Universidad de la República
Facultad de Ingeniería

Redes Neuronales Recurrentes Aplicadas a Sistemas de Localización Indoor en Redes WLAN

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Valentina Chumino, Richard Rodriguez, Andrés Stalker

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Federico La Rocca ..... Universidad de la República
Germán Capdehourat ..... Universidad de la República

Tribunal

Pablo Belzarena..... Universidad de la República
Pablo Cancela ..... Universidad de la República
Claudina Rattaro ..... Universidad de la República

Montevideo
martes 14 septiembre, 2021

Redes Neuronales Recurrentes Aplicadas a Sistemas de Localización Indoor en Redes WLAN, Valentina Chumino, Richard Rodriguez, Andrés Stalker.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 98 páginas.
Compilada el martes 14 septiembre, 2021.
http://iie.fing.edu.uy/

Locura es pretender hacer una y otra vez lo mismo, esperando obtener resultados diferentes.

Albert Einstein

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

El presente trabajo no podría haber sido concretado sin el apoyo y colaboración de ciertas personas e instituciones, a las cuales queremos expresarles nuestro debido agradecimiento.

Queremos agradecer a Federico y Germán, por la tutela realizada. A la Facultad de Ingeniería y a la Universidad de la República, por la formación que nos han dado, y por abrirnos las puertas a pesar de la situación sanitaria de nivel nacional y mundial por la que se cursó este año. A nuestras familias, parejas y amigos, por la compañía y paciencia.

A todos ustedes.

Gracias.

Esta página ha sido intencionalmente dejada en blanco.

A nuestras familias, que siempre nos han apoyado.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

En los últimos años se ha disparado la cantidad de aplicaciones que aprovechan la ubicación de dispositivos móviles con distintos fines, ya sean comerciales, educativos, o de entretenimiento. La fuente más utilizada para localizar los dispositivos es el GPS (Global Positioning System), pero el mismo presenta dificultades para conseguir buena precisión dentro de entornos cerrados. Este trabajo tiene como foco estudiar el problema de localización de dispositivos móviles en interiores, en una aplicación concreta.

Se continúa con lo presentado en el proyecto de grado "Localización Indoor Basada en Wi-Fi" (mayo 2019) en el que se utilizó un sistema basado en WiFi que proponía una solución efectiva para el problema en cuestión. El sistema contó con una aplicación para brindar una solución al Museo Nacional de Artes Visuales, proporcionando contenido audiovisual en función de la posición del usuario.

En este proyecto se estudian e implementan alternativas a la solución previamente mencionada, buscando mejorar la precisión y estabilidad del sistema de localización a través de la utilización de Redes Neuronales Recurrentes. Se hace foco especialmente en técnicas de aprendizaje automático que se nutran de la información temporal de la trayectoria del usuario. Por otro lado, se estudian técnicas de *crowdsourcing* y *crowdsensing* que alimenten al sistema con la información alternativa recopilada activa o pasivamente por los usuarios del sistema. En particular, se implementa una solución de *crowdsourcing* que se encuentra adecuada al contexto de trabajo.

Se realizan diversas evaluaciones dentro de un espacio cerrado disponible en Facultad de Ingeniería, UdelaR. Se elige trabajar en este lugar debido a que la situación sanitaria del país imposibilita trabajar en centros culturales como el Museo Nacional de Artes Visuales. Las evaluaciones realizadas nos llevaron a concluir que se puede alcanzar un error menor a 2 m en la localización del usuario, contando únicamente con la infraestructura WiFi ya existente en Facultad.

Además, se realizan estudios comparativos entre el desempeño del sistema diseñado en el proyecto anterior y las soluciones aquí presentadas. Bajo las mismas condiciones, se logró mejorar la precisión en un 30 %.

Esta página ha sido intencionalmente dejada en blanco.

Prefacio

El objeto de estudio de la presente tesis, propuesto por los Directores de Tesis, generó un gran interés en el grupo por el estudio de técnicas de aprendizaje automático que tan vasto uso se observa en la ingeniería de hoy en día para la resolución de problemas de todo tipo. Esto último, sumado a la cercanía de algunos integrantes del grupo a las redes inalámbricas, llevó a la decisión de retomar el trabajo realizado previamente por otros estudiantes de Facultad de Ingeniería para el Museo Nacional de Artes Visuales.

Lamentablemente, considerando la situación sanitaria del país presente en el transcurso de este proyecto, no fue posible trabajar en una solución para el Museo Nacional de Artes Visuales. En su lugar, se buscó un espacio disponible en Facultad de Ingeniería, UdelaR, para realizar los estudios necesarios para el desarrollo del sistema de localización y comparaciones de funcionamiento respecto al sistema original.

En este escenario se pierde en cierta medida el requerimiento del cliente, más allá de la localización en sí misma, sin intereses secundarios como los había previamente en el recorrido a ciegas de la muestra "Aquí Soñó Blanes" del Museo Nacional de Artes Visuales. De cualquier manera, se puede tomar este trabajo como base para realizar una aplicación de navegación en Facultad de Ingeniería, útil para aquellos estudiantes poco familiarizados con las instalaciones y que necesiten mayor orientación dentro de sus instalaciones.

Equipo Locindoor

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Agradecimientos III

Resumen VII

Prefacio IX

1. Introducción **1**
1.1. Motivación 1
1.2. Antecedentes 2
1.3. Planteamiento del Problema y Principales Resultados 3

2. Estado del Arte **7**
2.1. Técnicas de Posicionamiento en Interiores 7
2.1.1. Triangulación vs Huellas Digitales 8
2.2. FIND3 9
2.2.1. Componentes del Sistema 10
2.2.2. Algoritmos Utilizados por FIND3 11
2.3. Redes Neuronales Recurrentes como Algoritmo de Predicción . . . 13
2.3.1. Función de Coste 15
2.3.2. Long-Short Term Memory Units 16
2.3.3. DL-RNN 19
2.3.4. MISO, MIMO y P-MIMO 21
2.4. Crowdsourcing vs Crowdsensing 23
2.4.1. Fingerprint Crowdsourcing 23
2.5. Análisis de Desempeño 25
2.5.1. Clasificación 25
2.5.2. Regresión 26

3. Implementación del sistema **29**
3.1. Introducción 29
3.2. Arquitectura de los Servidores 30
3.3. Relevamiento de Huellas 33
3.4. Aplicación Android Para Toma de Huellas 35
3.5. Aplicación Android de Usuario 36
3.6. Comunicaciones Entre las Componentes 37
3.6.1. Relevamiento 37

3.6.2. Clasificación 38
3.6.3. Crowdsourcing 39
3.7. Procesamiento de los Datos 39
3.7.1. Generación de Trayectorias Virtuales 41
3.8. Implementación de algoritmos MIMO y P-MIMO 44
3.8.1. Hiperparámetros 45
3.8.2. Fase en Línea 46
3.9. Implementación del Algoritmo DL-RNN 46
3.9.1. Hiperparámetros 46
3.10. Elección de los Hiperparámetros 47

4. Evaluación del Sistema **49**
4.1. Rendimiento de los Modelos 49
4.2. Resultados en Función del Largo de las Trayectorias 51
4.3. Resultados en Función del Tamaño de las Zonas 53
4.4. Resultados en Función de la Cantidad de Trayectorias 55
4.5. Resultados en Función de la Cantidad de Huellas 56
4.6. Resultados en Función de la Cantidad de APs 57
4.7. Comparación del Sistema Locindoor con Posifi 59

5. Conclusiones y Trabajo a Futuro **65**
5.1. Análisis de Desempeño 65
5.2. Conclusiones Finales 65
5.3. Trabajo a Futuro 67
5.3.1. Sistema y Aplicación 67
5.3.2. Infraestructura Dedicada 67
5.3.3. Crowdsourcing 67

A. Implementación de códigos **69**
A.0.1. Procesamiento de Datos 69

B. Matriz de Confusión para 90 Zonas **73**

Referencias **75**

Índice de tablas **78**

Índice de figuras **80**

Capítulo 1

Introducción

1.1. Motivación

Actualmente, la gran mayoría de las personas contamos con un dispositivo móvil con el cual accedemos a Internet. Obtener con precisión las posiciones de estos dispositivos (usuarios) se puede aprovechar para la promoción de diversas aplicaciones comerciales, industriales, educativas o de salud. Centros comerciales, aeropuertos y campus universitarios pueden utilizarla para darle a sus clientes un sistema de navegación dentro de sus edificaciones, acompañado de diversas aplicaciones. Incluso se puede realizar analítica sobre el flujo de personas, sitios más concurridos, horarios de mayor actividad y perfiles de usuarios que pueden ser útiles a la hora de toma de decisiones en estas instituciones. Además de usuarios de aplicaciones móviles, se puede incluso extender su uso a la rama del Internet de las Cosas (IoT por su sigla en inglés) para realizar trazabilidad de los múltiples dispositivos que una empresa pueda tener conectados a su red.

La tecnología de GPS (Sistema de Posición Global) se ha convertido en una herramienta de posicionamiento muy utilizada en el día a día. Sin embargo, la misma no es adecuada para la localización en interiores, ya que obstáculos como edificios y paredes bloquean las señales.

Algunas de las estrategias comúnmente utilizadas para el posicionamiento en interiores se basan en las señales recibidas por el dispositivo, ya sean Wi-Fi, Bluetooth, tecnología celular 2/3/4/5G, campo magnético y hasta GPS. Se predice entonces la localización del usuario, utilizando métodos directos como triangulación o indirectos como la comparación con medidas tomadas previamente en el recinto de interés. Otras estrategias se basan en la utilización de los sensores internos del dispositivo, como el acelerómetro y el giroscopio, para calcular variaciones de distancia realizadas por el usuario a partir de una ubicación inicial conocida. Como no se pueden realizar cálculos precisos debido al patrón de movimiento variable entre distintos usuarios y escenarios (caminar, subir una escalera, utilizar un ascensor), muchas veces se utiliza como método complementario a los anteriores con el fin de mejorar la precisión de las estimaciones realizadas [24].

Considerando los frecuentes despliegues de redes WLAN (Wireless Area Net-

works) en muchos escenarios interiores de interés, parece casi natural pensar en aprovechar la señal recibida por el dispositivo de los distintos Access Points Wi-Fi (AP) y trabajar con los valores de RSSI (Received Signal Strenght Indicator) percibidos. En este sentido, se encuentran dos esquemas de trabajo entre los cuales optar: el escaneo activo y el escaneo pasivo de la señal.

El escaneo activo del RSSI refiere a que el dispositivo reporta al sistema las potencias de las señales de los APs para luego recibir su ubicación estimada. Este tipo de escaneo es el adecuado para resolver el problema sin tener control de la red Wi-Fi desplegada en el sitio. La alternativa es el esquema de escaneo pasivo del RSSI, en el que los APs miden la señal enviada por el dispositivo y la envían al sistema encargado de estimar la ubicación. Es adecuado si se tiene control de la red Wi-Fi desplegada, y deslinda al sistema de incluir una aplicación móvil para el dispositivo.

La utilización de métodos que comparan las medidas de señal para el usuario en una ubicación determinada, ya sea de manera activa o pasiva, son un buen acercamiento al problema pero presentan problemas de estabilidad y precisión. Las medidas para una misma ubicación pueden variar en el tiempo, y confundirse con ubicaciones cercanas. Esto se podría intentar corregir teniendo en cuenta la información que se tiene de considerar las medidas del usuario en instantes de tiempo anteriores al que se está intentando clasificar actualmente.

Dado que el usuario se mueve por una trayectoria continua y con una velocidad limitada dentro de un recinto, la ubicación actual del dispositivo está correlacionada con las ubicaciones anteriores. Por este motivo, la utilización de Redes Neuronales Recurrentes para aprender de las mediciones secuenciales de RSSI a lo largo de trayectorias determinadas, podrían llegar a mejorar la precisión de la localización del dispositivo y reducir los "rebotes" entre ubicaciones adyacentes que se perciben en predicciones que no consideran la información pasada.

1.2. Antecedentes

En mayo de 2019, Antonio Bracco, Federico Grunwald y Agustín Navcevich, presentaron en Facultad de Ingeniería, UdelaR, el proyecto de grado "Localización indoor basada en Wi-Fi". El trabajo propuso una solución al problema de localización de dispositivos móviles en interiores, utilizando medidas de RSSI del dispositivo referenciadas a distintos Access Points Wi-Fi en la zona determinada como de interés [11].

El proyecto antes mencionado parte del framework de FIND3 [5]. El mismo propone una arquitectura de servidores, cuenta con una aplicación Android denominada "Posifi APP Scanner" adaptada para la recolección de señales RSSI y otra aplicación Android denominada "MNAVegante" [12] para el usuario final. Esta última tomaba las medidas de RSSI del dispositivo, las enviaba al servidor y devolvía la ubicación estimada para luego mostrarle al usuario el contenido multimedia acorde a la misma.

La predicción de la ubicación se basó en un sistemas de huellas digitales, que es un enfoque para resolver el problema de localización indoor en redes WLAN

sin tener que modelar la propagación de la señal. Se utiliza el escaneo activo de la señal de RSSI en una serie de puntos dentro de la zona de interés. Mediante una etapa de aprendizaje intermedia, el sistema genera un mapa de huellas digitales asociadas a las distintas ubicaciones. Este mapa es luego utilizado para estimar la posición de las huellas que el usuario enviará en futuras instancias.

El problema entonces se transforma en uno de clasificación. Primero se realiza un mapeo del sitio, en donde se divide el área de interés en distintas zonas, cuyos centros serán los Puntos de Referencia (de aquí en adelante, PR). Se procede a tomar un vector de huellas de RSSI obtenidas de los distintos APs en cada uno de los PR. Luego de procesar y depurar estos datos, se busca comparar el vector enviado por el usuario con los del mapa de huellas e inferir mediante algún algoritmo de qué zona es la más probable que haya venido ese vector.

El sistema presentado implementó un meta-algoritmo de aprendizaje automático denominado AdaBoost (Adaptive Boosting) [25]. AdaBoost utiliza diversos sub-algoritmos (clasificadores) para aprender a tomar decisiones a partir de ellos y utiliza como estrategia ponderar a cada clasificador en función de qué tan buena fue su estimación, en un subconjunto de medidas de entrenamiento, evaluadas sobre un subconjunto de medidas de evaluación. Entre los algoritmos utilizados se encuentran algunas técnicas de aprendizaje automático supervisados, como KNN (K-Nearest Neighbour) [7] y redes neuronales como el perceptrón [26].

El sistema implementado alcanzó un 96,8 % de exactitud utilizando el índice de Youden [11]. Sin embargo, se observó que las predicciones tenían problemas de estabilidad, porque en ciertos momentos variaban con rapidez entre dos zonas contiguas. Es por esto que se tuvo que acompañar esta implementación con un módulo que compare la nueva predicción con las diez anteriores y decida si tomarla como válida o no.

Se consideran estos resultados con el fin de tomarlos como punto de partida para el presente proyecto, teniendo en cuenta que se trabajará en un nuevo escenario. Los mismos fueron conseguidos tomando 10.000 huellas en el relevamiento, distribuidas entre 16 zonas de tamaño variable, en una red de 15 APs (al menos 3 APs cada 50 m) en donde en cada zona se detectaron al menos 3 APs. Los APs no detectados se fijaron con un valor de -90 dBm por defecto. Únicamente se consideraron APs de ANTEL configurados con potencia fija.

También se consideró el sistema filtrando las MACs de los APs que operan en 5GHz (dejando únicamente aquellos que operan en 2.4 GHz, ya que es la que todos los dispositivos móviles del mercado cuentan) alcanzando una precisión del 92 %. Por último, se redujo a 10 APs llegando a una precisión de 94 % (incluyendo la banda de 5 GHz).

1.3. Planteamiento del Problema y Principales Resultados

El presente trabajo tiene como objetivo principal mejorar la precisión y estabilidad de localización de usuarios en interiores mediante una aplicación móvil

para Android, tomando como referencia el proyecto "Localización indoor basada en Wi-Fi", realizado en el período 2018/19. Para esto se evalúan e implementan distintas técnicas de aprendizaje automático con el fin de considerar simultáneamente la relación espacial y la relación secuencial temporal. Adicionalmente, se realiza un estudio de las técnicas disponibles de *crowdsourcing* y *croudsensing* que se podrían integrar al sistema. *Crowdsourcing* refiere a una contribución activa de los usuarios de la aplicación, en la que se aporta información necesaria para ampliar el mapa de huellas digitales, ya sea en cantidad de huellas por localización o en cantidad de puntos de referencia. En cambio, *crowdsensing* refiere a que esta contribución sea pasiva, con algoritmos que corran en segundo plano mientras la aplicación está siendo utilizada.

En este contexto, se decide mantener ciertas hipótesis respecto al trabajo de referencia que nos permitan enfocar en nuevos métodos de aprendizaje automático, investigando en particular la incorporación de la información temporal del usuario (trayectorias). Se mantiene el esquema de escaneo activo de los datos, es decir que el dispositivo del usuario es quien toma las señales recibidas. En particular, la elección de la señal RSSI como parámetro de entrada para la toma de decisiones sigue siendo la mejor alternativa en este contexto, dado que en Facultad de Ingeniería existen varias redes de Wi-Fi con cobertura en amplias zonas del edificio.

Este trabajo plantea el enfoque de huellas digitales (en inglés fingerprinting) con distintos métodos de aprendizaje automático, en particular redes RNN (Recurrent Neuronal Networks). De esta manera se pretende llegar a predicciones más consistentes con la movilidad de un usuario dentro del área de interés, buscando aprovechar únicamente la infraestructura existente de facultad (que tiene menor densidad de APs que el MNAV) y buscando minimizar los trabajos en campo para la fase de relevamiento.

Una red neuronal recurrente es una clase de red neuronal artificial, donde los resultados de salida dependen no solo del valor de entrada actual sino también de los datos históricos. Se utilizan en situaciones donde los datos tienen una correlación secuencial. Considerando que en localización de interiores el usuario se mueve por una trayectoria continua y con una velocidad limitada, la ubicación actual del mismo está efectivamente correlacionada con las ubicaciones anteriores. Se considera entonces que las redes RNN son una buena forma de tomar las mediciones secuenciales de RSSI a lo largo de una trayectoria, a fin de calcular la ubicación del dispositivo con una mejor la precisión que otros métodos.

En primer lugar se implementan tres de los modelos estudiados: MIMO (Multiple Input, Multiple Output), P-MIMO (una variación del MIMO que se detallará más adelante) y DL-RNN (Deterministic Linearized Recurren Network). La localización del dispositivo en estos métodos se expresa en términos de coordenadas cartesianas expresadas en metros, por lo que el error es medible de forma objetiva. Luego se procede a la recolección y tratamiento de los datos en campo, para poder realizar la evaluación de los modelos. Se estudian diversas variables que podrían afectar en el error del sistema, como son el largo de la trayectoria considerada, el tamaño de las zonas, la cantidad de trayectorias utilizadas para el entrenamiento de la redes, la cantidad de huellas recolectadas en cada zona y la cantidad de

Access Points visibles en en área de interés. El error más bajo obtenido fue de 1,6 m.

Seguido de esto, se implementó la solución utilizando el modelo que se encontró más adecuado para el caso de estudio y con mejor precisión. Para esto se definió una arquitectura del sistema, incluyendo la configuración de los servidores en la nube, conectividad e implementación de los algoritmos seleccionados. Se realizaron diversas adaptaciones para que el sistema almacene la trayectoria del usuario. También se implementó una interfaz web para cargar al servidor el contenido multimedia a mostrar al usuario en los servidores.

Por último, se adaptó la aplicación MNAVegante al contexto actual: nuevo cliente, nuevo contenido multimedia y nueva cadencia de envío y recepción de datos. Se realizó una actualización al sistema operativo Android 9 y posteriores para funcionar en dispositivos del mercado actual, y se agregó un módulo con técnicas de crowdsourcing que permitan al usuario contribuir con información útil para volver a entrenar las redes neuronales.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Estado del Arte

En este capítulo se detallan las técnicas, métodos y conceptos estudiados para la implementación realizada. En primer lugar se analizan las distintas técnicas de posicionamiento en interiores consideradas. Luego se explica el framework FIND3 utilizado y por último se realiza una introducción a las Redes Neuronales Recurrentes, en particular a los modelos específicos de interés para la resolución del problema de posicionamiento en interiores.

2.1. Técnicas de Posicionamiento en Interiores

Si bien el presente trabajo se enfocó en utilizar redes Wi-Fi para resolver la localización del usuario, existen diversas tecnologías que permiten atacar este problema, y pueden llegar a brindar información complementaria. En particular se detallan a continuación aquellas características en las que se encontró que no era necesario agregar infraestructura dedicada para el proyecto, sino que aprovechan los sensores del terminal y las condiciones del entorno [24] [37] .

- **Acelerómetro y giroscopio**

  Con el avance de la tecnología integrada a los dispositivos móviles personales, cada vez es más frecuente encontrar que los mismos incluyan uno o ambos sensores inerciales. El acelerómetro permite detectar la orientación del terminal (si el mismo está horizontal, vertical, boca arriba, boca abajo, etc.) y la velocidad a la que se mueve, con lo que podemos construir las trayectorias recorridas por el usuario, si sabemos su punto de partida. A esto se le puede complementar con la información que brinda el giroscopio, el cual permite detectar los giros del dispositivo.

  La desventaja de trabajar con estos sensores, es que los patrones de movimiento varían usuario a usuario (largo y cadencia del paso) y entre diferentes entornos (escaleras, ascensores, corredores), con lo que es difícil conseguir un algoritmo lo suficientemente robusto y versátil a la vez [31] .

- **Magnetómetro**

Otro sensor que hoy en día la mayoría de los terminales traen integrado, es el magnetómetro. Con él se pueden detectar las variaciones del campo magnético terrestre que son causados por las estructuras metálicas de los edificios. Si bien las medidas del campo magnético no son únicas para una posición del edificio (puntos alejados del área a trabajar pueden presentar el mismo valor de señal recibida), en áreas suficientemente reducidas, incluir este parámetro puede ayudar a desambiguar la predicción entre zonas que presentan una huella de RSSI similar. La desventaja mayor que presenta es la sensibilidad a interferencias causadas por mobiliario metálico, u otros factores que puedan llegar a modificar el entorno [35].

- **Cámaras fotográficas**

  Algunos sistemas utilizan reconocimiento de imágenes para comparar fotografías que el usuario pueda enviar de ciertos puntos de referencia, estimando una posible distancia a estos puntos. Esto podría ayudar a mejorar la precisión en la localización del dispositivo pero es más frecuente utilizar la información en conjunto con la predicción por Wi-Fi obtenida en ese instante, para luego corregir o mejorar los algoritmos de predicción principales [36].

2.1.1. Triangulación vs Huellas Digitales

Ya sea que se esté trabajando en un esquema de escaneo activo o pasivo de los valores de RSSI en una red WLAN (Wireless Local Area Network), para cada instante en que se realice la obtención de datos tendremos un conjunto de valores de señal: los vistos por cada AP dentro de la solución para la MAC del dispositivo o los que ve el dispositivo para cada AP del cual recibe señal. Este conjunto de medidas será enviada a un servidor que utilizará alguna estrategia para estimar la ubicación del usuario. Las dos más utilizadas se detallan a continuación [24]:

- **Triangulación de los datos**

  La estrategia aquí consiste en estimar la distancia entre el terminal y cada AP a partir del valor de RSSI percibido, para luego utilizar un algoritmo de trilateración que permita obtener una ubicación aproximada del mismo.

  Por más que se tengan en cuenta los complejos modelos de propagación disponibles, las estimaciones que se pueden obtener suelen ser muy variables. Aquí no solo influye el recorrido por el aire que realice la señal de Wi-Fi, sino que también se pueden ver afectados por rebotes y obstáculos, sobretodo en áreas interiores donde paredes, personas y mobiliario estarán presentes. No se encontró adecuado realizar una implementación de modelos de triangulación para el contexto de este trabajo, debido a que se entiende que sería muy complejo de realizar en recintos interiores.

- **Sistema de Huellas digitales**

  La segunda estrategia estudiada se basa en la idea de utilizar algoritmos que comparen las medidas obtenidas en cierto instante, con una base de datos que cuente con una gran cantidad de medidas realizadas previamente.

Esto agrega el trabajo de realizar una fase previa (que denominaremos "offline") de relevamiento donde se recogen las denominadas huellas digitales que formarán un "mapa" de ubicaciones y niveles de señal. El mapa debe ser discretizado en zonas acotadas para que el relevamiento sea factible, y se obtendrán una o varias huellas en el PR (Punto de Referencia) de cada zona.

Los algoritmos son entonces entrenados con las huellas obtenidas y testeados para obtener una precisión estimada de lo dispositivos. Luego, en el momento en que un usuario desea conocer su ubicación (que denominaremos fase "online"), se ejecutan estos algoritmos con la huella enviada por el mismo y se le devuelve la predicción. Cabe resaltar que en muchos casos la precisión suele mejorar a medida que más cantidad de huellas se obtienen para el relevamiento, etiquetándolas en zonas más reducidas, lo que en general implica un trabajo más arduo en la fase de recolección. La figura 2.1 muestra de manera gráfica el procedimiento antedicho.

