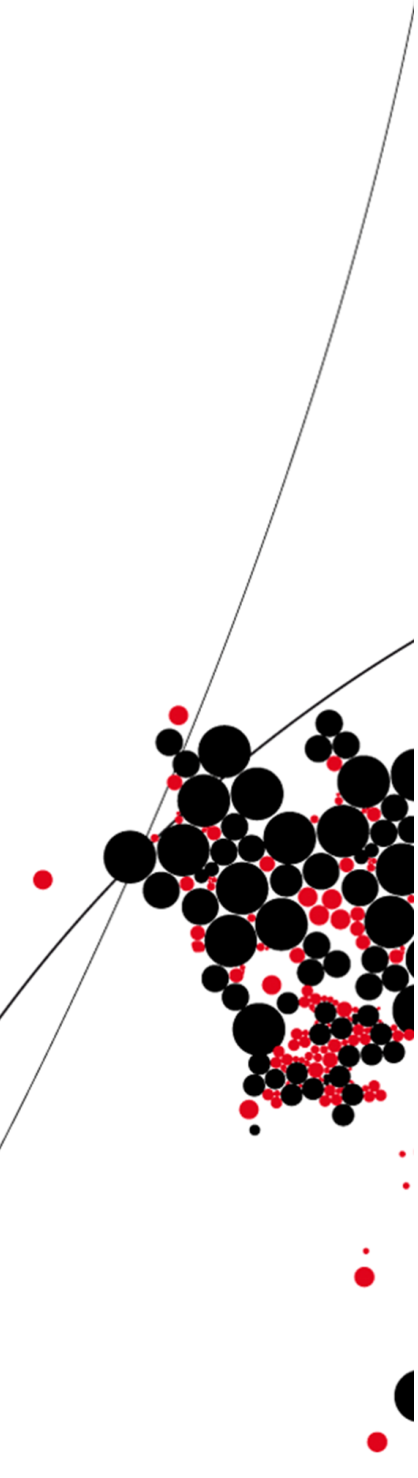
UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

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

Shreya Sinha

M.Sc. Thesis Embedded Systems
September 2021



Supervisors:
dr. Le Viet Duc

Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

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

***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, this challenge is addressed by developing a framework that completely automates the CNN architecture design process. The automated architectures based on VGG blocks have shown superior performance compared to standard architectures such as VGG-16. Further, three heuristics are explored for automation: Bayesian optimisation, Hyperband, and Random Search, which demonstrate their importance towards the automated CNN architecture development for WiFi fingerprinting. Experiments are conducted on publicly available datasets and, a comparative study between the automated architectures and other models is presented. This work would, therefore, facilitate the CNN design process for WiFi indoor localisation.**

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

I. Introduction

Indoor localisation is a concept of locating a person or a device in an indoor environment. It 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, indoor localisation still remains an open challenge as GPS does not usually work indoors.

Among many of the solutions studied for indoor localisation, WiFi-based is one of the most attractive solutions because: it is easy to deploy and maintain, wireless LANs (WLANs) are ubiquitous, and smartphones are nowadays WiFi-enabled [2] making scanning of WiFi signals an easy process. Among various techniques, fingerprinting [3] is a common and popular one to identify the location of a user by characterising his radio signal environment [4]. 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 present location. These measurements are then matched with the radio map to determine the user's position with the aid of position estimation algorithms.

One such potential position estimation algorithm is Convolutional Neural Networks (CNN) [5], which is a deep learning algorithm popularly used to solve image recognition and processing problems. It has shown good results with images and therefore, due to its ability to learn complex relationships between RSSI and location coordinates as demonstrated in [6, 7], can serve as a promising solution for WiFi localisation.

The performance of a CNN model depends heavily on its architecture, which is a possible combination of the number of layers and associated hyperparameter values set within the layer. Although previous works could shortlist each potential architecture, and then train and validate it to choose the best among them, a satisfactory solution may not be an optimal solution because it is difficult to come up with the number of layers needed and tackle multiple possible hyperparameter combinations. This is because the computational complexity also increases greatly during runtime as the list of possible hyperparameter combinations that needs to be tuned grows. Moreover, designing the architecture requires expertise in both CNN's working and an understanding of the problem domain [8]. In practical scenarios such as WiFi-based fingerprint localisation, this is often more challenging to do so. Even the popular hand-crafted architectures, such as VGGNet based on VGG blocks [9], whose performance has been proven on many data types and applications [10, 11], may still not guarantee good accuracy for WiFi fingerprint localisation.

A. Problem Statement

The designing of CNN architecture to deliver good performance requires expertise in CNN's working. To this day, the development of completely automatic CNN architectures for WiFi fingerprint localisation with promising performance, constrained by limited computational resources, is still in its infancy. Additionally, tuning the hyperparameters for optimising the architectures requires considerable time and effort. Therefore, for optimisation of CNN architectures, three heuristic computational paradigms, such as Bayesian optimisation

[12], Hyperband [13] and Random Search [14] still needs to be explored. The main objective of this research therefore, is to generate a framework for automating the CNN architecture design process based on the VGG block for WiFi localisation, and explore the three heuristics for optimisation of such architectures.

B. Research Questions

How do we solve indoor localisation problem using WiFi fingerprinting technique [3] and CNN [5] as position estimation algorithm, given that CNN is complex to design and tune for a novice?

The main question can be sub-divided into:

1) How do we design a framework for automating the development of CNN architecture for WiFi fingerprint localisation?
2) How do we optimise such architectures using heuristic techniques such as Bayesian optimisation [12], Hyperband [13] and Random Search [14]?
3) Given the automated architectures, on what basis can we decide which heuristic technique performs the best?

C. Methodology

To this end, the development of completely automatic CNN architectures for WiFi fingerprint localisation with promising performance, constrained by limited computational resources, is still in its infancy. The target of this research is to develop an automated framework for CNN architecture designing based on the VGG block, and use the heuristic computational paradigms, such as Bayesian optimisation [12], Hyperband [13] and Random Search [14] for optimisation. To evaluate the framework, we use publicly available datasets such as Indoor Location, and Navigation competition series on Kaggle [15], as well as the UJIIndoorLoc dataset [16]. 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 research question is achieved by breaking it down to following goals.

1) Designing a framework to automatically deliver a CNN architecture based on VGG Blocks for WiFi fingerprint localisation.
2) Designing a framework which automates the hyperparameter tuning process of the automatically designed CNN architectures using the three heuristics for optimisation.

The proposed framework assumes no prior knowledge of the user about the CNN design process, investigated dataset and genetic algorithms.

D. Thesis Outline

The thesis follows the following structure:

Chapter 2: **Background and Related Work** - In this section, we first discuss theory regrading indoor localisation using WiFi fingerprint technique and basics of CNN architecture. We then discuss the theory behind the heuristic algorithms for optimisation. This section is later followed by some state-of-the-art solutions involving WiFi Indoor localisation using CNN and its architecture optimisation.

Chapter 3: **Completely Automating the CNN design process** - In this section we discuss an overview of our suggested framework. We discuss the prerequisites required for WiFi based indoor localisation using CNN. We also provide in depth discussion of model backbone, building blocks and hyperparameters tuned for model optimisation.

Chapter 4: **Experimental Setup** - In this section we discuss the datasets in detail, we perform data analysis and provide some additional information regarding the scenario where experiments were conducted and tools used for implementation.

Chapter 5: **Experimental Results** - In this section, we compare the performances of our automated CNN architectures with other machine learning and deep learning algorithms. Further, we compare the working of all three optimisation heuristics to understand their importance towards CNN architecture optimisation. In the end, we study the best and worst architecture returned by all the three heuristics and form a comparative study between them.

Chapter 6: **Discussion and Conclusion** - In the discussion section, we reflect on the study and our findings. Finally, we summarise our findings and results in the conclusion section followed by recommendations to improve or extend the existing work.

II. Background and Related Work

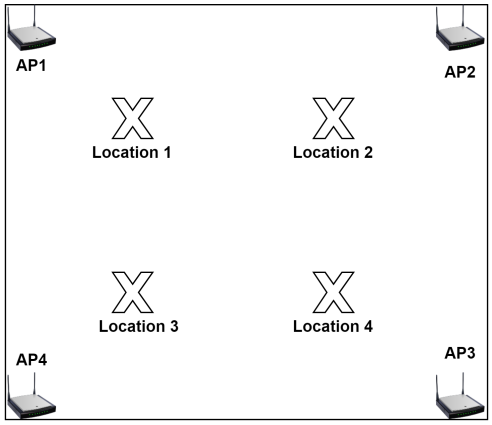
In this section, the theory behind WiFi fingerprinting technique is presented. It is then followed with a basic background about the CNN algorithm and the default available VGG-16 CNN architecture. A basic background of the three optimisation heuristics is also presented, and the section is concluded with some related works relevant to this study.

