A Deep Learning approach for indoor user localization in Smart Environments

Fabrizio De Vita, Dario Bruneo
Department of Engineering,
University of Messina, Italy
Email: {fdevita,dbruneo}@unime.it

***Abstract*—Nowadays, smart environments are becoming an integral part of our everyday lives. Objects are becoming smarter and the number of applications where they are involved increases day by day. In such a context, indoor localization is a key aspect for the development of smart services which are strictly related to the user position inside an environment. In this paper, we present a deep learning approach to estimate the indoor user location starting from its Wi-Fi fingerprint composed by those signals perceived in the environment. We show some experimental results that demonstrate the feasibility of the proposed approach.**

***Index Terms*—Indoor localization; Machine Learning; Deep Learning; TensorFlow; Smart Environments**

I. Introduction

The advancement of the ICT technology and the increase of device computational power allowed the construction of a network of objects which are able to communicate and make decisions based upon some criteria. We refer to this phenomenon with the term Internet of Things (IoT) where common use objects (usually blended with the environment) are often equipped with a small intelligence and act as sensors and actuators [1]. In today's world, IoT finds one of its maximum expression with *smart environments*, where thousands of objects are connected together acting as bridges toward the physical world and exposing a set of services that can be accessed by any kind of device with the aim to improve the quality of our lives in many ways [2]. In such a context, Cloud Computing plays an important role providing storage and computational power services in order to manage and process all the data coming from objects [3]. In the last few years, the market related to this area has evolved and the economic potential of smart applications and IoT in general has strongly changed, so the number of scenarios where these technologies can be implemented increases day by day. In particular, the market related to location-aware services is expanding [4]. Besides mapping and navigation, the information relative to the position can be exploited in several ways, especially if it is used in combination with the already mentioned IoT and Cloud technologies, in order to develop location-aware smart services [5]. When talking about localization, we have to consider two possible scenarios: outdoor and indoor. In outdoor scenarios, the Global Positioning System (GPS) is today the de-facto standard; it uses a minimum of 24 artificial satellites for the data transmission reaching a precision in the order of a meter. However, GPS cannot be used in indoor environments since the signals used by this technology are heavily attenuated by the building walls, causing a considerable loss in terms of accuracy. On the contrary, indoor localization has no standard and continues to be an open research problem with some already available solutions which are very different in terms of technologies and adopted algorithms [5].

In this work, we are interested in estimating if a user is located inside a specific smart environment in order to allow her/him to interact with the corresponding smart objects. Then, our main requirement is to distinguish among different (even adjacent) locations (e.g., rooms) rather than understanding the actual position of an user inside that specific location. Moreover, we are interested in realizing a low-cost system that does not require any specific hardware and that can be used with a current smartphone.

To do that, we present an indoor localization system that is based on the concept of *Wi-Fi fingerprints*: the set of wireless signals perceived by an user in a specific location. We used a Deep Learning approach in order to design a Deep Neural Network (DNN) that takes as input a wireless fingerprint and generates as output the most probable location associated to it. Since this method relies on Wi-Fi signals distributed inside an indoor environment, the only hardware needed is a device capable to scan for Wi-Fi radio signals.

The paper contributions are the following: *i)* we designed an indoor localization system that, thanks to its supervised learning approach, is able to easily adapt to different environments; *ii)* we have realized an Android app that contains the trained DNN and that can be used without reverting to the Cloud; *iii)* we created a labeled Wi-Fi fingerprint dataset that can be further used to test different localization techniques; *iv)* we provided a quantitative performance analysis to test the accuracy of the system in a real environment.

The paper is organized as follows. Section II contains an analysis of the indoor localization state of the art. Section III provides a description of the Wi-Fi fingerprint dataset we built. Section IV presents our deep learning approach and the techniques adopted for the implementation. Section V describes the Android application we designed while Section VI shows the experimental results of the realized case study. Finally, Section VII concludes the paper and gives some details about future developments.

978-1-5386-4705-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SMARTCOMP.2018.00078

II. Indoor localization

The proliferation of wireless technologies together with mobile computing devices and IoT generated the interest in the development of location-aware systems. Location-based Services (LBS) are becoming a vital part of our life and represent the core of many applications which provide several services depending on the user location.

In our vision, the main goal of a smart environment is to support its guests by executing a series of tasks (which can be fully or partially autonomous); to do that, a smart environment must exhibit context-awareness in order to understand its current state and determine what types of action to execute. IoT is the key technology to make smart environments a reality, objects in this sense are fundamental for the data collection and processing. As already said, IoT builds a network that allows objects to communicate; communication modules, Cloud Computing and Machine Learning are the main components of what we call *Smart Ecosystems* that enable users to interact with smart objects scattered inside several environments in a fluid and transparent way [6], [7].