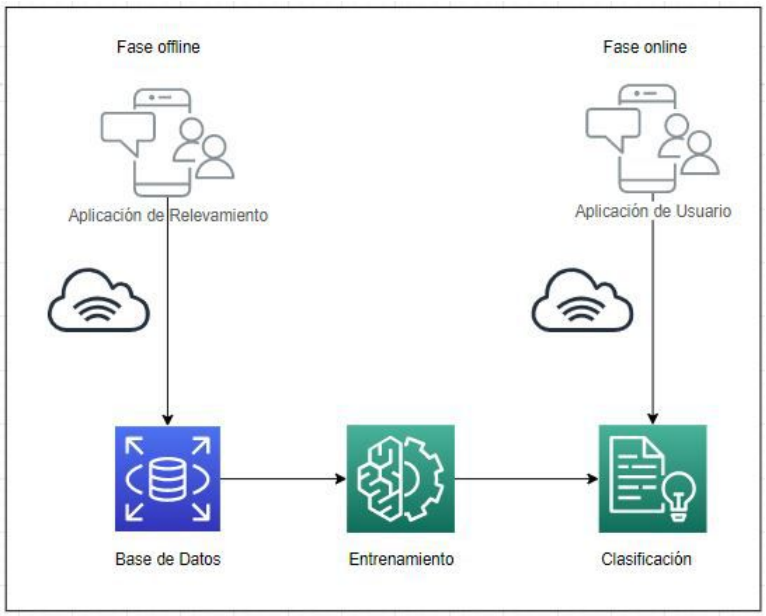


Figura 2.1: Diagrama del problema de huellas digitales.

2.2. FIND3

Como se indicó anteriormente, este trabajo pretende continuar lo realizado por el proyecto de grado "Localización indoor basada en Wi-Fi". El mismo adaptó un framework de código abierto denominado FIND3 (Framework for Internal Navigation and Discovery) [5], que se basa en un esquema de recolección activa de los datos de RSSI percibidos por un terminal Android con Wi-Fi y conexión a

Internet. Cuenta con un esquema de servidores y un meta algoritmo denominado AdaBoost que pondera las predicciones de varios algoritmos independientes entre ellos, para llegar a una localización aproximada del terminal en un instante dado.

2.2.1. Componentes del Sistema

Los componentes del sistema implementado son los siguientes:

- Servidor principal

  Es el servidor que se comunica con los terminales y los otros componentes, como la base de datos y el servidor de aprendizaje automático. Cuenta con conexión a Internet y una dirección IP pública.

  Se integra además una base de datos relacionales para guardar la información de los dispositivos, el historial de ubicaciones para cada uno de ellos y la información sobre cada zona. También cuenta con una API para la comunicación con la aplicación Android de los terminales.

- Servidor de aprendizaje automático

  Es el servidor que recibe las huellas enviadas por las aplicaciones Android, a través del servidor principal. En la fase offline, recibe y almacena los datos enviados y entrena los algoritmos de predicción de ubicación. En la fase online, habiendo guardado el estado de los algoritmos ya entrenados, los ejecuta para devolverle al servidor principal la ubicación estimada.

- Aplicación de Android para recolección de huellas
  FIND3 presenta una aplicación de Android que facilita la tarea del relevamiento de huellas y generación del mapa de RSSI. En esta aplicación se genera un identificador del dispositivo, y se envían al servidor principal cada cierto tiempo los valores de RSSI con una marca temporal.

- Página Web
  El sistema provee también una página web que está armada para mostrar las huellas obtenidas por el usuario y la precisión con la que se clasificó cada una de ellas utilizando el índice de Youden (definido en la sección 2.5.1).

Cabe resaltar que en este trabajo, si bien se analizaron otros posibles frameworks para realizar la implementación, como AnyPlace [2], se decidió continuar con lo implementado sobre FIND3, ya que el proyecto anterior realizó diversas modificaciones y agregados que resultaron de utilidad para nuestro equipo de trabajo. En particular, se realizaron adaptaciones a la aplicación Android de toma de huellas para acelerar el relevamiento permitiendo tomar una huella de RSSI cada pocos segundos. También se construyó una aplicación Android de usuario con una interfaz amigable y contenido adaptado para que el usuario reciba contenido multimedia diferente en cada zona predicha, y una página web que permite subir a la nube dicho contenido de manera amigable.

Si bien para implementar nuestro sistema hubo que actualizar y readaptar el software, además de volver a implementar el sistema de comunicaciones, partir con esta base redujo considerablemente el trabajo de desarrollo de software. Esto nos permitió hacer foco en los métodos de inteligencia artificial para la predicción de huellas, los cuales se detallan en la sección que sigue.

2.2.2. Algoritmos Utilizados por FIND3

Para abordar el problema de huellas digitales, se parte de un conjunto de medidas recolectadas previamente, y asignadas a una zona definida. Llamaremos ejemplo a cada una de estas medidas, un vector de n dimensiones (la señal recibida por n APs) que, apareado a su etiqueta correspondiente (en nuestro caso los identificadores de cada zona), los algoritmos utilizarán para obtener una hipótesis (modelo o función) mediante un modelo de aprendizaje. Esta hipótesis es a priori la que mejor se ajusta a los datos de entrenamiento y que permite generar la salida ante nuevos datos de entrada.

Para realizar esta tarea, FIND3 utiliza un meta algoritmo denominado AdaBoost [25] (abreviación de Adaptative Boosting), que parte del concepto que combinar los resultados de un grupo de algoritmos es mejor que elegir uno solo de ellos. La estrategia utilizada es variar el peso de cada uno de los ejemplos en la fase de entrenamiento. Los ejemplos que fueron clasificados correctamente en una iteración, reciben una ponderación más baja en la siguiente. Por otro lado, los ejemplos que fueron clasificados incorrectamente, reciben más importancia en la siguiente. De esta manera el entrenamiento se concentra en realizar mejores predicciones sobre los ejemplos que aún no han logrado ser clasificados de manera adecuada.

A continuación se pasan a detallar aquellos métodos de inteligencia artificial utilizados [32]:

- **SVM lineal**

  El método de vectores de soporte de máquinas es un algoritmo de aprendizaje supervisado. Es un clasificador discriminatorio, que dado el espacio que contiene los datos de entrenamiento etiquetados, genera un hiperplano óptimo que divide los ejemplos en dos clases. Los datos más cercanos al hiperplano que quedan de cada lado del mismo se utilizan como soporte de una banda entre los datos.

  Repitiendo este algoritmo, se llega a un espacio dividido por regiones disjuntas, en donde un nuevo dato se clasificará en la región que la contenga, o la que se encuentre a una menor distancia euclidiana del mismo.

- **Árboles de decisión**

  Los árboles de decisión son una técnica que se basa en la idea de un diagrama de flujo, en la que vamos realizando distintas preguntas que van acotando la cantidad de soluciones posibles al problema. En la fase de entrenamiento, se busca la característica que permita imponer una condición que divida al

árbol en dos ramas de la mejor manera posible. Se define a ese atributo como un nodo de decisión, y se procede a iterar las veces que sea necesario en cada rama, hasta que el subconjunto generado tenga un único elemento o hasta que todos los ejemplos del subconjunto tengan la misma etiqueta.

El problema aparece en decidir cuál es el largo óptimo de árbol (cuántos nodos consecutivos se buscará generar), y lo que usualmente se hace es calcular el costo de dividir el conjunto de ejemplos y tratar de reducir este costo.

- **Bosques aleatorios**

  Los bosques aleatorios son uno de los algoritmos de inteligencia artificial más utilizados. Es un método de conjuntos basado en el concepto de "divide y vencerás". Un bosque está compuesto de árboles, que mientras más arboles tenga, más robusto será el algoritmo. Los árboles se crean mediante conjuntos de datos tomados al azar, se obtiene predicciones de cada árbol y se elige la mejor predicción mediante votación.

  En un problema de clasificación, cada árbol vota y se asigna la etiqueta más popular. El algoritmo construye un árbol de decisión para cada conjunto de muestras y obtiene un resultado de predicción de cada uno. Se realiza una votación para cada resultado previsto y se selecciona el resultado con más votos como predicción final. Una gran cualidad de este algoritmo es que saca a la luz la importancia de cada característica a la hora de tomar decisiones.

- **K-Nearest Neighbours (KNN)**

  Es otra técnica de inteligencia artificial que, como su nombre lo indica (k vecinos más cercanos), busca clasificar la muestra enviada buscando encontrar los k ejemplos más cercanos [7]. Si cada ejemplo x_i es un vector de n valores en un espacio de características, se utiliza la distancia euclideana n-dimensional entre la muestra a clasificar y y cada uno de los ejemplos almacenados en la fase de entrenamiento, y se asigna a esta nueva muestra la etiqueta más frecuente dentro de los k (hiperparámetro configurable) ejemplos más cercanos.

  En este método, todos los atributos cuentan con el mismo peso a la hora de decidir qué etiqueta asignarle a la muestra recibida, corriendo el riesgo que alguno de ellos traiga información irrelevante y contribuya a una decisión errónea. Es por eso que es importante el tratamiento previo de los datos, eliminando en este caso señales de AP desconocidos o muy lejanos.

- **Multi Layer Perceptron (MLP)**

  El perceptrón es una red neuronal de las más sencillas. Una neurona es una unidad de cómputo que tiene conexiones de entrada a partir de los que recibe valores que utilizará para realizar una suma ponderada de ellos, agregando un sesgo o variable de entrada fija y se le aplica una función no lineal. Estos pesos son los parámetros del modelo, y son los que se buscará optimizar en la fase de entrenamiento para que los resultados de esta suma sean lo más parecidos a la salida esperada según los ejemplos utilizados. En otras

palabras, se busca reducir la función de coste (el error en la predicción de la neurona) mediante el método del gradiente descendiente para resolver problemas linealmente separables.

Posteriormente, se desarrolló el MLP, o perceptrón multicapa, que está formado por capas ocultas intermedias entre la entrada y salida, de tal manera que permite resolver problemas que no son linealmente separables. Cada capa oculta tendrá un número de neuronas que recibirán los mismos datos de la capa previa, lo que permite generar conocimiento jerarquizado, y especializar neuronas dentro de una capa para analizar cierto conjunto de características.

Al valor de salida de cada capa se la distorsiona a través de una función de activación no lineal, en general una función sigmoide, de tal forma que la red neuronal no pueda simplificarse al procesamiento de una sola neurona por ser una composición de funciones lineales. El entrenamiento de la red suele realizarse utilizando *backpropagation*, o propagación hacia atrás, en donde se reparte el error de cada neurona entre las neuronas de la capa anterior en función de los pesos de entrada de la primera.

Existen otros algoritmos que FIND3 tiene previstos para considerar dentro del AdaBoost, pero en el proyecto de referencia se descartaron por ocupar demasiada capacidad de procesamiento en la nube sin aportar mejoras sensibles en la precisión del sistema global. Cabe resaltar además, que ninguno de estos algoritmos tiene por sí mismo presente el histórico de muestras enviados por el usuario, sino que clasifica cada muestra para un instante dado.

2.3. Redes Neuronales Recurrentes como Algoritmo de Predicción

En esta sección se presentarán las Redes Neuronales Recurrentes (RNN) como algoritmos de regresión o clasificación, que permitan resolver el problema de huellas digitales utilizando las medidas que el usuario va enviando a medida que realiza una trayectoria [4].

Las redes RNN son una de las principales arquitecturas del Aprendizaje Profundo y las primeras clases de redes neuronales diseñadas para tratar datos secuenciales en el tiempo. Estas redes tienen memoria, es decir, consideran tanto entradas como salidas de tiempos anteriores para el tratamiento de los datos actuales.

Son fuertemente utilizadas en problemas en que la relación secuencial entre datos es tanto o más importante que los datos de entrada para ese instante a la hora de obtener una salida precisa. Un ejemplo de esto es la aplicación al reconocimiento de voz, en la que una secuencia de datos representa una palabra dentro de un cierto alfabeto, y no interesa reconocer cada sonido por separado sino estudiarlos como un conjunto ordenado para un determinado lapso de tiempo.

Estas redes usan el concepto de recurrencia: para generar la salida l^t en un cierto instante de tiempo t, a través de la función de activación a^t, no solo se

utiliza la entrada x^t para ese instante, sino también la activación definida para el instante anterior:

$$l^t = h(K3 \times a^{t-1} + c)$$

en donde

$$a^t = g(K1 \times a^{t-1} + K2 + x^t + b)$$

y $K1$, $K2$, $K3$, b y c son coeficientes constantes en el tiempo de ejecución (los parámetros que la red deberá aprender) y g,h las funciones de activación de las neuronas. En la figura 2.2 se puede observar una esquematización de estas operaciones.

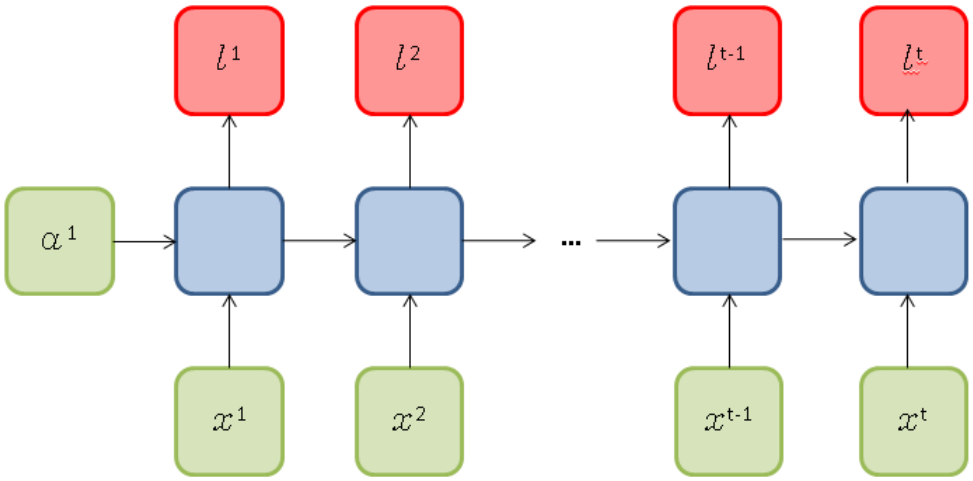


Figura 2.2: Arquitectura de una RNN genérica.

Los coeficientes se ajustan en la fase de entrenamiento. Se define la cantidad de épocas (epochs), que son las veces que la red iterará para ajustar los coeficientes antedichos y un tamaño de lote (batch size), que es la cantidad de muestras que se tomarán en cada iteración del tiempo de ejecución. También se define el optimizador a utilizar, que es el método o algoritmo que permite ajustar los coeficientes. El método de propagación hacia atrás (backpropagation) es de los más populares: va midiendo las derivadas parciales del error con respecto a los pesos y los va ajustando gradualmente con el fin de minimizar el mismo.

Como se puede observar, la dependencia temporal es únicamente con la salida del instante anterior, que a su vez tiene dependencia con el instante previo, y así una cantidad T de veces. Esto genera un compromiso entre la estabilidad y la cantidad de memoria, ya que la información de varios pasos en el pasado se comienza a perder a medida que bajamos el peso de $K1$. Existen actualmente generalizaciones de las RNN que tienen memoria a más largo plazo, como las redes Long-Short Term Memory Units (LSTM) o las Gated Recurrent Units (GRU).

2.3.1. Función de Coste

Existen distintas funciones de coste (o pérdida) que se pueden utilizar para minimizar los parámetros antedichos. Cuál de ellas utilizar queda a discresión de quien implementa la red, en función de qué interesa resaltar. A continuación se describen algunas de las más utilizadas para las RNN trabajando en regresión:

El **error cuadrático medio** (RMSE), se puede calcular como:

$$Loss = \frac{\sum_{i=1}^{N} \|\hat{l}_i - l_i\|_2}{N}$$

donde $\hat{l}_i$ es la predicción para el paso i-ésimo, l_i la ubicación real de dicho paso y N la cantidad de ejemplos de testing. En otras palabras, es el promedio de las distancias euclídeas entre las salidas de la red, una medida de la diferencia entre lo predicho y lo conocido.

El **error absoluto medio** (MAE), mide el promedio del valor absoluto del error a lo largo de las N muestras, y se puede calcular como:

$$Loss = \frac{\sum_{i=1}^{N} |\hat{l}_i - l_i|}{N}$$

Esta medida de error es mucho menos sensible a errores altos para muestras particulares, ya que errores mayores a 1 no se verán elevados al cuadrado como en la función de coste anterior.

El **error log-cosh** (MAE), mide el logaritmo del coseno hiperbólico del error a lo largo de las N muestras, y se puede calcular como:

$$Loss = \frac{\sum_{i=1}^{N} log(cosh(\hat{l}_i - l_i))}{N}$$

Esta última se aproxima mucho al MSE para errores bajos (menores a uno) y al valor MAE parara errores altos.

Es de tener en cuenta que, debido a la tendencia exponencial (creciente o decreciente) que puede tener la influencia de los datos pasados en las predicciones actuales, los datos de salida en una RNN podrían terminar divergiendo (o convergiendo a cero), presentándose así el problema conocido como gradiente explosivo (o desvaneciente).

Otro fenómeno a considerar es el sobre entrenamiento (overfitting), que sucede cuando la red aprende a predecir muy bien los datos del conjunto de entrenamiento, pero pierde generalidad y no logra predecir nuevas muestras con diferencias razonables al mismo. Uno de los motivos puede ser que se corran demasiadas epochs en el sistema. En ese sentido, se puede realizar un algoritmo que permita correr suficientes epochs como para reducir la función de coste, pero que detenga el entrenamiento (early stopping) cuando se detecte que nuevas muestras empiezan a tener un error creciente. Para eso, se define un conjunto de validación (para el entrenamiento de la red y el análisis de la función de coste) y otro conjunto más pequeño de testing (para ir evaluando el error al predecir nuevas muestras).

También puede ocurrir el sobre entrenamiento si los datos de entrenamiento comparten demasiada información entre ellos. Aquí también sucederá que la red aprenderá a predecir sobre datos particulares, perdiendo la generalidad necesaria para clasificar nuevos datos. Una de las estrategias utilizadas para atacar este problema, es el aumento de datos de entrada mediante la adición de ruido en alguna de la dimensiones que puede no ser del todo representativa.

2.3.2. Long-Short Term Memory Units

Las redes LSTM (Long-Short Term Memory Units) son un tipo de red neuronal recurrente que permiten entrenar con éxito arquitecturas muy grandes, evitando el problema anteriormente mencionado de la divergencia del gradiente mediante un mecanismo que decide qué información se va a almacenar y cuál va a ser eliminada, obteniendo una memoria más grande a largo plazo de los datos pasados. Estas redes fueron introducidas por Hochreiter Schmidhuber en 1997 [27], y se fueron refinando y popularizando a lo largo de los años, siendo todavía una de las redes más utilizadas.

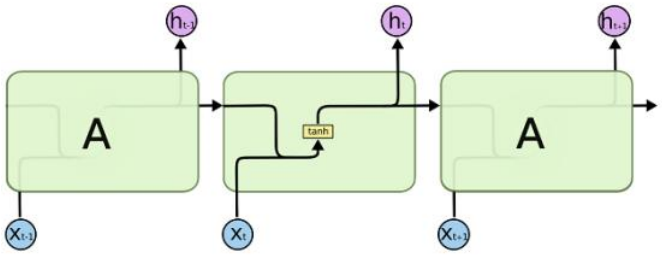


Figura 2.3: Arquitectura de una RNN [20].

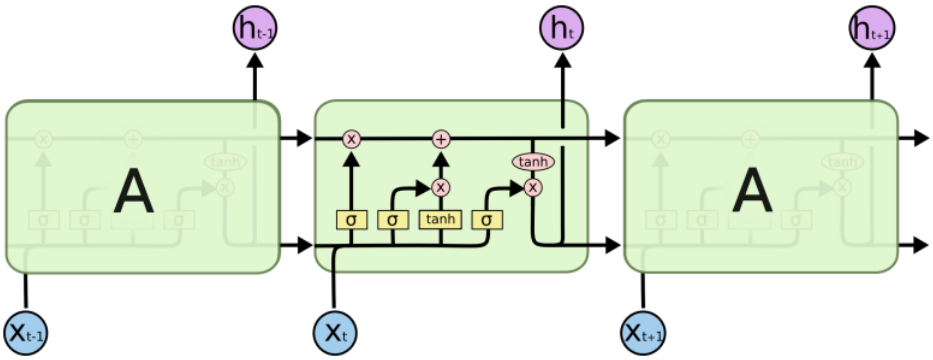


Figura 2.4: Arquitectura de una LSTM.

En la figura 2.4 podemos ver la estructura de una LSTM. En comparación con la RNN ilustrada en 2.3 vemos que la red LSTM contiende cuatro puertas

(o capas) que controlan cómo fluye la información dentro de la unidad. Estas puertas son llamadas, puerta del olvido, puerta de entrada, puerta del candidato y puerta de salida. En el esquema cada línea representa por donde se transporta un vector de información desde la salida de un nodo hasta la entrada de otro y las operaciones puntuales dentro de la unidad son representadas por las figuras rosadas. Las variables de entrada a la capa son X_t, que representa el valor de la secuencia en el instante t y h_{t-1}, que es la salida de la secuencia de datos en el instante anterior. En la siguiente sección se detalla paso a paso el funcionamiento de estas redes [22] [28].

Funcionamiento de una Red LSTM

La clave de las redes LSTM es el estado de la celda, la línea horizontal que recorre la parte superior del diagrama (figura 2.6), C_{t-1} indica el estado de la celda en el paso anterior. La información fluye a lo largo de la línea, con solo algunas interacciones lineales menores. Esta línea se encarga del estado de la memoria a largo plazo de la red. Como se mencionó, cada celda LSTM tiene la capacidad de agregar o eliminar información, esto lo hace utilizando las puertas de la siguiente manera:

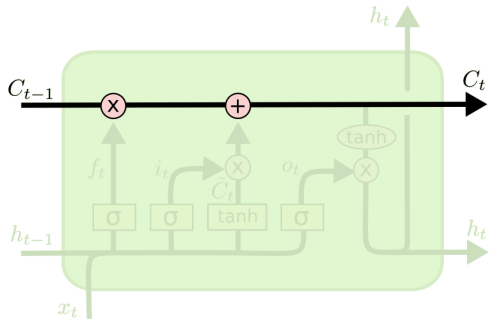
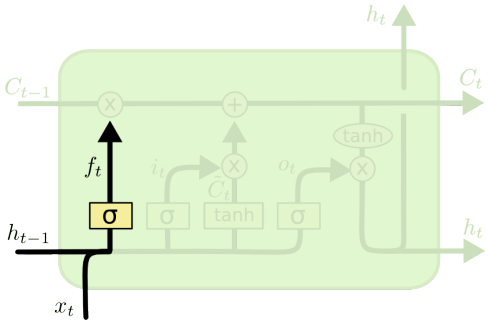


Figura 2.5: Estado de la celda.

El primer lugar está compuesto por la "puerta de olvido" que es la que se encarga de decidir qué información va a ser eliminada del estado actual de la celda, discriminando qué información es más o menos relevante. Está compuesta por una función sigmoide (σ - "forget gate layer"), este tipo de función transforma valores de entrada al intervalo $(0, 1)$.

En la figura 2.6, se representa con f_t a la salida de la función sigmoide obtenida cuando su entrada es la concatenación de X_t (entrada en el instante t) y h_{t-1} (salida de la secuencia de datos del paso anterior).

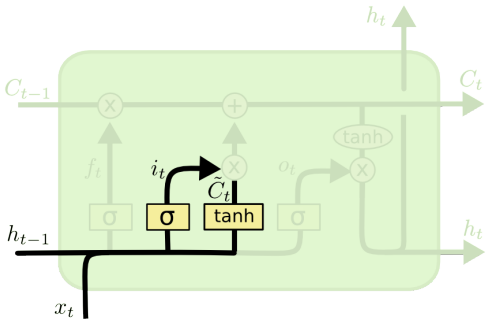
El segundo paso es decidir qué información nueva se va a almacenar en el estado de memoria, esto se desglosa en dos partes, por un lado tenemos la puerta llamada "puerta de entrada" formada por una función sigmoide (σ - "input gate layer") que decide qué valores se actualizarán y en qué medida (ponderando entre 0 y 1 cada uno). Luego, una capa del tipo *tanh* que crea un vector de valores posibles a ser agregados al estado actual, representado en la figura 2.7 como $\widetilde{C}_t$.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

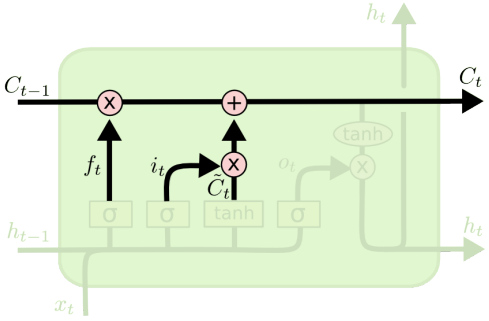
Figura 2.6: Primer paso - "forget gate layer".

Ambas funciones se multiplican, y el resultado se suma para obtener la salida de esta etapa.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 2.7: Segundo paso LSTM.



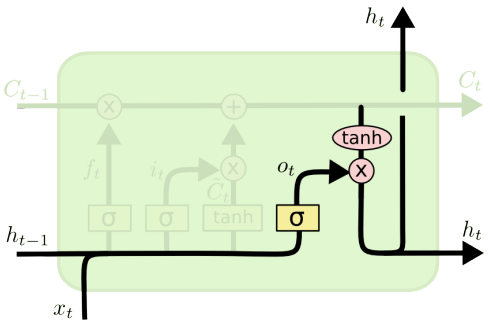
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 2.8: Tercer paso LSTM - Actualización de estado.