A. Fingerprinting Technique

Among many technologies used to solve indoor localisation problem, WiFi based [17, 18] is a popular technique due to it's universality being the main advantage. Often, there is no need to add any additional hardware devices to deploy WiFi in buildings and individuals can use WiFi with ease on their mobile devices. WiFi, which stands for Wireless Fidelity, uses radio frequencies to send signals between two or more devices. WiFi localisation i.e to estimate a user's position

in an indoor space using WiFi, is popularly solved via two methods: triangulation and fingerprinting [3]. In triangulation, the user's mobile device captures the radio signals emitted by the WiFi infrastructure. Based on the strength of the signals received by the device, the three most strongest signals are selected and the distance between the device and the corresponding WiFi hardware is calculated. Finally, with the help of a position estimation algorithm, the final position of the user is calculated based on these distances. However, one drawback of this method is that due to the obstacles present in the room, there can be severe errors in calculating distances. One way the researchers have overcome this challenge is by applying fingerprint technique, which has also proved to be more accurate [19].

Fingerprint based technique [3] is conducted in two phases: offline and online. In the offline phase, the user moves around the indoor space with a mobile device such as a smartphone. The device is responsible for collecting the radio signals from each of the Access Point (AP) deployed in the space. An AP is responsible for projecting WiFi signals in the designated area, and thus the mobile device collects the WiFi signals from all the detected AP in user's area. Received Signal Strength Indicator (RSSI) measures how strong or weak the signals are, which are received by the mobile device. For example, if the mobile device detects an RSSI value of -30 dBm from AP1 and -80 dBm from AP2, indicates that the user is located closer to AP1 in comparison to AP2, and therefore receives stronger signals from AP1. A radiomap is created based on all the RSSI values collected from all detected APs at particular location. In the online phase, the user collects the RSSI values at their present location, which is then matched against with the radio map created in the offline phase. With the help of a position estimation algorithm, the position of the user is determined. Fig. 1 depicts the offline stage and radio map created.



AP1	AP2	AP3	AP4	Location
-30	-50	-80	-49	1
-51	-30	-50	-82	2
-50	-81	-48	-31	3
-82	-52	-31	-51	4

Fig. 1: WiFi RSSI values collection from 4 APs at fixed known location followed by radiomap creation

Although this technique promises accurate results, the site survey process, i.e. collecting RSSI fingerprints from fixed interested locations is time consuming and labour intensive. The data collection process needs to be repeated several times at the same reference points. To overcome this challenge, crowdsourcing [20] method can be implemented. In this method, several users move along a trajectory in the indoor space collecting WiFi signals from the detected APs. Thereby, all the parts of the indoor space is covered and the data is then consolidated to create the radiomap.

B. CNN - Basic Structure

Convolutional Neural Network (CNN) [5] has provided excellent results in certain fields such as image processing and even voice recognition. They are a class of deep artificial neural networks which are able to maintain spatial integrity of input images, i.e. the images can be fed into the network in their original 2-D form if they are greyscale images, or 3-D form if they are RGB images. After the images are fed into the CNN, the process follows the below order.

1) Extract features: CNNs extract features with the help of convolutional filters, which are in practice set of weights applied to the input image. Convolutional filters slides over the input image, and create a filtered version of the image known as feature map. Different filters are capable of creating different feature map from the same input image. Convolutional filters can have both positive and negative weights for creating such feature maps.
2) Processing by activation function: The feature maps are further processed by an activation function. Generally, ReLU [21] is a popular activation function used in CNN. Activation functions introduce non linearity in the CNN architecture or model helping it learn complex information in the data. ReLU, for example, ensures that only the nodes with positive activation, i.e the nodes with a value greater than 0.0 will send their values forward to the next step. The nodes with negative activation send a value of 0.0. The nodes that are positively activated are likely to contain useful information and focusing on only these can possibly lead to better results.
3) Dimensionality reduction via pooling: The goal of pooling is to reduce the size of feature maps without the loss of information. This can reduce the amount of processing required in the further stages as well as reduce the time during training the model. A filter is passed over the feature maps for pooling. Common

variants of such filters are MaxPool, AveragePool and SumPool [22]. As the name suggests, MaxPool takes the maximum pixel value within the filter, AveragePool averages the pixel values within the filter and SumPool sums them up.

CNN can be very deep networks comprising of several layers of convolutional and Pooling filters. However, the last few layers of the network, are fully connected (FC) layers. The output of the convolutional or pooling layers is passed to the FC layers. There can be one or several FC layers depending on the problem but the final output layer is adjusted according to whether the problem is a classification or a regression problem.

Once the architecture is ready, the network is trained using few but not all data samples known as training data. Training the network means finding the appropriate weights of the neural network. This is achieved by updating the weights of the network in response to the errors made by the network on training data. Loss functions specified in the output layer is used to calculate the error i.e the difference between the predicted output and the actual output across the training samples. The type of loss function employed depends on the problem. For indoor localisation, the goal of the network in all cases will be to minimise the loss. Therefore, an optimiser is used, which is an algorithm used to change the parameters of the neural network such as weights to reduce the losses. Example of such optimisers are Adam [23], Adadelta [24] and it should not be confused with the optimisation algorithms such as Bayesian optimisation, which are used to change the hyperparameters of the architecture and will be discussed further. Therefore, training the network is a long process and needs to be repeated several times to achieve optimal performance.

C. CNN - VGG-16

VGG-16 was first presented in [9]. 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 connected layers with different dimensions is placed, the value of which depends on the problem. A complete VGG-16 architecture is shown in Fig. 2.

Fig. 2: A complete VGG-16 architecture

D. CNN - Overfitting

Once the basic structure of the CNN is created, training the neural network is the next important step. It is the process of finding appropriate weights and biases of the network. The available dataset is usually divided into training and testing data. The training data is used to train the network repeatedly to find the best possible values of weights and biases, by checking how closely the computed output is to the known correct output values. Once the training phase is completed, the testing data is fed as input to the model once, and the accuracy of the model is computed by finding the extend to which the computed output differs from the known correct output. Therefore, the models learns the appropriate values of weights and biases from the training data and the final performance of the model is calculated on the basis of how well it performs over the test data. However, one big challenge faced during neural network training is the problem of overfitting [25]. It is the process when the model learns the training data well but fails to perform satisfactorily on the test data. There are some ways to tackle this problem such as:

1) k-fold cross validation [26]: In this case, the training data is divided into k disjoint subsets of approximately equal sizes. The CNN architecture or model is trained using k-1 subsets, which now together form the training data. The remaining one subset is now called the validation data, against which the model's performance is measured. The procedure is repeated until each of the k subset has as served as the validation dataset. The total performance of the model is the average of the performance of the model over k validation datasets. Fig. 3 illustrates this process with 5 fold cross validation. The original training data is divided into 5 subsets, therefore k=5. In the first iteration or fold, the first subset serves as the validation set and remaining are the new training datasets. In the second iteration, the second subset is the validation set and so on. The final performance of the model is still tested on the test data however, applying the k-fold cross validation technique [27] gives the scientists a better intuition on how the model performs on unseen data while training.

Iteration 1	Val	Train	Train	Train	Train
Iteration 2	Train	Val	Train	Train	Train
Iteration 3	Train	Train	Val	Train	Train
Iteration 4	Train	Train	Train	Val	Train
Iteration 5	Train	Train	Train	Train	Val

Fig. 3: k-fold cross validation with 5 folds

2) Early Stopping [28]: Training the model for too long can also lead to overfitting. The strategy of early stopping is used to stop the training process after some time, once

the accuracy of the model stops improving. The training time of the model depends on the number of epochs. i.e the model will be trained for a longer time if the number of epochs is 100, in comparison to when we set the epoch value 10. As shown in Fig. 4, the vertical axis represents the number of epochs and the horizontal axis is the error. The blue line represents the training error and the red line represents the validation error. If the model continues to be trained after a certain point, the validation error and training error will deviate from each other significantly, leading to overfitting. If we stop the training of the model earlier, we run into the problem of underfitting [29]. Therefore, the goal is to stop the training of the model at the exact point to prevent both the problems.

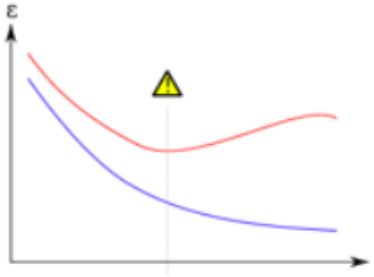


Fig. 4: CNN architecture or model's training process. The training and validation error plotted with respect to the number of epochs

3) Dropout Layer: The process of dropout [30] is a popular technique to prevent overfitting. Each layer in the neural network consists of several nodes or neurons. These neurons are initialised with weights randomly at the beginning of the training process, making the error higher at the beginning. In dropout technique, few neurons are dropped out from the network during each iteration. It can be seen as them being temporarily removed from the network. The removal of neurons from the network is random, and a tunable parameter 'p' determines the percentage of neurons to deactivate. The dropout procedure can be seen as averaging the affect of several different networks, which forces the network to not rely on one feature, but learn robust features that are useful.

Training the neural network is a time consuming process, especially for a deep architecture. Batch Normalisation is a widely adopted technique to enable faster and more stable training of neural network. Normalising the inputs of the neural network to a zero mean and a constant standard deviation had been known since a long time to benefit the training process [31]. Batch normalisation layers in addition, can be inserted in the CNN architecture the same way as convolutional and FC layers. There might be a little disagreement within the scientific community regarding batch normalisation improving the training process or in general accuracy of the results. However the successful proliferation of batch normalisation in several areas of deep learning [32, 33] cannot be ignored. The authors in [34] claim that the internal covariate shift is the reason for slow training of deep architectures. They describe this phenomenon as the change in the distribution of network activations due to the change in network parameters during training. For a sequential neural network, the output of the first layer is the input to the second layer, the output of the second layer is the input to the third and so on. Therefore, when the parameters of a layer change, so does the distribution of the inputs to the next layer. These shifts can be problematic for larger and deeper networks. Batch normalisation method is intended to reduce this effect by normalising the distribution of inputs of each layer to have zero mean and standard deviation of one.