In such a context, the main problem is to determine the motion activity of the mobile client and use this information in order to locate the user in the environment. Due to the complex nature of an indoor environment, the definition of a localization technique is characterized by several challenges mostly caused by the presence of obstacles like walls, doors, human beings, and others which influence the propagation of electromagnetic waves. There are several methods that can be used to implement a location technique and in general it is possible to classify them into three categories: proximity detection, triangulation, and scene analysis.

A. Proximity Detection

It is one of the simplest methods to implement which determines the position of a mobile client by using the Cell of Origin technique [5]; it considers the location of the base station to be the location of the user and for this reason it is not very accurate. If more beacons detect the presence of the mobile client, it simply forwards the position information to the the beacon from which it receives the strongest signal. Nowadays this method continues to be used by several technologies like GSM, Bluetooth, and radio services.

B. Triangulation

Triangulation uses geometric properties in order to determine the location of the target [5]. It can be divided (depending on the geometric approach adopted) into two categories: *angulation* and *lateration*. Angulation techniques measure the angle of incidence of a radio-frequency signal and exploit this information in order to determine the position of an object in the space. Lateration methods are able to determine the position of an object by measuring the distance among several reference points; in this sense it is possible to evaluate it in several ways using techniques based on time or signal properties like the attenuation.

C. Scene Analysis

Most indoor localization approaches adopt "fingerprint" matching as a basic scheme for the location estimation; the method consists in collecting features of the scene (fingerprints) in each location belonging to the area of interest and building a fingerprint dataset [5]. This kind of techniques is divided into two phases:

- offline phase during which the fingerprints are collected by measuring or computing them analytically;
- online phase which allows to locate an object by matching the perceived fingerprint with the closest contained in the fingerprint dataset.

Even if different fingerprint can be exploited (bluetooth beacons, RFID tags, etc.), the most adopted technique is the one that uses Wi-Fi fingerprints. In fact, such a method does not require neither additional hardware (already installed Wi-Fi access points can be used) nor nodes synchronization and it can be implemented totally via software; this dramatically reduce the complexity and the cost of the system. The major drawback of the method is determined by the laborious and time consuming dataset construction; furthermore, if some access points are removed or new ones are added it could be necessary to update the dataset. System accuracy depends on the number of the access points as well as on the density of the data fingerprints. Moreover, it must be taken into account that the Wi-Fi signals in an indoor environment fluctuate over the time even if there are no environmental changes. In general, since the method works with radio signals, all the problems related to the reflection and diffraction which influence the signals propagation are the main challenges to deal with and usually this is done by frequently updating the dataset in order to obtain the widest possible fingerprint range.

Our solution belongs to this category since it relies on a fingerprint radio map based on Wi-Fi signals. In the literature, we found some papers that face this problem with different approaches. The authors in [8] provide an extensive dataset. However, to localize the user, they only use a basic machine learning technique based on K-Nearest Neighbor (KNN) that is insufficient to correctly map the user especially when the surrounding environment is highly mutable in terms of signals. The approach described in [9] examines 20 well known machine learning algorithms and focuses in particular on two of them: the K* algorithm which uses an entropy-based distance function and the RBF regressor which uses a radial basis function network. In this paper, authors trained their system to localize the user inside a single location split in several partitions. Differently, our goal is to build a system capable to localize the user in several locations where the Wi-Fi signals can have a very high variability. The authors in [10] propose a solution made by the combination of Principal Component Analysis (PCA) used to extract important information from a pre-defined radio map together with popular machine learning models such as Decision Trees, KNN, Random Forests, and Support Vector Machines. Using this technique the authors obtain good results, but similarly to the previous paper, they

are able to localize the user in a single location split in several part with a total number of six Wi-Fi access points. In our environment we have a larger number of access points so the use of the PCA by considering only the main principal components causes a higher loss in terms of information which can be a problem when the environment is too large and the user position is determined by considering the "contribute" of each access point perceived in a location. Finally, one of the main difference between our approach and the others is the use of deep learning instead of traditional machine learning algorithms. In such a context, where signals can have a very high variability, we believe that deep learning can be a valid solution to build an elastic model capable to understand the complex relationship between a Wi-Fi fingerprint and its corresponding location especially when the number of signals in the environment is very large.

III. Wi-Fi fingerprint dataset

In this section, we provide details on the Wi-Fi fingerprint dataset that has been created in order to design our machine learning algorithm for user localization. Since we planned to use a supervised machine learning approach, first of all it is necessary to build a dataset containing all the labeled data necessary for the model training. Even if some Wi-Fi fingerprint datasets are available on-line, we have chosen to build a new dataset based on wireless signals present in one of the floors of our University Department, for the following reasons: *i*) in this way, it is possible to understand the performance of the approach during real experiments on the field; *ii*) this solution will be then used to interact with objects scattered in the smart environments we have created as an open lab.