En este momento la red ya puede actualizar el estado de la memoria cambiando C_{t-1} por el nuevo estado $\widetilde{C}_t$, esto se hace multiplicando el estado anterior por f_t, olvidando una cierta cantidad de información en función de f_t. A continuación se suma i_t a la memoria, actualizando la misma con nuevos valores. Este paso se representa en la figura 2.8.

Por último, se debe construir la función de decisión de salida. Para esto, una cuarta capa, también del tipo sigmoide, tomará la función de decisión anterior

y decidirá qué valores de estado se propagarán a la siguiente etapa, y en qué medida. La salida de esta capa se multiplica por el estado de la salida filtrado por una función *tanh*, para que sus valores queden entre -1 y 1, y el resultado será la función de decisión a pasarle a la siguiente etapa, como se ve en la figura 2.9.



$$o_t = \sigma\left(W_o\ [\,h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 2.9: Cuarto paso LSTM - salida.

De esta manera, se consigue que el estado de salida en cada etapa considere los estados anteriores con dependencias a largo plazo (cientos o miles de pasos), evitando que la suma de sus derivadas se vaya a cero, lo cual se consigue con la implementación de la capa "forget gate", que le permite regular qué información pasar de estado a estado.

En este apartado se utilizaron imágenes extraídas de [3], dado que son muy representativas.

2.3.3. DL-RNN

Una posible de aplicación de estas redes al problema de huellas digitales en redes Wi-Fi, la DL-RNN (Deterministic Linearized Recurrent Neuronal Network), propone encadenar dos redes RNN para mejorar su precisión, basándose en la idea de aprender no solamente del histórico de medidas enviadas por el usuario, sino del histórico de predicciones realizadas por la propia red [33]. En este sentido, utilizaríamos una primera RNN (de mapeo) que entrene normalmente a partir del conjunto de ejemplos relevados en la fase offline, obteniendo una predicción gruesa de la ubicación del usuario. Luego se usarían las salidas intermedias de cada etapa oculta como entrada en las etapas de una segunda RNN (de filtrado), que en base a lo aprendido por la red llegaría a una predicción más precisa.

La estructura de esta red consiste en una RNN (Location matching RNN) que mapea una secuencia de medidas $\hat{a} - (\hat{f}_1, ..., \hat{f}_s)$ en una estimación aproximada de secuencia de ubicación $\hat{b} - (\hat{l}_1, ..., \hat{l}_s)$, y otra RNN (Location filtering RNN) que mapea la salida de la RNN anterior en una estimación precisa de ubicación.

Estas dos RNN, del tipo LSTM, se conectan en cascada y son entrenadas con la misma función de pérdida como se muestra en la figura 2.10.

- **Location matching RNN:** Como se mencionó, la red utilizada es la LSTM, donde r_t denota las señales RSSI de entrada para el tiempo t. La unidad de

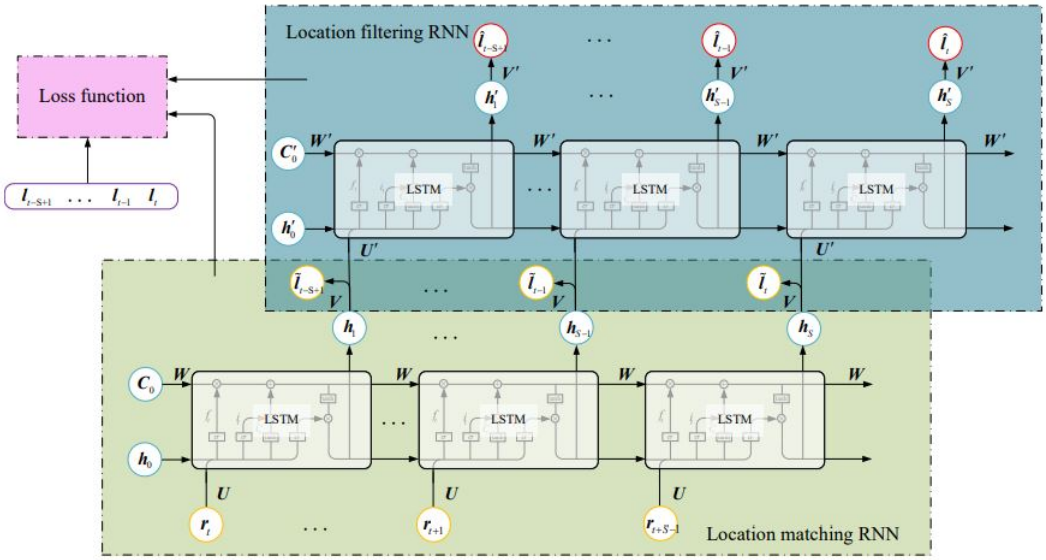


Figura 2.10: Arquitectura DL-RNN.

capa oculta h_t y la estimación de la posición aproximada de salida $\tilde{l}_t$ se actualizan para cada tiempo.

En esta configuración, la red aprende una relación aproximada de mapeo, y luego las características aprendidas por la red se usarán para un filtrado de posición más preciso, con el fin de obtener una estimación de posición precisa.

- **Location filtering RNN**: Esta red utiliza la salida de la red Location matching RNN. Realiza un filtrado de las características de ubicación aproximada para obtener una estimación de ubicación en la trayectoria más precisa.

Para evitar el problema del gradiente desvaneciente, se utilizan neuronas LSTM con funciones de activación del tipo *tanh* para cada red.

La función de coste para cada ejemplo se calcula como la suma ponderada de los costes de cada red por separado:

$$Loss = \frac{\lambda_1}{2} \sum_{i=1}^{T} \|\hat{l}_{1i} - l_i\|_2 + \frac{\lambda_2}{2} \sum_{i=1}^{T} \|\hat{l}_{2i} - l_i\|_2$$

En donde $\hat{l}_{1i}$ y $\hat{l}_{2i}$ indican la salida de la primera y segunda red respectivamente, T es la cantidad de pasos considerados hacia el pasado, $\lambda_1 + \lambda_2 = 1$ son los pesos de cada red individual, hiperparámetros de la DL-RNN a fijar (en general se toma λ_1 menor a λ_2).

Este modelo se probó en la Universidad Estatal de Río de Janeiro, tomando 157 PR separados unos 6 metros entre ellos y con unas 180 a 240 señales recibidas de Wi-Fi en dichos PR. El sistema tomó secuencias de 11 instantes correlacionados y

alcanzó una precisión promedio de 3,05 metros, con tan solo un 5 % de las medidas obteniendo un error de más de 6,5 metros. En las mismas condiciones, una red KNN dio una precisión de 4,9 metros en promedio, con un 90 % de las medidas percibiendo un error de más de 9 metros.

Realizando pruebas sobre los datos recabados en esta Universidad para ajustar los hiperparámetros, la bibliografía recomendó establecer la primer etapa como una red LSTM de sólo una capa con 100 neuronas, y la segunda etapa también como una LSTM de una capa pero con 50 neuronas. Se definió un batch size de 100 trayectorias y un procedimiento de early stopping para prevenir el sobre entrenamiento.

2.3.4. MISO, MIMO y P-MIMO

Otra propuesta de aplicación de las RNN a nuestro problema, compara varias estructuras posibles utilizando redes RNN como base, pero de distintas maneras [29].

El primer modelo propuesto es llamado MISO (Multiple Input, Single Output), en la que la red toma cada vector de huellas RSSI del usuario a lo largo de su trayectoria considerada como entradas independientes y tiene como salida única la etiqueta de ubicación estimada. En la figura 2.11 se puede apreciar la estructura del modelo MISO.

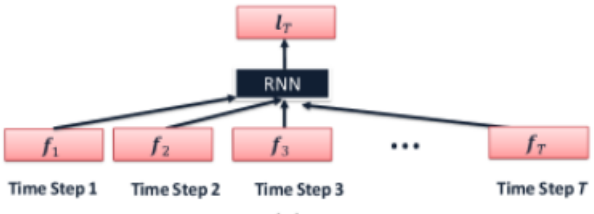


Figura 2.11: Arquitectura MISO.

Si al vector de huellas se le agrega la etiqueta de ubicación real relevada en el entrenamiento, entonces al modelo se le denomina A-MIMO. Para testeo y en la fase offline, se agrega la etiqueta predicha en el paso anterior. La función de coste para ambos modelos se define como el RMSE usual:

$$Loss = \|\hat{l}_i - l_i\|$$

El segundo modelo es un MIMO (Multiple Input, Multiple Output), en que se encadenan T redes RNN, siendo T la cantidad de pasos considerados en el histórico de la trayectoria del usuario. Las entradas de la RNN i-ésima serán las medidas de RSSI del usuario para el instante i dentro de la trayectoria ($1 \leq i \leq T$) y la predicción de la RNN anterior, de haberla (la primera red tiene única entrada). Si al vector de huellas de RSSI le agrega la etiqueta de ubicación real relevada, se le denomina A-MIMO. De manera análoga a la A-MISO, tanto en testing como en la fase offline se agrega la etiqueta predicha en el paso anterior. En la figura 2.12 se puede apreciar la estructura del modelo MIMO.

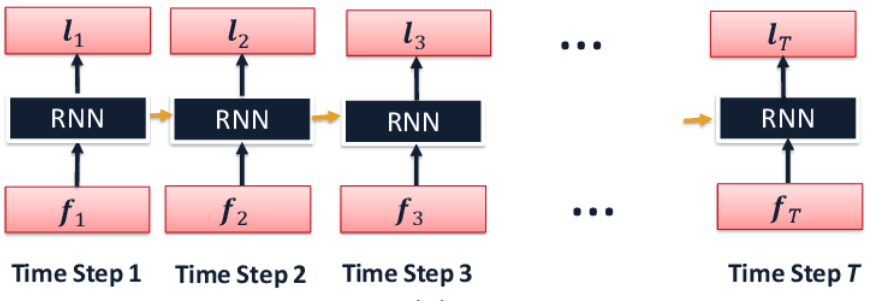


Figura 2.12: Arquitectura MIMO

Por último, se propone el modelo P-MIMO, que se puede ver como una red A-MIMO en la que la etiqueta agregada es siempre la predicha en el paso anterior, tanto en entrenamiento como en testing. En la figura 2.13 se detalla la estructura del modelo P-MIMO.

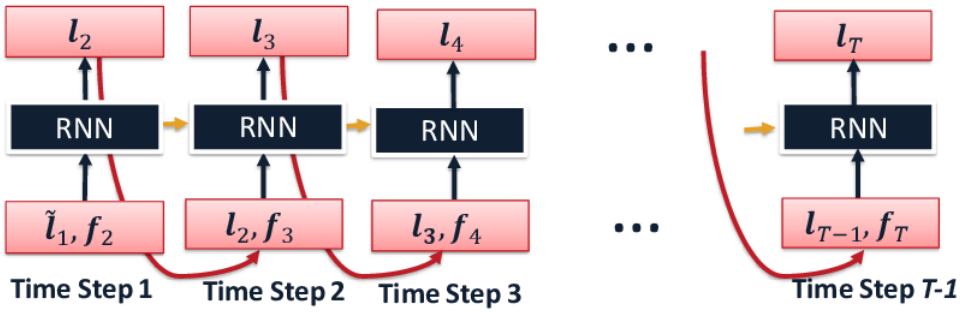


Figura 2.13: Arquitectura PMIMO

Las funciones de coste para estas últimas tres redes, con múltiple salida, se define como:

$$Loss = \frac{\sum_{i=1}^{T} \|\hat{l}_i - l_i\|_2}{T}$$

Los algoritmos fueron implementados en la Universidad de Victoria, en un área de $21m \times 16m$ en las que se definieron 365 PRs, con 11 MACs de APs percibidas por el dispositivo en estas zonas. Se implementaron con neuronas LSTM, se entrenaron las redes con 365.000 trayectorias y se compararon los errores en metros (definidos como la distancia euclídea entre la posición relevada y la estimada para el dataset de testing). En este escenario tanto P-MIMO como MIMO dieron errores promedio menores a 1 m (0,75 m y 0,80 m respectivamente) mientras que las otras redes dieron errores mayores a 1 m.

En esta bibliografía se realizaron diversas pruebas sobre los parámetros utilizados. Se estudió el largo de trayectorias (5, 10 y 40 pasos), concluyendo que 10 pasos daban los mejores resultados. Se varió también el tamaño del dataset de trayectorias, observando que por encima de 10.000 trayectorias los errores en predicción no disminuían considerablemente. Se estudió la cantidad de capas y de neuronas

por capa, resultando en un óptimo de 2 capas ocultas de 100 neuronas cada una. Se utilizó un optimizador ADAM con learning rate de 0.001 y un dropout rate de 0.1.

2.4. Crowdsourcing vs Crowdsensing

Como se vio anteriormente, la tarea offline de relevamiento de huellas previo al entrenamiento puede ser costosa en tiempo y dinero, ya que los algoritmos suelen necesitar una gran cantidad de huellas por zona y los métodos de relevamiento no suelen permitir relevar más de una huella cada unos pocos segundos. Si consideramos un área extensa en la que nos interese implementar un sistema de localización en interiores, nos vamos a encontrar con la desagradable noticia de que precisaremos quizás alguna decena de horas de relevamiento realizadas por un humano.

En particular, se necesitan entre 3 y 10 segundos para tomar una medida de señal en una ubicación determinada, y un sistema robusto necesita varias medidas distintas tomadas en la misma zona. Más adelante, en este trabajo, se verá que se hicieron varios relevamientos llegando a un total de 8000 huellas para cubrir una zona de 360 metros cuadrados, llevando un trabajo acumulado de diez horas. A su vez, es necesario la participación de dos personas en estos relevamientos, una de ellas manipulando el dispositivo móvil y la otra verificando que los datos se van cargando al servidor en la nube.

Claro está que los sistemas además comienzan a realizar mejores predicciones mientras mayor sea la cantidad de huellas que dispongan en cada zona, o mientras más pequeñas sean las zonas y por ende más cerca estén los puntos de referencia.

Parece una pregunta natural preguntarse si hay alguna manera de trasladar entonces parte el esfuerzo del relevamiento al usuario final de la aplicación: se parte de un sistema entrenado con un conjunto más pequeño de ejemplos, y dejamos que los usuarios nos compartan (consciente o inconscientemente) información que podamos utilizar para que nuestro algoritmo sea más preciso con el tiempo. Además, se genera una condición de robustez ya que la información que se aporte a medida que pase el tiempo va a serle fiel a ciertos cambios del entorno, como movimiento de obstáculos para la señal de Wi-Fi (mobiliario metálico, mampostería liviana, cartelería, etc.).

2.4.1. Fingerprint Crowdsourcing

Es entonces que surge el concepto de *crowdsourcing*, bajo la idea de que la unidad es más potente que la individualidad. En efecto, si tenemos un conjunto de individuos capaces de voluntariamente (y presuntamente sin intenciones de perjudicar nuestro sistema) aportar información a medida que realizan sus tareas personales en el recinto que alberga nuestra solución, entonces podríamos no solo partir de un relevamiento más sencillo, sino que también garantizaríamos la durabilidad de los resultados acertados de la misma.

Pasemos a analizar distintas estrategias que se pueden realizar para llevar a los usuarios a añadir información a nuestro sistema:

- **Contribución activa**

  La idea de esta estrategia es buscar contribuyentes fiables que puedan utilizar algún tipo de referencia (ya sea en un plano, o indicadores de proximidad a un sitio reconocible) para que activamente nos proporcionen una pareja de huella de RSSI y ubicación necesaria para contribuir a la fase de entrenamiento. Los usuarios deberían estar seleccionados, validados o motivados como "no maliciosos" para evitarnos la posibilidad de tener contribuciones que finalmente terminen siendo perjudiciales para el sistema.

  Un ejemplo de esto, sería el caso en que tuviesen una motivación económica indirecta, por ejemplo si el sistema se implantara en un centro comercial: cuando el usuario presione un botón en una aplicación de descuentos cuando está frente a una caja realizando su pago, recibirá una bonificación en su compra.

  Otra alternativa es solicitarle al usuario que se acerque hasta el punto de referencia para realizar el aporte, o incluso forzarlo a realizar una trayectoria fija, con pocos grados de libertad (como por ejemplo recorrer un pasillo) en la que sabemos con certeza los PR por los que pasará.

  En contraposición, podemos buscar una estrategia más reactiva, en la que la aplicación le consulte al usuario si se encuentra próximo a cierta referencia altamente reconocible, si es que creemos que realmente se encuentra allí. Y es que tanto en caso afirmativo como negativo tendremos información útil de su respuesta.

  En algunas implementaciones, como Google Project o TraviNavi [38] [36], el usuario puede contribuir indirectamente con su ubicación tomando una fotografía de un sitio de referencia. Un complejo sistema de reconocimiento de imágenes, como SfM [23] (Structure from Motion), podría detectar vértices en una edificación o cartelería y utilizarlo para estimar la distancia entre el terminal y el PR, para así agregar un nuevo PR y aumentar la densidad de huellas digitales del sistema.

- **Contribución pasiva**

  Una alternativa es tomar los datos, con autorización del usuario pero sin su participación directa. A esto a veces se le denomina Crowdsensing debido a que es el sistema quien automáticamente estima mediante un método alternativo la ubicación del usuario, y envía al sistema la pareja de huella y localización [34]. Una estrategia bastante común es utilizar otros sensores del dispositivo móvil, como ser acelerómetro y giroscopio, para predecir pequeñas variaciones de movimiento: si damos por válida la ubicación inicial del usuario, podremos aumentar la densidad de nuestros PR con tan solo enviar una medida nueva cuando se haya movido unos centímetros en una dirección conocida.

Otra manera de implementar Crowdsensing es manteniendo una dimensión más en el mapa de huellas digitales. Se elige una segunda magnitud a percibir por el terminal (campo magnético, señal celular 2/3/4G, señal FM o beacons Bluetooth) y utilizar una medida para corregir la otra. Estas implementaciones son bastante frecuentes en sistemas como MaLoc, Mawi.

2.5. Análisis de Desempeño

Las redes RNN pueden trabajar tanto como algoritmos de clasificación como de regresión. Se detallan a continuación las métricas consideradas en cada escenario para realizar la evaluación del sistema.

2.5.1. Clasificación

En este caso, se separa el área de interés en zonas rectangulares (preferentemente zonas de dimensiones idénticas) y se eligen los PR de tal manera que coincidan con los centros de los mismos. Se numeran las zonas y se pretende clasificar una medida nueva dentro del conjunto de zonas disponibles.

En este escenario, esta clasificación puede ser positiva (se acertó en la predicción con la zona real de donde se tomó la muestra) o negativa (la predicción dio una zona distinta a la real).

Dado un conjunto de ejemplos con su número de zona conocidos, se puede realizar un diagnóstico con el algoritmo y compararlo a los resultados reales, resultando en un diagnóstico verdadero (el resultado del diagnóstico fue igual a la clasificación) o falso (el diagnóstico y la clasificación dieron diferentes). De aquí en adelante, se utilizarán las abreviaciones para las cuatro posibles combinaciones: VP (Verdadero Positivo), VN (Verdadero Negativo), FP (Falso Positivo) y FN (Falso Negativo).

Se eligieron tres estadísticas principales para estudiar el rendimiento del sistema. De aquí en adelante se abreviarán:

Se define la precisión (según índice de Youden) como:

$$J = especificidad + sensibilidad - 1 \tag{2.1}$$

en donde,

$$Especificidad = \frac{VP}{VP + FN} \tag{2.2}$$

$$Sensibilidad = \frac{VN}{VN + FP} \tag{2.3}$$

El valor puede variar entre -1 a 1 y resulta en una medida del rendimiento del diagnóstico, siendo 1 la prueba perfecta. Para evaluar el resultado del índice en el caso de uso, se utiliza un set de datos de prueba sobre el cual evaluar los distintos algoritmos.

Se define tambien la exhaustividad como

$$Exhaustividad = \frac{VP}{VP + FN} \tag{2.4}$$

que permite representar cuántas huellas que son efectivamente de una zona, son finalmente clasificadas en esa zona. Esta métrica en cierta medida permite complementar la precisión y da una buena idea de en qué zonas los usuarios pueden tener mayores problemas para ser localizados

Por último, se considera la matriz de confusión, que permite reconocer la predicción cruzada entre las distintas zonas. Para la posición de la fila i y columna j en la matriz, el valor representa cuántas huellas correspondientes a la zona i fueron clasificadas en la zona j. Es una manera visual muy intuitiva: en la diagonal de la matriz se verán las predicciones positivas, mientras que es fácil detectar cuando los ejemplos de una zona son clasificados sistemáticamente de manera equivocada, por ejemplo en una zona adyacente.

2.5.2. Regresión

Otro acercamiento al problema es asignarle a cada PR_i un par coordenado (x_i, y_i) que refiera a la distancia en metros de un punto en plano definido como origen, pasándole a la red este par de etiquetas junto a la huella tomada en la fase de entrenamiento.

Ante un nuevo ejemplo, la salida de la red será un nuevo par coordenado que no tiene por qué coincidir con las coordenadas de ningún PR. Algunas de las redes consideradas tienen múltiples salidas, una para cada paso en la trayectoria de entrada. En estos casos, se considera como predicción de la localización por parte del sistema, a la salida correspondiente al último paso.

Para una huella conocida, con localización real l_i, se define el error en la predicción resultante $\hat{l}_i$ como la distancia euclidiana entre los pares coordenados. El error de la red entonces será el promedio de los errores de un set de ejemplos de tamaño N:

$$E = \frac{\sum_{i=1}^{n} ||\hat{l}_i - l_i||}{N} \tag{2.5}$$

Como manera gráfica de estudiar el comportamiento del sistema, se considera una gráfica que muestra el error en cada predicción del conjunto de testing. Se calcula el error en metros obtenido para todos ellos y se grafica en función de x. En el proceso, se relevan también el error máximo y mínimo de la red, así como el porcentaje de predicciones que quedaron por debajo de ciertos umbrales de interés.

También se analizan algunas trayectorias elegidas aleatoriamente de entre las incluidas en el conjunto de testing. Se comparan las trayectorias reales con las que se forman con las predicciones para cada paso por la red, para evaluar si hay convergencia a medida que transcurren los pasos. Esto puede dar una noción sobre el tiempo que demora el sistema en estabilizarse, ya que si las predicciones iniciales son muy alejadas de las reales, durante los primeros segundos desde que el usuario inicie la primer solicitud de localización el sistema estará dando información errónea que puede llevar a una mala experiencia de usuario.

Para volver al problema de clasificación, se realiza un algoritmo que toma el par coordenado de salida y decide en qué zona corresponde clasificarlo. De esta

manera se pueden volver a aplicar las métricas mencionadas en el punto anterior, y en particular, entregar al sistema una etiqueta de zona con la cual decidir qué información multimedia mostrarle al usuario final en su aplicación.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Implementación del sistema

En el presente capítulo se desarrolla el modelo de trabajo, algoritmos implementados y herramientas utilizadas. Primeramente se realizará una introducción a la estructura de trabajo. Luego, se detallarán la arquitectura implementada de los servidores, el método de recolección de los datos en campo, y el funcionamiento de las aplicaciones móviles que se desarrollaron. Por último se explican las interacciones entre las distintas componentes del sistema para los casos de uso de interés, el procesamiento de los datos y la implementación de los algoritmos estudiados.

3.1. Introducción

Considerando los resultados de las referencias analizadas, se decidió evaluar tres métodos centrados en las redes LSTM. Estos métodos se implementaron en Python y se entrenaron con un conjunto de huellas de ejemplo, simulando resultados y comparándolos de manera de elegir el óptimo. Se desarrollan entonces los modelos MIMO (múltiple entrada y múltiple salida), P-MIMO (múltiple entrada y múltiple salida) y DL-RNN (dos RNN en cascada). Como vimos, estos métodos de inteligencia artificial, aplicados a la localización de huellas digitales de Wi-Fi constan de dos fases, la fase offline para el entrenamiento y la online para la clasificación en tiempo real del usuario [17].

En la fase offline, se procede a dividir el área de interés, en nuestro caso el hall del primer piso de Facultad de Ingeniería, UdelaR, en zonas acotadas. Luego se utiliza un dispositivo móvil para tomar varias medidas de señal en cada una de estas zonas y enviarlas a un servidor específico (de Entrenamiento-Principal). Se construyen trayectorias que hagan recorridos por las mismas y se le asigna a cada paso de la trayectoria una medida de señal de entre las relevadas. El largo (cantidad de pasos) de las trayectorias a utilizar se discutirá más adelante.

Se construyen tres dataset disjuntos, matrices que para cada trayectoria indican a qué zona corresponde cada paso y qué vector de señales RSSI le corresponde. Un dataset se utiliza para el entrenamiento de la red neuronal, otro para la validación de la red, y por último uno para el testing de los datos, que nos permitirá evaluar el rendimiento del modelo. Una vez entrenada la red en el servidor de Entrenamiento-

AI, se guardan los parámetros resultantes que no variarán de aquí en más, y se implementa en otro servidor (de Clasificación-AI) una función de mapeo que recibe nuevas trayectorias y las clasifica utilizando los algoritmos implementados.

En la fase online, el dispositivo con acceso a Internet utiliza una aplicación Android para enviar periódicamente al servidor de Clasificación-Principal las señales de RSSI de los APs que percibe, una huella $f_i \in R^n$. Este servidor recibe la huella, la almacena en una base de datos como siguiente paso de una trayectoria etiquetada con el identificador del usuario y le solicita al servidor de Clasificación-AI la estimación de posición. La almacena en la base de datos como la última predicción encontrada y queda disponible para futuras consultas. Cada vez que el dispositivo envía una huella, solicita la última predicción disponible y la almacena internamente. Luego, identifica si esta predicción es diferente a la última almacenada (la primera vez que recibe una predicción siempre se considera como diferente), y de ser así le solicita a un servidor de información multimedia una imagen correspondiente a esta ubicación. Se le muestra entonces al usuario un croquis del plano del Hall de Facultad de Ingeniería con una marca aludiendo a su ubicación según fue predicha.

