Trabajo Fin de Máster
Máster Universitario en Ingeniería Electrónica, Robótica y Automática

Estudio del empleo de redes neuronales para la fusión de datos procedentes de sensores lidar y radar

Autor: Rafael Uceda Gallegos

Tutor: Fernando Caballero Benítez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Máster

Máster Universitario en Ingeniería Electrónica, Robótica y Automática

Estudio del empleo de redes neuronales para la fusión de datos procedentes de sensores lidar y radar

Autor:

Rafael Uceda Gallegos

Tutor:

Fernando Caballero Benítez

Profesor Titular

Dpto. Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Máster: Estudio del empleo de redes neuronales para la fusión de datos procedentes de sensores lidar y radar

Autor: Rafael Uceda Gallegos
Tutor: Fernando Caballero Benítez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Quisiera ser breve. Agradezco este trabajo y, sobre todo, lo que implica haber llegado hasta aquí, a mis padres. Gracias por confiar en mí y apoyarme en todos los ámbitos de mi vida.

Rafael Uceda Gallegos
Sevilla, 2023

Resumen

El objetivo principal de este trabajo de fin de máster es investigar y analizar el desempeño de diversas arquitecturas de redes neuronales en la resolución del problema de la fusión de datos de sensores de lidar y radar. La combinación de estas dos fuentes de información puede ofrecer una ventaja significativa al superar las limitaciones individuales de cada sensor y mejorar la fiabilidad de las mediciones obtenidas. En este sentido, se realizará un estudio de las técnicas de desarrollo, entrenamiento y evaluación de modelos, con el objetivo final de determinar cuál de las tipologías de redes neuronales estudiadas proporciona los mejores resultados para el problema en cuestión.

Abstract

The main objective of this project is to investigate and analyze the performance of different neural network architectures in solving a specific problem, namely, the fusion of data from lidar and radar sensors. The combination of these two sources of information can offer a significant advantage by overcoming the individual limitations of each sensor and improving the reliability of the measurements obtained. In this regard, a study of developing, training, and model evaluation techniques will be carried out, with the ultimate goal of determining which of the neural network typologies studied provides the best results for the problem at hand.

Índice Abreviado

Resumen III
Abstract V
Índice Abreviado VII

1 Introducción 1
1.1 Entorno de trabajo 2
1.2 Datos de entrenamiento 4
1.3 Consideraciones previas 4

2 Estado del arte 7

3 Red neuronal artificial 11
3.1 Primera aproximación 12
3.2 Segunda propuesta: Dropout 13
3.3 Tercera propuesta: Batch Normalization 14
3.4 Conclusiones 15

4 Red convolucionales 17
4.1 Redes convolucionales 2D 17
4.2 Redes convolucionales 1D 20

5 Conclusiones 29

Índice de Figuras 31
Bibliografía 33

Índice

Resumen III
Abstract V
Índice Abreviado VII

1 Introducción 1
1.1 Entorno de trabajo 2
1.2 Datos de entrenamiento 4
1.3 Consideraciones previas 4

2 Estado del arte 7

3 Red neuronal artificial 11
3.1 Primera aproximación 12
3.2 Segunda propuesta: Dropout 13
3.3 Tercera propuesta: Batch Normalization 14
3.4 Conclusiones 15

4 Red convolucionales 17
4.1 Redes convolucionales 2D 17
4.1.1 Unet 2D 17
4.2 Redes convolucionales 1D 20
4.2.1 Primera aproximación 20
4.2.2 Estudio de hiperparámetros 22
4.2.3 Unet 1D 24
4.2.4 Unet ++ 25

5 Conclusiones 29

Índice de Figuras 31
Bibliografía 33

1 Introducción

Una parte fundamental para el desarrollo de sistemas robóticos autónomos es la percepción del entorno. Esto permite a los agentes móviles reconocer el medio de trabajo, localizarse en él y, posteriormente, navegar.

Para llevar a cabo dicha percepción, un robot móvil debe llevar embarcados una serie de sensores que traduzcan estímulos del entorno en datos manejables por su sistema de procesamiento. Existe una amplia gama de sensores empleados en robótica móvil, entre ellos se encuentran lidares y radares, que son objeto de estudio en este documento.

Estos sensores se engloban dentro de la categoría de sensores empleados para la construcción y locacalización en mapas [4].