The data for the dataset construction were collected at the 7th floor of the Engineering Department of the University of Messina in the set of locations $\mathcal{L}$ composed of six different rooms and of the main corridor which connects them, as depicted in Fig. 1. We started building a dataset containing, as *data points*, the information about all the Wi-Fi signals perceived inside each environment by performing multiple Wi-Fi scans at different time instants in different locations $l \in \mathcal{L}$. Data received from Wi-Fi scans will be composed by the Service Set Identifier (SSID), the MAC address (MAC), and the Wi-Fi signal strength (RSSI) of the perceived Wi-Fi beacon frames.

In order to manage the issues related to those mobile devices announcing Wi-Fi hotspots, that can give rise to distorted location information, we used the SSID to filter received signals where substrings containing keywords associated to default hotspot names (e.g., *iphone*, *huawei*, *android*, etc.) are found. Of course, such a method does not allow us to filter custom mobile SSIDs but we found that it is enough to remove the majority of unwanted signals.

Let us define a *labeled Wi-Fi fingerprint* $w_t^{(l)}$ as the t-uple composed of the $(MAC, RSSI)$ couples received in the location $l \in \mathcal{L}$ at the timestamp t:

$$w_t^{(l)} =< (MAC_1, RSSI), ..., (MAC_q, RSSI) > . \quad (1)$$

In this context, we are using the timestamp t as a unique identifier for the set of all the collected labeled Wi-Fi fingerprints. In fact, even if multiple devices are used to collect labeled data in parallel, the probability to perform a scan in the same timestamp (expressed in milliseconds) is very low. Then, we indicate with $\mathcal{T}$ the ordered set of timestamps at which the Wi-Fi scans have been executed. In the following, we will refer to the subscript t to indicate the t-th labeled data.

Every MAC present in $w_t^{(l)}$ will correspond to a single *feature* of the Wi-Fi fingerprint dataset, while the corresponding RSSI will be the value of that feature in the t-th example.

The final set of features $\mathcal{F}$ is then obtained through the union of all the distinct MAC addresses found during the multiple scans:

$$\mathcal{F} = \bigcup_{t \in \mathcal{T}} \{MAC_x\} : (MAC_x, RSSI) \in w_t^{(l)}. \quad (2)$$

In this way, as long as we perform Wi-Fi scans, the dataset will increase both "horizontally" by adding new features (i.e., MAC addresses not perceived before) and "vertically" by adding new data points (i.e., new labeled fingerprints).

The machine learning algorithm that we are going to use, assumes that each data point is composed by exactly the same set of features. For this reason, we define a generalized labeled Wi-Fi fingerprint $\hat{w}_t^{(l)}$ as a labeled Wi-Fi fingerprint containing all the MAC addresses present in $\mathcal{F}$ and where for each couple $(MAX_x, -) \notin w_t^{(l)}$ the corresponding value of RSSI is set to the value -200. The idea to codify the absence of a specific MAC address during a Wi-Fi scan with the value -200 comes from the fact that the RSSI value usually lays in the range between 0 and -100 dBm (where values in proximity of 0 mean a very strong received signal). In this sense, the value -200 means a signal whose received power is very low (virtually absent) thus representing all those MAC addresses that are not perceived during the scan.

Finally, the set $\mathcal{W}$ of labeled Wi-Fi fingerprints corresponding to $\mathcal{L}$ is obtained by merging all the generalized labeled Wi-Fi fingerprints collected in $\mathcal{T}$:

$$\mathcal{W} = \{\hat{w}_t^{(l)}\}, t \in \mathcal{T}. \quad (3)$$

Table I shows a possible view of the dataset $\mathcal{W}$ where each row corresponds to the RSSI values of a generalized labeled fingerprint $\hat{w}_t^{(l)}$ for each value of $t \in \mathcal{T}$ while each column corresponds to a MAC address $MAC \in \mathcal{F}$. Moreover, the last column corresponds to the location $l \in \mathcal{L}$ where the generalized labeled fingerprint $\hat{w}_t^{(l)}$ has been taken.

MAC_1	MAC_2	MAC_3	...	$MAC_{\|\mathcal{F}\|}$	$\mathcal{L}$
$RSSI$	-200	$RSSI$	...	$RSSI$	$Room0$
...	...	...	...	...	...
-200	-200	-200	...	$RSSI$	$Room0$
-200	$RSSI$	$RSSI$	...	-200	$Room1$
...	...	...	...	...	...
-200	-200	$RSSI$	...	$RSSI$	$MainCorridor$

TABLE I
LABELED Wi-Fi FINGERPRINT DATASET STRUCTURE.

Fig. 1. The floor map with the locations that have been monitored.