Un conjunto acotado de zonas fueron seleccionadas para probar un sistema de crowdsourcing. Estas zonas (como por ejemplo, la que incluye la puerta a la mesa de entrada de trámites de Facultad) se eligieron por ser fácilmente reconocibles por cualquier usuario. Cuando la aplicación recibe una predicción que coincida con alguna de las mismas, se le despliega un pop-up al usuario consultándole si efectivamente se encuentra allí. El usuario puede seleccionar entre dos opciones: Sí o No. En caso de que su respuesta sea afirmativa, se envía la huella nuevamente al servidor de Entrenamiento-Principal, para que la guarde en la base de datos. Esta información, puede ser posteriormente aprovechada para realizar futuros entrenamientos de la red neuronal, generando así robustez al sistema frente a cambios en el tiempo del mapa de RSSI del área de interés, y contribuyendo a la red con información proveniente de distintos dispositivos físicos.

3.2. Arquitectura de los Servidores

Este proyecto utilizó los servicios de hosting brindados por AWS (Amazon Web Services). Se entendió que la experiencia presentada por el proyecto "Localización Indoor Basada en WiFi" relativa a esta elección fue positiva, presentando una solución funcional y con un rendimiento adecuado para el escenario trabajado, por lo que se procedió a realizar una configuración que partió de esa arquitectura de servidores aunque con modificaciones sustanciales.

Según se puede ver en la figura 3.1, se levantaron seis servidores, cada uno en su contenedor de Docker. Se distribuyeron en tres instancias separadas de EC2-T3 (Amazon Elastic Compute Cloud), uno para el entrenamiento de las redes (medium), otro para la clasificación de las trayectorias del usuario (large) y un tercero para consultas de contenido multimedia (medium), cada una de ellas con su conexión a Internet correspondiente. Además, se contrataron dos buckets S3. El primero para el almacenamiento de los archivos .csv de entrenamiento de las redes

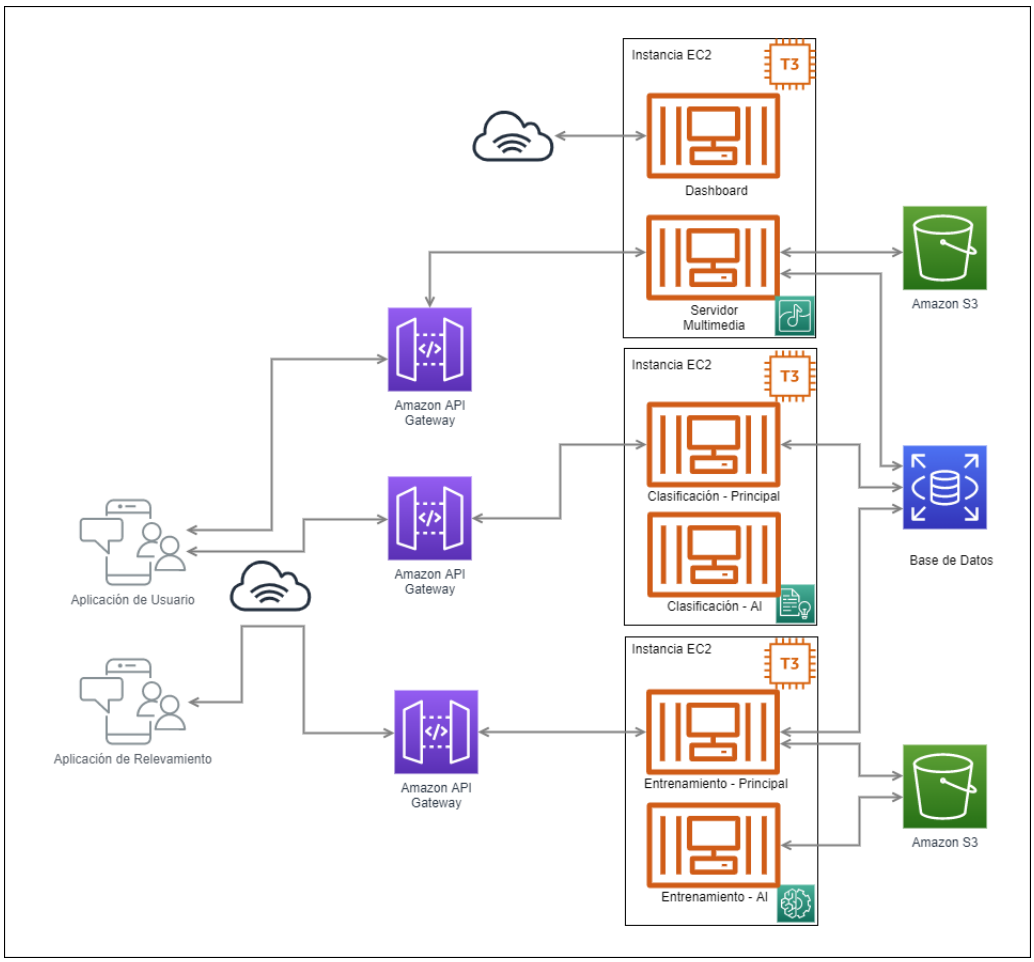


Figura 3.1: Arquitectura de los servidores.

neuronales y los parámetros resultantes del mismo. En el segundo, se guardan las imágenes que se muestran al usuario en su aplicación, según la zona clasificada.

A dos de los servidores se les denomina “principales”, ya que se encargan de comunicarse con las aplicaciones de relevamiento y de usuario, así como con los servidores de AI. El servidor Entrenamiento-Principal recibie vía API las medidas de RSSI del relevamiento junto con la etiqueta de la ubicación. Luego, almacena esta información en una base de datos relacional, armada sobre PostgreSQL [15], y la envía al servidor de inteligencia artificial (Entrenamiento-AI, levantado en la misma instancia) que toma las huellas y las divide en los tres conjuntos necesarios, creando los archivos .csv.

PostgreSQL es un framework de manejo de bases de datos relacionales de código abierto. Es una de las tantas opciones que brinda AWS para el manejo de las bases de datos en sus servidores, una de las más populares, y la utilizada por el proyecto anterior.

El servidor Clasificación-Principal recibe vía API las medidas RSSI del usuario junto con su identificador de usuario y su correspondiente timestamp. Almacena

la información en la base de datos y la envía al servidor de Clasificación-AI, en su misma instancia de EC2, quien implementa el algoritmo entrenado y guarda en la base de datos el resultado de la predicción. Por último, el servidor de Clasificación-Principal queda esperando que se le responda con la localización predicha. La aplicación recibirá la información actualizada, una vez realice un GET solicitándola. El tratamiento de los datos, para modificar el JSON enviado por el usuario en información almacenable en sus bases de datos, se configuró y adaptó en Go [8].

Go es un lenguaje de código abierto orientado a objetos, desarrollado por Google. El mismo fue utilizado en el framework de FIND3 para la implementación de los servidores y por lo tanto se tuvo que estudiar para realizar las modificaciones realizadas.

Como se puede ver, se realizaron diversas modificaciones al sistema base, principalmente separando el entrenamiento de la clasificación en instancias distintas. Esto es, porque el servidor de entrenamiento se tuvo que diseñar desde cero, y las nuevas implementaciones requirieron instalar TensorFlow [19] y Keras [6] para poder realizar el entrenamiento de las redes neuronales recurrentes. Además, se tuvo que modificar las comunicaciones con la base de datos para almacenar trayectorias de T pasos, en vez de huellas simples, necesarias para el entrenamiento.

TensorFlow es una biblioteca de código abierto (con APIs para varios lenguajes aunque en este trabajo se utilizó como librería de Python), desarrollada específicamente para realizar algoritmos de aprendizaje automático. Provee las herramientas necesarias para elaborar, entrenar y analizar redes neuronales. Keras es una librería específica para Python, compatible con TensorFlow, que contiene los módulos necesarios para construir una red neuronal, como las capas, funciones de activación y optimizadores. Ambas herramientas demandan más capacidad de procesamiento y espacio en disco de lo considerado para correr las funciones de clasificación en tiempos razonables. Debido a esto, se tuvo que contratar una instancia en AWS de 4GB de RAM y 100GB de disco duro.

Por último, para el servidor de contenido multimedia, que se levantó en una tercer instancia, no se realizaron cambios mayores a los implementados por el proyecto anterior. Sin embargo, se tuvo que volver a levantar la página web (dashboard) que oficia de interfaz para cargar en el servidor el contenido a mostrar al usuario para cada ubicación. Esto es una tarea que se realiza por única vez, salvo que se deseen cambiar las imágenes de localizaciones que recibe el usuario desde su aplicación.

La figura 3.2 muestra tablas creadas en la base de datos relacional de la solución, con sus datos y las relaciones entre ellos. Se partió de la estructura de datos propuesta por el proyecto anterior, agregándose A continuación se detalla cada tabla con el fin de explicar la

- gps: guarda datos de gps, no se utiliza en esta solución.
- pieces: guarda información de las zonas (identificador, nombre y url hacia la imágen).
- devices: guarda el identificador del dispositivo y le asigna una clave que se utiliza para guardar las huellas asociadas al dispositivo en la tabla sensors.

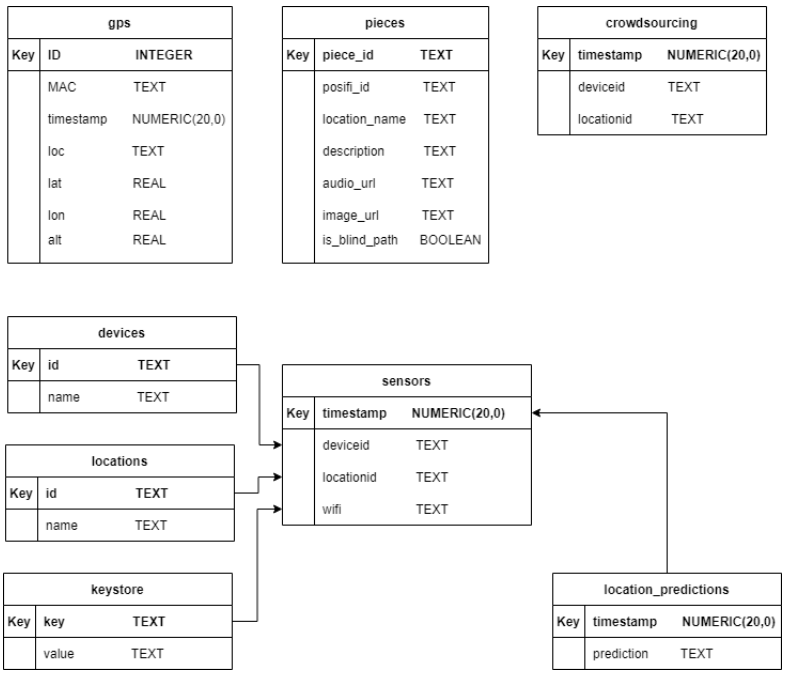


Figura 3.2: Modelo relacional de la base de datos

- locations: guarda el identificador de la zona y le asigna un nombre que es utilizado para referenciar las huellas asociadas a esa zona a la hora de realizar el relevamiento. Se relaciona con la tabla sensors como locationId.

- keystore: guarda las MAC address de los APs y les asocia una clave. La misma es utilizada para referenciar, en la tabla sensors, los valores de RSSI asociados a cada MAC en las huellas.

- sensors: guarda las huellas asociadas a cada dispositivo para ser clasificada, se relaciona con las tablas devices, locations, keystore y location_predictions.

- location_predictions: guarda las zonas predichas. El atributo Timestamp es el mismo que el de la tabla sensors, porque es de esta tabla que se obrtiene la huella para clasificar.

3.3. Relevamiento de Huellas

Partiendo de un área de interés, se divide la misma en m zonas, donde a cada zona le corresponde un punto de referencia PR_i, siendo este el centro físico de la zona. Este punto de referencia tiene asociada una ubicación en coordenadas $l_i = (x_i, y_i)$.

Al escanear las huellas RSSI se genera por cada ubicación (loc_i) una tabla de dimensiones $(n, p + 1)$, donde tenemos n huellas para cada zona y las medidas disponibles (RSSI) de los p puntos de acceso Wi-Fi (APs) en todas las frecuencias

de la portadora detectados. Se tiene una columna adicional para la etiqueta de la localización relevada.

Una huella se representa con un vector de la forma $f_i = {RSSI_1}^i, {RSSI_2}^i, ..., {RSSI_p}^i$, asignando al ultimo elemento de la fila la etiqueta de la zona (loc_i) a la cual pertenece.

Por lo tanto para cada zona i, obtendríamos la siguiente tabla:

$$F_i = \begin{pmatrix} Ap_1 & Ap_2 & ... & Ap_p & zona \\ {RSSI_1}^1 & {RSSI_2}^1 & ... & {RSSI_p}^1 & loci \\ {RSSI_1}^2 & {RSSI_2}^2 & ... & {RSSI_p}^2 & loci \\ ... & ... & ... & ... & loci \\ {RSSI_1}^n & {RSSI_2}^n & ... & {RSSI_p}^n & loci \end{pmatrix}$$

Con el propósito de ejecutar los experimentos a fin de seleccionar el método más adecuado se optó por utilizar el sector del hall del primer piso de Facultad de Ingeniería. El mismo se dividió en 90 zonas de $2m \times 2m$, y se tomaron huellas en cada celda.

En la figura 3.8 se muestra de manera más esquemática la división del mismo.

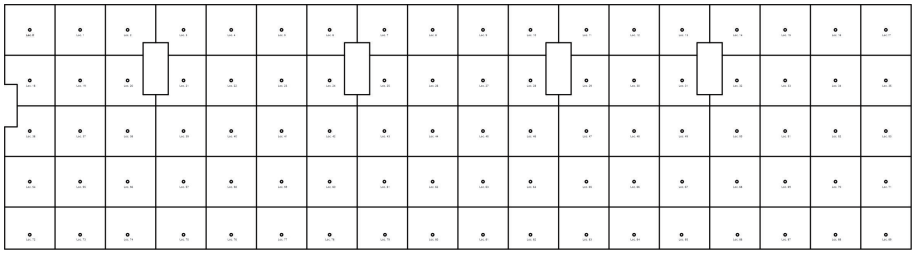


Figura 3.3: Esquema - Hall del primer piso de facultad de Ingeniería.

Estas huellas se utilizaron para entrenar las redes implementadas con trayectorias ficticias y simular los resultados, con el fin de realizar evaluaciones preliminares y decidirse por un algoritmo a implementar en la arquitectura del sistema.

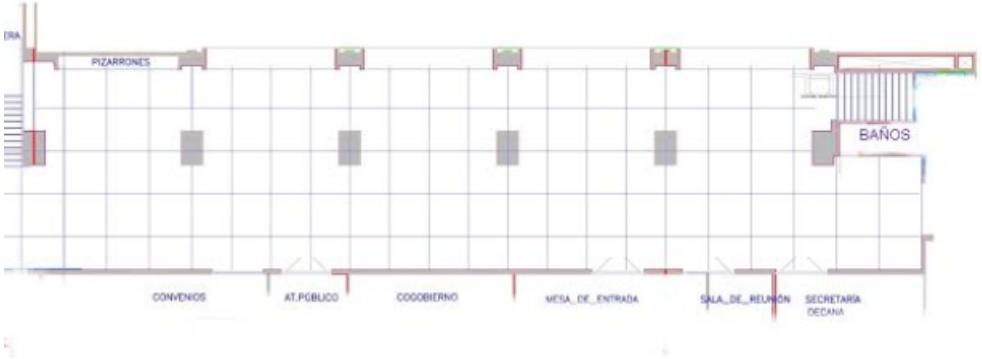


Figura 3.4: Plano - Hall del primer piso de facultad de Ingeniería.

3.4. Aplicación Android Para Toma de Huellas

Con el fin de realizar un relevamiento eficiente, que nos permita tomar varias huellas de RSSI en cada una de las zonas definidas, se optó por no utilizar la herramienta dispuesta por FIND3 [5] sino actualizar la aplicación Posifi_App_Scanner del proyecto "Localización Indoor Basada en Wi-Fi" que es una modificación a la anterior. La misma permite enviar datos con una frecuencia mucho más alta, del orden de un segundo, reduciendo considerablemente los tiempos de relevamiento.

En ese sentido, hubo que utilizar Android Studio [1] para actualizar las librerías Node.js [13] y npm.js [14] de la aplicación, ya que la misma no se podía correr en las nuevas versiones de Android (en particular Android 9). Android Studio es una herramienta disponible para Linux, Windows y MAC que permite trabajar de manera cómoda un proyecto de desarrollo de una aplicación para Android. Permite visualizar y actualizar las versiones de las múltiples librerías actualizadas. Cuenta con un editor de código, un emulador de la aplicación en desarrollo y un debugger.

Además, se realizaron algunas modificaciones en el código escrito en React Native [18] para adecuarla al nuevo sistema de comunicación implementado. React Native es un framework de JavaScript diseñado específicamente para escribir código nativo de aplicaciones móviles multi-plataforma (para Android e IOS). Esto quiere decir, que es una interfaz común que luego utilizará APIs de traducción a Objective-C (IOS) o Java (Android).

La figura 3.5 muestra la pantalla de recolección de huellas. En la misma se debe ingresar la URL del servidor Entrenamiento-Principal en el campo "Host", un identificador del dispositivo en el campo "Device", el identificador de la zona en el campo "Place" y el nombre del proyecto (en nuestro caso "locindoor") en el campo "Family". La aplicación envía vía API la información ingresada junto con la huella escaneada, en un JSON que el servidor Entrenamiento-Principal tomará como entrada para construir un archivo .csv que concentre todas las huellas enviadas.

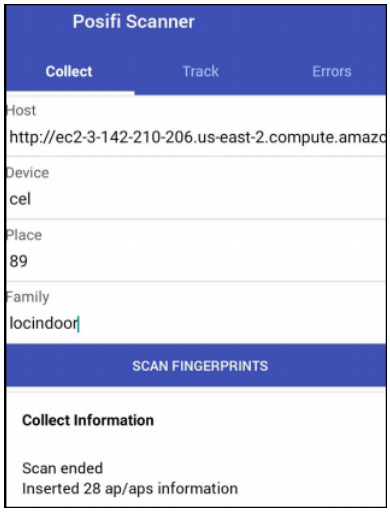


Figura 3.5: Aplicación para toma de huellas

3.5. Aplicación Android de Usuario

Para brindarle al usuario final una aplicación amigable que le permita navegar por el área de interés y visualizar de manera ágil su ubicación desde su terminal, se decidió adaptar la aplicación MNAVegante.

La aplicación MNAVegante realizaba ya la recolección de las señales RSSI del terminal, las enviaba vía API en un JSON a un servidor en la nube y obtenía la localización online. Luego, si la zona recibida se comparaba a un histórico interno de su propio storage asíncrono para decidir si le mostraba un nuevo contenido multimedia o esperaba a recibir más localizaciones que validen el movimiento inter-zonas. En caso de decidir que el usuario efectivamente se había desplazado y no era un error en la predicción, solicitaba vía API el contenido multimedia a mostrar. Para tomar esta decisión realizaba una comparación de las últimas 10 predicciones disponibles.

Además, tenía dos opciones de navegación, el recorrido a ciegas y la muestra "Aquí soñó Blanes", que básicamente corrían el mismo algoritmo pero mostraban información distinta, pretendiendo dar una experiencia de usuario adecuada tanto para videntes y no videntes, pudiendo llevar al usuario audio o texto que detallara la obra mostrada en las zonas definidas con este propósito.

Para aprovechar esta aplicación, se tuvo que actualizar las librerías de npm.js y node.js desde Android Studio para poder ser instalada en Android 9 y posteriores, modificando el código en React Native para que se comunicara con los nuevos servidores levantados en el marco de este trabajo. Además, se modificó la estética de la aplicación acorde al contexto del uso que se le daría dentro de Facultad de Ingeniería, solicitando al servidor únicamente imágenes ilustrativas de la ubicación del usuario, como se ve en la figura 3.6, y no archivos de audio e información de texto como fue ideada originalmente.

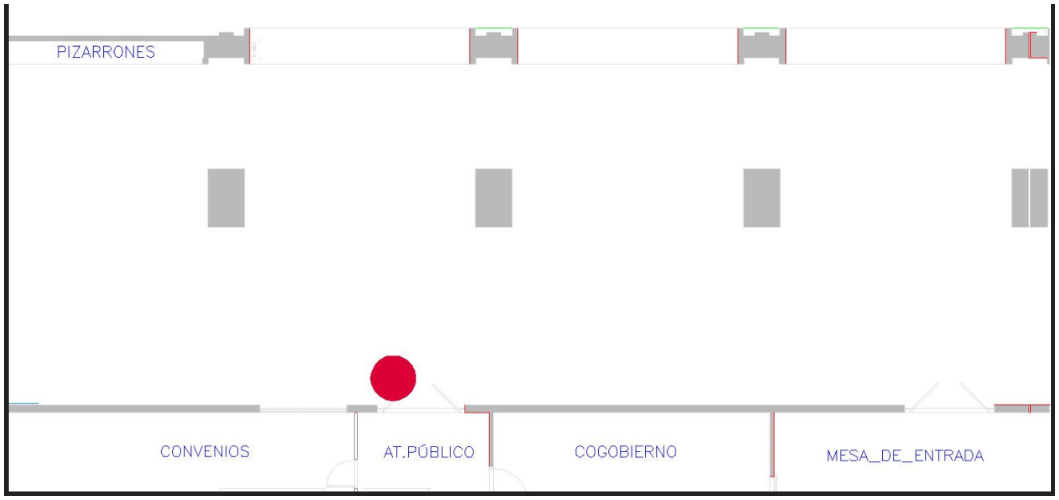


Figura 3.6: Imagen mostrada en la aplicación para la zona 78.

Por otro lado, se tuvo que cambiar la frecuencia con la que se envían los datos al servidor. Se buscó una coherencia entre la cadencia con la que el terminal envía

sus huellas de RSSI con la distancia temporal entre pasos configuradas en la red neuronal, teniendo presente efectos de delay y jitter presente en las comunicaciones cliente-servidor. Al final, se terminó coincidiendo en que una huella por segundo era la frecuencia óptima tanto para la generación de trayectorias de entrenamiento como para la frecuencia de localización, ya que con zonas de dos metros cuadrados difícilmente el usuario cambiara de ubicación en menos de un segundo.

Recordemos además que nuestra arquitectura maneja la dependencia temporal de los datos enviados directamente en los algoritmos de inteligencia artificial, así que no era ya necesario que se guardase el histórico en el storage de la aplicación.

Por último, se agregó un módulo de Crowdsourcing. El mismo revisa constantemente que la última predicción de ubicación recibida esté dentro de un conjunto de zonas predefinidas. Se eligieron aquellas zonas próximas a un sitio claramente distinguible, como lo es la entrada de decanato o de mesa de entrada de trámites. Cuando esta condición se cumple, al usuario le aparece en su pantalla un pop-up consultándole si efectivamente se encuentra en esa ubicación, según se puede ver en la figura 3.7.

La respuesta, sea afirmativa o negativa, se envía al servidor junto con la huella tomada en el instante en que el usuario ingresa su opinión y se guarda en una base de datos para posteriormente ser contrastada con los datos de entrenamiento. De momento, es necesario correr un proceso manual para agregar al conjunto de ejemplos de entrenamiento las huellas que se hayan considerado coincidentes con lo predicho.

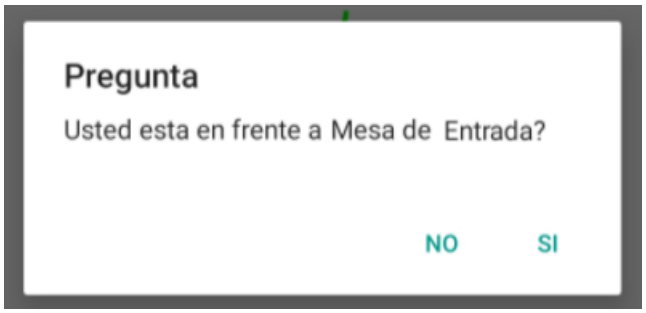


Figura 3.7: Pop-up de consulta al usuario

3.6. Comunicaciones Entre las Componentes

A fin de resumir las comunicaciones descritas previamente para facilidad del lector, se presentan a continuación los detalles de las mismas para los distintos casos de uso implementados:

3.6.1. Relevamiento

Para realizar las tareas de campo requeridas previas al entrenamiento de las redes neuronales en la fase offline, cada vez que se requiere almacenar una huella

de RSSI en una ubicación a relevar, se realizan las siguientes interacciones entre componentes (aplicación de relevamiento, servidor Entrenamiento-Principal):

1. La aplicación hace un POST a la URL del servidor Entrenamiento-Principal (*/data*) con los datos de Host (URL del servidor), Device (identificador del dispositivo), Place (identificador de la zona) y Family (identificador del área de interés).
2. El servidor crea una tabla en la base de datos para la Family ingresada, en caso de que sea una que no conoce.
3. El servidor principal guarda la huella en la base de base de datos, dentro de la tabla correspondiente.