La construcción de mapas requiere la utilización de sensores que midan distancias. Existen tres enfoques básicos para medir la distancia:

1. Sensores basados en la medición del tiempo de vuelo (TOF) de un pulso de energía emitido que viaja hacia un objeto reflectante y luego retrocede al receptor. Los sensores TOF son populares debido a su alta precisión y velocidad de respuesta. Estos sensores se utilizan en aplicaciones generales de robótica, como robots de limpieza y drones, y también en automóviles autónomos.
2. Sensores basados en la medición de fase (o detección de fase). Implica la transmisión de ondas continuas en lugar de los cortos pulsos utilizados en los sistemas TOF. Estos sensores miden la fase de la señal reflejada y utilizan esta información para determinar la distancia.
3. Los sensores basados en radar de frecuencia modulada (FM). Estos dispositivos son similares a los sensores de medición de fase basados en ondas continuas, pero utilizan ondas de radio en lugar de ondas de luz. Esta técnica se utiliza en aplicaciones como la medición de la velocidad de un vehículo o la detección de obstáculos.

En general, la elección del sensor adecuado depende de la aplicación y las condiciones específicas del entorno. Es importante evaluar la precisión, el alcance y la velocidad de respuesta de los diferentes sensores disponibles para seleccionar el más adecuado para cada tarea de mapeo. El uso del lidar como sensor de percepción del entorno está muy extendido en la mayoría de aplicaciones robóticas. Sin embargo, presenta dificultades cuando se trata de ambientes con partículas en suspensión. Esto implica que construir un sistema de localización y mapeo robusto que funcione en condiciones adversas sigue siendo un problema abierto.

El radar es otro tipo de sensor activo, cuyo espectro electromagnético suele estar en la banda de frecuencia mucho más baja (GHz) que el lidar (desde THz a PHz). Por lo tanto, puede operar de manera más confiable en la mayoría de las condiciones climáticas y de luz. Además, tiene cualidades distintas de las del lidar, por ejemplo, mayor alcance de detección, estimaciones de velocidad relativa a partir del efecto Doppler y medición de rango absoluto. Recientemente, el radar

ha sido considerado como indispensable para la autonomía segura y se ha adoptado cada vez más en la industria automotriz para la detección de obstáculos y los sistemas avanzados de asistencia al conductor.

En resumen, cuando se trata de condiciones adversas como polvo en suspensión, humo o niebla, el sensor radar ofrece mediciones menos erráticas que el lidar, sin embargo, la resolución angular del lidar es muy superior a la del radar.

La combinación de mediciones de lidar y radar permite obtener una representación más completa y precisa de los obstáculos, mejorando la reconstrucción del entorno en situaciones donde un solo sensor no sería suficiente.

No obstante, la fusión de ambos sensores presenta desafíos técnicos significativos, tales como la heterogeneidad de los datos, la falta de correspondencia entre ellos, el ruido y la incertidumbre. Estos desafíos pueden dificultar la integración efectiva y limitar la precisión y robustez del sistema. Por lo tanto, es esencial desarrollar métodos de fusión de datos que aborden estos retos tecnológicos.

La fusión de datos más tradicional se ha abordado generalmente mediante el llamado filtro de Kalman [8].

El filtro de Kalman es un algoritmo matemático que se utiliza ampliamente en la fusión de datos y la estimación de estado. Fue desarrollado por Rudolf Kalman en la década de 1960 y se utiliza en una amplia variedad de aplicaciones, desde la navegación por satélite hasta la robótica y la ingeniería de sistemas. El filtro de Kalman se basa en la teoría de control y utiliza un modelo matemático para predecir el estado futuro de un sistema, apoyándose en las mediciones obtenidas mediante diversos sensores.

El filtro de Kalman se divide en dos etapas: predicción y actualización. En la etapa de predicción, el filtro utiliza el modelo matemático del sistema para predecir el estado futuro del sistema y la incertidumbre asociada con la predicción. En la etapa de actualización, el filtro combina la predicción con las mediciones de los sensores para obtener una estimación más precisa del estado actual del sistema. El filtro también actualiza la incertidumbre asociada con la estimación en función de la precisión de los sensores.

Una de las ventajas clave del filtro de Kalman es su capacidad para manejar mediciones ruidosas y datos faltantes de manera efectiva. El filtro utiliza una combinación ponderada de la predicción y las mediciones de los sensores para obtener una estimación del estado del sistema. Sin embargo, el filtro de Kalman también tiene algunas limitaciones, como la necesidad de un modelo matemático preciso del sistema y la asunción de que el ruido en las mediciones del sensor sigue una distribución gaussiana.

