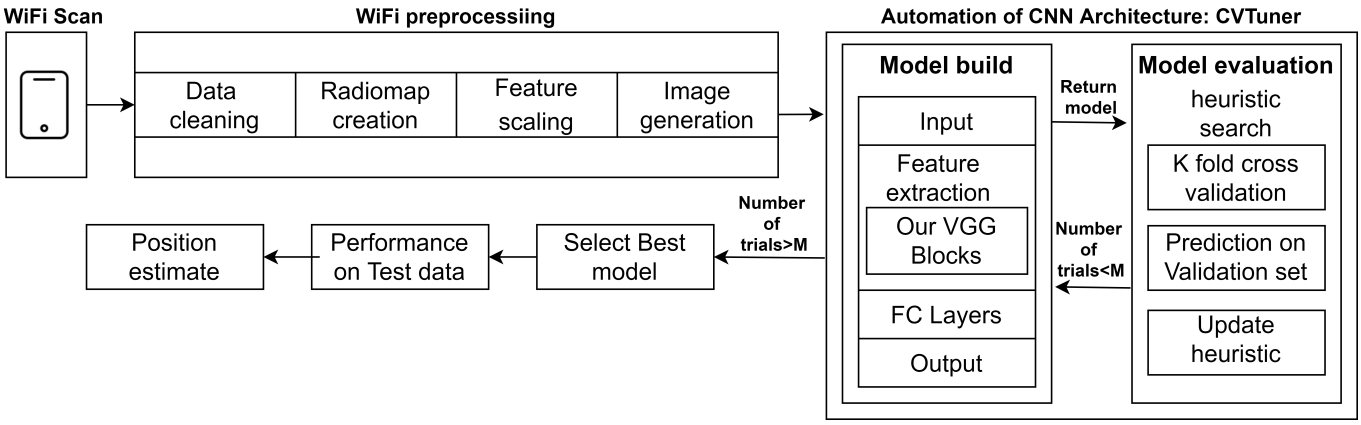


Fig. 2: Overview of the WiFi fingerprinting localisation with hypertuned CNN architecture

D. Model Optimisation

For architecture optimisation, a custom CVTuner is designed. The choice of heuristic optimisation algorithm, the objective to be minimized, the max_trials i.e the maximum number of different architectures to try out, and the model building function are passed as parameters to instantiate the CVTuner. The model-building function, as presented in Algorithm 1, is a user-defined function that takes an argument *hp* from which one can sample hyperparameters within a specified range. The model backbone is present in this function. The details of the body of the CVTuner is further summarized in Algorithm 2. Since we are dealing with a regression problem, and the goal of the architecture is to predict both x and y labels, we use mean square error (MSE) as the loss function.

Algorithm 1 Model Build function to tune the selected hyperparameters in order to build the CNN architecture

procedure BUILD-MODEL(hp) ▷ hp = hyperparameter
  Model = Sequential Model
  Model ← Add Layer (Input, Shape=(h, w, c))
  for $i = [1, n]$ **do** ▷ n = number of maximum VGG units to tune
    Model ← Add Layer (Convolutional 2D,
      Number of filters = hp.Choose(64 / 128 / 256 / 512))
    Model ← Add Layer (Batch Normalization)
    Model ← Add Layer (Convolutional 2D,
      Number of filters = hp.Choose(64 / 128 / 256 / 512))
    Model ← Add Layer (Batch Normalization)
    Model ← Add Layer (MaxPool2D)
  Model ← Add Layer (Flatten)
  Model ← Add Layer (Dense,
      Units = hp.Int(min = 64, max = 512))
  Model ← Add Layer (Dropout,
      Rate = hp.Float(min 0, max 0.5))
  Model ← Add Layer (Dense,
      Units=2, Activation='linear')
  Learning rate = hp.choose(0.01 / 0.001 / 0.0001)
  Model ← Compile(Loss function = Mean Squared Error
      Optimizer = Adam (learning rate))
return Model

E. Hyperparameters

In the following, we discuss some of the hyperparameters that we define in the CVTuner for optimisation.