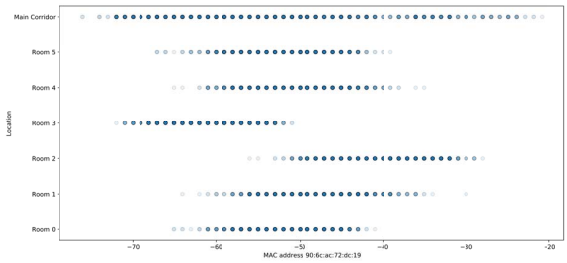


Fig. 2. Scatter plot showing the distribution of RSSI values related to a specific MAC address with respect to different locations.

Once we obtained a consistent number of records (approximately after performing scans in different hours for one week), we started a data analysis phase in order to better understand the nature of collected data. In particular, we made several scatter plots to understand how the RSSI values changed as the user moved between locations as shown. For example, in Fig. 2 we plot the RSSI values of a particular MAC address with respect to the different locations $l \in \mathcal{L}$ and it is possible to see how, in that case, RSSI values tend to distribute among different average values when the location changes, thus allowing us to exploit such a data variation in order to localize the user in the environment.

However, we also learned that the signal associated to some MAC addresses had a very high variation rate (e.g., they were present just for a limited period). For this reason, it has been necessary to apply a filtering process to the dataset in order to remove all the features related to those access points that were perceived only a few times during the scans, thus representing potentially dangerous outliers for the model learning. In order to automate the filtering process, we established to remove a MAC address MAC_i when:

$$n_l^{MAC_i} \leq n_l * \delta, \forall l \in \mathcal{L} \tag{4}$$

where:

- $n_l^{MAC_i}$: represents the number of labeled fingerprints associated to location l where the MAC address MAC_i has a value different from -200;
- n_l: represents the total number of labeled fingerprints associated to location l.
- δ is a threshold with $0 \leq \delta \leq 1$

Regarding the threshold δ, it represents the percentage of the data relative to a location that must be exceeded to maintain the MAC address in the dataset and in our case is fixed to 0.2 (thus representing the 20% of the total number of records relative to a generic location). Through the filtering process we were able to remove a total of 61 MAC addresses thus reducing the overall model complexity and, at the same time, removing noisy data not useful for learning. Finally, the obtained dataset structure is characterized by a number of features $\|\mathcal{F}\|$ equal to 93, a number of locations $\|\mathcal{L}\|$ equal to 7, and a number of generalized labeled fingerprints $\|\mathcal{T}\|$ equal to $11,524$.

At the end of preprocessing phase, we used the PCA technique to visualize the dataset in a 2-dimensional space by plotting the two principal components returned by the PCA algorithm and labeling the points with the corresponding locations, as shown in Fig. 3. Looking at the plot of Fig.

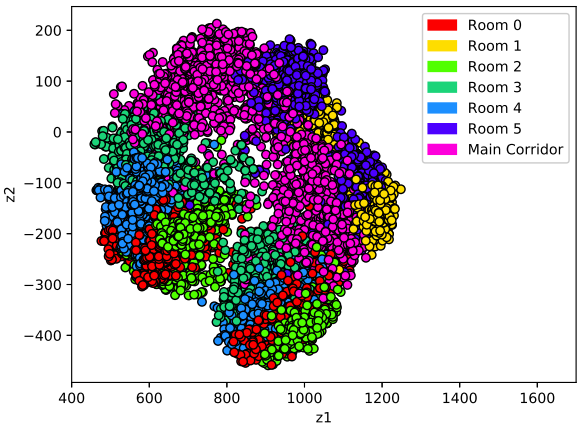


Fig. 3. Dataset visualization in a 2-dimensional space obtained by plotting the two principal components (z1,z2) resulting from applying the PCA technique.

3, it is possible to observe that, even if points related to different locations are not separated, all the points relative to a specific location are well concentrated in a restricted area with a negligible number of outliers thus making this dataset a good starting point for the machine learning model design.

IV. Deep Learning Approach

The problem to find the correct location of a user given a Wi-Fi fingerprint is a supervised multi-class classification problem. If the number of features which define the dataset is too large, especially when classic machine learning approaches are used, it is necessary to apply a huge and time consuming pre-processing phase in order to understand and select only those features that contain the information useful to build a good model capable to estimate the output. Moreover, the indoor localization problem highlights an issue related to the fact that environments could differ very much each other in terms of Wi-Fi signal distribution. Think for example to the difference in number of Wi-Fi access points that we could find in a University Campus with respect to a home location. For all these reasons, we decided to face this classification problem using a Deep Learning approach as a general purpose learning system.