E. Hyperparameter Tuning

Hyperparameter tuning [35] is the process of finding the right combination of hyperparameters, to maximize the performance of CNN. Hyperparameters in neural network are variables that people can set manually and it defines the overall structure of the network [36]. For example, in case of CNN, the hyperparameters can be the learning rate of the optimiser, the variable 'p' controlling the percentage of neurons to dropout, the size of filters in Conv2D layer and MaxPool layers or even the number of layers in neural network. Tuning the hyperparameters can improve the performance of CNN significantly, however as the complexity of the neural network increases, hyperparameter tuning can become increasingly difficult. Model complexity, for example, can include how deep the neural network is, along with the appropriate number of filters in each layer. Simpler models might not be able to learn much from the dataset while complex models may overfit the data. Therefore, hyperparameter tuning allows the scientists to come up with the correct CNN architecture configuration according to the problem statement. Since the hyperparameters are problem dependant and therefore an architecture with certain hyperparameter values that work for one dataset might not work for another. Hyperparameter tuning also affects the overall training time. Changing the hyperparameters can change the training time from few minutes to few hours. Hence, hyperparameter tuning requires a balance of coming up with an optimised architecture and letting it train for a satisfactory amount of time, which is a tedious and difficult task and demands an expert's experience and knowledge.

Hyperparameter tuning can be mainly divided into two categories: manual or automatic. Manual search tries different combinations of hyperparameters by hand and depends on the expert's intuition in identifying the important hyperparameters that will have greater impact on the results. For non-experts, this is a difficult process. As the number of possible hyperparameters to tune and the range they can take increases, the number of possible combinations that can be created also

increases and hyperparameter tuning can become a complex process. To overcome this challenge, automatic algorithms for hyperparameter tuning are proposed. Some of them which interests our topic of study are listed below.

1) Grid Search: it is one of the most popularly used methods to explore the hyperparameter configuration space [35]. The Grid Search algorithm in general tries all possible combination pairs. Therefore, for example if the the learning rate can take two values of 0.01 and 0.001 and the rate of dropout needs to be checked between two values 0.3 and 0.4, there will be four different combinations to try and therefore four different architectures to train and validate. The algorithm tries all the possible combinations and therefore performs an exhaustive search of the given hyperparameters and their range. The advantage of Grid Search is that it is easy to implement and is effective for small search spaces. The biggest limitation is, as the number of hyperparameters to tune grows, or the range of values hyperparameters can take increases, the search for an optimised architecture by trying out all possible combinations becomes computationally expensive.
2) Bayesian Optimisation: The Bayesian optimisation [37] 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 Bayesian probabilistic model, and an acquisition function, which determines the samples to search next [37]. 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. Bayesian optimisation, therefore provides an advantage over Grid Search in case the hyperparameter search space is large. However, since it follows a sequence of order, it is difficult to parallelise it.
3) Hyperband: Hyperband [13] 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 [13].
4) Random Search: Random Search [14] bases its optimisation strategy on a stochastic process. It randomly selects samples between the upper and lower bound of each hyperparameter and then trains the model using these until the budget mentioned, such as max_trials is exhausted. Therefore, unlike Bayesian optimisation, it does not take past evaluations into account. It is much more efficient than Grid Search in large search spaces [14], however it's main drawback is that a large number of evaluations conducted are unnecessary since it does not exploit previously known regions.

F. Related Work

Numerous studies have addressed fingerprinting-based WiFi localisation. Initially, machine learning algorithms (ML) such as K-nearest neighbors (k-NN) [38], Support vector machine (SVM) [39], Random forests and, Decision trees were explored. In [40], 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 [41], a comparative study is conducted between six different types of ML algorithms to measure their performance over a common dataset. The authors in [42] incorporated an unsupervised ML method such as Principle Component Analysis (PCA) to improve the localisation performance.

Since Deep Learning can solve a complex problem better than standard machine learning, Deep-Neural-Network (DNN)-based methods have also been studied [43, 44]. The main drawback of these methods is the poor accuracy they achieve when the dataset is insufficient [45]. Reference [45] analyses WiFi localisation using CNN, where the authors convert RSSI vectors into 2-D images for multi-floor classification. Similarly, CNN-LOC [46] 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 [47].

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 [8]. Moreover, an exact CNN architecture such as VGG-16 [9] might be unable or ineffective to solve some problems. As a result, architecture optimisation algorithms based on heuristic computational paradigms, such as, random search [14], Bayesian optimisation [12], and Hyperband [13], have been proposed. However, to our knowledge, none of the previous works has proposed a fully automated end to end CNN architectural development using heuristics for optimisation for WiFi fingerprint localisation.

III. Completely Automated CNN design for Fingerprinting localisation

In this section, an overview of the proposed framework is given. The framework helps in understanding the automated end-to-end process of developing the CNN architecture for WiFi localisation. For a deeper understanding, the process of restructuring the data to create the input images for the CNN model, the proposed CNN building blocks, and finally the optimisation using the hyperparameter tuning technique is also discussed.

A. Overview

The framework for developing an automated CNN architecture is shown in Fig. 5. The initial phase is the data collection phase performed by recording WiFi scans using a smartphone.

The recorded data is then treated in two ways depending on the dataset used. In case of Kaggle dataset [15], the collected data is cleaned, useful information is extracted, the data is rearranged in the fingerprint radiomap format and missing values for any label corresponding to a particular AP is filled up. Additionally, dummy columns are added with fake APs in case the original total number of AP is not a perfect square. In case of UJIIndoorLoc [16], the data preprocessing stage involves adding a value in case there are any missing values for a particular label corresponding to a particular AP. Further, similar to Kaggle dataset, dummy columns with fake APs are added to make the total number of APs a perfect square. Input images for the CNN architecture is then generated by reshaping the original 1-D RSSI array into a 2-D array for each label, which serves as an input image for the CNN algorithm.

The third stage of our framework involves developing a workflow for automating the CNN design process by finding the best possible selection of hyperparameters. In this stage we develop a custom tuner called *CVTuner* which will help in achieving our goal. It consists of two components, model building and model evaluation. The model building function contains an argument for sampling the hyperparameters. This function along with the choice of heuristic algorithm for architecture optimisation are used to instantiate the CVTuner. The model evaluation stage comprises of initiating the tuner search process by passing the necessary arguments. Depending on the heuristic algorithm specified, several different architectures are explored and the best one is retained.

Finally, we analyse the performance of the best automated architecture against the test data and perform position estimation over it.

B. Data Collection

The Kaggle dataset [15] is provided by an indoor positioning company XYZ10 in collaboration with Microsoft Research. It was collected with the help of several users walking along a particular path in a shopping mall. During the walk, android smartphones are held firmly in front of the chest of the user, enabling the user to collect sensor data. The sensor data is recorded in several trace files, and consists of readings belonging to different sensors such as bluetooth, WiFi, accelerometer, gyroscope etc. The Kaggle dataset is cleaned to extract only the WiFi scans which were recorded in the trace files.

The UJIIndoorLoc [16] 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. The dataset consists of only WiFi sensor readings.

C. Data Preprocessing

In case of Kaggle dataset [15], it is first cleaned by arranging the data is the correct format. The clean data consists of rows of APs and their readings followed by their timestamp. Therefore AP1,AP2...APn,Timestamp represents the cleaner WiFi scan. The location information for each WiFi scan in the form of 'x' and 'y' coordinates is computed. This is achieved by using the complimentary files provided along with the dataset. A radio map is thus fabricated combining all the required information. Suppose that during each WiFi scan, the user receives RSSI values from neighbouring N APs, namely, the data **w** = $(ap_1, ap_2...., ap_N)$ is a 1-D vector of length N, where ap_i denotes the RSSI value obtained from the AP i. Each data **w** 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. Since the CNN requires a 2-D array input, 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 and rearrange the 1-D array into a 2-D array.

Similarly, UJIIndoorLoc dataset [16] also consist 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. To make the number of AP a perfect square, we add dummy APs with the value of -105 dBm. Similar to Kaggle dataset, we rearrange the 1-D array into a 2-D array for each label.

We further scale both the datasets by applying a standard scalar. The standard scalar transforms the data in a way that mean is 0 and standard deviation is 1. This may help to speed up the training process.