Algorithm 2 Algorithm to perform K Fold cross validation for each of model returned by the Model Build function, and calculate their Validation and Test loss.

procedure RUN TRIAL($self, trial, x, y, args, kwargs$)
  Batch size ← hp.choose(8 / 16 / 24 / 32)
  Epochs ← A constant value
  CV Split← Split the Training data as train and val using K fold
  Value losses = []
  for train indices, value indices in CV **do**
    X-train ← X[train index]
    X-val ← X[val index]
    Y-train ← Y[train index]
    Y-val ← Y[val index]
    Train Model ← Train, Val, Epochs and Batch size
    Predict ← Label predictions of validation set X-val
    Predict ← Inverse-transform(Predict)
    Y-val ← Inverse-transform(Y-val)
    ValueLosses ← Average (MeanSquareErrors
      (Predict, Y-val))
  TestPredictions ← Label predictions of test set X-test
  TestPredictions ← Inverse-transform(TestPredictions)
  MSETest ← MeanSquareErrors(TestPredictions, Y-test)
  Save the model

- Batch Size: The performance of a CNN model is affected by its batch size. A Larger batch size speeds up the computations, but too large a batch size may result in a poorly generalised model.
- Number of VGG Blocks: We also leave it up to the tuner to determine the number of VGG blocks to tune. Too many blocks can lead to overly complex architecture. Fewer layers may yield inaccurate results.
- Filters in the Conv2D layer: The filter extracts the distinct set of features from the input. Since an input may have different features, we need *n* multiple filters to extract the important features from the input.
- Dense Layer Units: The dense layer in a CNN is a fully connected layer, and each neuron receives input from all neurons in the previous layer. The units of the dense layer define the form of the input that is passed to the subsequent layer.
- Learning Rate: The learning rate is also an important hyperparameter that controls how much a model must

be changed in response to the estimated error obtained each time the model's weights are updated. If the learning rate of an optimiser is set too low, training becomes a tedious process. If it is too high, it can cause the model to converge to a suboptimal solution.

- Rate of dropout Layer: Another method to prevent a model from overfitting is to add a dropout layer. We add a dropout layer between two dense ones in our model. The rate of the dropout layer determines the percentage of neurons that get deactivated in that particular layer, thus affecting the performance of the architecture.

V. Experimental Setup

In this section, we implement the previously described framework to solve the indoor WiFi localisation challenge. We discuss the dataset overview, and the techniques used to evaluate the dataset for further analysis.

A. Dataset Description

The Kaggle data is provided by an indoor positioning technology company XYZ10 in collaboration with Microsoft Research. It consists of multiple traces of sensor data collected from over 200 buildings. However, for our study, we consider the data collected over two floors: F1 and F2, from the same building. We run the regression over both floors separately. No information is provided about the model of the smartphone or the operating system running on the device. For this study, we therefore assume that identical devices were used for data collection. The UJIIndoorLoc dataset was created in 2013 and covers three buildings of Universitat Jaume I. More than 20 different users helped to create the dataset, which can be used for both floor classification, or latitude-longitude regression. For our study, we consider the data collected on a particular floor, of a particular building for regression.

B. Data Analysis and tuner configuration

For the Kaggle dataset, the values of M are 939 and 561 for F1 and F2, respectively. The value of M is 1137 for the UJIIndoorLoc dataset. Therefore, we consider our dataset to be scarce in terms of the input required for neural networks. When an AP i is not detected at a particular location, we insert a value of -100 for such missing cases in our Kaggle dataset, and a value of -105 for the UJIIndoorLoc dataset. Finally, we perform the standard scalar technique on the inputs before restructuring them as 2D images for the CNN model.

The CNN positioning algorithm is performed by splitting the datasets as train and test in the ratio of 80:20. We use the same set of training and test data for all further experiments. The training data is fed into our *CVTuner*, along with a selection of heuristic algorithm to determine the best hyperparameters. The objective of the tuner is set to find the minimum validation loss, which is computed by averaging out the loss obtained after applying k-Fold Cross-Validation technique with 5 folds over the training data. We specify the number of max_trials, which represents the number of hyperparameter combinations that will be tested by the heuristic algorithm as 150 due to time constraints.