Pese a la gran efectividad de los filtros de Kalman, en este trabajo se propone el empleo de redes neuronales para estudiar su potencial en este marco de trabajo y para evadir la dependecia de modelo matemático que posee dicho filtro.

En resumen, el objetivo principal de este proyecto es evaluar distintas estructuras de redes neuronales fusionando datos de lidar y radar. Un esquema representativo se muestra en la figura 1.1.

1.1 Entorno de trabajo

Para llevar a cabo la implementación de los distintos modelos de redes neuronales se opta por el lenguaje de programación Python y la librería Keras [1].

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, que se ha convertido en uno de los lenguajes más populares en la industria de la tecnología. Fue creado en 1991 por Guido van Rossum y se caracteriza por su sintaxis clara y sencilla, lo que lo hace ideal tanto para principiantes como para programadores experimentados. Además, es compatible con una gran cantidad de plataformas y sistemas operativos, lo que lo convierte en una herramienta muy versátil.

Una de las características más importantes de Python es su amplia comunidad de desarrolladores, que han creado una gran cantidad de bibliotecas y frameworks que facilitan el desarrollo de aplicaciones complejas. Algunas de estas bibliotecas son especialmente útiles para el análisis de datos, como NumPy, Pandas y Matplotlib, mientras que otras son ideales para el desarrollo web, como Django y Flask. Python también es utilizado en áreas como el aprendizaje automático, la inteligencia artificial y la robótica, gracias a herramientas como TensorFlow, Keras y PyTorch.

Otra ventaja de Python es su facilidad de aprendizaje y uso, ya que cuenta con una amplia documentación y recursos en línea, así como una sintaxis clara y fácil de leer. Esto lo convierte en un lenguaje ideal para enseñar programación en instituciones académicas y cursos en línea. Además, Python es conocido por ser un lenguaje muy productivo, lo que significa que los desarrolladores pueden escribir código de manera más rápida y eficiente, lo que les permite centrarse en la lógica y la solución de problemas en lugar de en detalles técnicos de bajo nivel.

Keras es una biblioteca de aprendizaje profundo de código abierto escrita en Python. Se utiliza principalmente para desarrollar y entrenar redes neuronales profundas, incluyendo redes neuronales convolucionales (CNN), redes neuronales recurrentes (RNN) y redes neuronales con memoria a largo plazo (LSTM). Fue desarrollado por Francois Chollet y es mantenido por la comunidad de código abierto.

Keras se centra en la facilidad de uso, lo que la convierte en una de las herramientas más populares para el desarrollo de modelos de aprendizaje profundo. Es una interfaz de alto nivel que proporciona una abstracción intuitiva de la complejidad de la programación de redes neuronales profundas. Ofrece una amplia variedad de capas y modelos predefinidos, lo que facilita la creación de modelos de redes neuronales personalizados.

Keras también cuenta con una gran comunidad de desarrolladores y usuarios, lo que significa que hay muchos recursos y tutoriales disponibles en línea. Además, es compatible con otras bibliotecas populares de aprendizaje automático, como TensorFlow y PyTorch, lo que permite una integración sin problemas en proyectos más grandes. En resumen, Keras es una herramienta esencial para cualquier persona que quiera construir y entrenar modelos de aprendizaje profundo de manera rápida y eficiente.

Para facilitar la programación y la ejecución de los distintos códigos desarrollados en este trabajo, se emplea Google Colab como entorno de ejecución. Google Colab es una plataforma en línea gratuita que permite a los usuarios escribir y ejecutar código en lenguaje Python. Desarrollada por Google, esta herramienta ofrece una gran cantidad de recursos, como el uso gratuito de GPUs, TPUs y la integración con Google Drive. Google Colab es especialmente útil para la implementación de proyectos de aprendizaje automático y redes neuronales, ya que permite utilizar bibliotecas populares como TensorFlow, Keras, PyTorch y OpenCV.

El principal beneficio de utilizar Google Colab es la disponibilidad de recursos de alta gama sin tener que invertir en hardware costoso. Además, al estar basado en la nube, los usuarios pueden acceder a sus proyectos desde cualquier dispositivo con conexión a internet. Otra ventaja importante es la facilidad de colaboración, ya que los usuarios pueden compartir sus notebooks y trabajar en equipo en tiempo real.