Deep Learning [11] is a class of machine learning algorithms that uses several layers (composed by non-linear processing units) connected in cascade so that the input of a layer is given by the output of the previous one. This technique is able to discover intricate relationships between its internal parameters implementing the so called features extraction which consists in the process during which given a set of input features, it is possible to remove those values which are redundant or not informative easing the learning process and more in general the model complexity. Typical deep learning architectures are DNNs which find several applications such as computer vision, speech recognition, and natural language processing. A DNN can be considered as an artificial neural network with a high number of hidden layers between the input and the output; this kind of architectures allow DNNs to fit extremely complex non-linear relationships but, at the same time, make harder the training phase in terms of *i*) a higher number of required labeled data; *ii*) data overfitting; *iii*) computation time.

With respect to DDNs, the multi-class classification problem can be formulated [12] as finding the best parameter vector $\boldsymbol{\theta}$ for a parametric family of probability distributions $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ where $\mathbf{x} \in \mathbb{R}^n$ is a vector of features corresponding to an unlabeled Wi-Fi fingerprint with $n = \|\mathcal{F}\|$ and $\mathbf{y} \in [0,1]^p$ is a vector containing the different locations with $p = \|\mathcal{L}\|$ codified through one-hot-encoding (e.g., the vector $\mathbf{y} = [0,0,0,1,0,0,0]^T$ will correspond to $Room3$).

Starting from the dataset obtained as discussed in Section III, indicating with $m = \|\mathcal{T}\|$ the number of generalized labeled Wi-Fi fingerprints, we define the *design matrix* $\mathbf{X} \in \mathbb{R}^{m \times n}$ where each row contains the RSSI values of the corresponding fingerprint $\hat{w}_t^{(l)}$. Moreover, we defined a *label matrix* $\mathbf{Y} \in [0,1]^{m \times p}$ where each row corresponds to the one-hot-encoded location associated with the corresponding row in $\mathbf{X}$. Finally, we split the design and the label matrices in three parts (by randomly partitioning the set $\mathcal{T}$ in three subsets $\mathcal{T}_{train}$, $\mathcal{T}_{valid}$, and $\mathcal{T}_{test}$ representing, respectively, the 65%, the 15%, and the 20% of the entire dataset) thus obtaining the *training set*, the *validation set*, and the *test set* with the corresponding matrices $\mathbf{X}^{(train)}$,$\mathbf{Y}^{(train)}$,$\mathbf{X}^{(valid)}$,$\mathbf{Y}^{(valid)}$,$\mathbf{X}^{(test)}$,$\mathbf{Y}^{(test)}$.

Then, we defined a DNN composed by a feedforward fully-connected network with:

- A number of input units, corresponding to the number of features n, equal to 93.
- A number of output units, corresponding to the number of locations p, equal to 7.

Other DNN hyperparameters have been selected by running multiple tests to find the optimal configuration. In particular, in order to establish the number of hidden layers between the input and the output, we adopted a progressive training approach by adding each time a new hidden layer until we reached a satisfying result in terms of accuracy and training time, using the validation set. We found that a number of hidden layers equal to 6 is enough to obtain a good network topology which is capable to fit well the data and correctly predict the user position inside the environment. Moreover, we decided to use a number of neurons for the hidden layers equal to 80. The architecture of the DNN is shown in Fig. 4.

With respect to the activation function of each neuron, we decided to use the *Rectified Linear Unit* (ReLu) which resulted in a faster learning that required less training iterations if compared with other activation functions like the sigmoid. In order to correctly train the DNN, it is first of all necessary to define a cost function J which gives us the information relative to the DNN learning status. The adopted cost function is the *cross entropy* which fits well to classification problems especially if it is associated with the *softmax* function. Softmax

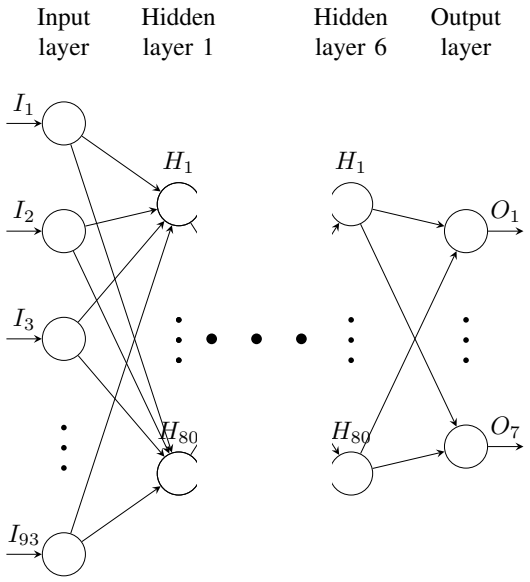


Fig. 4. DNN architecture.

is a function that given as input a vector $\mathbf{z} \in \mathbb{R}^p$, generates as output a vector $\tilde{\mathbf{y}} \in \mathbb{R}^p$ whose values add up to 1:

$$\tilde{y_j} = \textit{Softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{p} e^{z_k}}, j = 1, ..., p. \quad (5)$$