VI. Experimental Results

In the experiments, we study the performance of our automated CNN architectures and compare it to the state-of-the-art VGG-16 architecture. We also interest the readers with the working of heuristic algorithms for an understanding of the underlying process for model selection. This can be crucial because although each of the algorithms have certain limitations, they play an important role in automating the CNN architecture.

A. Comparison of Accuracy

Through our framework we generate three automated CNN models, each generated through Bayesian optimisation, Hyperband and Random Search. We compare the performance of these against VGG-16 architecture, kNN and decision tree regressor, as summarized in Table I. The main performance criteria is the root mean square error of each of the architecture. The root mean square error is used in regression models to measure the differences obtained between the predicted observations and the actual observations. As seen from Table I, the Bayesian and Hyperband automated architectures both perform better than Random Search in all the cases, with lower RMSE values. This is no surprise as the Bayesian algorithm works with underlying probabilistic models and Hyperband optimises Random Search. While conducting the experiments we observed that while Bayesian optimisation is also capable of achieving such accuracy within 30 trials, this number in one case was as high as 117 for Hyperband. Despite that, all three heuristics performed better than the standard VGG-16 and other regression ML models. In addition, the depth of the automated models is fewer than VGG-16, thereby making the training process of models faster.

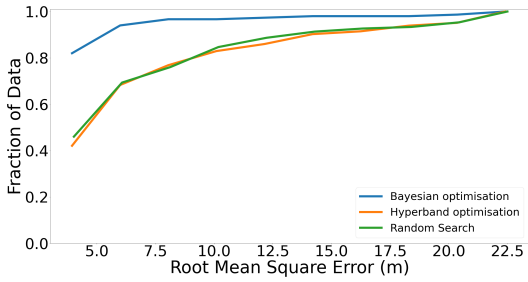
B. Comparison of Heuristic Algorithms

For all three datasets, we perform the hyperparameter tuning using Bayesian optimisation a total of five times. The same is carried out for the remaining two heuristics. For each dataset, we plot the RMSE values returned by the heuristic algorithms during one such experiment. As we perceive from Figures Fig. 4a, Fig. 4b and Fig. 4c, initially the Bayesian algorithm tries to find the best model for reaching the objective function and eventually the set of best hyperparameters over the model. Therefore, towards the end of each experiment, all the trials have approximately similar performance. In contrast, the Hyperband as shown in Figure Fig. 5a, Fig. 5b and Fig. 5c and RandomSearch in Fig. 6a, Fig. 6b and Fig. 6c executes random configurations, the only difference being that Hyperband optimizes the RandomSearch method by limiting the time spent on each configuration.

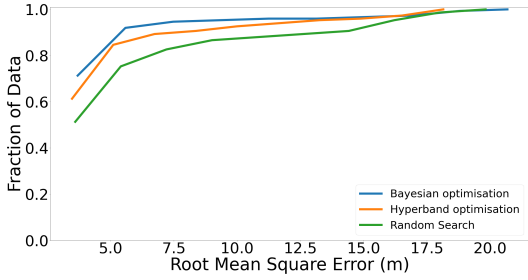
For a better understanding of the performances, we plot the cumulative density function graph of all three heuristics for all three datasets as shown in Fig. 3. As we notice, The RMSE for approximately 90 percent of the trials in Bayesian optimisation is similar and shows good performance. This is in contrast with Hyperband and Random Search where 90 percent of the performances also include high values of RMSE.

TABLE I: Performance comparisons between the CNN architecture automatically designed by our framework and other models on the Kaggle dataset Floor F1, Floor F2 and UJIIndoorLoc dataset Floor 0 Building 0

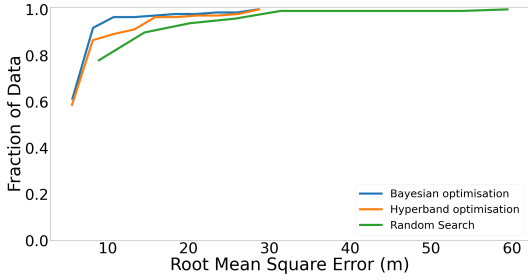
Model Architecture	Kaggle [11] floor F1 RMSE	Kaggle [11] floor F2 RMSE	UJIIndoorLoc [12] RMSE
Decision Tree [14]	3.35 m	3.61 m	5.13 m
kNN [13]	1.80	1.82 m	4.22 m
VGG-16 [6]	22.41 m	17.29 m	27.96 m
CNN-Bayesian optimisation [8]	1.89 m	1.81 m	3.06 m
CNN-Hyperband [9]	1.97 m	1.69 m	2.93 m
CNN-Random Search [10]	2.31 m	1.76 m	3.05 m