Las huellas con sus respectivas etiquetas de localización son luego extraídas de la base de datos en formato .csv, procesados mediante funciones de Python en el servidor de Entrenamiento-AI y separados en los dataset de entrenamiento, validación y testing. Para cada dataset se genera la cantidad de trayectorias predefinida, seleccionando para cada ubicación una huella aleatoria dentro del dataset correspondiente. Se arman las matrices tridimensionales y se ingresan como datos de entrada para el entrenamiento de las redes en este mismo servidor. Por último se ejecuta manualmente el entrenamiento de la red y se guardan los parámetros internos para futuras clasificaciones.

3.6.2. Clasificación

Para que un usuario reciba su localización estimada en tiempo real (fase online), debe abrir la aplicación de usuario y entrar en la opción "¿Dónde estoy?". A partir de ese momento, sucederán las siguientes interacciones entre componentes (aplicación de usuario, servidor Clasificación-Principal, servidor Clasificación-AI, servidor Multimedia):

1. La aplicación consulta internamente su identificador de dispositivo. De no tener uno, lo genera aleatoriamente.
2. La aplicación tomará periódicamente las señales de RSSI de todas las MAC percibidas en su interfaz de RF y realizará un parse a un JSON.
3. La aplicación realiza un POST a la URL del servidor Clasificación-Principal (*/data*), enviando la huella en formato JSON.
4. Este servidor agrega la huella recibida a la siguiente posición libre de la tabla de huellas, indicando a qué dispositivo corresponde.
5. El servidor arma la trayectoria en un objeto de GO y la envía al servidor Clasificación-AI.
6. El servidor Clasificación-AI toma la trayectoria como entrada para ejecutar la red neuronal pre-entrenada y devolver una ubicación en coordenadas (x, y).

7. El servidor Clasificación-AI corre un algoritmo de Python que transforma las coordenadas en una etiqueta de ubicación y la pasa al servidor Clasificación-Principal.

8. El servidor Clasificación-Principal almacena la ubicación estimada en una tabla *location* específica para ese dispositivo.

9. La aplicación realiza un request periódico de su última localización disponible, a la URL del servidor Clasificación-Principal (*/predictions*).

10. En caso de recibir una localización distinta a la anterior, la aplicación Android realiza un request a la URL del Servidor Multimedia con esta nueva etiqueta y descarga la imagen a mostrar al usuario.

Este procedimiento se realiza una vez por segundo para garantizar una buena experiencia de usuario y coherencia con la diferencia temporal calculada entre pasos para la etapa de generación de trayectorias.

3.6.3. Crowdsourcing

Cuando la aplicación recibe por primera vez una localización que corresponde a un cierto conjunto predefinido internamente, despliega un pop-up al usuario consultándole si se encuentra próximo a una referencia fácilmente distinguible. El usuario responde afirmativa o negativamente según su percepción personal. Es entonces que suceden las siguientes interacciones entre componentes (aplicación de usuario, servidor principal):

- La aplicación de usuario realiza un POST al servidor de Entrenamiento-Principal, de un JSON con la última huella de RSSI confeccionada, su identificador de usuario, la etiqueta de la zona que disparó la consulta al usuario y la respuesta binaria a la consulta (0 o 1).

- El servidor pincipal guarda la información en una tabla de su base de datos, en una tabla dedicada a recolectar estas huellas

Posteriormente, es necesario realizar un procedimiento manual en el que las huellas recolectadas con respuesta afirmativa (1) sean agregadas a los dataset de entrenamiento, validación y testing, para armado de nuevas trayectorias necesarias para un re-entrenamiento de la red neuronal.

3.7. Procesamiento de los Datos

Las huellas digitales se envían al servidor de Entrenamiento-Principal en un JSON con el siguiente encabezado: [timestamp, name, wifi], donde en la columna wifi se encuentra un objeto con todas las señales RSSI detectadas en la celda de nombre “name”. Luego, han de ser procesados en el servidor de Entrenamiento-AI

para la construcción de trayectorias y el formato necesario para cada uno de los modelos implementados.

Para la elaboración del código a fin de realizar el procesamiento de estos datos, se utilizó *Jupyter Notebook* [16]. La misma es una aplicación web de código abierto, de fácil acceso, que permite crear y compartir documentos que contienen código, ecuaciones, visualizaciones y texto narrativo. Por esto, es muy útil para la limpieza y transformación de datos, implementación de modelos de aprendizaje automático, visualización de datos, realización de gráficos, etc.

El lenguaje utilizado fue Python [30] [17] con las librerías NumPy [9] y Pandas [10]. NumPy es una biblioteca que brinda soporte para trabajar con vectores y matrices multidimensionales, permitiendo crear y operar con estos elementos buscando la rapidez en las operaciones detrás de sus funciones. Pandas, en cambio, es una extensión de NumPy que se utiliza para manipular y analizar datos, introduciendo funciones que permiten operar con tablas numéricas y series temporales. Además del manejo de matrices NumPy para la manipulación de los datos, se utilizó en gran parte el formato de datos DataFrame de Pandas, el cual es una estructura de datos etiquetada bidimensional con columnas de tipos potencialmente diferentes. Este tipo de formato acepta entradas como, ndarrays, listas, dictados, series, numpy.ndarray, ndarray estructurado o de registro y hasta otro DataFrame, haciendo que sea un formato ideal para el manejo del tipo de datos que utilizamos.

Con estas herramientas se realizó una transformación de los datos, de forma que puedan ser introducidos en los tres modelos de redes neuronales implementadas. Las redes seleccionadas tienen como entrada un tensor de la forma [Tamaño del lote, n° de pasos de tiempo de una secuencia, n° de unidades de una secuencia de entrada]. En el caso de la DL-RNN, la entrada podrá ser un tensor de cuatro dimensiones, en la que se agregan múltiples vectores de huellas RSSI para cada zona, y no solo una huella como es en el caso de MIMO y P-MIMO.

El procesamiento de los datos se realizó en las siguientes etapas:

1. Adaptación de los datos enviados por la aplicación de escáner a una matriz donde cada fila representa un vector de señales RSSI, siendo las columnas los diferentes APs detectados y la última le corresponde a la zona donde fueron tomadas (etiqueta).

2. Filtrado de las huellas para quitar las señales recibidas por APs que no pertenecen a Facultad de Ingeniería o en su defecto a su proveedor de servicios de Internet. Se debe asegurar que los dispositivos van a permanecer medianamente fijos entre el relevamiento y las pruebas de campo y APs desconocidos pueden provenir de fuentes efímeras (como un terminal móvil siendo utilizado como Wi-Fi Hot Spot).

3. Limpieza de datos para eliminar huellas duplicadas en la misma zona de trabajo. Esto es para prevenir el efecto de las ráfagas de datos idénticos que provengan de la aplicación de relevamiento, en caso de que no haya realizado un nuevo escaneo entre dos medidas consecutivas. Además, se pretende posteriormente armar las trayectorias con huellas distintas para los conjun-

tos de entrenamiento, validación y testing (de lo contrario se podría sobre entrenar la red).

4. Normalización de los datos entre valores 0 y 1, dado que las redes con las que experimentemos utilizarán sigmoide o tanh como funciones de transferencia, por lo que los valores normalizados ofrecen mejores resultados a la hora de generar la salida en la red.

5. División del dataset. Para cada zona, se reparten las huellas disponibles en: 70 % para entrenamiento, 15 % para validación y 15 % para testing.

6. Creación de una matriz de trayectorias para cada uno de los dataset resultantes. Se generan aleatoriamente por un algoritmo que se detallará a continuación.

3.7.1. Generación de Trayectorias Virtuales

Como se mencionó, se crearon de manera virtual posibles trayectorias que un usuario recorrería dentro del edificio siguiendo las sugerencias de las referencias estudiadas [29]. En la figura 3.8 se puede apreciar la estructura de las matrices de datos generadas.

	l_1	l_2	l_3	...
l_1	0	0,5	6,0	
l_2	0,5	0	1,5	
l_3	6,0	1,5	0	
...				

Matriz de distancias euclidianas

	l_1	l_2	l_3	...
l_1	0,3117	0,3079	0,0035	
l_2	0,3079	0,3117	0,1927	
l_3	0,0035	0,1927	0,3117	
...				

Matriz de probabilidades

	l_1	l_2	l_3	...
l_1	0,3117	0,6259	0,6291	
l_2	0,3079	0,6258	0,8183	
l_3	0,0035	0,1962	0,5139	
...				

Matriz de distribución acumulativa

Figura 3.8: Matrices - generación de trayectorias.

En primer lugar se crea una matriz de tamaño 90×90 que contiene las distancias euclidianas entre cada punto de referencia PR_i. A partir de estas distancias se crea una matriz de probabilidades normalizada utilizando la siguiente ecuación:

$$P(l_i) = \frac{1}{2\sigma^2(1 - e^{\frac{d_{max}^2}{2\sigma^2}})} e^{\frac{(x_i - x_{pre})^2 + (y_i - y_{pre})^2}{2\sigma^2}}$$

Parámetro	Valor
σ	2m
Δt	1.0s
d_{max}	1.4 m

Tabla 3.1: Parámetros utilizados para la generación de Trayectorias

donde:

- $\sigma = v_{max}\Delta t$ - es la distancia máxima que un usuario puede moverse en un tiempo Δt, tomando v_{max} como la velocidad máxima estándar de una persona en el interior de un edificio. Se considera que la velocidad en interiores de una persona varía entre 0.4m/s a 2m/s [21], por lo que se tomó $v_{max} = 2m/s$.
- d^2_{max} - es la distancia máxima entre la ubicación considerada l_i y la ubicación más alejada dentro de la celda.
- (x_{pre}, y_{pre}) - es la ubicación anterior a estar en la posición $l_i = (x_i, y_i)$

La ecuación de probabilidad tiene la forma de una distribución gaussiana con la media como ubicación anterior y la desviación estándar como σ. Todos los lugares que tengan la misma distancia física con l_i obtendrán la misma probabilidad de ser elegidos como el siguiente punto de la trayectoria. Los parámetros utilizados para la generación de las trayectorias se muestran en la tabla 3.1.

Luego de creada la matriz de probabilidades se pasa a generar una matriz de distribución acumulativa sumando el $P(l_i)$ para cada ubicación l_i. Finalmente, para obtener la siguiente ubicación l_j de cualquier ubicación l_i en la trayectoria, se elige un número aleatorio R ($0 < R < 1$) y se toma como l_j la ubicación que tiene el valor en la matriz de distribución acumulativa más cercano con R.

En la imagen 3.9 se pueden observar a modo de ejemplo 6 trayectorias elegidas al azar de las 50.000 generadas por el algoritmo, cada una de ellas con 10 pasos (T=10).

Se debe considerar que la distancia recorrida entre dos pasos contiguos está definida por esta ecuación, entonces se comportará de manera aleatoria, y por lo tanto en un gran número de trayectorias generadas tendremos muestras de trayectorias representativas de todos los posibles recorridos dentro del recinto. Por otro lado, veremos que algunas trayectorias incluirán pasos contiguos en la misma localización (simulando una persona que no se está moviendo, o moviéndose lento sin cambiar de zona) y otras incluirán pasos que varíen a zonas no contiguas (simulando personas corriendo, por ejemplo).

Se procedió a analizar los datos obtenidos y se vio que un 20 % de las trayectorias daban saltos de dos o más zonas en un solo paso. Considerando que las zonas tienen 2m de ancho y que la velocidad máxima considerada para un usuario en movimiento es de $2m/s$, se decidió descartar estas trayectorias. Esto se hizo porque utilizarlas para el entrenamiento o validación implicaría que la red aprenda de un

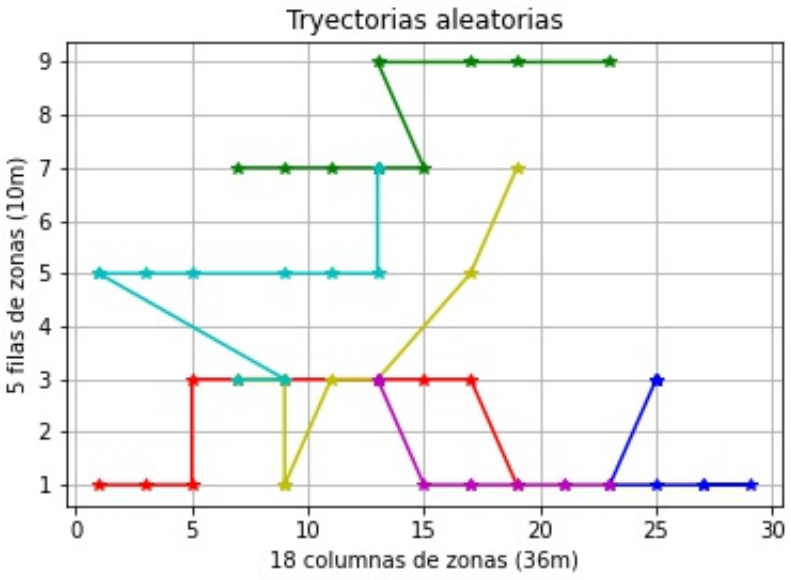


Figura 3.9: Trayectorias generados por el algoritmo

comportamiento límite, que un humano seguramente no realizaría dentro de una Universidad.

De esta manera se generaron las tres matrices, una de 30.000 trayectorias para formar el dataset de entrenamiento, una de 3.000 trayectorias para el conjunto de validación y otra de 3.000 para el conjunto de pruebas.

A partir de la matriz de trayectorias se formaron dos tipos de tensores de datos utilizados en las redes implementadas, en función de lo que cada red necesita como entrada. En lo que sigue, T es el número de pasos de las trayectorias:

- **Dataset_simple**: consta de un tensor de 3 dimensiones (Trayectorias, TimeStep o pasos, huellas) de la forma:

$$D_{sim} = \begin{pmatrix} (TimeStep)_1 & (TimeStep)_2 & ... & (TimeStep)_T \\ f_{ts1} & f_{ts2} & ... & f_{tsT} \\ f_{ts1} & f_{ts2} & ... & f_{tsT} \\ ... & ... & ... & ... \\ f_{ts1} & f_{ts2} & ... & f_{tsT} \end{pmatrix}$$

  donde $f_{tsi} = (RSSI_1, RSSI_2, ..., RSSI_n, (x_i, y_i))$ es el vector de señales RSSI, etiquetado con las coordenadas de la zona l_{tsi}. Este vector de potencia se obtiene eligiendo una huella de forma aleatoria correspondiente a la zona y al dataset para el cual estamos construyendo la trayectoria.

- **Dataset_múltiple**: consta de un tensor de 4 dimensiones de la forma:

$$D_{mul} = \begin{pmatrix} (TimeStep)_1 & (TimeStep)_2 & ... & (TimeStep)_T \\ F_{ts1} & F_{ts2} & ... & F_{tsT} \\ F_{ts1} & F_{ts2} & ... & F_{tsT} \\ ... & ... & ... & ... \\ F_{ts1} & F_{ts2} & ... & F_{tsT} \end{pmatrix}$$

  donde F_{tsi} es la matriz definida en la sección 3.1, etiquetado con las coordenadas de la zona l_i.

3.8. Implementación de algoritmos MIMO y P-MIMO

Se implementaron los modelos de múltiple entrada y múltiple salida, MIMO y P-MIMO de acuerdo a la referencia [29].

- MIMO - donde múltiples entradas RSSI f_i para cada tiempo i de la trayectoria producen múltiples ubicaciones de salida l_i.

- P-MIMO - toma múltiples RSSI y múltiples ubicaciones predichas previamente para la entrada y produce múltiples ubicaciones para la salida. Por lo que específicamente el valor de ubicación que da el algoritmo es la ubicación predicha (l_i) del paso de tiempo anterior.

Las redes neuronales se definen en Keras como una secuencia de capas, para estos dos modelos el contenedor de cada capa es la clase Sequential. Hay dos tipos de modelos disponibles en Keras el modelo secuencial y la clase de modelo que se usa con API funcional, esta última es la utilizada para la implementación DL-RNN. La capa de entrada LSTM se especifica mediante el argumento *input_shape* en la primera capa oculta de la red.

Para MIMO se utilizó una capa recurrente LSTM y para P-MIMO se utilizaron dos de estas capas, de manera que la salida de la primera capa ingrese como entrada en la segunda. Se configuraron las capas de entrada, salida y las ocultas con sus hiperparámetros correspondientes (se muestran en el apéndice 1 estos modelos).

Para minimizar la función pérdida (loss) en el entrenamiento en ambos modelos, se eligió la siguiente, acorde a la bibliografía estudiada:

$$Loss = \frac{\sum_{i=1}^{T} \|\hat{l}_i - l_i\|_2}{T}$$

donde l_i representa la posición verdadera de la muestra i-ésima, $\tilde{l}_i$ presenta la posición predicha de la muestra i-ésima.

Esta función no es predeterminada de Keras por lo que se definió la misma en Python, a fin de ser utilizada en la compilación de la red LSTM.

En la imagen 3.10 se representa el modelo P-MIMO implementado.

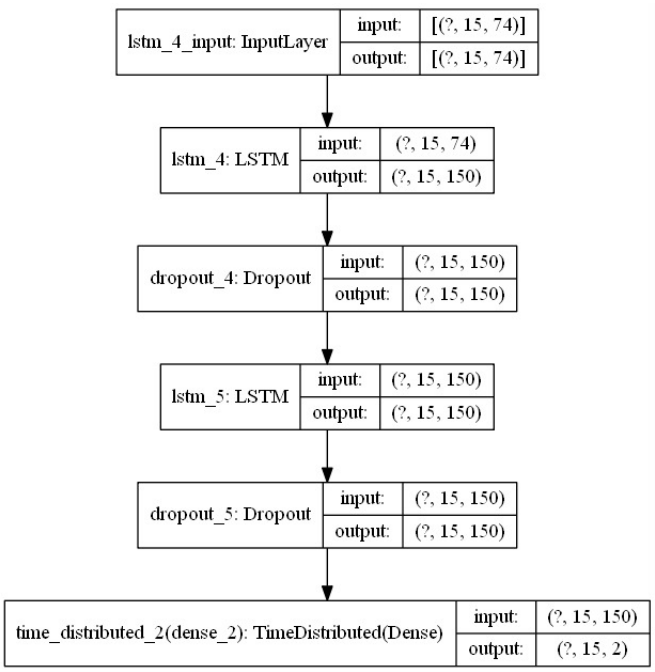


Figura 3.10: Arquitectura PMIMO

Parámetro	Valor
Tipo de RNN	LSTM
Tamaño de la trayectoria	15
N° de neuronas por capa oculta	150
Dropout	0.2
Optimizador	Adam

Tabla 3.2: Hiperparámetros utilizados en los algoritmos de MIMO y PMIMO

3.8.1. Hiperparámetros

En cualquier entrenamiento de las redes neuronales existen hiperparámetros que se deben ajustar en función de cómo se ha desarrollado el entrenamiento, de la evolución de la función de coste, de la calidad de la salida de la red o del tiempo de ejecución de la misma. En la tabla 3.2 se resume la configuración de los hiperparámetros seleccionados, siendo igual para los dos algoritmos propuestos.

A fin de evaluar estos modelos se utilizó para el entrenamiento el *dataset_simple* para los ambos casos. Como se mencionó, el mismo consta de 30000 trayectorias de 15 pasos (T=15), donde cada paso cuenta con una lista de huellas y la coordenada correspondiente a la ubicación.

3.8.2. Fase en Línea

Para la fase en línea la ubicación de salida l_i aparecerá en varios TimeStep, l_i^j es la ubicación de salida l_i del time step j.

Ejemplo: en el time step 1, l_1^T se estima con la información de T-1 pasos anteriores, en el time step 2, el número de pasos anteriores disminuye a T-2, y así hasta el time step T-1 donde l_{T-1}^Tse predice sin paso previo. Por lo tanto, usando el promedio de la ventana deslizante, se promedia el error de la ubicación predicha l_T en múltiples pasos de salida y se aumenta la precisión de localización.

El resultado final de salida l_T será el promedio del conjunto de salida anterior:

$$l_T = \frac{\sum_{j=1}^{T-1} l_T^i}{T-1}$$

3.9. Implementación del Algoritmo DL-RNN

Al igual que los modelos anteriores DL-RNN utiliza secuencias de mediciones para aprender la correlación de la trayectoria de los usuarios.

Para este modelo nos basamos en el documento [33], el mismo consta de dos RNN conectadas en cascada. La primera RNN es para la coincidencia de ubicación, hace coincidir las mediciones de huellas digitales del historial con las ubicaciones estimadas correspondientes. La segunda RNN se utiliza para el filtrado de ubicación, que toma la última capa de la primera RNN como entrada y genera una estimación de ubicación más precisa.

Como se vio anteriormente, la función de pérdida elegida para este modelo consta de dos partes: el error de Location matching RNN y el error de Location filtering RNN. El total de las pérdidas L se expresa de la siguiente manera:

$$L(\{l_1,...,l_s\},\{\tilde{l}_1,...,\tilde{l}_s\},\{\hat{l}_1,...,\hat{l}_s\}) = \frac{\lambda_1}{2}\sum_{t=1}^{s}\|\hat{l}_t - \ l_t\|^2 + \frac{\lambda_2}{2}\sum_{t=1}^{s}\|\hat{l}_t - \ l_t\|^2$$

donde $\|.\|$ es la distancia euclidiana, λ_1 y λ_2 representan los pesos de las pérdidas 1 y 2 respectivamente y $\lambda_1+\lambda_2 = 1$.Esta función no es predeterminada de keras por lo que se definió la misma en Python, a fin de ser utilizada en la compilación de la red LSTM.

En la figura 3.11 se muestra la estructura de la red implementada.

3.9.1. Hiperparámetros

En la tabla 3.3 se resume la configuración de los hiperparámetros utilizados para este modelo.

En este caso para la evaluación de este modelo se utilizo el **dataset_múltiple**. El mismo cuenta con 5000 trayectorias de 15 pasos (T=15), donde cada paso consta de una matriz de huellas de la ubicación.

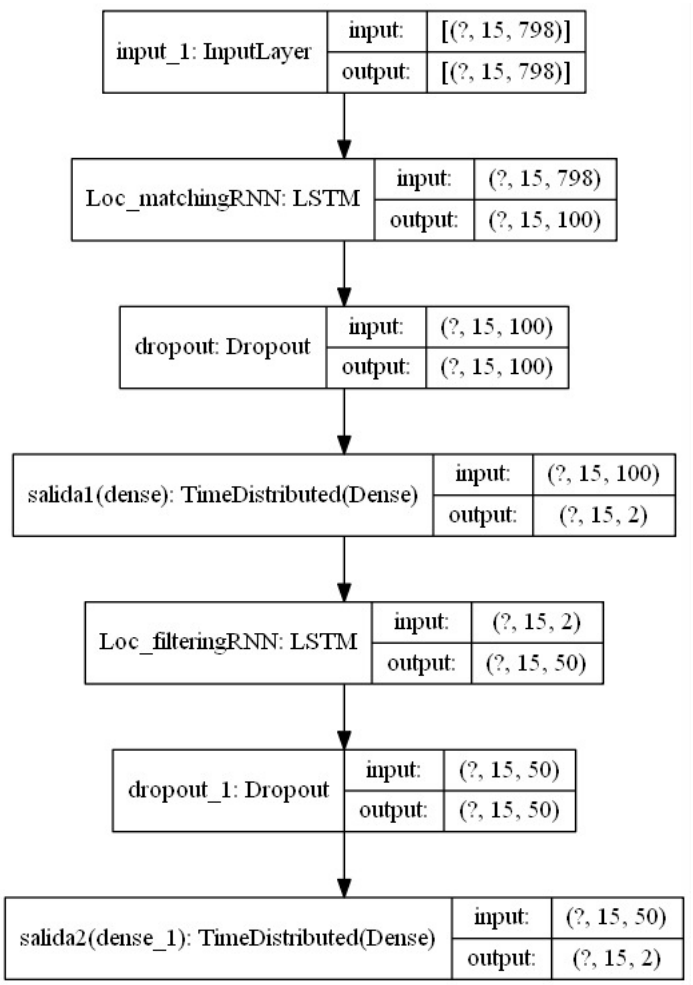


Figura 3.11: Arquitectura DL-RNN Implementada.

3.10. Elección de los Hiperparámetros

La elección de los hiperparámetros fue de manera similar para los tres modelos, por lo que se desarrolla la obtención de cada uno de ellos de manera general. Los mismos se determinaron a través del conjunto de validación y se seleccionó por valor óptimo de búsqueda. Se encontró que los hiperparámetros de mayor impactos son: la cantidad de épocas y el tamaño del lote. El parámetro epochs refiere a cuántas veces el conjunto de entrenamiento recorre el modelo de la red, la elección de la cantidad de epochs que se utilizan en cada modelo se realiza mediante pruebas, eligiendo el más adecuado.

En la figura 3.12 se puede observar a modo de ejemplo los valores de la función loss en cada una de las epochs, al aumentar este parámetro el rendimiento del modelo va mejorando hasta obtener un comportamiento asintótico. Se aplica un mecanismo de early stopping para que finalice el entrenamiento en el momento en que se observe que el error del conjunto de validación (val_loss) crece durante al

Parámetro	Valor
Tipo de RNN	DL-RNN
Tamaño de la trayectoria	15
N° de neuronas en LocMatchingRNN	100
N° de neuronas en LocFilteringRNN	50
Dropout	0.1
Optimizador	RMSprop(1e-3)
λ_1	0.1
λ_2	0.9

Tabla 3.3: Hiperparámetros utilizados en el algoritmo DL-RNN

menos 5 épocas.