In our context, the vector $\tilde{\mathbf{y}}$ represents the output of the DNN where each element $\tilde{y_j}$ is the probability to be in location j given as input a fingerprint $\mathbf{x}$. Then, in order to define the cost function J we have to compare $\tilde{\mathbf{y}}$ with the correct output $\mathbf{y}$ using the cross entropy, thus maximizing the likelihood of the training data:

$$J = -\sum_{t \in \mathcal{T}_{train}} \mathbf{Y}_{t,*}^{(train)} \cdot \log(\tilde{\mathbf{y}}^{(t)}) \quad (6)$$

where the sum is extended to all the examples of the training set and $\tilde{\mathbf{y}}^{(t)}$ is the DNN output referred to the t-th training example.

The training goal is to minimize the cost function J by changing weights and biases of the DNN. To do this we use the adaptive moment estimation (also known as *Adam optimizer*) that maintains a different learning rate for each DNN weight and adapts it during the training phase. Finally, in order to prevent overfitting we adopted a regularization approach by adding a penalization term to all the neural network weights. Table II summarizes all the adopted DNN hyperparameters.

For the DNN implementation we used TensorFlow, an open source framework developed by Google (http://tensorflow.org) to solve a wide range of machine learning problems.

Fig. 5 shows how the cost function J decreases at each iterative step during the so called training epochs. The curve trend allows us to demonstrate that the DNN is learning in a correct way. This is also confirmed by the accuracy (obtained by feeding the DNN with the examples present in the test set $\mathbf{X}^{(test)}$ and comparing the results with the correct output $\mathbf{Y}^{(test)}$) that reached a value equal to $99,6\%$ representing a good result in terms of generalization.

# Hidden Layers	6
# Neurons per layer	80
Activation function	ReLu
Learning rate	0.001
Regularization term	0.01
Training epochs	25000

TABLE II
DNN HYPERPARAMETERS.

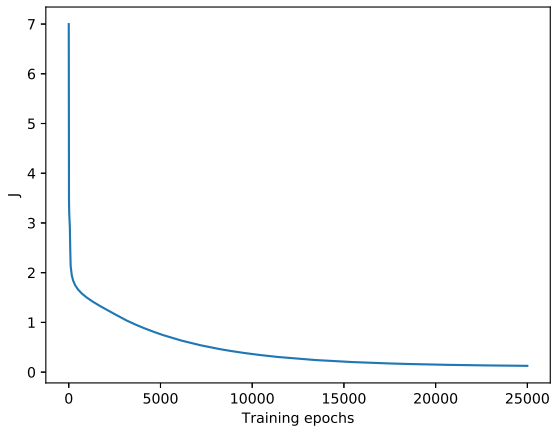


Fig. 5. The value of the cost function J with respect to the training epochs.

V. SYSTEM IMPLEMENTATION

To realize the system and test it in a real scenario, we realized two software sub-systems: *i)* a Python server, hosted in the Cloud and with TensorFlow installed, that has been used for the DNN training; *ii)* an Android app that has been used to create the dataset as well as to localize the users.

During the dataset creation phase, the Android app has been installed on several mobile devices in order to collect a large number of data through crowdsensing. The collected data is then sent to the Cloud server for the data processing phase and for the DNN training. Once the training phase is finished, the trained DNN is then sent back to the Android app in order to be used during the tracking phase. In fact, one of the features of the Tensorflow framework is the possibility to export the trained model and run it on a mobile phone; this can be done by using the Python front end which allows the user to store in a binary file the whole neural network description and then load into an Android environment.

By doing this, all the heavy computation necessary for the DNN training is done on the Cloud and only when the trained model is created it is sent to the app thus obtaining a faster application that runs locally on the smartphone without the need of an Internet connection for the user localization. In

```
int predict(float[] input){
     float[] result = new float[OUTPUT_SIZE];
     inferenceInterface.feed(INPUT_NODE,input,
         INPUT_SIZE);
     inferenceInterface.run(OUTPUT_NODES);
     inferenceInterface.fetch(OUTPUT_NODE,result);
     return argmax(result);
}
```

Listing 1. Prediction Method.

particular, our trained DNN occupies $148KB$ which is a fair good result if we think to its complex topology.

Once the trained DNN is loaded into the smartphone, thanks to the advanced APIs provided by Tensorflow, it is possible to retrieve the results using the method shown in Listing 1 where the `inferenceInterface` object acts like a communication bridge and interconnects the DNN with the Android application. The code consists of three methods:

- `feed()`: it takes as input 3 parameters: the name of the input layer, the input data, the input size. It allows to feed the DNN with new input data.
- `run()`: it takes as input only one parameter which consists in the name of the output layer, it runs the DNN computation and generates the output.
- `fetch()`: it takes as input the name of the output layer and a float array which will contain the DNN output. As already suggested by the name, it allows to fetch the results and store them.