Por último, Google Colab ofrece una interfaz intuitiva y fácil de usar, lo que permite a los usuarios centrarse en el desarrollo de su código sin tener que preocuparse por la configuración del entorno. Los usuarios pueden cargar y descargar archivos directamente desde Google Drive, lo que facilita la gestión de proyectos. En general, Google Colab es una herramienta imprescindible para cualquier persona interesada en el aprendizaje automático y la ciencia de datos.

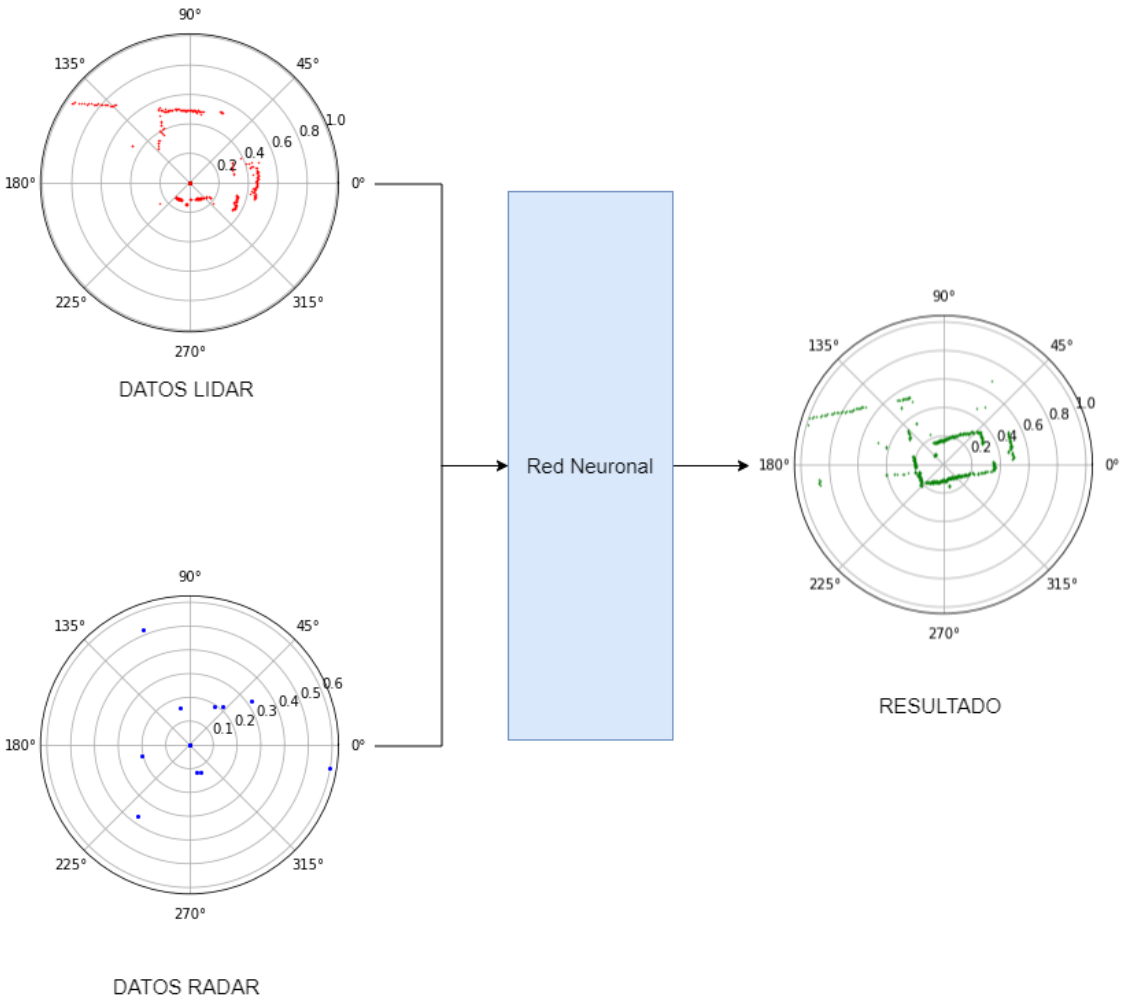


Figura 1.1 Representación gráfica del esquema propuesto.

1.2 Datos de entrenamiento

El conjunto de datos consta de grupos de tres escaneos, que están referidos al mismo marco de coordenadas y tienen las mismas características. Estos escaneos se generan a partir de (en orden de aparición):

- Escaneo LiDAR: obtenido de un LiDAR OS1-16
- Escaneo RADAR: obtenido de 4 sensores RADAR.
- Escaneo Ground Truth: obtenido de la posición del robot y del mapa precalculado del entorno.