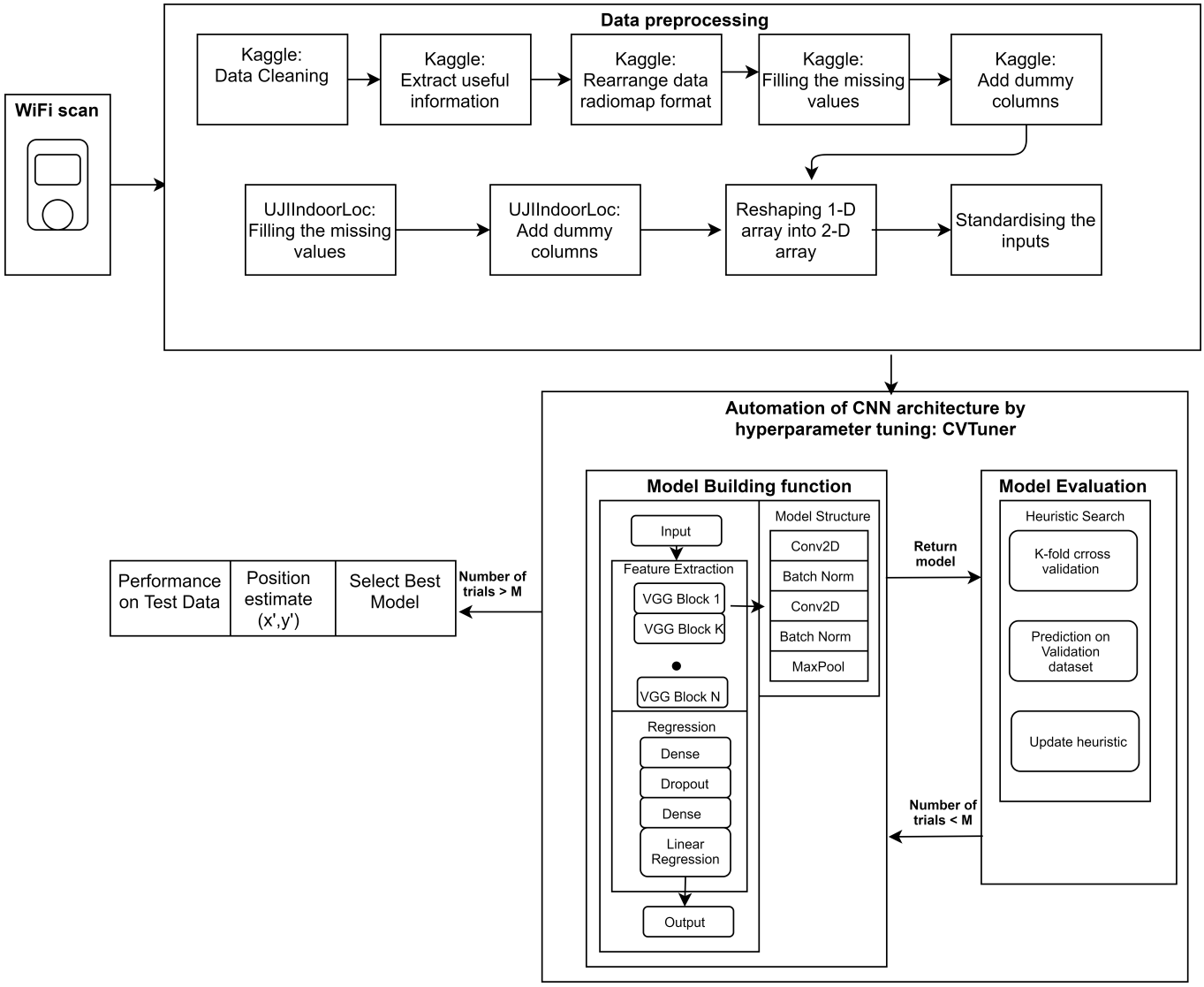


Fig. 5: Overview of the WiFi fingerprinting localisation with hypertuned CNN architecture

D. 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 [9], a VGG block consists up of two Convolutional layers and one MaxPool layer. In our proposed model backbone, we additionally add Batch Normalization after each convolutional layer to standardize the inputs of each layer. This might reduce 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.

E. 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 which takes an argument *hp* from which one can sample hyperparameters within a specified range. The details of the body of the CVTuner is further summarized in Algorithm 2. For initiating the CVTuner search process for an optimised architecture, certain arguments are passed such as training data, batch size etc. However, in our case, we also hyperparameter tune the batch size, as presented in line 2 of algorithm 2. The value of epoch is constant, but the technique of early stopping is employed while training the model to prevent underfitting and overfitting. Finally, we split the training data into k folds and apply k-fold cross validation [26] technique to analyse the architecture and calculate the loss. The total loss of the architecture is the average of the loss over k folds. The total number of architecture or trials

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

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, Early stop, 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

tried out by the CVTuner is the value of the max_trials. Each architecture is assigned a unique trial id which can be accessed to retrieve architecture information such as the number of VGG units, batch size, rate of dropout unit etc. This is achieved as it is possible to call the model building function from within the body of the CVTuner. Therefore, for every new trial, a new architecture with new set of hyperparameters is used for evaluation. However, for k-fold cross validation, the trail id remains the same and therefore the k-fold cross validation takes over the same architecture. The next set of hyperparameters of an architecture for a new trial depends on the choice of heuristic optimisation algorithm specified while instantiating the tuner. For example, in case of Random Search [14], past configurations of hyperparameters are not taken into account and the new set of hyperparameter configuration is completely random. However, in case of Bayesian optimisation [37], this is not the case. Finally, 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 as it measures the average squared difference between the predicted values and the actual values. We finally calculate the root mean squared error (RMSE) i.e root of MSE for architecture performance and comparisons.

F. Hyperparameters

In the following, we discuss some of the hyperparameters that we define in the model building function and CVTuner for hyperparameter tuning.

- Batch Size: The performance of a CNN model is affected by its batch size [48]. Larger batch size speeds up the computations, but too large a batch size may result in a poorly generalised model. Therefore we set the range to tune the batch size in CVTuner.
- Number of VGG Blocks: We also leave it up to the CVTuner to determine the number of VGG blocks to tune. Too many units can lead to an overly complex architecture. Fewer layers may yield inaccurate results. This is achieved by using a for loop in the model building function. The for loop executes *n* number of times, creating n number of VGG blocks. The values of *n* is taken up by the CVTuner from a specified range.
- 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. It can be challenging to come up with the number of units in the dense layer, especially for a

novice.

- 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. We therefore, let the automation process decide the ideal learning rate.
- Rate of 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 significantly.

IV. Experimental Setup

In this section, we implement the previously described framework to solve the indoor WiFi localisation challenge. We describe the dataset, the techniques used to optimise the architecture and the environment where the experiments were conducted.

A. Dataset Description

The Kaggle dataset [15] consists of multiple traces of sensor data collected from over 200 buildings. However, for our study, we only consider the data collected from two floors: F1 and F2, from the same building named 5da138364db8ce0c98bc00f1. 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 total number of APs for floor F1 (including dummy APs) is 676 for floor F2 is 289. The total number of data samples i.e the value of M is 939 for F1 and 561 for F2.

The UJIIndoorLoc [16] dataset also consists of data collected over several buildings and floors. However, since we only want to perform position estimation for x and y coordinates, we consider the data collected only on Building number 0, floor number 0. The total number of APs (including dummy APs) is 529. 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.

B. CVTuner configuration and prerequisites

The CNN positioning algorithm is performed by splitting the datasets as train and test in the ratio of 80:20 for both Kaggle F1, F2 and UJIIndoorLoc dataset Building 0 floor 0. Going forward we will address this simply as UJIIndoorLoc for the purpose of this study. Therefore, 20 percent of the total data is kept aside as test data. The data split is performed randomly. *CVTuner* is a custom tuner designed for the purpose of architecture optimisation. To instantiate the tuner, a choice between Bayesian optimisation, Random Search and Hyperband can be given. In case of Bayesian optimisation and random search, we also need to specify the value of max_trials. We have set this value as 150 for all the experiments, keeping the timing constraints for training a model and hyperparameter tuning in mind. In case of Hyperband, the process for finding the optimised architecture continues until in the end only one architecture remains. Therefore, we cannot specify max trials value in case of Hyperband. Instead, for a fair comparison, we stop the optimisation process for Hyperband once 150 trials are carried out by it. In addition, we also need to specify the model building function for instantiating the CVTuner. For the purpose of clarity, model building function is described in the next section in detail. Next, to initiate the tuning search process, we pass the training data and training labels to the tuner search process. This begins the search for the CNN optimised architecture keeping the choice of heuristic algorithm for optimisation as a baseline. For the purpose of searching promising architectures, we perform k fold cross validation with 5 folds over the training data and average out the validation loss over 5 folds. The method of early stopping is employed while training the model. The parameter 'p' is set at 20. This is called the patience parameter and is the number of epochs with no improvement after which the training stops. The value of p is problem dependant and the value of 20 is set after trying different values within the range 10-100. The batch size is also tuned in the CVTuner. The option to tune between the values 8, 16, 32 and 64 is given. Theses values are the most popular batch size values used while training the neural network.

C. Model building function description

The model building function uses an argument hp from which hyperparameters can be sampled. For hyperparameter tuning, the range of values is specified in this function. The range for all the hyperparameters is listed below.