(a) Kaggle Floor F1



(b) Kaggle Floor F2



(c) UJIIndoorLoc Floor F0

Fig. 3: CDF depicting the performance of all three heuristics for each dataset

C. Comparison of the automated architectures

For a comparative study, we retrieve the best and poorest architectures returned by each of the heuristic algorithms for Kaggle Dataset F1. The best model corresponds to the model with the lowest RMSE on the test data and the poorest model corresponds to the model that achieved the maximum RMSE value. The complete comparison of the architectures is shown in the Table II. For all heuristics, the best generated CNN architectures require only three VGG blocks; VGG-16, on the other hand, has five VGG blocks. However, the location estimation accuracy of the generated CNN architectures is much higher than that of VGG-16.

From Table II, it can also be seen that the difference in the number of neurons for each layer gives a significant improvement in terms of location estimation accuracy, the RMSE is approximately 2 m for the best models and 22 m for the poorest models. The arrangement of neurons across the layers also does not follow any particular pattern, such as monotonically increasing, decreasing, or constant. Therefore, it is challenging to craft the best CNN model for fingerprint localization unless one has extensive knowledge about CNN and invests a lot of effort.

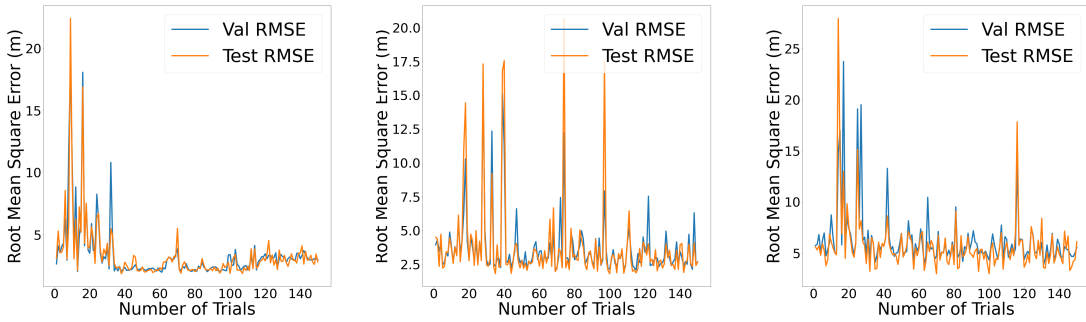
VII. Conclusion

This paper proposed a completely automated CNN architecture design based on VGG blocks for fingerprinting localisation. We base our study on a publicly available dataset uploaded on Kaggle as part of their Indoor Location and Navigation competition series. The WiFi localisation is carried out by generating a WiFi database and matching it against a CNN positioning algorithm. The CNN models are automatically designed by our proposed framework and, especially with the Bayesian optimisation algorithm, yield a high localisation accuracy and outperform the standard VGG-16. The significant dependence of a model's performance on its architecture is also shown by drafting a comparison between the worst and the best model. This dependence proves that a slight modification in the CNN architecture would lead to a massive change in localisation accuracy. In other words, we recommend adopting our fully automated CNN designing framework for indoor localisation with WiFi fingerprinting. Although the VGG blocks and the heuristics algorithms are used in the proposed framework, the readers are not required to have expertise in these when they are using the proposed framework. It is straightforward to extend this framework with other Neural Networks such as ResNet and DenseNet blocks.