Para este trabajo se parte de un dataset formado por 1755 ficheros de texto con las mediciones realizadas por lidar, radar y ground truth del entorno. Estas mediciones representan distancias en coordenadas polares con saltos de 0.00999999977648 radianes (629 mediciones para completar la circunferencia).

1.3 Consideraciones previas

Tras un breve análisis inicial de los datos de entrenamiento se llegan a las siguientes conclusiones:

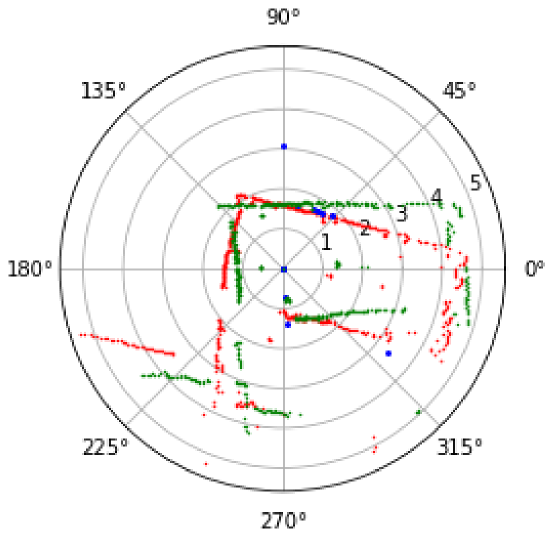
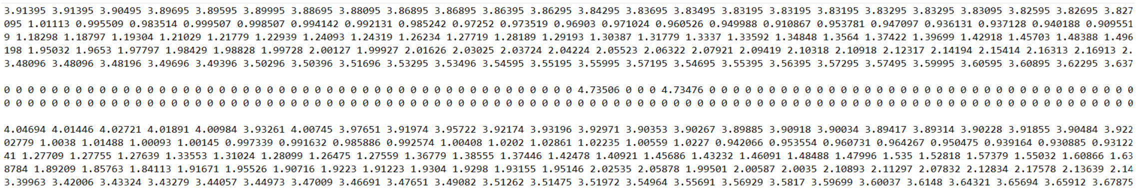


Figura 1.2 Representación gráfica de uno de los experimentos. Rojo: lidar, azul: radar, verde: ground truth.



3.91395 3.91395 3.90495 3.89695 3.89595 3.89995 3.88695 3.88095 3.86895 3.86895 3.86395 3.86295 3.84295 3.83695 3.83495 3.83195 3.83195 3.83195 3.83295 3.83295 3.83095 3.82595 3.82695 3.827
095 1.01113 0.995509 0.983514 0.999507 0.998507 0.994142 0.992131 0.985242 0.97252 0.973519 0.96903 0.971024 0.960526 0.949988 0.910867 0.953781 0.947097 0.936131 0.937128 0.940188 0.909551
9 1.18298 1.18797 1.19304 1.21029 1.21779 1.22939 1.24093 1.24319 1.26234 1.27719 1.28189 1.29193 1.30387 1.31779 1.3337 1.33592 1.34848 1.3564 1.37422 1.39699 1.42918 1.45703 1.48388 1.496
198 1.95032 1.9653 1.97797 1.98429 1.98828 1.99728 2.00127 1.99927 2.01626 2.03025 2.03724 2.04224 2.05523 2.06322 2.07921 2.09419 2.10318 2.10918 2.12317 2.14194 2.15414 2.16313 2.16913 2.
3.48096 3.48096 3.48196 3.49696 3.49396 3.50296 3.50396 3.51696 3.53295 3.53496 3.54595 3.55195 3.55995 3.57195 3.54695 3.55395 3.56395 3.57295 3.57495 3.59995 3.60595 3.60895 3.62295 3.637

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4.73506 0 0 0 4.73476 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