El batch es un subconjunto reducido de las trayectorias para entrenar la red durante cada época. Al igual que para las épocas se realizaron varias pruebas, encontrando para el caso de MIMO y P-MIMO que utilizando un batch de 30 trayectorias se obtiene un buen rendimiento y no lleva mayor costo computacional. En el caso de DL-RNN se llegó a un batch de 100 trayectorias como valor más adecuado.

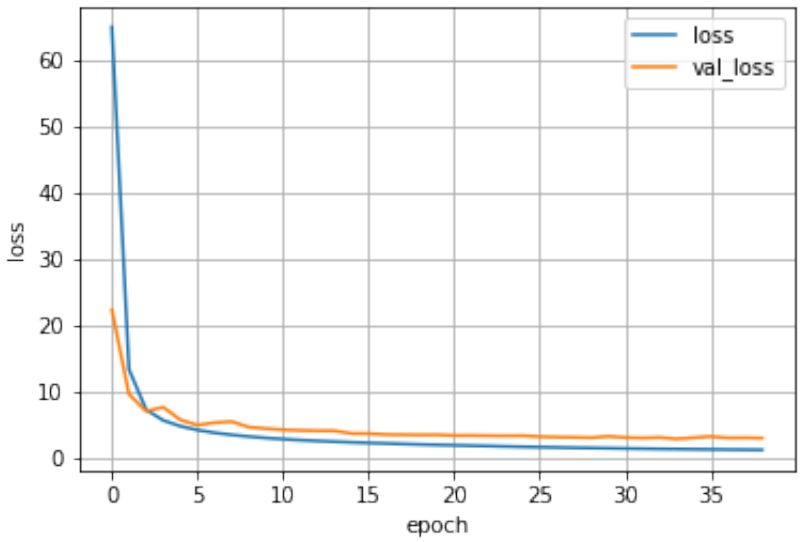


Figura 3.12: Rendimiento del modelo en función de la cantidad de epochs.

Capítulo 4

Evaluación del Sistema

Con las huellas filtradas y procesadas, para considerar únicamente los APs conocidos y eliminar huellas duplicadas se generaron tres conjuntos (para entrenamiento, validación y testing). Para cada conjunto se generó un cierto número de trayectorias de 15 pasos, eliminando aquellas trayectorias que implicaban saltos de más de 2 mts en un solo paso. El resultado fue de 30.000 trayectorias para entrenamiento, 3.000 para validación y 3.000 para testing.

Luego de elegidos los hiperparámetros se procedió a entrenar y evaluar las tres redes estudiadas. De aquí en más, se considerará el problema de regresión, por lo que las evaluaciones se realizarán midiendo el error promedio de la red para los casos de testing, en metros. Como las redes MIMO, P-MIMO y DL-RNN tienen múltiple salida, se toma como error la distancia euclidiana entre l_i y $\tilde{l}_i$ para $i = T$, es decir, para la salida del último instante de tiempo considerado.

Luego, en ciertos casos que se encuentran de interés, se analizarán los resultados como problema de clasificación mapeando las coordenadas de las ubicaciones predichas a los identificadores de las zonas y evaluando el sistema en función de la precisión.

4.1. Rendimiento de los Modelos

En primer lugar se compararon los modelos MIMO y P-MIMO y se encontró, de acuerdo a lo informado en la referencia, que el modelo P-MIMO muestra mejor resultado que el modelo MIMO. Se tiene un error promedio de 2,80 m en MIMO y de 2,39 m en P-MIMO.

Para la red DL-RNN, el error promedio es de 2,0 m. En la gráfica 4.1 se muestran todos los errores de las predicciones del modelo para el conjunto de testing. Se puede ver que el máximo error es 19,1 m, pero el 99 % de los puntos se encuentran por debajo de los 7,1 m.

La figura 4.2 muestra un ejemplo particular del conjunto de testing. Se debe recordar que, si bien la función loss va considerando las diferencias entre la línea azul y la línea roja para cada paso, el error final de la red sólo considerará la diferencia en el último. Esto es debido a que nuestro sistema utilizará únicamente

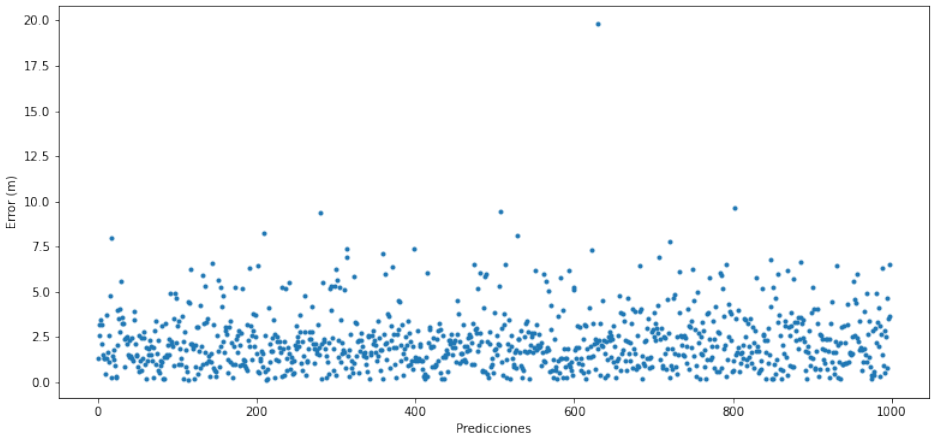


Figura 4.1: Gráfica del error en los ejemplos de testing para DL-RNN.

esa salida para devolverle la predicción al usuario.

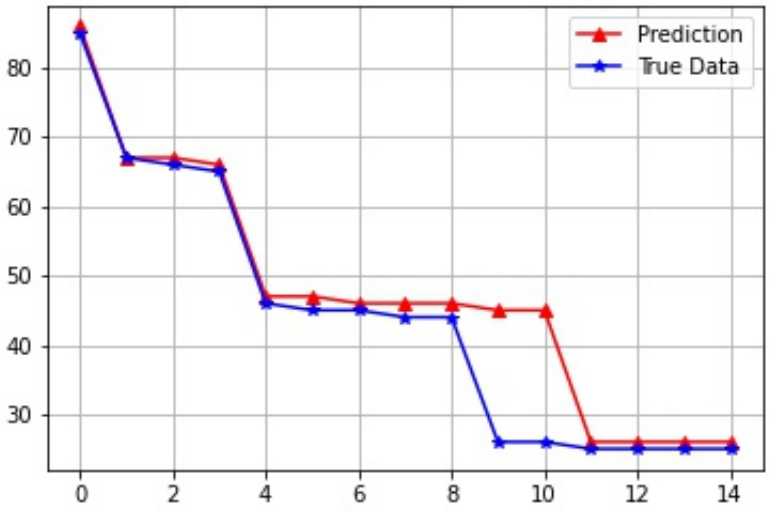


Figura 4.2: Trayectoria real vs predicción de una DL-RNN.

Se comparan entonces de forma empírica los resultados del testing entre MIMO, P-MIMO y DL-RNN para los mismos dataset de entrenamiento, validación y prueba. En la tabla 4.1 se muestran los errores medidos y los tiempos de entrenamiento en cada caso. Se puede deducir que utilizando el modelo DL-RNN se obtiene un mejor rendimiento a menor costo computacional. Esto era de esperar ya que la estructura DL-RNN no da una estimación directa de las medidas RSSI, sino que primero da una aproximación de la ubicación para luego filtrarla y dar

una ubicación más precisa, generando una mejor estimación del trayecto que al usar los otros modelos.

	MIMO	P-MIMO	DL-RNN
Error (m)	2,8	2,4	2,0
Tiempo (hs)	12,0	6,0	2,5

Tabla 4.1: Comparativo de error medio por red

En cuanto a la implementación final, se vio que la estructura DL-RNN solicita múltiples entradas del usuario para cada paso. Esto implicaría tomar varias huellas por segundo y sin garantía de que fueron tomadas en la misma zona. Además, la capacidad de procesamiento necesaria para entrenar la red con un tensor 4-dimensional es muy alta, llevando períodos de más de 24 hs de entrenamiento utilizando los servidores en la nube. Algo similar sucede cuando se crea en tiempo real (en la fase online) la matriz de entrada para realizar la clasificación.

A pesar de todo esto, es posible entrenar la DL-RNN con una única entrada, lo cual viabiliza su utilización para una implementación real. Teniendo esto en cuenta y viendo que el entrenamiento de una red DL-RNN en estas condiciones llevó mucho menos tiempo que que para la P-MIMO, se seleccionó la DL-RNN como modelo para el sistema final. A pesar de eso, se vio que la red P-MIMO también era implementable y con resultados también positivos.

Mapeando las salidas de la red DL-RNN de coordenadas a localizaciones, se vuelve al problema de clasificación. En el Anexo 2 se muestra la figura B.1, que es la matriz de confusión generada para las 90 zonas. Como se puede observar, la mayoría de los valores se encuentran próximos a la diagonal de la matriz, lo que indica que la mayoría de confusiones son entre zonas adyacentes en una misma columna. También existen diversas confusiones entre zonas adyacentes en una misma fila, se ven como franjas paralelas a la diagonal debido al salto de numeración entre zonas (se numeraron por columna: 0-17, 18-35, 36-53, 54-71 y 72-89), con una diferencia de 18 zonas hacia arriba o abajo, causa de este fenómeno.

De aquí en más se realizarán evaluaciones de desempeño para la red DL-RNN en función ciertos parámetros del sistema considerados, como ser cantidad y largo de las trayectorias, resolución espacial y cantidad de APs percibidos con buen nivel de señal (menores a -70 dbm). Para el análisis del largo de trayectorias se evalúan las tres redes, buscando confirmar que la elección tomada es la correcta independientemente de este factor.

4.2. Resultados en Función del Largo de las Trayectorias

Si bien la literatura utilizada recomienda realizar el entrenamiento de la red con trayectorias de 15 pasos, se decidió analizar el rendimiento de las redes con trayectorias de diferentes tamaños.

La motivación para esto, es tanto práctica como teórica: dado un sistema que toma un paso por segundo, se debe esperar 20 segundos del usuario enviando datos

hasta completar una trayectoria real. Mientras tanto, la red clasificará utilizando una trayectoria con la información disponible hasta el momento, ya que el servidor rellenará los pasos sin información repitiendo la huella más antigua enviada por el usuario. Esto deriva en un tiempo muy largo de estabilidad del sistema. Por otro lado, se evaluará qué tanto aporta un histórico tan prolongado a la precisión del sistema.

Se procedió a armar conjuntos de trayectorias de 5, 10, 15 y 20 pasos, a partir del mismo dataset de huellas, siguiendo el procedimiento detallado anteriormente. Se entrenaron las redes MIMO, P-MIMO y DL-RNN, observando que el error baja conforme las trayectorias se hacen más largas, con lo cual se puede deducir que efectivamente a trayectorias más largas, la red tendrá mayor precisión. Lo anterior se puede apreciar de manera clara en la imagen 4.3.

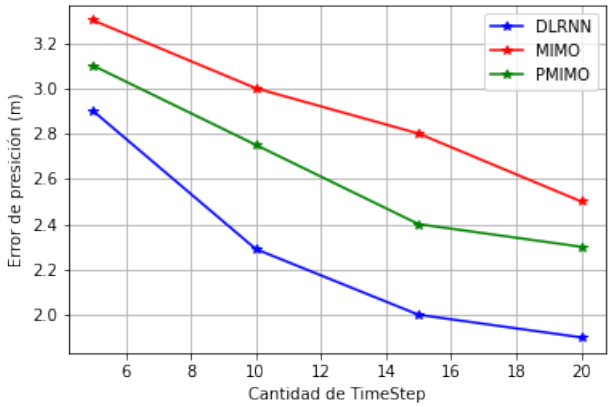


Figura 4.3: Comparativo de errores por largo de trayectoria.

Sin importar el largo de la trayectoria, la red DL-RNN resultó en los errores más bajos de predicción. Los mismos se detallan en la tabla 4.2.

Pasos	Error (m)
5	2,9
10	2,29
15	2,0
20	1,9

Tabla 4.2: Error de DL-RNN por cantidad de pasos.

En suma, vemos que si bien agrega precisión el trabajar con 20 huellas, la diferencia no es remarcable (tan solo 10 cm). Por este motivo, en la práctica se decidió utilizar la DL-RNN de 15 pasos para la implementación final del sistema, lo que permite mantener un error aceptable mientras que se llega a la estabilidad en tiempos más razonables.

4.3. Resultados en Función del Tamaño de las Zonas

Considerando la poca cantidad de APs con buen nivel de señal que se perciben desde algunas zonas, se decidió realizar pruebas con una resolución espacial más baja, viendo si el error de las redes también bajaba en el proceso. Para esto, se procedió a concatenar bloques de 4 ó 6 zonas contiguas, formando cuadrados de 4 metros de lado o rectángulos de 4 metros de ancho por 6 metros de largo como se puede ver en la figura 4.4. Esto redujo la cantidad de zonas, las 90 zonas previas se transformaron en unas nuevas 18 zonas.

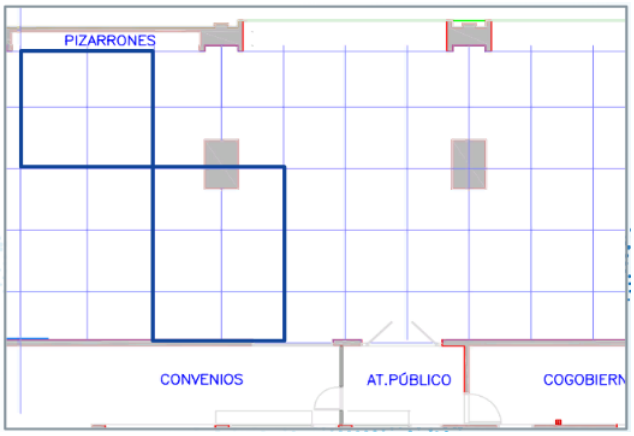


Figura 4.4: Reagrupación de las zonas

Se utilizaron nuevamente las 36.000 trayectorias generadas anteriormente para 90 zonas, re etiquetando sus ubicaciones según el nuevo mapa. Esto se hizo así, en contraposición a generar trayectorias a partir del dataset de huellas con ubicaciones mapeadas ya que se entendió que de esta manera se evita seleccionar aleatoriamente huellas del etiquetado nuevo que corresponden a sub-zonas del etiquetado anterior que sean inconsistentes con la trayectoria generada.

Para visualizarlo con un ejemplo, consideremos una trayectoria en las 90 zonas que muestra una secuencia de zonas visitadas $(1, 2, 3, 4)$. En el nuevo mapeo a 18 zonas, esta misma trayectoria resulta en una secuencia $(1, 1, 2, 2)$. Se busca evitar que se le asignen, huellas correspondientes al trayecto $(2, 1, 4, 3)$ del mapeo inicial, que podría suceder al asignar una huella aleatoria del mapeo nuevo, ya que sería muy errático para una persona real caminando y asumiría desplazamientos de más de 2 m entre pasos.

Luego de realizado esto se entrenaron las redes P-MIMO y DL-RNN. Las figuras 4.6 y 4.5 muestran las trayectorias generadas y predichas en cada caso, observando cómo ambas convergen para la última ubicación de la misma (la que utilizará el sistema para devolver al usuario). También se aprecia que la DL-RNN tiene mejores predicciones intermedias, paliando el problema de estabilidad mencionado en la sección 4.2.

En la tabla 4.3 se puede apreciar que el error baja conforme se agrandan las zonas consideradas. En las próximas secciones se volverá a hablar del compor-

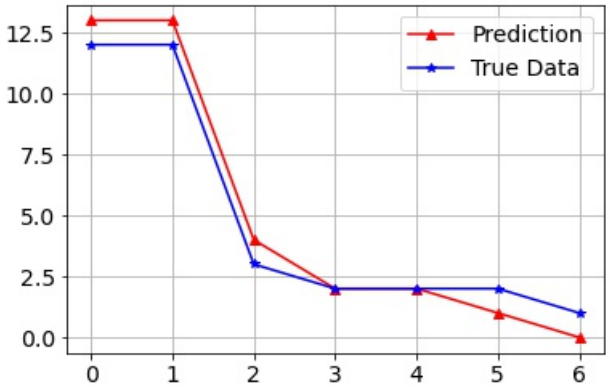


Figura 4.5: Trayectoria real y predicha para P-MIMO.

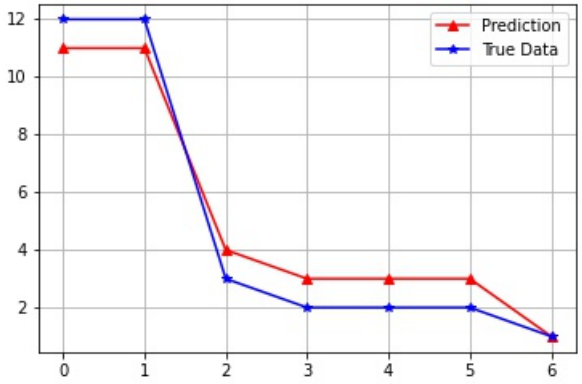


Figura 4.6: Trayectoria real y predicha para DL-RNN

.

tamiento del error en función de la resolución espacial, mostrando conclusiones adecuadas al mismo.

Zonas	Error (m)
90	2,0
18	1,5

Tabla 4.3: Error de DL-RNN en función de la resolución espacial

4.4. Resultados en Función de la Cantidad de Trayectorias

Para realizar los conjuntos de datos de entrenamiento, validación y testing, se siguió la recomendación de la bibliografía estudiada en la que se indicaba que por encima de las 10.000 trayectorias no había una mejora sustancial en el rendimiento del sistema. Sin embargo, se decidió realizar una prueba sobre la DL-RNN con 110.000 trayectorias de entrenamiento, 6.000 para validación y 6.000 para testing. Se obtuvo un error promedio de 2,1 m.

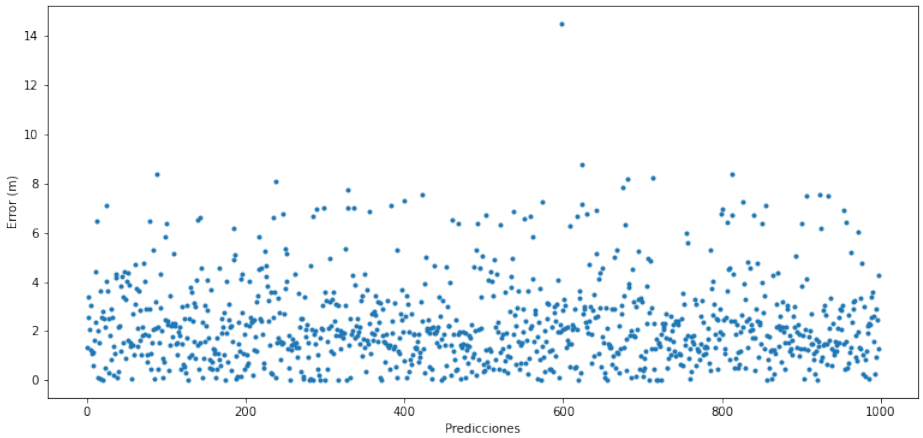


Figura 4.7: Dispersión del error para DL-RNN con 100.000 trayectorias.

En la figura 4.7 se puede observar que la distribución de error se comporta de manera similar que con un tercio de las trayectorias de entrenamiento, obteniendo un error máximo de 14,5 m y un 99 % de los datos por debajo de 6,2 m. Se ve que el sistema no mejora notablemente, pero los costos computacionales sí son elevados, como se puede apreciar en la tabla 4.4, donde t_1 refiere al tiempo de generación de trayectorias y t_2 al tiempo de entrenamiento de la red.

Trayectorias	Error (m)	t_1	t_2
35.000	2,0	1,0h	2,0h
110.000	1,9	3,0h	4,0h

Tabla 4.4: Resultados para DL-RNN en función de la cantidad de trayectorias

Se puede concluir que aumentar la cantidad de trayectorias utilizando el mismo dataset inicial de 884 huellas no mejoró sensiblemente el error del sistema, por lo que se buscan otras formas alternativas a esta.

4.5. Resultados en Función de la Cantidad de Huellas

Como se mencionó anteriormente, en el procesamiento de los datos se realiza un filtrado previo al entrenamiento para descartar huellas idénticas del relevamiento tomadas en la misma zona. Esto se realiza con el fin de evitar el sobre entrenamiento de la red.

Este filtrado puede reducir considerablemente el tamaño del conjunto de huellas que se utilizan para realizar trayectorias para el entrenamiento, requiriendo un esfuerzo mayor de trabajo de campo y aumentando los costos de implementación de la solución. Por estos motivos, se realiza una comparación en función de la cantidad mínima de huellas únicas, tomadas en una misma zona.

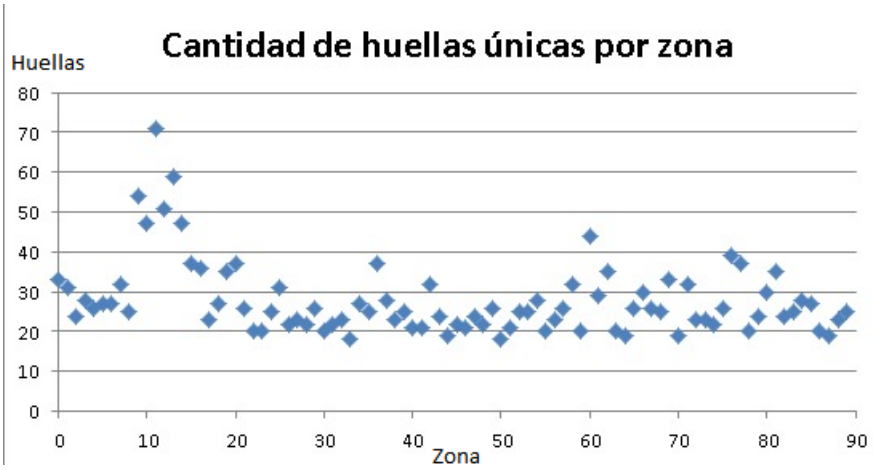


Figura 4.8: Cantidad de huellas por zona.

Se procedió a hacer un relevamiento más largo, de 2535 huellas únicas, permaneciendo al menos 4,5 minutos en cada una de las zonas, tiempo en que se vio que en la mayoría de las ubicaciones se obtuvo por lo menos 20 muestras diferentes. Este relevamiento llevó 8 hs de trabajo en campo y como se ve en la 4.8, tan solo en 4 lugares se encontró que habían menos de 20 (17 a 19 huellas únicas).

Se volvieron a generar 30.000 trayectorias para entrenamiento, 3.000 para evaluación y 3.000 para testing, se entrenó la DL-RNN y se alcanzó un error de 1,6 m como se ve en la figura 4.9. Un 95 % de los casos resultaron en un error menor a 4 m y un 72 % menor a 2 m. Esto implica una reducción en un 20 % del error encontrado para el dataset de 884 huellas, que había alcanzado 2,0 m de error.

Luego, se procedió a evaluar el rendimiento del sistema para 18 zonas con este nuevo dataset, repitiendo el procedimiento de la sección anterior. Se logró aquí un error de 1,7 m, según se muestra en la figura 4.10, con un 92 % de los casos con error menor a 3 m y un 68 % menor a 2 m.

Se concluye entonces que el error en metros encontrado en las predicciones no disminuyó al reducir la resolución espacial del mapa de huellas. El efecto observado con un dataset de huellas más pequeño puede estar sesgado a la selección de huellas para testing, y con un dataset más representativo el error converge independientemente del tamaño de las zonas elegidas.

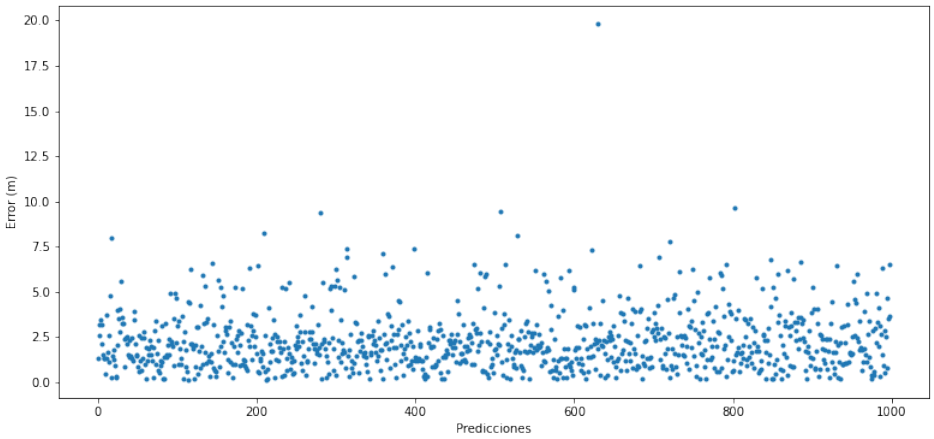


Figura 4.9: Error para DL-RNN en 90 zonas.