1) Number of VGG blocks: The number of VGG blocks can range from a minimum of 1 to a maximum of 3. Since each VGG block comprises of 2 Conv2D layers, 2 Batch Normalisation layers and one MaxPool layer, a model with 3 VGG blocks is deep enough to give good performance on our datasets. Increasing the number of blocks to potentially take higher values affect the training time of the architectures, in turn increasing over all time to carry out 150 trials for hyperparameter tuning.
2) Filters: In Conv2D layer, filter values is a choice between 64, 128, 256 and 512. Therefore during hyperparameter tuning, only the specified filter values can be sampled from. We have chosen such these values as these are the most popular filter values used for training the CNN architecture.
3) Dense Layer Units: The dense layer units can take values in the range of 64 to 512, with a step size of 64. Although dense layer units can be any integer value, we restrict the range only to certain values keeping the training time of CNN networks and tuning them in mind.
4) Learning Rate: The model optimiser used is Adam, which is the most popular choice amongst scientists. The

TABLE I: RMSE of 5 experiments conducted over three datasets: Kaggle F1, F2 and UJIIndoorLoc using Bayesian Optimisation, Random Search and Hyperband

Dataset	Heuristic	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean RMSE	Standard deviation
F1	BO	2.17 m	2.21 m	1.89 m	2.17 m	2.15 m	2.118 m	0.12 m
	RS	2.35 m	2.51 m	2.58 m	2.31 m	2.56 m	2.46 m	0.12 m
	HP	2.02 m	2.22 m	2.17 m	1.97 m	2.24 m	2.12 m	0.12 m
F2	BO	1.89 m	1.81 m	2.08 m	2.02 m	2.11 m	1.98 m	0.12 m
	RS	2.27 m	1.78 m	1.76 m	1.99 m	2.21 m	2.00 m	0.23 m
	HP	2.22 m	2.11 m	2.07 m	1.69 m	2.29 m	2.07 m	0.23 m
UJIIndoorLoc	BO	3.59 m	3.06 m	3.12 m	3.57 m	3.34 m	3.33 m	0.24 m
	RS	3.19 m	4.01 m	4.44 m	3.78 m	3.05 m	3.69 m	0.57 m
	HP	3.67 m	2.93 m	3.37 m	3.80 m	3.36 m	3.42 m	0.33 m

TABLE II: 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 floor F1 RMSE	Kaggle floor F2 RMSE	UJIIndoorLoc RMSE
Decision Tree	3.35 m	3.61 m	5.13 m
kNN	1.80	1.82 m	4.22 m
VGG-16	22.41 m	17.29 m	27.96 m
CNN-Bayesian	1.89 m	1.81 m	3.06 m
CNN-Random Search	2.31 m	1.76 m	3.05 m
CNN-Hyperband	1.97 m	1.69 m	2.93 m

learning rate of Adam optimiser is a hyperparameter and can be tuned to take up 0.01, 0.001 and 0.0001.

5) Rate of dropout Layer: The dropout rate can be sampled from a minimum value of 0 to a maximum value of 0.5. The step size of dropout rate is 0.025. We have set the upper limit as 0.5 since a dropout value of more than that significantly affects the performance of the CNN architecture.

D. Environment

All the experiments are carried out on Google Colaboratory [49]. It is a product from Google allowing users to write python code especially for machine learning and deep learning. It gives free access to powerful hardware options such as CPU, GPU and TPU. The GPU is allocated to the users randomly. For most of our experiments, we were allocated the Tesla P100-PCIE-16GB GPU. A Jupyter notebook environment is provided for interactive development.

E. Libraries and Tools

For carrying out the experiments, we use Keras [50], TensorFlow [51] and scikit-learn [52] libraries. All the libraries are open source and are designed to carry out machine learning and neural networks operations. Some important functions and classes used for conducting experiments are listed below.

1) train test split(): The scikit-learn library provides this function which takes an entire dataset as input and returns a dataset split into two subsets: train and test.
2) StandardScalar(): This is also a scikit-learn function for standardising the inputs.
3) Earlystopping: Kera's early stopping class is used by specifying only few important arguments such as setting the quantity to be monitored as val loss, minimising it and setting the patience parameter p to 20.
4) Tuner: The base Tuner class is responsible for hyperparameter search process. Although built-in tuner classes are available for Bayesian optimisation, Random Search and Hyperband, we don't use any of those and create our own custom CVTuner. The Tuner class of KerasTuner [53] can be subclassed to customise our own tuning and hyperparameter search process. Advanced training techniques such as k-fold cross validation and early stopping is implemented for this study, and therefore is a part of our cutstom CVTuner.
5) tuner.search(): This process initiates the search process of the tuner for an optimised CNN architecture. Parameters such as training data, callbacks etc are given for initiating the process.

V. Experimental Results

In this section, we study the performance of our automated CNN architectures in detail and compare it to the state of the art VGG-16 architecture. Further, we compare the architectures with other popular ML algorithms for regression. We also interest the readers with the working of heuristic algorithms for a better 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: Automated Architectures

We conduct a total of five experiments using the Bayesian optimisation heuristic algorithm on Kaggle F1, F2 and UJIIndoorLoc dataset, to find the the best performing automated CNN architecture. We repeat the process with other heuristics algorithms i.e. with Hyperband and Random Search again for a total of five times on all the three datasets. The max_trial value for each of the experiments is set to

150. The main performance criteria is the RMSE of each of the automated CNN architecture. Lower value of RMSE indicates a better performing CNN architecture in comparison to an architecture with higher RMSE. This is presented in Table I. For all the three heuristics: Bayesian optimisation, Random Search and Hyperband, all the experiments results in good architectures and acceptable errors. As it can be seen from the Table I, Bayesian optimisation is conducted a total of five times, and the RMSE value of the best performing automated CNN architecture i.e the architecture with the least RMSE value found in 150 max_trials of each trial is retained. The process is also repeated for the other heuristics algorithms.

The mean performance of Bayesian optimisation over 5 experiments of Kaggle F1, F2 and UJIIndoorLoc dataset is 2.118 m , 1.98 m and 3.33 m while the standard deviation is 0.12 m, 0.12 m and 0.24 m. Similarly, the mean of Random Search experiments for all datasets is 2.46 m, 2.00 m and 3.69 m while the standard deviation is 0.12 m, 0.23 m and 0.57 m. In case of Hyperband, the mean is 2.24 m, 2.07 m and 3.42 m and the standard deviation is 0.12 m, 0.23 m and 0.33 m. As seen in the Table I, the performance of individual experiments in terms of RMSE is very close to each other. The mean RMSE value of Bayesian optimisation over 5 trials is the least in all the datasets by an extremely small margin. However, the standard deviation of Bayesian optimisation is also the least in all datasets. This indicates that the performance of Bayesian optimisation shows consistent results in coming up with an automated CNN architecture for all 5 trials. This is due to the fact that Bayesian optimisation takes past evaluations into account while tuning the hyperparameters. The CNN architectures generated via Random Search optimisation algorithm are also at par with Bayesian optimisation. However, since each architecture is tuned randomly and the history of the trials is not taken into account, we see that the standard deviation of Random Search is more than Bayesian optimisation. This gives a wider range of error values that next set of automated CNN architecture can possibly show. In case of Hyperband, we had to manually stop the optimisation process once the number of max_trials reached 150. Therefore, there is a possibility that the performance of Hyperband improves significantly if we let it search for hyperparameters further, however it comes at the cost of additional computation time and powerful hardware resources. In case of 150 trials, Hyperband has better overall average in comparison with Random Search in two datasets: Kaggle F1 and UJIIndoorLoc.

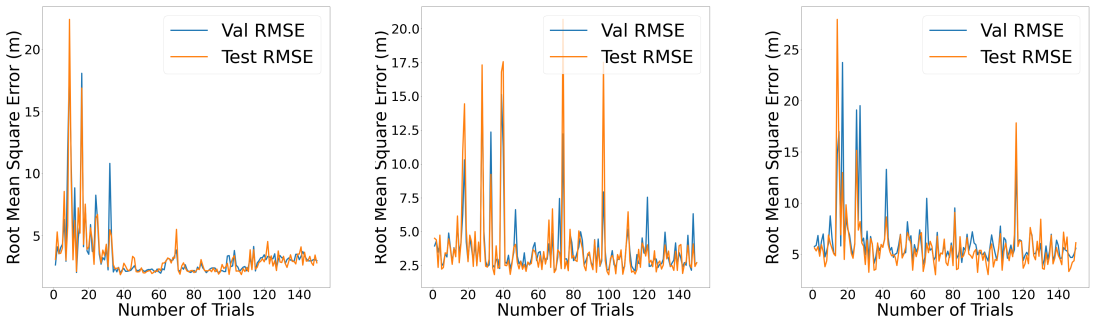
B. Comparison of Accuracy: Other Architectures

From Table I, which lists down the RMSE of the best performing CNN architecture for each experiment, we now select the best CNN architecture among them and compare the performance of these against decision tree regressor, k-nearest neighbors (kNN) algorithm and VGG-16 architecture, as summarized in Table II. kNN algorithm is a popular position estimation algorithm for WiFi indoor localisation. For a fair comparison, we perform hyperparameter tuning for decision tree regressor and kNN using grid search optimisation technique. We also performed k fold cross validation with 5 folds in case of VGG-16 architecture. However in case of VGG-16, unlike our other automated CNN architectures, we keep the architecture as it is and do not tune any hyperparameters except the batch size. As seen from Table II, all the automated architectures perform better than the decision tree regressor. The performance of our automated CNN architectures is similar to kNN in case of Kaggle datasets and slightly better in case of UJIIndoorLoc. Finally, all three heuristics performed significantly better than the standard VGG-16 architecture in all cases. In addition, the number of layers or the depth of the automated models are fewer than that of VGG-16, thereby making the training process of individual models faster. Therefore, all our automated architectures have shown acceptable performance.