4.04694 4.01446 4.02721 4.01891 4.00984 3.93261 4.00745 3.97651 3.91974 3.95722 3.92174 3.93196 3.92971 3.90353 3.90267 3.89885 3.90918 3.90034 3.89417 3.89314 3.90228 3.91855 3.90484 3.922
02779 1.0038 1.01488 1.00093 1.00145 0.997339 0.991632 0.985886 0.992574 1.00408 1.0202 1.02861 1.02235 1.00559 1.0227 0.942066 0.953554 0.960731 0.964267 0.950475 0.939164 0.930885 0.93122
41 1.27709 1.27755 1.27639 1.33553 1.31024 1.28099 1.26475 1.27559 1.36779 1.38555 1.37446 1.42478 1.40921 1.45686 1.43232 1.46091 1.48488 1.47996 1.535 1.52818 1.57379 1.55032 1.60866 1.63
8784 1.89209 1.85763 1.84113 1.91671 1.95526 1.90716 1.9223 1.91223 1.9304 1.9298 1.93155 1.95146 2.02535 2.05878 1.99501 2.00587 2.0035 2.10893 2.11297 2.07832 2.12834 2.17578 2.13639 2.14
3.39963 3.42006 3.43324 3.43279 3.44057 3.44973 3.47009 3.46691 3.47651 3.49082 3.51262 3.51475 3.51972 3.54964 3.55691 3.56929 3.5817 3.59699 3.60037 3.6148 3.64321 3.65694 3.65912 3.67875

Figura 1.3 Vista previa de uno de los experimentos.

- Dado un experimento, los datos procedentes de lidar distintos de cero son más abundantes que los procedentes de radar. Esto se observa claramente en la figura 1.2 donde los puntos rojos (lidar) son más abundantes que los azules (radar).
- No se puede afirmar que los datos provenientes de uno de los sensores contengan menos error que los del otro. En la figura 1.2 se comprueba que existen diferencias de medida con respecto al ground truth tanto en los puntos rojos (lidar) como en los azules (radar).

Se plantean dos enfoques en el tratamiento de los datos: como vectores de tamaño [629,1] en coordenadas polares y como imágenes en coordenadas cartesianas.

2 Estado del arte

En situaciones de emergencia donde el humo reduce la visibilidad general del ambiente, los sistemas de detección tradicionales como los escáneres lidar pueden no ser suficientes para proporcionar información precisa del entorno. En estas situaciones, el uso de sensores radar puede ser de gran ayuda, ya que presentan un mejor rendimiento en ambientes con partículas en suspensión y pueden proporcionar información complementaria en tiempo real.

Esto ha sido objeto de estudio anteriormente, y, como se concluye en el artículo *Evaluation of Navigation Sensors in Fire Smoke Environments* [13], mientras que los sensores lidar se ven afectados en entornos en los que el humo no deja ver más allá de los 4 metros, los sensores radar no presentan ninguna alteración. Dichas conclusiones extraídas del artículo se presentan en la figura 2.2.

La fusión de datos de lidar y radar se presenta como una solución prometedora para mejorar la precisión y la fiabilidad de la detección de objetos en entornos desafiantes. La combinación de los datos de estos dos tipos de sensores puede proporcionar una imagen más completa y mejorar la detección de puntos característicos en situaciones de poca visibilidad.

Además, el uso de redes neuronales para la fusión de datos de lidar y radar puede mejorar aún más la precisión y la eficacia de estos sistemas. Las redes neuronales pueden aprender a integrar la información de los dos tipos de sensores de manera más efectiva y proporcionar una mejor estimación de la posición y orientación de los objetos en el entorno.

Figura 2.1 Robot móvil en entorno con fuego y humo. Fuente: [9].

-

Instrument	Dense smoke, low temperature	Light smoke, high temperature
Electromagnetic instruments		
LIDAR instruments	Attenuation at 4 m visibility; failure at 1 m visibility	No effect
Visual cameras	Attenuation at 8 m visibility; failure at 1 m visibility	No effect
Kinect™ depth sensor	Poor results even with >8 m Visibility (combination of particle blocking and sensor being flooded by light from fire)	Sensor flooded by light from fire during whole test
Night vision	Failure at about 4 m visibility	Sensor flooded by light from fire during whole test
Thermal cameras	No effect	No effect
Radar	No effect	No effect
Other		
Sonar	Some attenuation with temperature change	Attenuation with temperature change

Figura 2.2 Rendimiento de distintos sensores en entornos con humo. Procede de [13].

La fusión de datos de sensores mediante redes neuronales ha sido motivo de estudio en los últimos años. Numerosos trabajos de investigación han explorado el uso de redes neuronales para la fusión de datos de diferentes sensores, como cámaras, lidar, radar y GPS, con el objetivo de mejorar la precisión y la fiabilidad de los sistemas de percepción para aplicaciones como la conducción autónoma y la robótica.