References

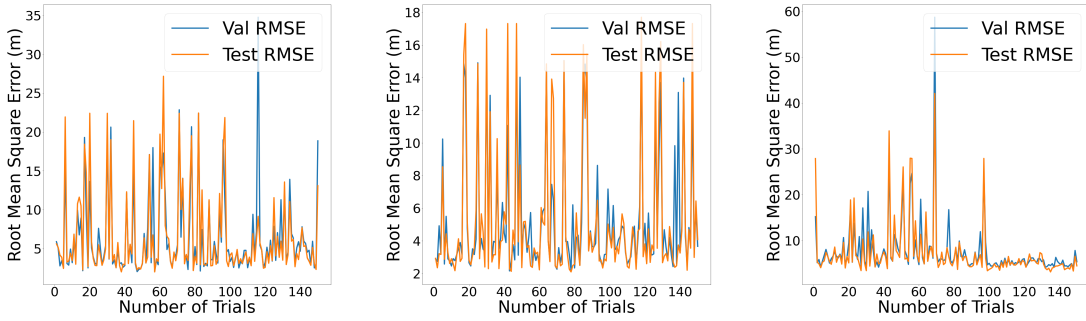
[1] R. Harle, "A Survey of Indoor Inertial Positioning Systems for Pedestrians," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[2] L. Li, X. Guo, N. Ansari, and H. Li, "A Hybrid Fingerprint Quality Evaluation Model for WiFi Localization," *IEEE Internet of Things Journal*, vol. 6, pp. 9829–9840,



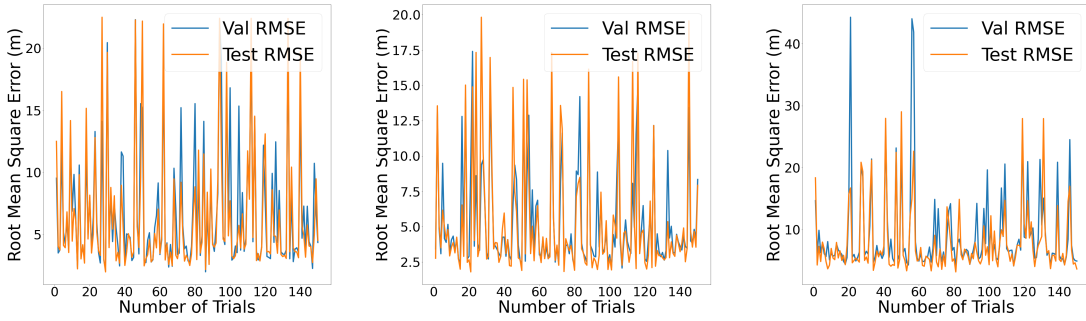
(a) (b) (c)

Fig. 4: Search for the hyperparameters via Bayesian optimisation Algorithm: (a) Floor F1, (b) Floor F2 and (c) UJIIndoorLoc



(a) (b) (c)

Fig. 5: Search for the hyperparameters via Hyperband optimisation Algorithm: (a) Floor F1, (b) Floor F2 and (c) UJIIndoorLoc



(a) (b) (c)

Fig. 6: Search for the hyperparameters via Random Search Algorithm:(a) Floor F1, (b) Floor F2 and (c) UJIIndoorLoc

Dec. 2019. Conference Name: IEEE Internet of Things Journal.

[3] C. Basri and A. El Khadimi, "Survey on indoor localization system and recent advances of WIFI fingerprinting technique," in *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, (Marrakech, Morocco), pp. 253–259, IEEE, Sept. 2016.

[4] X. Song, X. Fan, C. Xiang, Q. Ye, L. Liu, Z. Wang, X. He, N. Yang, and G. Fang, "A novel convolutional neural network based indoor localization framework with wifi fingerprinting," *IEEE Access*, vol. 7, pp. 110698–110709, 2019.

[5] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely Automated CNN Architecture Design Based on Blocks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, pp. 1242–1254, Apr. 2020. Confer-

TABLE II: Comparison of the hyperparameters returned for the best and the poorest architecture for Kaggle F1