C. Comparison of Heuristic Algorithms

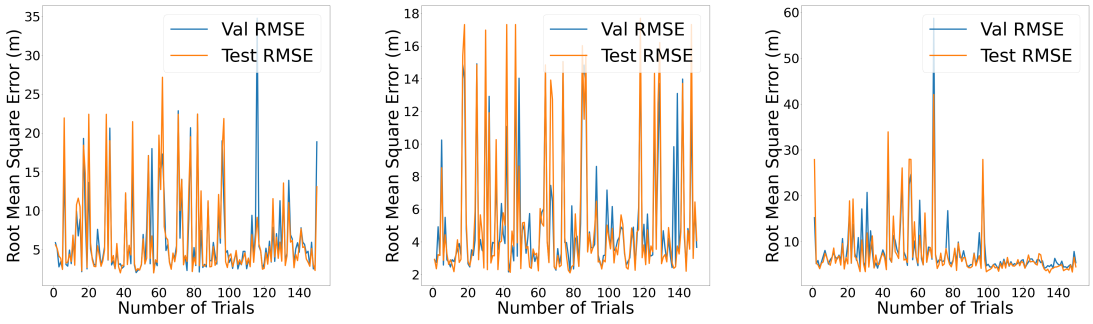
For each dataset: Kaggle F1, F2 and UJIIndoorLoc, we plot all the RMSE values returned by the heuristic algorithms during one experiment. As we perceive from Figures Fig. 6a, Fig. 6b and Fig. 6c, initially the Bayesian algorithm tries to find the best model for reaching the objective function and eventually the set of best hyperparameters over the model. Therefore, towards the end of each experiment, the all the trials have approximately similar performance. In contrast, the Hyperband as shown in Figure Fig. 7a, Fig. 7b and Fig. 7c and RandomSearch in Fig. 8a, Fig. 8b and Fig. 8c executes random configurations, therefore, throughout the experiment we see fluctuating performances of the architectures. Bayesian optimisation has the potential to come up with a good performing architecture even when the number of trials is less, as seen from the graphs. While this depends more on luck in case of Hyperband and Random Search. For example, Fig. 6a shows that several good performing architectures were discovered by trial number 50 in case of Kaggle F1 dataset, while this is cannot be said for certain for the other two heuristics due to their random nature, as shown in Fig. 7a and Fig. 8a

For an even better understanding of the performance returned by the algorithms, we plot the cumulative density function graph of all three heuristics for all three datasets as shown in Fig. 9, Fig. 10 and Fig. 11. As we notice, The RMSE for approximately all the experiments in Bayesian optimisation for Kaggle F1 is similar and shows good performance. 95 percent of the trails have RMSE less than 10 m. This is in contrast with Hyperband and Random Search where 90 percent of the performances include high value of RMSE values of more than 10 m. Similarly in case of Kaggle F2, 95 percent of the automated architectures generated by Bayesian optimisation have RMSE value less than 5 m. This is in contrast with Hyperband and Random Search, where only approximately 80 percent of the automated CNN architectures show good performance of less than 5 m. In case of UJIIndoorLoc, 90 percent of the RMSE values of automated architectures



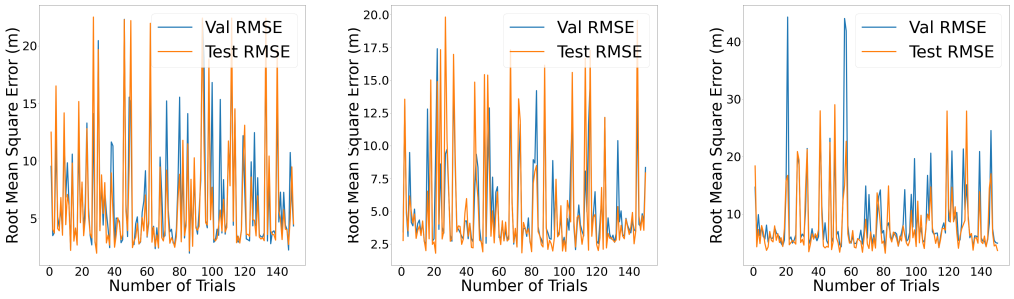
(a) (b) (c)

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



(a) (b) (c)

Fig. 7: Search for the hyperparameters via Hyperband optimisation Algorithm: (a) Floor F1, (b) Floor F2 and (c) UJIIndoorLoc



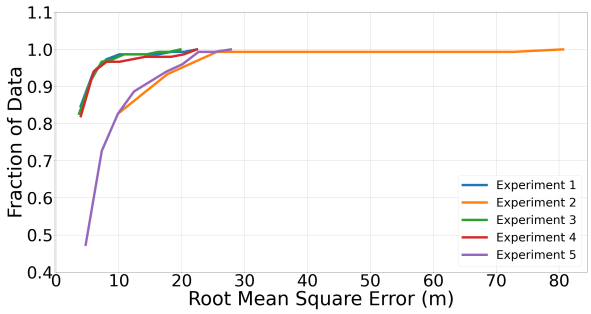
(a) (b) (c)

Fig. 8: Search for the hyperparameters via Random Search Algorithm:(a) Floor F1, (b) Floor F2 and (c) UJIIndoorLoc

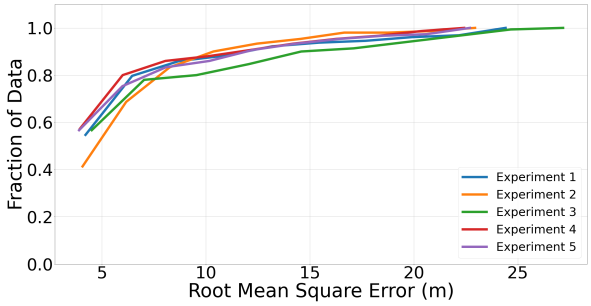
generated via Bayesian optimisation shows good performance in comparison to Hyperband and Random Search.

D. Comparison of the automated architectures

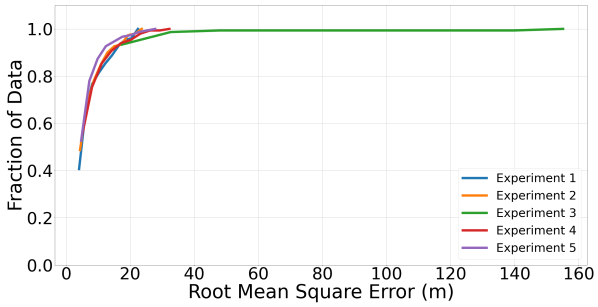
For a comparative study, we retrieve the best and poorest architectures of one of the selected experiment from each heuristic algorithms, for all the three datasets. The best model corresponds to the model with the lowest RMSE on the val data and the poorest model corresponds to the model that achieved the maximum RMSE value. For the Kaggle F1 dataset, the complete comparison of the architectures is shown in the Tables III, IV and V. In all the three heuristics cases, the difference in the best and poorest RMSE is approximately 20 m. While the best architectures have a good performance of approximately 2 m RMSE, the RMSE of the poorest architectures is approximately 22 m. Table III especially highlights the importance of hyperparameter tuning the CNN architecture. Both the architectures have the same batch size,



(a) Bayesian optimisation CDF graph

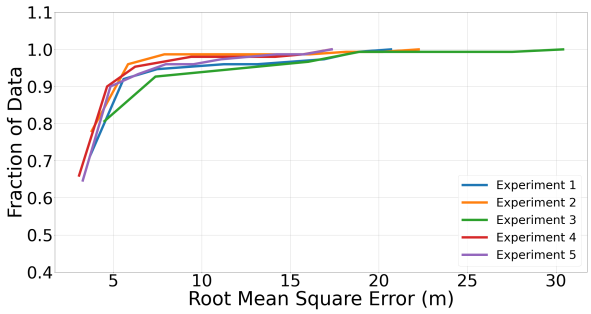


(b) Hyperband optimisation CDF graph

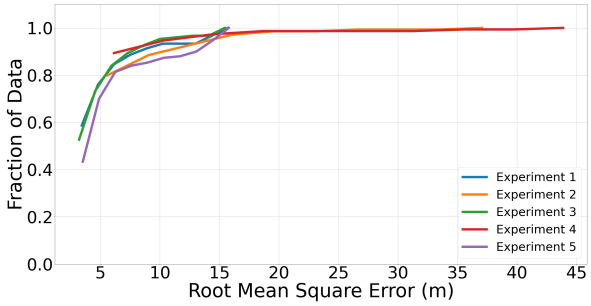


(c) Random Search CDF graph

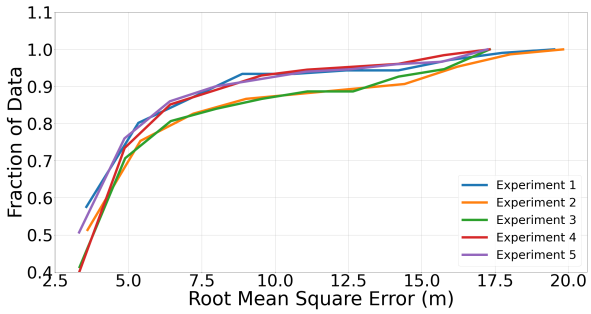
Fig. 9: CDF depicting the performance of all three heuristics for Kaggle Floor F1