Un trabajo interesante en esta línea de investigación es el de Xinxin Du et al. (2018) [6], que propone una arquitectura de red neuronal para la detección de vehículos mediante la fusión de datos de lidar y visión. La red neuronal consta de tres partes principales: la primera parte genera propuestas de ubicación de vehículos en la imagen utilizando la nube de puntos de lidar; la segunda parte refina la ubicación de las propuestas utilizando información de múltiples capas de la red neuronal; y la tercera parte lleva a cabo la tarea de detección utilizando una red neuronal que comparte algunas capas con la red de propuestas. La evaluación muestra la arquitectura definida es capaz de generar propuestas de alta calidad de manera más eficiente.

Otro ejemplo es el trabajo de Faten Hamed Nahhas et al. (2019) [10], que utiliza una técnica de fusión de datos lidar-ortofoto para la detección de edificios mediante aprendizaje profundo. En este trabajo, se utiliza un enfoque de análisis basado en objetos y se realiza una fusión de características a dicho nivel. Posteriormente, se utiliza una reducción de dimensionalidad basada en autoencoder para transformar las características de bajo nivel en características comprimidas y una red neuronal convolucional (CNN) para transformar las características comprimidas en características de alto nivel, que se utilizan para clasificar los objetos en edificios y fondo. La arquitectura propuesta se optimiza utilizando el método de búsqueda de cuadrícula y se analiza su sensibilidad a los hiperparámetros. Los resultados muestran que la reducción de dimensionalidad mediante autoencoder puede mejorar la precisión de la detección y que la selección adecuada de los hiperparámetros es crítica para mejorar la capacidad de generalización del modelo. Además, se compara el modelo propuesto con una máquina de vectores de soporte (SVM) y se muestra que el modelo propuesto supera a los modelos SVM en el área de trabajo. En el área de prueba, sin embargo, el modelo SVM logra una precisión superior a la del modelo propuesto sin reducción de

dimensionalidad.

Existen también estudios centrados en la fusión de sensores radar. Un ejemplo de ello es el trabajo de investigación presentado por Thomas et al. (2020) [11], que propone una arquitectura de fusión de radar y cámara basada en el aprendizaje profundo para la detección de objetos en el vehículo. El sistema propuesto, denominado CameraRadarFusion Net (CRF-Net), fusiona datos de cámara y de radar mediante deep learning para mejorar los resultados de detección de objetos. Además, se presenta la estrategia de entrenamiento BlackIn, que se inspira en la técnica de Dropout, para enfocar el aprendizaje en un tipo específico de sensor. Los resultados experimentales muestran que la red de fusión es capaz de superar a una red de detección de objetos basada únicamente en imágenes para dos conjuntos de datos diferentes.

Como se observa en los estudios publicados, las redes neuronales para la fusión de sensores se han consolidado como una herramienta muy útil en el campo de la percepción de los sistemas autónomos. Los resultados obtenidos por los distintos autores demuestran que esta técnica puede mejorar significativamente la capacidad de detección y reconocimiento de objetos, lo cual es esencial para la operación segura y eficiente de vehículos autónomos y robots móviles. Además, la fusión de datos de sensores mediante redes neuronales puede ayudar a superar las limitaciones de los sensores individuales, ya que cada tipo de sensor tiene sus propias fortalezas y debilidades. Por tanto, el uso de redes neuronales para la fusión de sensores se presenta como una línea de investigación prometedora que seguramente seguirá creciendo en los próximos años.

3 Red neuronal artificial

El primer esquema propuesto y el más simple es el de una estructura basada en capas ocultas conocida como red neuronal artificial. En la figura 3.1 se muestra un ejemplo gráfico de dicha estructura. Este esquema está compuesto fundamentalmente por una capa de entrada, más o menos capas ocultas en función de la complejidad del sistema y una capa de salida. En el caso de este trabajo, la capa de entrada tiene una forma matricial (629,2) para incluir en cada una de las columnas los datos de lidar y radar. La capa de salida tiene estructura vectorial (629,1) acorde a la forma del ground truth.

Se ha experimentado con distinto número de capas ocultas y unidades por capa. Hay varios factores a tener en cuenta al elegir el número de unidades y capas para una red neuronal:

- La complejidad del problema: Un problema más complejo requerirá una mayor cantidad de unidades y capas para poder modelarlo adecuadamente.
- Los datos de entrada: El número de unidades en la primera capa debe ser igual al número de características en los datos de entrada.
- El número de salidas deseadas: El número de unidades en la última capa debe ser igual al número de salidas deseadas.
- El tamaño de la muestra: Una muestra más grande puede requerir una mayor cantidad de unidades y capas para evitar el sobreajuste.