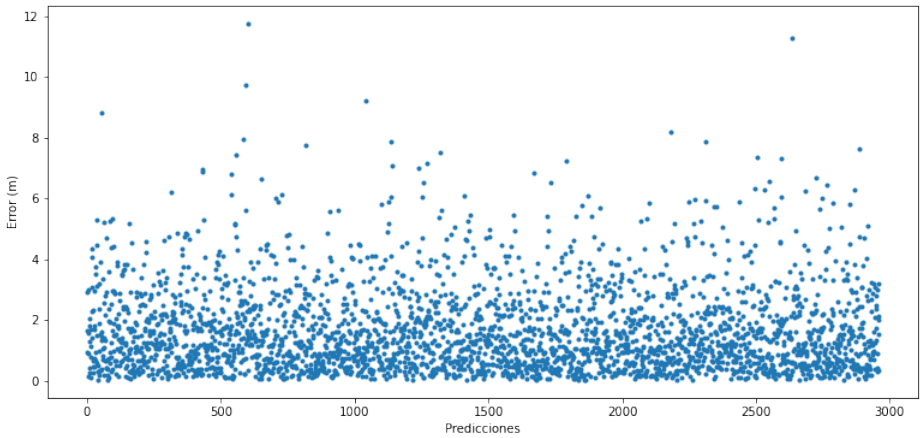


Figura 4.10: Error para DL-RNN en 18 zonas.

En cambio, sí se encontró una mejora en el problema de clasificación, logrando subir la precisión de 32 % para 90 zonas a 64 % para 18 zonas. Esto es razonable ya que, si bien el error es el mismo (a menos de 10 cm), el tamaño de las zonas elegidas aumenta considerablemente, y un mayor porcentaje de las predicciones caerán en las ubicaciones clasificadas.

4.6. Resultados en Función de la Cantidad de APs

Analizando los conjuntos de datos realizado en los relevamientos, se notó que algunas zonas tenían tan solo 2 APs con un nivel de señal alto (menor a -70 db)

y 6 APs con un nivel de señal bajo pero no nulo (menor a -90 db). Es de interés verificar si la falta de cobertura impactó en el rendimiento final del sistema, y por eso se decidió realizar un nuevo relevamiento agregando dos APs en el área de interés y analizar los resultados.

Conociendo la ubicación de los APs provistos por ANTEL para Facultad de Ingeniería (marcados en rojo), se eligieron dos ubicaciones en las que se entendió que se complementaría la cobertura existente (marcados en verde). En cada uno de estos sitios se colocó un equipo Huawei modelo CPE B593. La ubicación de estos equipos se muestra en la figura 4.10. Se procedió a relevar, obteniendo al menos 9 huellas únicas en cada una de las 90 zonas y entrenando la red DL-RNN con las mismas 30.000 trayectorias generadas anteriormente.

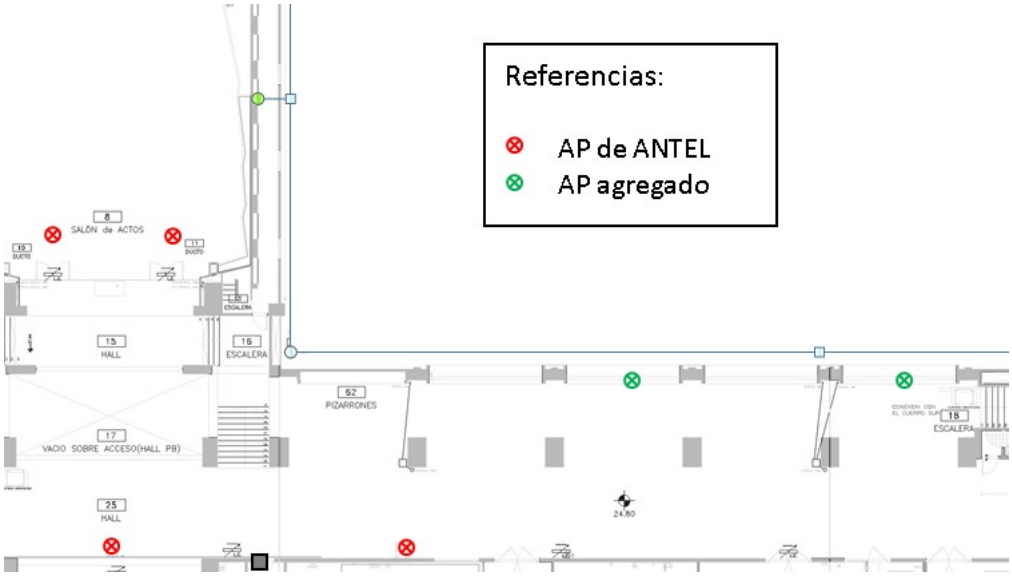


Figura 4.11: Ubicación de los APs adicionales.

En estas circunstancias se logró alcanzar un error de 1,7 m, viendo que aumentar la cobertura disponible no permite mejorar el rendimiento al sistema de regresión implementado.

Por consistencia con otras pruebas, se entrenó nuevamente la DL-RNN con 30.000 trayectorias para las 18 zonas estudiadas anteriormente. Como se ve en la 4.12, se alcanzó un error de 1,7 m, con un 94 % de los datos presentando error por debajo de los 3 m y un 72 % por debajo de los 2 m.

Luego, se procedió a pasar al problema de clasificación, realizando la matriz de confusión mostrada en la figura 4.13,y una precisión del 69 %, un 5 % mejor que en las mismas condiciones sin estos dos APs. En la misma se puede observar que la gran mayoría de las equivocaciones son en zonas adyacentes, ya sea en ubicaciones con etiquetas contiguas o con una diferencia de 9. Algunas zonas se encontraron difíciles de graficar, ya que la aleatoriedad con la que se generan las trayectorias no permite garantizar que las mismas terminan en cierta zona. Se ve que las zonas 7, 8, 16 y 17 tienen menos casos de test que las representan. Esto se da porque se

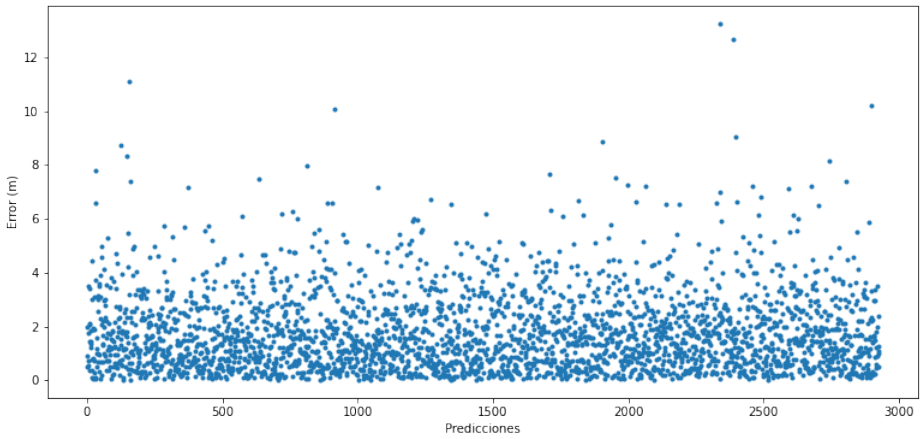


Figura 4.12: Precisión de la DL-RNN para el conjunto de testing con dos APs adicionales.

encuentran en un extremo del área de interés y tienen menos zonas adyacentes, por lo que la probabilidad de que una trayectoria de 15 pasos generada aleatoriamente termine en ellas, es más baja que para el resto.

Por último, se realizó un filtrado de APs, considerando únicamente aquellos provistos por ANTEL para Facultad de Ingeniería y los dos agregados específicamente para estas pruebas (aquellos indicados en la figura 4.11), con el fin de evaluar si los APs lejanos aportan o no información. Se encontró que el error subió a 3,4 m, lo que indica que efectivamente estos APs son necesarios para el correcto funcionamiento del sistema.

4.7. Comparación del Sistema Locindoor con Posifi

Para poder comparar los resultados de las redes neuronales recurrentes con la implementación de "Localización Indoor Basado en Wi-Fi (Posifi)" [11] con la solución presentada en este trabajo (LocIndoor), se decidió implementar ambas soluciones y evaluarlas dentro de las mismas condiciones. Se utilizaron 1579 huellas tomadas en Facultad de Ingeniería (UdelaR) en el área de interés de este proyecto, considerando la distribución de 18 PR y se dividieron en tres dataset, manteniendo la proporción de huellas por zona en cada uno de ellos:

- 70 % de los datos para el entrenamiento.
- 10 % de los datos para validación.
- 20 % de los datos para testing.

Con estos dataset se entrenó el sistema Posifi, llegando a unos valores de precisión de 37 % según se puede ver en la figura 4.15. Se generó la matriz de confusión que se puede ver en la figura 4.14, con el fin de poder comparar el rendimiento

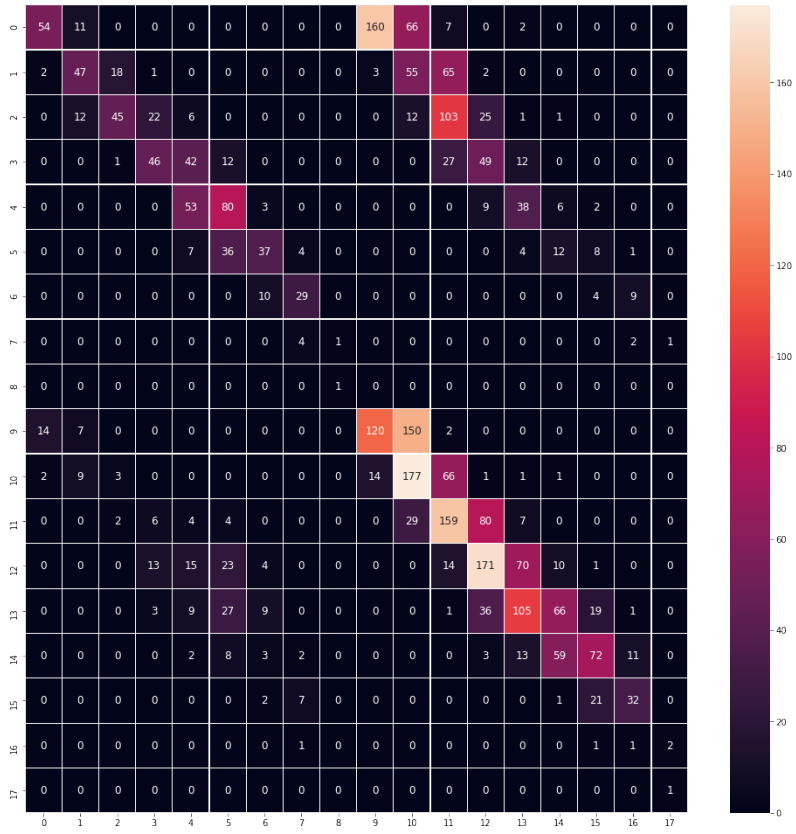


Figura 4.13: Matriz de confusión para DL-RNN en 18 zonas con dos APs adicionales.

en distintas zonas. Se cree que la disminución en precisión respecto a la solución presentada para el MNAV se debe a que el área de interés de este proyecto es un espacio sin divisiones intermedias, el hall de primer piso de facultad. El trabajo de referencia mostró también zonas más amplias y con mampostería entre muchas de sus zonas. Además, se corroboró que utilizó el dataset sin eliminar huellas duplicadas por zona (aunque al ser zonas más grandes presentaba un porcentaje sustancialmente menor de las mismas).

Para red DL-RNN se construyeron tres conjuntos de trayectorias de 15 pasos, utilizando como base los mismos set de datos. El sistema alcanzó un error promedio de 2.4 m.

Para poder realizar la comparación, se asignó a cada par coordenado (x, y) la etiqueta de su zona correspondiente, minimizando la distancia de cada predicción a los puntos de referencia considerados. Con las predicciones clasificadas, se procedió a graficar la matriz de confusión de la figura 4.14. Se calculó una precisión de 67 %, según se ve en la figura 4.17.

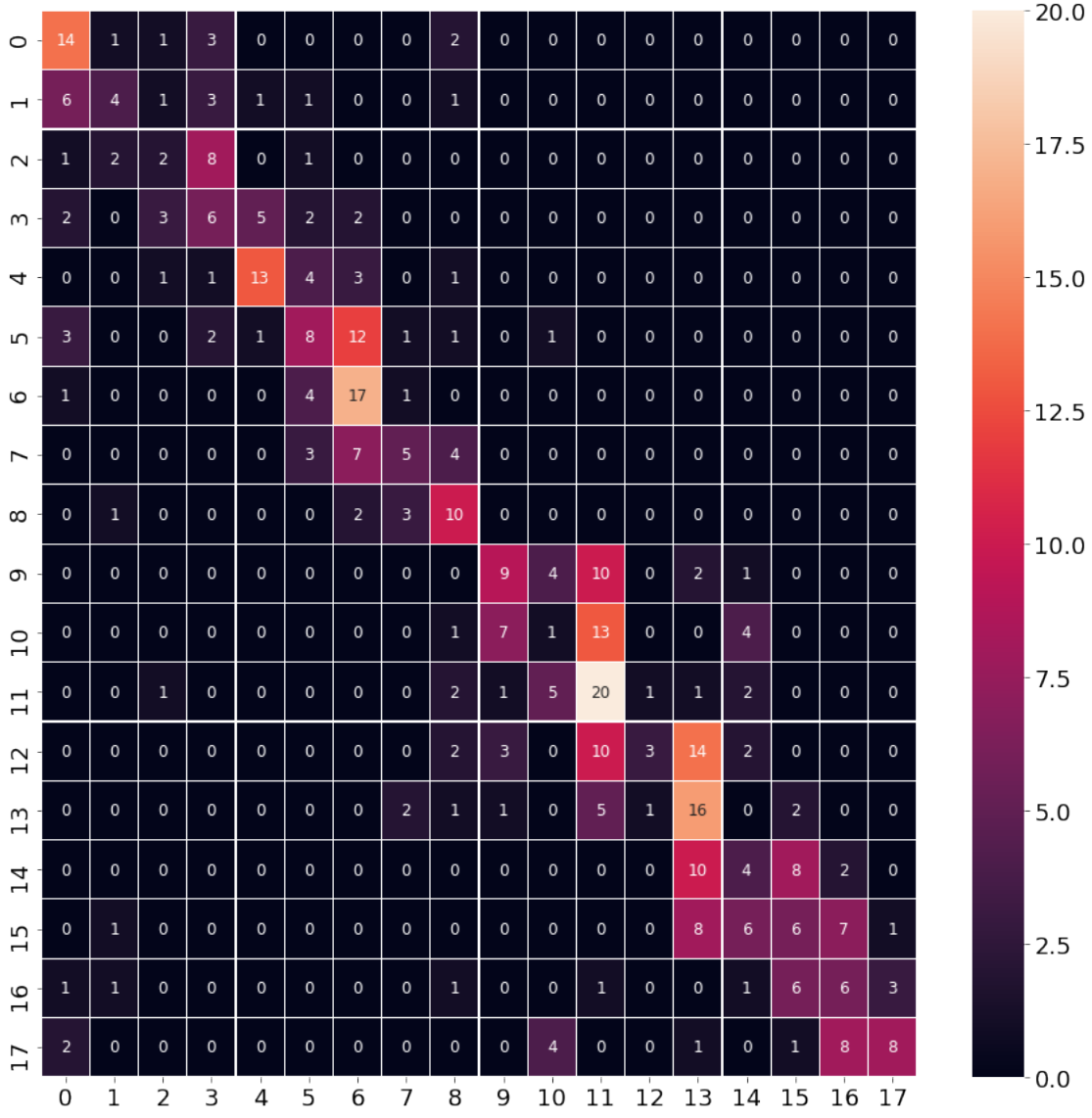


Figura 4.14: Matriz de confusión para el sistema Posifi

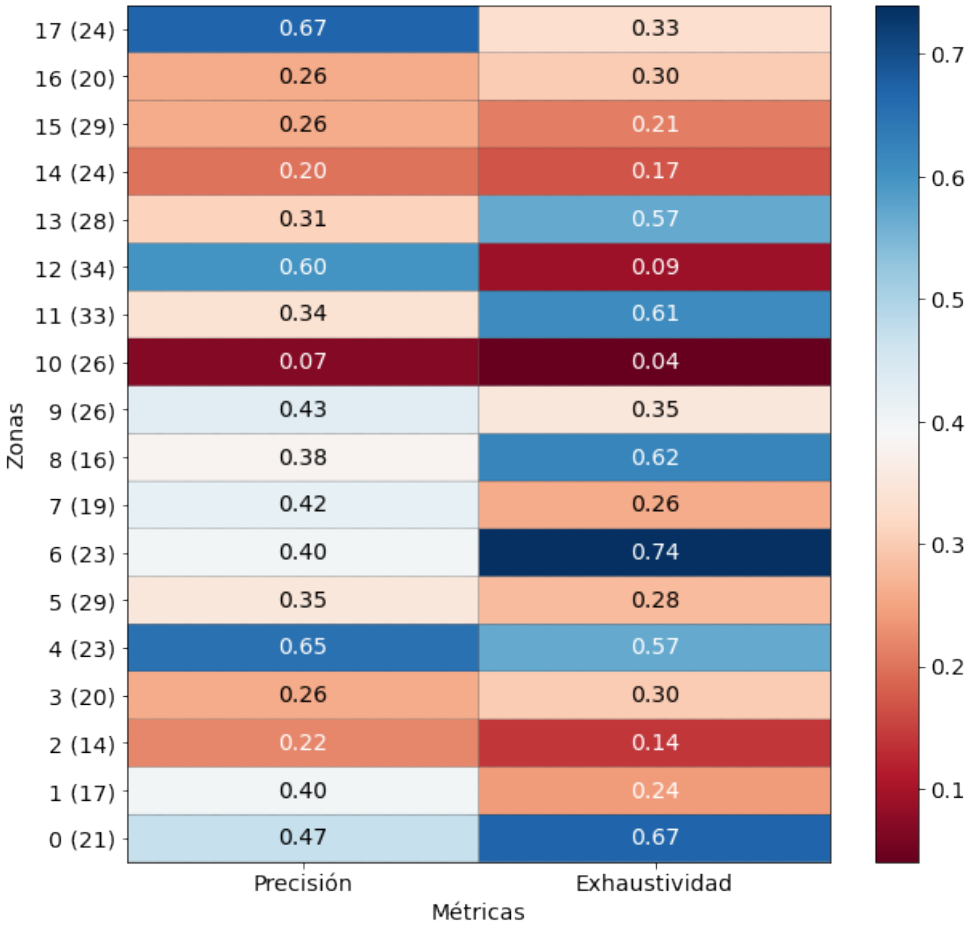


Figura 4.15: Métricas de clasificación para el sistema Posifi

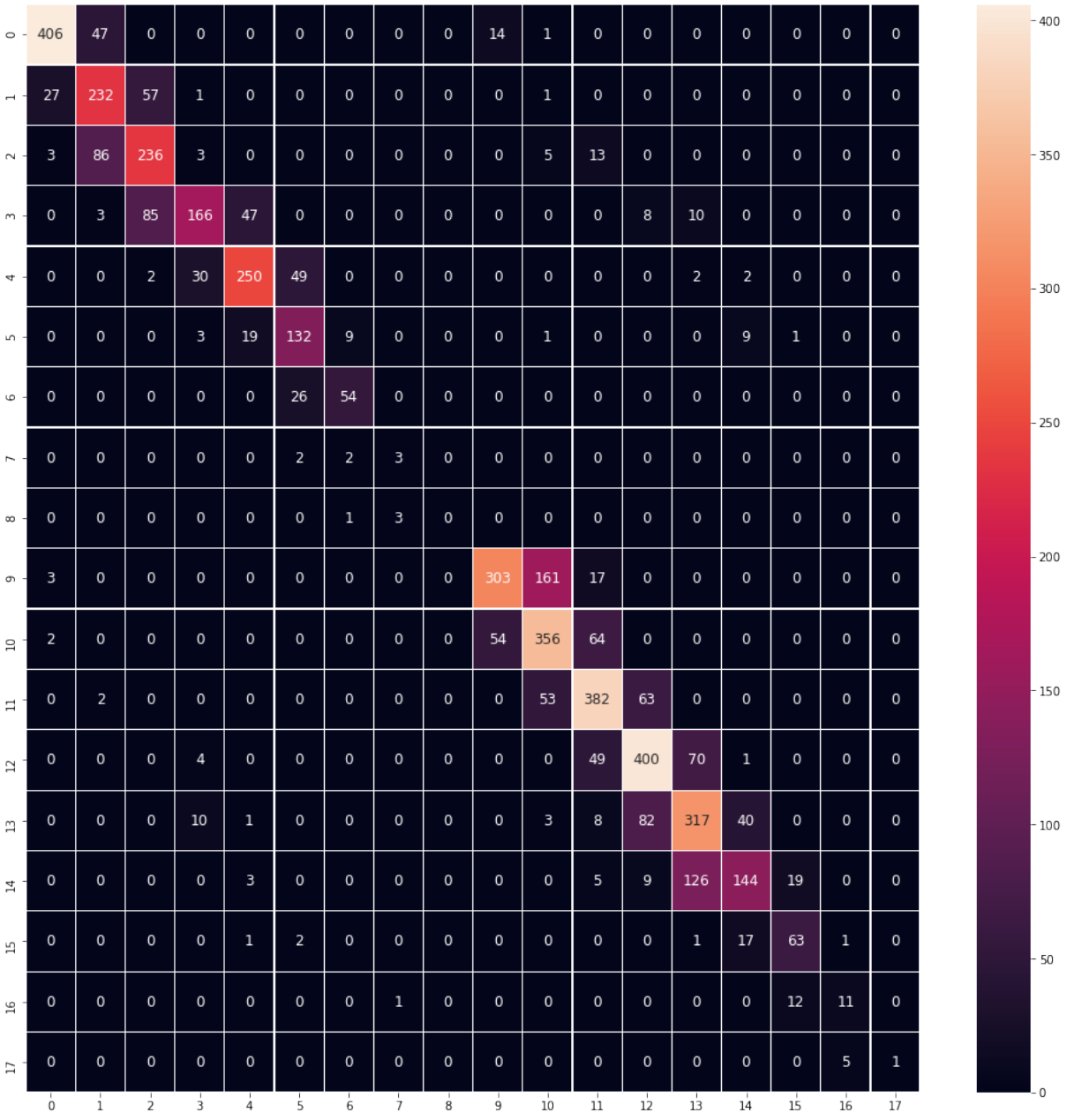


Figura 4.16: Matriz de confusión para el sistema Locindoor

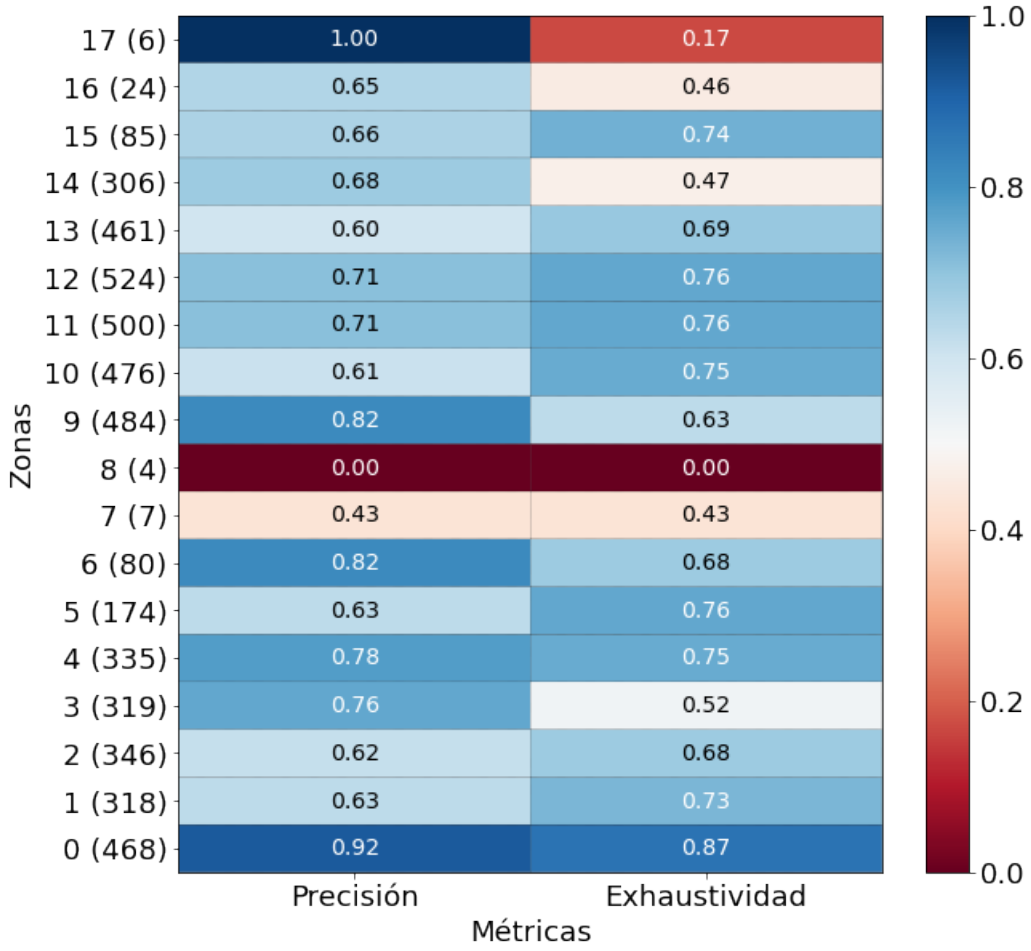


Figura 4.17: Métricas de clasificación para el sistema Locindoor.

Capítulo 5

Conclusiones y Trabajo a Futuro

En el presente trabajo se logró implementar un sistema de localización basado en Wi-Fi que, sujeto al escaneo propio del dispositivo Android del usuario, localiza al mismo dentro del Hall de primer piso de Facultad de Ingeniería, UdelaR.