From a structural point of view, the Android app is divided into two parts: *learning* and *tracking*. The learning part consists in collecting labeled Wi-Fi fingerprints through multiple Wi-Fi scans using the method `getScanResults()` provided by the Android class `WifiManager`. During this phase, the user is able to set the location where she/he is by pressing the area representing the room in the map as depicted in Fig. 6a. Moreover, during the "learn mode" the user can choose to send only one labeled wireless fingerprint or to send them continuously through a background process. Each time the scan returns a result all data together with the location label is stored inside a json file and sent to the server which acts like a data collector. With reference to Fig. 6b, when the *Track Me* button is pressed, the app starts the tracking phase and colors the area where the user is located in the environment. Like the previous case, the user can choose to scan for just one unlabeled fingerprint in order to be localized or to make a continuous scan to be automatically localized while she/he moves inside the environment. During this phase, the unlabeled data coming from Wi-Fi scans is fed to the trained DNN stored into the mobile phone to get the location prediction. Sometimes could happen to perceive MAC addresses that are not present in the design matrix. Since it is impossible to add new features to an already trained model, we apply a filtering process through which all the MAC addresses not present in the set of features $\mathcal{F}$ are automatically dropped.

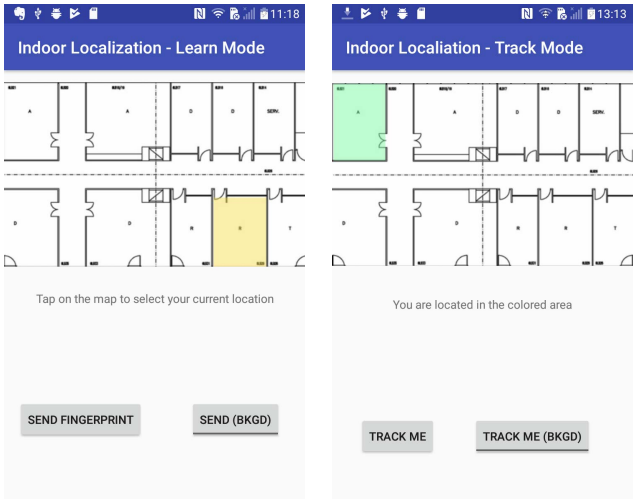
VI. Experimental Results

In this section, we present the experiments conducted to test the proposed indoor localization system. In particular, our goal is to test how the DNN performs in a real indoor environment by measuring the overall accuracy and system response time in predicting the user location. In fact, even if by querying the DNN using the data in the test set we reached an accuracy equal to 99.6%, data points belonging to the test set were acquired in a "static" way, i.e., by labeling fingerprints while users stayed in a specific location.

Now, we designed a "dynamic" experiment where a user is asked to follow a path and to explicitly notify (through a specific button in the Android app) the entrance in a specific location. In this way, we are able to trace the user location during the experiment time and to consider this information as a *ground truth*. At the same time, the Android app will predict the user location using the trained DNN. In particular, at a fixed time interval T, a Wi-Fi scan is performed and the corresponding unlabeled Wi-Fi fingerprint is passed to the DNN to predict the user location. For this experiment, we set the time interval T to 1 second.

We established a path to travel and fixed a residence time of 1 minute for each location (except for the $MainCorridor$) during which the user moves inside the room. With reference to Fig. 1, starting from $Room0$ the path we traveled is the following: $Room0 \rightarrow MainCorridor \rightarrow Room1 \rightarrow MainCorridor \rightarrow Room2 \rightarrow MainCorridor \rightarrow Room3 \rightarrow MainCorridor \rightarrow Room4 \rightarrow MainCorridor \rightarrow Room5 \rightarrow MainCorridor$. Fig. 7 shows the comparison between the ground truth and the DNN predictions over time. In the x-axis we have the experiment time (for an overall duration of 520 seconds) while the two colored bars represent the real and predicted user locations.

Generally, the obtained results are good, in fact the DNN is able to correctly predict the user location in most of the



(a) Learn Screen (b) Track Screen

Fig. 6. Android application screenshots.