(a) Bayesian optimisation CDF graph



(b) Hyperband optimisation CDF graph



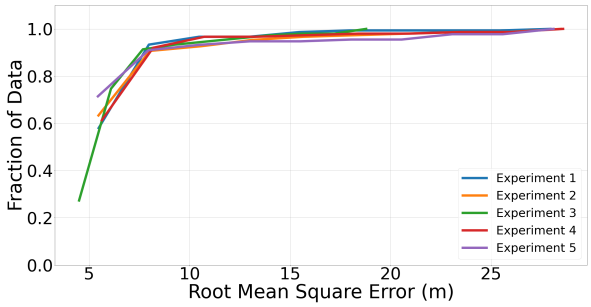
(c) Random Search CDF graph

Fig. 10: CDF depicting the performance of all three heuristics for Kaggle Floor F2

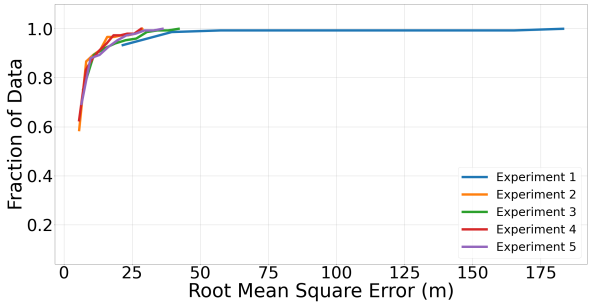
number of VGG blocks and the neurons present in the fully connected dense layers. The major difference between them lies in the neurons present across the layers in the VGG blocks, thereby resulting in completely different location estimation accuracies.

Similarly, for Kaggle F2 dataset, the complete comparison of the architecture is presented in tables VI, VII and VIII. Similar to the previous dataset, the RMSE of the best models is approximately 2m, while that of the poorest models ranges from 15 m to 45 m approximately. The best and the poorest architectures for the UJIIndoorLoc dataset is presented in tables IX, X and XI. In all the three cases, the RMSE of the best model is approximately 3 m and the poorest model is in the range 18 m - 22 m approximately.

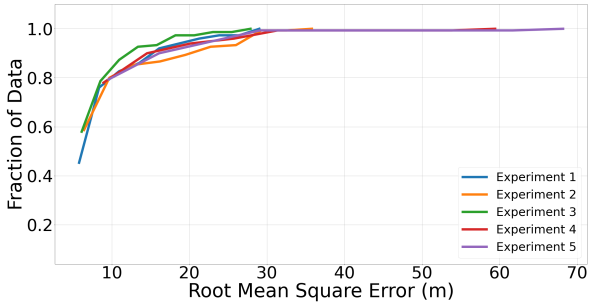
Comparing the automated CNN architectures with VGG-16, For all the heuristics, the best generated CNN architectures require only three VGG blocks; VGG-16, on the other hand, has five VGG blocks making it a deeper model to train. However, the location estimation accuracy of the generated CNN architectures is much higher than that of VGG-16. Additionally, it can be seen that the arrangement of neurons across the layers also does not follow any particular pattern, such as monotonically increasing, decreasing, or remaining constant. Therefore, it is challenging to craft the best CNN model for fingerprint localization unless one has an extensive knowledge about CNN and invests a lot of time and effort.



(a) Bayesian optimisation CDF graph



(b) Hyperband optimisation CDF graph



(c) Random Search CDF graph

Fig. 11: CDF depicting the performance of all three heuristics for UJIIndoorLoc

VI. Discussion and Conclusion

A. *Discussion - Answering the research questions*

The main research question asked at the beginning was how do we help solve indoor localisation problem using WiFi fingerprinting technique and CNN as position estimation algorithm, given that CNN is complex to design and tune for a novice?

To tackle the WiFi fingerprinting localisation challenge using CNN, we need to automate the CNN design process. This required creating an end to end process for automation, such that minimum manual intervention is needed. This would help a person who does not have any expertise in developing the CNN architecture. The goal to automate the process was further subdivided into following questions.

TABLE III: Comparison of the hyperparameters returned for the best and the poorest architecture for Kaggle F1: Bayesian optimisation

Architecture	**Best RMSE (1.89 m)**	**Poorest RMSE (22.41 m)**
Batch Size	16	16
Learning Rate	0.001	0.01
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-128	Conv2D-64
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-128
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-128	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-512	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 3	Conv2D-64	Conv2D-128
	BatchNorm	BatchNorm
	Conv2D-256	Conv2D-512
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-128	Dense-128
	Dropout-0.0	Dropout-0.25
	Dense-2	Dense-2

TABLE IV: Comparison of the hyperparameters returned for the best and the poorest architecture for Kaggle F1: Hyperband optimisation

Architecture	**Best RMSE (1.97 m)**	**Poorest RMSE (22.35 m)**
Batch Size	16	24
Learning Rate	0.001	0.01
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-64	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-64
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-512	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-256	Conv2D-512
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 3	Conv2D-256	Conv2D-512
	BatchNorm	BatchNorm
	Conv2D-128	Conv2D-64
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-320	Dense-128
	Dropout-0.0	Dropout-0.25
	Dense-2	Dense-2

1) How do we design a framework for automating the development of CNN architecture for WiFi fingerprint localisation?
   - This was achieved by using the Keras tuner class for model optimisation. The Keras Tuner class consisted of a model building function, which takes an

TABLE V: Comparison of the hyperparameters returned for the best and the poorest architecture for Kaggle F1: Random Search

Architecture	**Best RMSE (2.31 m)**	**Poorest RMSE (25.51 m)**
Batch Size	16	24
Learning Rate	0.001	0.01
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-512	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-512	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-256	Conv2D-64
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-64
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 3	Conv2D-256	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-128	Conv2D-128
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-448	Dense-448
	Dropout-0.0	Dropout-0.0
	Dense-2	Dense-2

TABLE VI: Comparison of the hyperparameters returned for the best and the poorest architecture for Kaggle F2: Bayesian optimisation

Architecture	**Best RMSE (1.81 m)**	**Poorest RMSE (22.24 m)**
Batch Size	8	8
Learning Rate	0.0001	0.01
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-64	Conv2D-512
	BatchNorm	BatchNorm
	Conv2D-512	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-64	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-512	MaxPool
	BatchNorm	Conv2D-512
	MaxPool	MaxPool
VGG Block 3	Conv2D-64	Conv2D-128
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-512	Dense-192
	Dropout-0.0	Dropout-0.0
	Dense-2	Dense-2

TABLE VII: Comparison of the hyperparameters returned for the best and the poorest architecture for Kaggle F2: Hyperband optimisation

Architecture	**Best RMSE (1.69 m)**	**Poorest RMSE (47.91 m)**
Batch Size	8	16
Learning Rate	0.001	0.01
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-512	Conv2D-64
	BatchNorm	BatchNorm
	Conv2D-256	Conv2D-512
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-256	Conv2D-64
	BatchNorm	BatchNorm
	Conv2D-128	MaxPool
	BatchNorm	Conv2D-64
	MaxPool	MaxPool
VGG Block 3	Conv2D-512	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-512
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-64	Dense-448
	Dropout-0.0	Dropout-0.0
	Dense-2	Dense-2

TABLE VIII: Comparison of the hyperparameters returned for the best and the poorest architecture for Kaggle F2: Random Search

Architecture	**Best RMSE (1.76 m)**	**Poorest RMSE (15.17 m)**
Batch Size	8	32
Learning Rate	0.001	0.001
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-64	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-128	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-64	Conv2D-512
	BatchNorm	BatchNorm
	Conv2D-64	MaxPool
	BatchNorm	Conv2D-64
	MaxPool	MaxPool
VGG Block 3	Conv2D-64	Conv2D-512
	BatchNorm	BatchNorm
	Conv2D-256	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-256	Dense-448
	Dropout-0.0	Dropout-0.25
	Dense-2	Dense-2

argument hp from which hyperparameters can be sampled from. It gives the programmers the freedom to design their own basic structure of the CNN architecture, as well as allows the them to specify the range of the hyperparameters to tune. Once the Keras Tuner is initiated, based on the underlying search algorithm for hyperparameter optimisation, different hyperparameter combinations would be tried out and tested. The search space can be explored and the best performing architecture can be retrieved. Therefore, it eliminates the need to have an expertise in CNN design process and working to come up with a good performing architecture.