Se estudiaron e implementaron Redes Neuronales Recurrentes como algoritmo de clasificación para resolver el problema de huellas digitales, y se realizaron diversas pruebas para analizar los valores óptimos de largo de trayectoria a considerar, tamaño de las zonas, cantidad de huellas a tomar en la fase de relevamiento.

Se estudiaron técnicas de *crowdsourcing* y *crowdsensing*, logrando implementar una técnica de *crowdsourcing* que permite al usuario aportar información sobre su ubicación actual en determinados escenarios.

5.1. Análisis de Desempeño

A continuación se realiza un breve resumen sobre el desempeño de las redes en función de las variables estudiadas: tipo de red, cantidad de APs filtrados, cantidad de zonas consideradas en la misma área de interés, cantidad de huellas relevadas por zona, trayectorias generadas para entrenamiento, largo en pasos de las trayectorias y error calculado medido en metros.

Además, se comparó el rendimiento de la DL-RNN de 15 pasos, 35.000 trayectorias, en 18 zonas, con el sistema que se tomó como base para este proyecto, consiguiendo aumentar la precisión en un 30 %.

5.2. Conclusiones Finales

El presente trabajo logró implementar un sistema de localización en interiores basado en Wi-Fi, utilizando Redes Neuronales Recurrentes en distintas arquitecturas, con un error de predicción final que se entiende adecuado para el caso de uso. Con este sistema, un estudiante de Facultad de Ingeniería puede utilizar una aplicación Android que opera en las versiones del sistema operativo que circulan en el mercado actual, para recibir una ubicación aproximada dentro de un

Red	AP s	Zonas	Huellas	Trayectorias	Pasos	Error (m)
MIMO	77	90	884	30.000	15	2,8
P-MIMO	77	90	884	30.000	15	2,4
DL-RNN	77	90	884	30.000	5	2,9
DL-RNN	77	90	884	30.000	10	2,3
DL-RNN	77	90	884	30.000	15	2,0
DL-RNN	77	90	884	30.000	20	1,9
DL-RNN	77	18	884	30.000	15	1,7
DL-RNN	79	90	884	30.000	15	1,7
DL-RNN	79	18	884	30.000	15	1,5
DL-RNN	77	90	884	110.000	15	1,9
DL-RNN	77	90	2.535	30.000	15	1,6
DL-RNN	79	18	2.535	30.000	15	1,7
DL-RNN	3	18	2.535	30.000	15	3,4

Tabla 5.1: Resumen de desempeño

área pre-determinada. Además podrá aportar información fresca para posteriores re-entrenamientos de la red predictora.

Se lograron determinar las mejores condiciones para el funcionamiento de la red, en cuanto a cantidad de huellas de RSSI relevadas, tamaño de las zonas de referencia y largo de la trayectoria del usuario a considerar, alcanzando un error estimado de 1,6 m sin necesidad de agregar infraestructura WLAN dedicada para esta aplicación. En estas condiciones, se alcanzó un 64 % de precisión. De cualquier manera, entendemos que se pierde información al pasar a clasificación, ya que el algoritmo calcula el error a la posición real del usuario y no necesariamente al centro de una zona. Es por esto que se considera como métrica principal de evaluación del sistema el error y no la precisión.

En las mismas condiciones, agregando infraestructura que complemente la cobertura existente en Facultad, no se logró bajar el error estimado de 1,6 m, pero sí aumentar la precisión del sistema de clasificación en 5 %. De cualquier manera, se vio que, despreciando el aporte de APs con poca potencia recibida, aumenta considerablemente el error estimado para la red, por lo cual se prefiere utilizar siempre toda la información disponible de APs que sabemos no sufrirán alteraciones imprevistas.

Por último se comparó este sistema, considerándolo como clasificador, a un proyecto previo que utilizaba varios algoritmos de inteligencia artificial sin memoria temporal. Se demostró una mejora del 30 % en la precisión del mismo.

5.3. Trabajo a Futuro

5.3.1. Sistema y Aplicación

A la hora de realizar las pruebas de campo se encontró que algunas funciones utilizadas en la aplicación quedaron obsoletas, quitándole a la misma la posibilidad de disparar el escaneo de señales de Wi-Fi a demanda. Esto llevó a que los resultados del análisis del sistema en campo fueran erráticos, ya que dependían de que el dispositivo por sí mismo realizara el escaneo y el comportamiento observado era variable y con un intervalo de tiempo entre escaneo demasiado alto para las trayectorias previstas.

Por cuestiones de tiempo no se pudo lograr resolver este problema, ya que se vio que implicaba realizar cambios en diversas partes del código e incluir nuevas librerías en el proyecto de Android Studio, pero se sabe que existen formas de solucionar esto. A pesar de todo, y en función de las pruebas realizadas y mostradas en la sección anterior, se entiende que el sistema funcionaría mejor que uno que utilice algoritmos de inteligencia artificial tradicionales, según se mostró en la sección 4.7.

Por otro lado, se entiende que la arquitectura del sistema podría mejorarse en cuanto a seguridad y disponibilidad. Esto se podría realizar con una redundancia sobre el servidor principal y la base de datos relacional, implementando además encriptamiento de los datos enviados entre cliente y servidor.

5.3.2. Infraestructura Dedicada

En el marco de una situación de una emergencia sanitaria mundial, y en particular nacional, no se pudo utilizar la infraestructura de WiFi instalada específicamente con propósitos de localización indoor en el Museo Nacional de Artes Visuales.

Como se mostró, la precisión del sistema mejora al agregar cierta cantidad de APs en el área de interés. Esto sugiere que, pudiendo hacer un dimensionamiento de la distribución de APs en Facultad de Ingeniería, en el que la razón de APs vistos desde cada zona y la cantidad de zonas considerada, sea más alta que lo disponible en el presente trabajo, se lograrían precisiones más cercanas al 90 %. De esta manera, se podría realizar una aplicación de usuario que permitiera guiar a un alumno hasta su salón de clases, u otras zonas de Facultad que sean de interés.

De cualquier manera, se vio que tomando las red RNN como solución a un problema de regresión, los errores promedio alcanzaron valores de 1,6 metros, con tan solo 2 APs mínimos vistos con buena señal (menores a -70 db) en una misma zona. Se entiende que éste es un buen resultado.

5.3.3. Crowdsourcing

En este trabajo se presentó e implementó una manera eficaz de recolectar datos de usuario para alimentar futuros entrenamientos de las redes estudiadas.

En un trabajo futuro esta recolección podría complementarse con técnicas de *crowdsensing*, tomando la zona validada por el usuario como punto de referencia para medir desplazamientos utilizando otros sensores del dispositivo, como ser acelerómetro. De esta manera, el usuario podría, de manera pasiva, aportar al sistema nuevos puntos de referencia para aumentar la densidad del mapa de huellas digitales.

Además, podría aportar medidas en otros puntos de referencia por los que se estima que está pasando en su recorrido, sin tener que preguntarle nuevamente dónde está. De esta manera, se aumentaría de manera confiable la cantidad de huellas disponibles por zona para futuras fases de entrenamiento, aspecto que vimos ayuda a la precisión de las predicciones. Además, este proyecto almacenó pero no utilizó la información que aporta el usuario cuando la zona predicha, y sugerida por el pop-up, no era efectivamente en la que se localizaba el mismo. Se debería analizar entonces qué huellas del dataset de entrenamiento llevaron a esta predicción y descartarlas o reemplazarlas, para garantizar la robustez del sistema en el transcurso del tiempo, a medida que el mapa de huellas de RSSI actuales varía respecto al considerado en la fase de relevamiento inicial.

Otro aspecto a mejorar sería automatizar los procesos de agregado de huellas a las matrices de entrenamiento, validación y testing. Si bien en este trabajo se implementaron los códigos necesarios para realizar estas tareas, los mismos no se realizan en el servidor en la nube y se tienen que correr manualmente. Para un sistema que salga en producción, se podría realizar un re-entrenamiento con una frecuencia que se vea adecuada según la cantidad de huellas nuevas que aporten los usuarios del mismo en cierto período de tiempo.

Apéndice A

Implementación de códigos

Como se mencionó, para implementar los modelos de LSTM que hemos descrito y probar su efectividad se ha optado por usar Python y diversas librerías útiles para el aprendizaje automático y la ciencia de datos en general. Podemos dividirlas en tecnologías usadas para el procesado de datos, la generación de trayectorias, para los modelos, y tecnologías de apoyo y evaluación.

Los códigos de programación que se han desarrollado, se agrupan en clases que hacen referencia a las etapas de este trabajo, al repasar estas clases se mencionarán las funciones que son fundamentales para la implementación, en el repositorio se encontraran además, funciones auxiliares extras a la implementación del sistema.

El desarrollo de códigos que se realizó es generalizable a otros sets de datos. Se creó un archivo configs.json donde se introducen o modifican los parámetros variables utilizados de todos los códigos.

Estos códigos se encuentran disponibles para el lector en los siguientes enlaces:

- Repositorio del Sistema Implementado:
  https://github.com/ChuVal/LocIndoorFING.git
- Repositorio de la aplicación de Scanner posifi:
  https://github.com/ChuVal/Scanner.git
- Repositorio de la aplicación de usuario:
  https://github.com/ChuVal/LocIndoorApp.git

La recolección de datos se realizó con la aplicación de scanner posifi, igualmente en el repositorio se encuentran los códigos necesarios para el procesamiento de datos tanto para esta aplicación como para la recolección utilizando Find3.

A.0.1. Procesamiento de Datos

El flujo del trabajo realizado con respecto a los datos, es el siguiente:

1. Recolección de datos

2. Preprocesamiento de datos
3. Generación de Trayectorias
4. Inicializar un modelo adecuado para el problema
5. Entrenar el modelo
6. Testear el modelo
7. Evaluación de modelos

Limpieza y Tratamiento de Datos

Los datos recolectados por la aplicación de escáner posifi se procesan por el servidor correspondiente, generando un archivo csv que contiene las huellas recolectadas con la dirección MAC de los APs detectados. En el caso de utilizar Find3 se extraen desde la base de datos en 2 archivos con formato csv, uno de estos archivos contiene las huellas RSSI recolectadas haciendo referencia a los AccesPoint detectados con un símbolo alfanumérico y el otro contiene la dirección MAC de estos APs en correspondencia con su símbolo.

Partiendo de los datos obtenidos en estos archivos se debieron adaptar los mismos a formato de datasets. Para este procesamiento se utiliza la clase *core/-data_processor.py*, la mismo maneja los datos como DataFrame, los preprocesa y transforma de manera de adaptarlos, logrando un dataset limpio y estructurado. Se obtiene una matriz donde cada fila son las muestras (huellas) y cada columna representa las características (APs) donde la última corresponde a la etiqueta (zona).

A los datos faltantes se le asigna el valor -100 dBm.

Filtrado de Ap

Luego de tener el datasets pronto, se debió hacer un filtrado de AP, para esto se utiliza la función *data_processor.py/solo_Fing*.

La lista de MACs correspondientes a los APs ubicados en Facultad de Ingeniería se encuentra en el archivo data/macs.txt.

Se definió la clase *class Aps()* a fin de manipular y adecuar los listados de direcciones MAC correspondientes a los Access Point.

Generación de Trayectorias

Como se menciono en el informe se crearon de manera virtual posibles trayectorias que un usuario podría tener dentro del edificio, en la clase *core/trajectories.py* se definen las funciones necesarias para está implementación, entre ellas tenemos:

```
1 class Trajectories():
2 def crear_mapa (self,configs):
3 def Mapa_Distancias(self, configs):
```

```
def Mapa_Probabilidades (self, MapaD_df,vmax,dmax,deltaT):
def Mapa_CDF (self, MapaP):
def generacion_trayectorias(self,T,cantidad,MapaCDF):
def trayectorias_3D (self,huellas,trayectorias):
def trayectorias_4D (self,huellas,trayectorias, cantidad):
```

En primer lugar se crea el Mapa de Distancias euclidianas entre los puntos referencia, luego se pasa a crear el mapa de probabilidades y por ultimo el CDF. Teniendo el Mapa_CDF creado se pasa a la generación de trayectorias aleatorias, de acuerdo a lo explicado en el informe. A la matriz obtenida se le hace una limpieza descartando trayectorias donde considera pasos que saltean más de una zona.

En cualquiera de los casos, los parámetros utilizados se establecen en el archivo config.json, esto nos permite variar elementos importantes sin necesidad de cambiar código, como por ejemplo lo son la cantidad de trayectorias que se van a generar, la cantidad de puntos referencia que se van utilizar, la distancia entre ellos, etc.

Ya generada la matriz de trayectorias aleatorias se pasa a cargar las huellas se utilizan dos algoritmos, el primero para agregar una huella por zona, *trayectorias_3D*, la elección de la huella se hace de forma random dentro del datasets y la zona correspondiente. Para el caso de la *trayectorias_4D* se cargan por zona una cantidad fija de huellas.

Creación de Modelos

Para las arquitectura MIMO y P-MIMO se implementaron los siguientes modelos secuenciales, utilizando los hiperparámetros mencionados en el informe:

MIMO

```
model = Sequential()
model.add(LSTM(
hidden_layer1, input_shape=(timestep,input_layer),
    return_sequences=True))
model.add(Dropout(dropout_rate))
model.add(TimeDstributed(Dense(output_layer)))
```

P-MIMO:

```
model = Sequential()
model.add(LSTM(
hidden_layer1, input_shape=(timestep,input_layer),
    return_sequences=True))
model.add(Dropout(dropout_rate))
model.add(LSTM(
hidden_layer2, input_shape=(timestep,input_layer),
    return_sequences=True))
model.add(Dropout(dropout_rate))
model.add(TimeDstributed(Dense(output_layer))) \\
```

En el caso de la DL-RNN el modelo implementado es el siguiente:

```
Huellas_inputs = keras.Input(shape=(timestep,input_layer))
x=Loc_matchingRNN=layers.LSTM(hidden_layer1, return_sequences=
    True,input_shape=(timestep,input_layer), name='
    Loc_matchingRNN' )(Huellas_inputs)
```

```
x = layers.Dropout(dropout_rate)(x)
salida1=layers.TimeDistributed(layers.Dense(output_layer),name='
    salida1')(x)
x=Loc_filteringRNN=layers.LSTM(hidden_layer2, return_sequences=
    True,           name='Loc_filteringRNN')(salida1)
x = layers.Dropout(dropout_rate)(x)
salida2=layers.TimeDistributed(layers.Dense(output_layer),name='
    salida2')(x)
model = keras.Model(inputs=[Huellas_inputs],outputs=[salida1,
    salida2])
```

Apéndice B

Matriz de Confusión para 90 Zonas

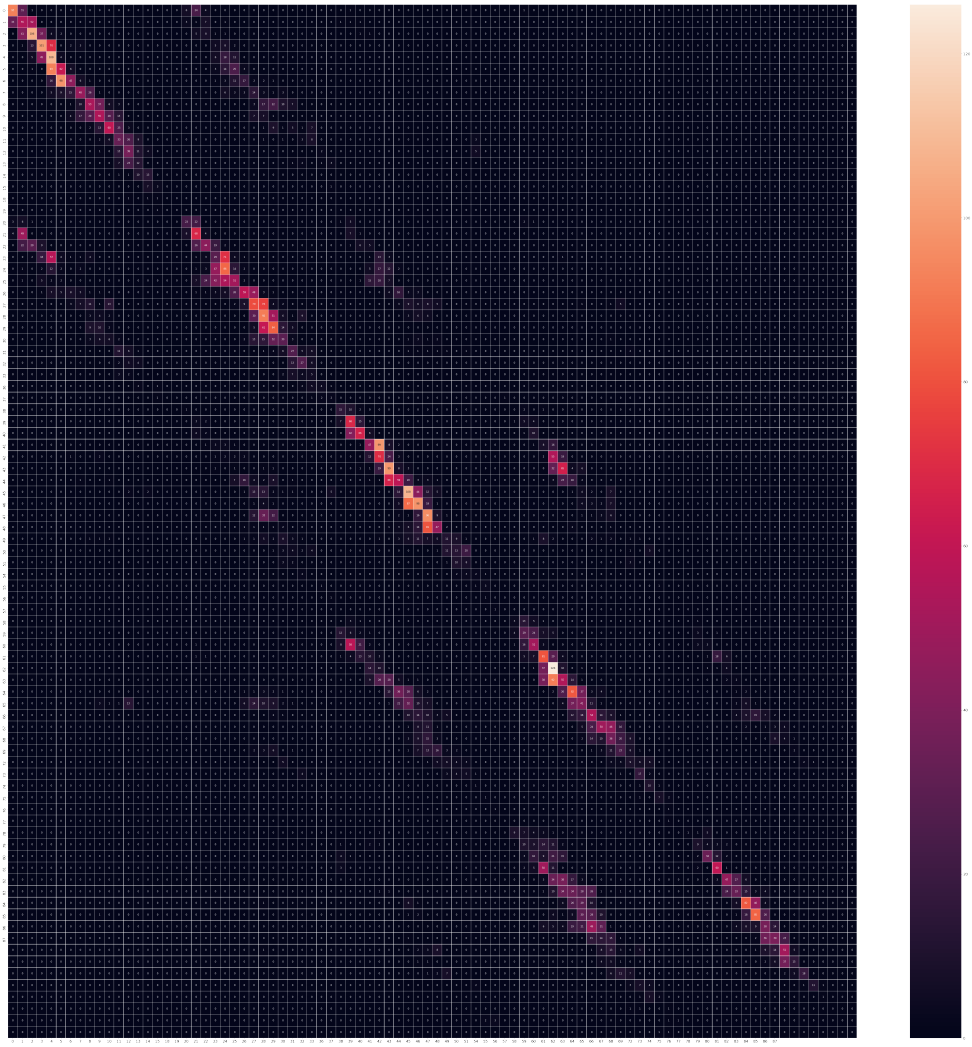


Figura B.1: Matriz de confusión para DL-RNN en 90 zonas

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] Android studio - https://developer.android.com/studio?hl=es-419.

[2] Anyplace - https://anyplace.cs.ucy.ac.cy.

[3] Comprensión de las redes lstm - https://colah.github.io.

[4] Deep learning - standford, by afshine amidi and shervine amidi - https://stanford.edu/ shervine/teaching/cs-230.

[5] Framework for internal navigation and discovery - https://www.internalpositioning.com/doc.

[6] Keras - https://keras.io.

[7] Knn - https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn.

[8] Lenguaje go - https://golang.org/.

[9] Librería numpy - https://numpy.org.

[10] Librería pandas - https://pandas.pydata.org/.

[11] Localizacion indoor basada en wi-fi, a. bracco, f. grunwald, a. navcevich - https://iie.fing.edu.uy/publicaciones/2019/bgn19/bgn19.pdf.

[12] Mnavigante (github) - https://github.com/agusnavce/mnavigator.

[13] Node.js - https://www.npmjs.com.

[14] Npm.js - https://nodejs.org/es.

[15] Postgresql - http://www.postgresql.org.

[16] Project jupyter - https://jupyter.org/.

[17] Python - https://www.python.org.

[18] React native - https://reactnative.dev.

[19] Tensorflow - https://www.tensorflow.org.

[20] Understanding lstm networks - standford, by christopher olah - https://web.stanford.edu/class/cs379c/archive/2018/class$_m$essages$_l$isting/content/artificial$_n$eural$_n$etw

[21] S. H. Creem-Regehr H. L. PickJr B. J. Mohler, W. B. Thompson and W. H. Warren. *Visual flow influences gait transition speed and preferred walking speed.*

[22] Jason Brownlee. *Long Short-Term Memory Networks With Python.* 2017.

[23] Li M. Ren K. Qiao C. Chen, S. *Crowd Map: Accurate Reconstruction of Indoor Floor Plans from Crowdsourced Sensor-Rich Videos.* IEEE 35th International Conference on Distributed Computing Systems, 2015.

[24] Piche R. Davidson, P. *A Survey of Selected Indoor Positioning Methods for Smartphones.* IEEE Communications Surveys Tutorials, 2017.

[25] Y. Freund and R. Shapire. *"A decision-theoretic generalization of on-line learning and an application to boosting.* Proceedings of the Second European Conference on Computational Learning Theory, 1995.

[26] J.F. Mas H. Taud. *Geomatic Approaches for Modeling Land Change Scenarios.* Lecture Notes in Geoinformation and Cartography, 2018.

[27] Sepp Hochreiter Jurgen Schmidhuber. *LONG SHORT-TERM MEMORY.* Neural Computation, 1997.

[28] Simeon Kostadinov. *Recurrent Neural Networks with Python Quick Start Guide.* 2018.

[29] X. Dong T. Lu-R. Westendorp M. T. Hoang, B. Yuen and K. Reddyn. *Recurrent Neural Networks for Accurate RSSI Indoor Localization.* IEEE Internet of Things Journal, vol. 6, no. 6, pp. 10639-10651.

[30] Arturo Fernández Montoro. *Python 3 al descubierto.* 2º edición - Alfaomega grupo editor, México.

[31] Marina M. K. Radu, V. *TrHiMLoc: Indoor smartphone localization via activity aware Pedestrian Dead Reckoning with selective crowdsourced WiFi fingerprinting.* International Conference on Indoor Positioning and Indoor Navigation, 2013.

[32] Toby Segaran. *Programming Collective Intelligence.* 1º edición - O'Reilly Media.

[33] Qun Wan Long He Xinrui Wang Siqi Bai, Mingjiang Yan and Junlin Li. *DL-RNN: An Accurate Indoor Localization Method via Double RNNs.* IEEE Systems Journal.

[34] Chen Q. Yang L. T. Chao. Wang, B. *Indoor smartphone localization via fingerprint crowdsourcing: challenges and approaches.* IEEE Wireless Communications, 23(3), 2016.

[35] Subbu K. P. Luo J. Wu J. Zhang, C. *GROPING: Geomagnetism and cROwd-sensing Powered Indoor NaviGation.* IEEE Transactions on Mobile Computing, 14(2), 2015.

[36] Shen G. Li L. Zhao C. Li M. Zhao F. Zheng, Y. *Travi-Navi: Self-Deployable Indoor Navigation System.* IEEE/ACM Transactions on Networking, 2017.

[37] Chen T. Guo D. Zhou, X. *From one to crowd: a survey on crowdsourcing-based wireless indoor localization.* Front. Comput. Sci. 12, 2018.

[38] Chen T. Guo D. Teng X. Yuan B. Zhou, X. *From one to crowd: a survey on crowdsourcing-based wireless indoor localization.* Frontiers of Computer Science, 12(3), 2018.

Esta página ha sido intencionalmente dejada en blanco.

Índice de tablas

3.1. Parámetros - generación de Trayectorias 42
3.2. Hiperparámetros utilizados en los algoritmos de MIMO y PMIMO 45
3.3. Hiperparámetros - modelo DLRNN 48

4.1. TablaErorres 51
4.2. TablaPasos 52
4.3. Error de DL-RNN en función de la resolución espacial 54
4.4. Resultados para DL-RNN en función de la cantidad de trayectorias 55

5.1. Resumen de desempeño 66

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

2.1. Diagrama del problema de huellas digitales. 9
2.2. Arquitectura de una RNN genérica. 14
2.3. Arquitectura de una RNN [20]. 16
2.4. Arquitectura de una LSTM. 16
2.5. Estado de la celda. 17
2.6. Primer paso - “forget gate layer”. 18
2.7. Segundo paso LSTM. 18
2.8. Tercer paso LSTM - Actualización de estado. 18
2.9. Cuarto paso LSTM - salida. 19
2.10. Arquitectura DL-RNN. 20
2.11. Arquitectura MISO. 21
2.12. Arquitectura MIMO 22
2.13. Arquitectura PMIMO 22

3.1. Arquitectura de los servidores. 31
3.2. Modelo relacional de la base de datos 33
3.3. Esquema - Hall del primer piso de facultad de Ingeniería. 34
3.4. Plano - Hall del primer piso de facultad de Ingeniería. 34
3.5. Aplicación para toma de huellas 35
3.6. Imagen mostrada en la aplicación para la zona 78. 36
3.7. Pop-up de consulta al usuario 37
3.8. Matrices - generación de trayectorias. 41
3.9. Trayectorias generados por el algoritmo 43
3.10. Arquitectura PMIMO 45
3.11. Arquitectura DL-RNN Implementada. 47
3.12. Rendimiento del modelo en función de la cantidad de epochs. . . . 48

4.1. Gráfica del error en los ejemplos de testing para DL-RNN. 50
4.2. Trayectoria real vs predicción de una DL-RNN. 50
4.3. Comparativo de errores por largo de trayectoria. 52
4.4. Reagrupación de las zonas 53
4.5. Trayectoria real y predicha para P-MIMO. 54
4.6. Trayectoria real y predicha para DL-RNN 54
4.7. Dispersión del error para DL-RNN con 100.000 trayectorias. 55
4.8. Cantidad de huellas por zona. 56

4.9. Error para DL-RNN en 90 zonas. 57
4.10. Error para DL-RNN en 18 zonas. 57
4.11. Ubicación de los APs adicionales. 58
4.12. Precisión de la DL-RNN para el conjunto de testing con dos APs adicionales. 59
4.13. Matriz de confusión para DL-RNN en 18 zonas con dos APs adicionales. 60
4.14. Matriz de confusión para el sistema Posifi 61
4.15. Métricas de clasificación para el sistema Posifi 62
4.16. Matriz de confusión para el sistema Locindoor 63
4.17. Métricas de clasificación para el sistema Locindoor. 64

B.1. Matriz de confusión para DL-RNN en 90 zonas 73

Esta es la última página.
Compilado el martes 14 septiembre, 2021.
http://iie.fing.edu.uy/