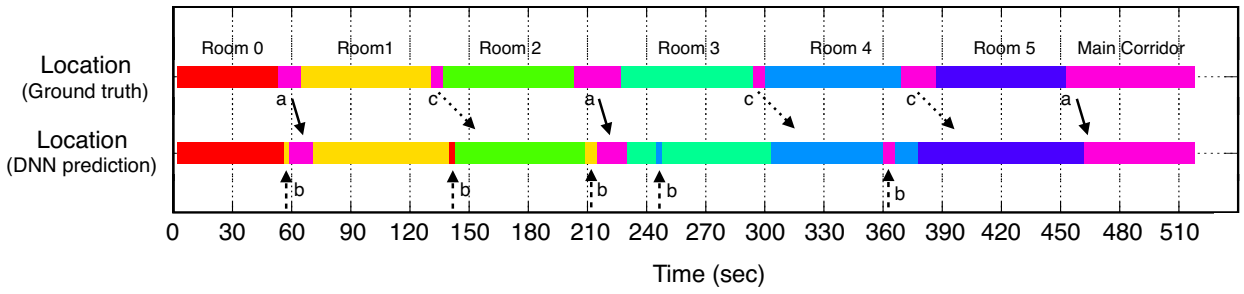


Fig. 7. DNN prediction of the user location during a specific path taken as ground truth.

cases. However, some errors are present, highlighted in Fig. 7 through arrows and grouped into three categories: a,b,c.

The first problem (category a) is related to the localization delay. Results show that the DNN is able to localize the user with an acceptable delay in the order of few seconds (as shown in Fig. 7 by the solid arrows labeled with a). This problem is mainly caused by the fact that the if in the ground truth trace the location is instantaneously updated by the user as she/he moves to a new location, in the prediction trace the time at which the DNN outputs the new location depends on the time interval T, on the time needed by the `WiFiManager` class to complete the Wi-Fi scan, and on the time needed by the DNN to perform the prediction.

The second typology of problem is related to the presence of some glitches in the DNN predictions (highlighted by dashed arrows labeled with b). These glitches happen especially when the user moves from a location to another and they are generated by the DNN itself which makes a wrong prediction associated to the given input fingerprint. However, such a behavior is acceptable since the DNN returns correct predictions for the most part of time the user remained inside a certain location.

Finally, dotted arrows labeled with c refer to the third problem that consists in the inability of the DNN to localize the user when she/he crosses the $MainCorridor$. This is due to the fact that, since the defined path has been traveled at a normal walking speed, when the user moves between two locations which are fairly near (e.g., with reference to Fig. 1 $Room1$ and $Room2$ or $Room3$ and $Room4$) the time necessary to cover the space which separates them is in the order of few seconds, so the DNN cannot catch such quick movements thus resulting unable to localize the user when she/he remains in the main corridor for a short amount of time. As already said, this happens only when the user moves between two near rooms, in fact as depicted in Fig. 7 in the last part of the experiment is possible to observe that the DNN is able to correctly localize the user in the $MainCorridor$ when the corresponding residence time is greater.

At the end of this experiment, the final DNN accuracy tested in a "dynamic" context was equal to 83.6% which is lower if compared with the result obtained during the "static" test (that was equal to 99.6%), nevertheless, it is a fair good result if we consider that all the above mentioned errors are negligible when applying this technique to a smart environment as an enabling technology to allow users to interact and access services provided by smart objects inside a specific location.

VII. Conclusions

In this paper, we presented a deep learning approach to tackle the problem of user localization in indoor smart environments. We created a dataset using the concept of Wi-Fi fingerprint and we designed a DNN to estimate the location of a user starting by the scan of the perceived Wi-Fi signals. Preliminary experimental results demonstrated the feasibility of the approach. Future works will be devoted to apply the concept of continuous learning to allow the DNN to adapt to environmental changes, to compare with other solutions, and to analyze the performance of the mobile application from the energy and bandwidth point of views.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1101–1102, 2012.

[3] D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito, "I/O-cloud: Adding an IoT Dimension to Cloud Infrastructures," *Computer*, vol. 51, no. 1, pp. 57–65, January 2018.

[4] Y. H. Wen, H. S. Chang, H. W. Kao, and G. H. Ju, "Location-aware services based on wi-fi network," in *The 16th Asia-Pacific Network Operations and Management Symposium*, Sept 2014, pp. 1–4.

[5] R. N. Zahid Farid and M. Ismail, "Recent advances in wireless indoor localization techniques and system," *Journal of Computer Networks and Communications*, vol. vol. 2013.

[6] A. K. Dey, G. D. Abowd, and D. Salber, "A context-based infrastructure for smart environments," in *Managing Interactions in Smart Environments*. London: Springer London, 2000, pp. 114–128.

[7] A. K. Bhattacharjee, D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito, "Extending bluetooth low energy pans to smart city scenarios," in *IEEE SMARTCOMP*, May 2017, pp. 1–6.

[8] Torres-Sospedra *et al.*, "Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems." in *IPIN*. IEEE, 2014, pp. 261–270.

[9] D. Mascharka and E. D. Manley, "Machine learning for indoor localization using mobile phone-based sensors," *CoRR*, vol. abs/1505.06125, 2015.

[10] A. Salamah, M. Tamazin, M. Sharkas, and M. Khedr, "An enhanced wifi indoor localization system based on machine learning," 10 2016.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.