	Bayesian optimisation		**Hyperband optimisation**		**Random Search**	
Architecture	**Best RMSE (1.89 m)**	**Poorest RMSE (22.41 m)**	**Best RMSE (1.89 m)**	**Poorest RMSE (22.38 m)**	**Best RMSE (1.96 m)**	**Poorest RMSE (22.49 m)**
Batch Size	16	16	8	16	24	8
Learning Rate	0.001	0.001	0.0001	0.001	0.001	0.001
Number of VGG Blocks	3	3	3	3	3	3
VGG Block 1	Conv2D-**128**	Conv2D-**64**	Conv2D-**64**	Conv2D-**512**	Conv2D-**128**	Conv2D-**512**
	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
	Conv2D-**64**	Conv2D-**128**	Conv2D-**128**	Conv2D-**256**	Conv2D-**512**	Conv2D-**512**
	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
	MaxPool	MaxPool	MaxPool	MaxPool	MaxPool	MaxPool
VGG Block 2	Conv2D-**128**	Conv2D-**256**	Conv2D-**512**	Conv2D-**512**	Conv2D-**256**	Conv2D-**256**
	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
	Conv2D-**512**	Conv2D-**256**	Conv2D-**128**	Conv2D-**128**	Conv2D-**64**	Conv2D-**64**
	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
	MaxPool	MaxPool	MaxPool	MaxPool	MaxPool	MaxPool
VGG Block 3	Conv2D-**64**	Conv2D-**128**	Conv2D-**256**	Conv2D-**256**	Conv2D-**512**	Conv2D-**128**
	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
	Conv2D-**256**	Conv2D-**512**	Conv2D-**512**	Conv2D-**256**	Conv2D-**64**	Conv2D-**256**
	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm	BatchNorm
	MaxPool	MaxPool	MaxPool	MaxPool	MaxPool	MaxPool
Fully Connected Layers	Dense-**128**	Dense-**128**	Dense-**384**	Dense-**64**	Dense-**256**	Dense-**256**
	Dropout-**0.0**	Dropout-**0.25**	Dropout-**0.0**	Dropout-**0.25**	Dropout-**0.0**	Dropout-**0.25**
	Dense-2	Dense-2	Dense-2	Dense-2	Dense-2	Dense-2

ence Name: IEEE Transactions on Neural Networks and Learning Systems.

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015. arXiv: 1409.1556.

[7] A. Krishnaswamy Rangarajan and R. Purushothaman, "Disease classification in eggplant using pre-trained vgg16 and msvm," *Scientific Reports*, vol. 10, p. 2322, Feb 2020.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

[9] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," 2018.

[10] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.

[11] "The Kaggle Dataset the kaggle dataset for indoor localization." Accessed: 2021-05-30.

[12] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[13] O. Kramer, *K-Nearest Neighbors*, pp. 13–23. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[14] Y.-Y. Song and Y. Lu, "Decision tree methods: applications for classification and prediction," *Shanghai archives of psychiatry*, vol. 27, pp. 130–135, Apr 2015. 26120265[pmid].

[15] J. Rojo, G. M. Mendoza-Silva, G. Ristow Cidral, J. Laiapea, G. Parrello, A. Simó, L. Stupin, D. Minican, M. Farrés, C. Corvalán, F. Unger, S. M. López, I. Soteras, D. C. Bravo, and J. Torres-Sospedra, "Machine Learning applied to Wi-Fi fingerprinting: The experiences of the Ubiqum Challenge," in *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–8, Sept. 2019. ISSN: 2471-917X.

[16] K. Sabanci, E. Yigit, D. Ustun, A. Toktas, and M. Aslan, "WiFi Based Indoor Localization: Application and Comparison of Machine Learning Algorithms," pp. 246–251, Sept. 2018.

[17] X. Wang, L. Gao, S. Mao, and S. Pandey, "CSI-based Fingerprinting for Indoor Localization: A Deep Learning Approach," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2016.

[18] J.-W. Jang and S.-N. Hong, "Indoor Localization with WiFi Fingerprinting Using Convolutional Neural Network," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, (Prague), pp. 753–758, IEEE, July 2018.

[19] A. Mittal, S. Tiku, and S. Pasricha, "Adapting Convolutional Neural Networks for Indoor Localization with Smart Mobile Devices," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, (Chicago IL USA), pp. 117–122, ACM, May 2018.

[20] T. Qu, M. Li, and D. Liang, "Wireless indoor localization using convolutional neural network," *Journal of Physics: Conference Series*, vol. 1633, p. 012125, Sept. 2020.

[21] L. Wu, G. Perin, and S. Picek, "I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis," p. 23.

[22] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, p. 295–316, Nov 2020.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.