2) How do we optimise such architectures using heuristic

TABLE IX: Comparison of the hyperparameters returned for the best and the poorest architecture for UJIIndoorLoc: Bayesian optimisation

Architecture	**Best RMSE (3.06 m)**	**Poorest RMSE (27.92 m)**
Batch Size	8	24
Learning Rate	0.0001	0.01
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-64	Conv2D-128
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-512
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-64	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 3	Conv2D-512	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-64
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-512	Dense-64
	Dropout-0.0	Dropout-0.25
	Dense-2	Dense-2

TABLE X: Comparison of the hyperparameters returned for the best and the poorest architecture for UJIIndoorLoc: Hyperband optimisation

Architecture	**Best RMSE (2.93 m)**	**Poorest RMSE (18.02 m)**
Batch Size	8	32
Learning Rate	0.001	0.001
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-64	Conv2D-128
	BatchNorm	BatchNorm
	Conv2D-128	Conv2D-128
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-256	Conv2D-128
	BatchNorm	BatchNorm
	Conv2D-64	MaxPool
	BatchNorm	Conv2D-512
	MaxPool	MaxPool
VGG Block 3	Conv2D-512	Conv2D-512
	BatchNorm	BatchNorm
	Conv2D-64	Conv2D-128
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-192	Dense-64
	Dropout-0.0	Dropout-0.25
	Dense-2	Dense-2

techniques such as Bayesian optimisation, Hyperband and RandomSearch?

- For optimisation using heuristic techniques, it is possible to use the built-in Bayesian optimisation, Hyperband and Random Search by Keras Tuner. However, for this work, we create our own custom Tuner called CVTuner for optimisation. This is because during the hyperparameter search process and coming up with an optimised architecture, we also incorporate early stopping and k-fold cross validation technique to prevent overfitting. We can therefore present our results with a stronger confidence. For Bayesian optimisation and Random Search, we specify the value of max_trials. This parameter takes in a number and creates as many different hyperparameter combinations as specified. Therefore, if the max_trial is 5, then 5 unique architectures will be created with different hyperparameter values. We can then retrieve the best performing architecture, i.e an architecture with the least RMSE on val and test data. In case of Hyperband, we chose to stop the process once certain number of trials were carried out.

TABLE XI: Comparison of the hyperparameters returned for the best and the poorest architecture for UJIIndoorLoc: Random Search

Architecture	**Best RMSE (3.78 m)**	**Poorest RMSE (27.92 m)**
Batch Size	16	16
Learning Rate	0.001	0.01
Number of VGG Blocks	3	3
VGG Block 1	Conv2D-128	Conv2D-64
	BatchNorm	BatchNorm
	Conv2D-128	Conv2D-256
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 2	Conv2D-256	Conv2D-256
	BatchNorm	BatchNorm
	Conv2D-128	Conv2D-512
	BatchNorm	BatchNorm
	MaxPool	MaxPool
VGG Block 3	Conv2D-512	Conv2D-512
	BatchNorm	BatchNorm
	Conv2D-256	Conv2D-512
	BatchNorm	BatchNorm
	MaxPool	MaxPool
Fully Connected Layers	Dense-64	Dense-256
	Dropout-0.0	Dropout-0.5
	Dense-2	Dense-2

3) Given the automated architectures, on what basis can we decide which heuristic technique performs the best?
   - We conducted five experiments in total using Bayesian optimisation, Random Search and Hyperband on three datasets. We retrieve the best performing architecture over five experiments for each of the heuristics, for each of the dataset. The average error of Bayesian optimisation over five experiments was the least for all the datasets. Additionally, the standard deviation of Bayesian optimisation was also the least for all datasets. The average error of Bayesian optimisation is not significantly different than the other two heuristics, however the low standard deviation gives the impression that Bayesian optimisation provides consistent results. Therefore,

for the 150 max_trials per experiment limitation we had, Bayesian optimisation heuristic technique performed the best due to consistent results in comparison with Hyperband and Random Search.

B. Conclusion

This study proposed a completely automated CNN architecture design based on VGG blocks for WiFi fingerprinting localisation. We base our study on publicly available datasets uploaded on Kaggle, as part of their Indoor Location and Navigation competition series and UJIIndoorLoc, a popular database for WiFi localisation. The WiFi localisation is carried out by cleaning the data, generating a WiFi radiomap and using the CNN as position estimation algorithm for location prediction. The CNN architectures are automatically designed by our proposed framework and, especially with our optimisation algorithms, yield a high localisation accuracy and outperform the standard VGG-16. This shows the need to develop a framework for automating the architectures as hand crafted or semi automated architectures might not perform well in all cases. The significant dependence of a model's performance on its architecture is also shown by drafting a comparison between the best and the worst architecture. This dependence proves that a slight modification in the CNN architecture would lead to a massive change in localisation accuracy. Additionally, we also study the working of the three heuristics for optimisation: Bayesian optimisation, Hyperband and Random Search. Bayesian optimisation showed consistent results due to the underlying principle of taking the past evaluations into account for hyperparameter search process.

C. Future Work

This existing work itself can be extended in several ways for different applications. There is a possibility to change the model building function by adding additional VGG blocks, in case a deeper architecture needs to be implemented. Also, the range of hyperparameters can be changed in such a way that more options for hyperparameter tuning is available. For example, in our experiments, we have given only four choices for filters in Conv2D layer: 64, 128, 256 and 512. It can be changed to take a minimum value of 32 and a maximum value of 512, with a step size of 16. Therefore, a wider range of hyperparameter is available for tuning. This of course, comes with an added complexity and therefore increases the overall training time for each architecture. Although the VGG blocks and the heuristics algorithms are used in the proposed framework, it is not necessary to have expertise in these while using the proposed framework. It is also straightforward to extend this framework with other networks such as the ResNet and the 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, Dec. 2019. Conference Name: IEEE Internet of Things Journal.

[3] E. Mok and G. Retscher, "Location determination using wifi fingerprinting versus wifi trilateration," *Journal of Location Based Services*, vol. 1, no. 2, pp. 145–159, 2007.

[4] 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.

[5] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[6] 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.

[7] M. Ibrahim, M. Torki, and M. ElNainay, "Cnn based indoor localization using rss time-series," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 01044–01049, 2018.

[8] 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. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

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

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

[11] Z. Liu, J. Wu, L. Fu, Y. Majeed, Y. Feng, R. Li, and Y. Cui, "Improved kiwifruit detection using pre-trained vgg16 with rgb and nir information fusion," *IEEE Access*, vol. 8, pp. 2327–2336, 2020.

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

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

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

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

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

2017.
[17] Y. Wang and X. Jia, "An indoor wireless positioning system based on wireless local area network infrastructure," 2003.
[18] M. Cypriani, F. Lassabe, P. Canalda, and F. Spies, "Open wireless positioning system: A wi-fi-based indoor positioning system," in *2009 IEEE 70th Vehicular Technology Conference Fall*, pp. 1–5, 2009.
[19] B. Jang and H. Kim, "Indoor positioning technologies without offline fingerprinting map: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 508–525, 2019.
[20] J. Niu, B. Wang, L. Cheng, and J. J. P. C. Rodrigues, "Wicloc: An indoor localization system based on wifi fingerprints and crowdsourcing," in *2015 IEEE International Conference on Communications (ICC)*, pp. 3008–3013, 2015.
[21] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2019.
[22] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," 2020.
[23] S. Bock and M. Weiß, "A proof of local convergence for the adam optimizer," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.
[24] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012.
[25] D. M. Hawkins, "The problem of overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, pp. 1–12, Jan 2004.
[26] Y. Bengio and Y. Grandvalet, "No unbiased estimator of the variance of k-fold cross-validation," *J. Mach. Learn. Res.*, vol. 5, p. 1089–1105, Dec. 2004.
[27] Y. Jung, "Multiple predicting k-fold cross-validation for model selection," *Journal of Nonparametric Statistics*, vol. 30, no. 1, pp. 197–215, 2018.
[28] L. Prechelt, *Early Stopping - But When?*, pp. 55–69. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.
[29] H. Allamy, "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)," 12 2014.
[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, Jan. 2014.
[31] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Efficient BackProp*, pp. 9–50. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998.
[32] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.
[33] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.
[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.
[35] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, p. 295–316, Nov 2020.
[36] P. Neary, "Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning," in *2018 IEEE International Conference on Cognitive Computing (ICCC)*, pp. 73–77, 2018.
[37] P. I. Frazier, "A tutorial on bayesian optimization," 2018.
[38] S. Zhang, X. Li, M. Zong, X. Zhu, and R. Wang, "Efficient knn classification with different numbers of nearest neighbors," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1774–1785, 2018.
[39] T. Evgeniou and M. Pontil, *Support Vector Machines: Theory and Applications*, pp. 249–257. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.
[40] 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.
[41] 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.
[42] A. H. Salamah, M. Tamazin, M. A. Sharkas, and M. Khedr, "An enhanced WiFi indoor localization system based on machine learning," in *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–8, Oct. 2016. ISSN: 2471-917X.
[43] 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.
[44] R. Ayyalasomayajula, A. Arun, C. Wu, S. Sharma, A. R. Sethi, D. Vasisht, and D. Bharadia, "Deep learning based wireless localization for indoor navigation," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, (London United Kingdom), pp. 1–14, ACM, Apr. 2020.
[45] 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.
[46] 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.
[47] 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.
[48] P. M. Radiuk, "Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for

Diverse Datasets," *Information Technology and Management Science*, vol. 20, Jan. 2017.
[49] E. Bisong, *Google Colaboratory*, pp. 59–64. Berkeley, CA: Apress, 2019.
[50] F. Chollet *et al.*, "Keras," 2015.
[51] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.
[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[53] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, "Keras tuner," 2019.