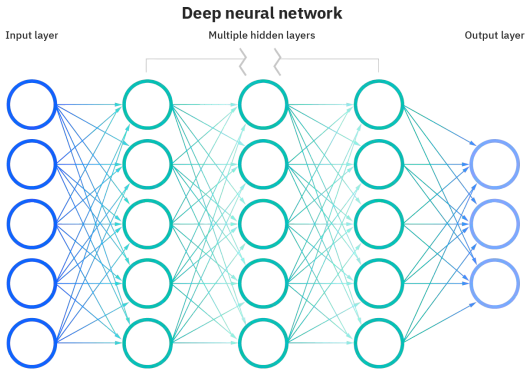


Figura 3.1 Esquema representativo red neuronal artificial.

3.1 Primera aproximación

Se ha propuesto una primera estructura de red basada únicamente en capas densas. Su representación se muestra en la figura 3.2. Para llevar a cabo su implementación en Keras, se emplea la función `model.add(Dense())` que añade capas densas al modelo. Una muestra de dicha implementación se observa en el código 3.1. Este es solo un ejemplo de los distintas variantes que se han entrenado alterando el número de capas y/o unidades por capa.

Código 3.1 Red neuronal artificial simple.

```
model = Sequential()
model.add(Dense(units=500,
                activation='relu', input_shape=(629,2,1)))
model.add(Dense(units=200, activation='relu'))
model.add(Dense(units=200, activation='relu'))
model.add(Dense(units=200, activation='relu'))
model.add(Dense(units=200, activation='relu'))
model.add(Dense(units=200, activation='relu'))
model.add(Dense(units=200, activation='relu'))
model.add(Dense(units=200, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['
    accuracy'])
model.build()
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer
    _names=True)
model.summary()
```

Tras el entrenamiento, se muestra una predicción del conjunto de validación en la figura 3.3. Se aprecia que los resultados no son favorables, la red ha aprendido a arrojar como predicción un círculo centrado en el origen. Ocurre un comportamiento similar con el resto del conjunto de validación.

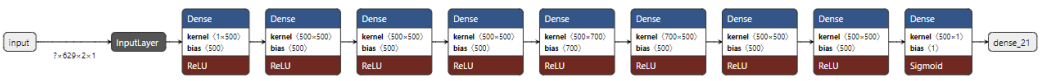


Figura 3.2 Esquema red inicial.

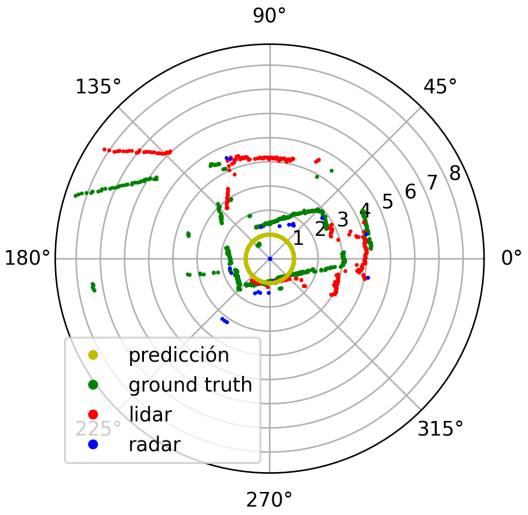


Figura 3.3 Resultado primera propuesta red neuronal.

3.2 Segunda propuesta: Dropout

Para dotar a la red inicial de más robustez y facilidad a la hora de reconocer los patrones del dataset, se propone incluir capas de dropout [2]. Las capas de dropout son una técnica de regularización utilizada para prevenir el sobreajuste en las redes neuronales profundas. Dropout funciona al "eliminar" aleatoriamente (es decir, establecer en cero) un cierto número de unidades de salida de una capa durante el entrenamiento. Las unidades eliminadas no se tienen en cuenta durante la propagación hacia adelante o hacia atrás, por lo que la red no puede confiar en la presencia de ciertas características, obligando a la red a aprender características más robustas que sean útiles en muestras con mayor variabilidad. La implementación en Keras se ha realizado con la función `Dropout(n)`. El parámetro "n" se refiere a la fracción de las neuronas de la capa que se van a desactivar aleatoriamente durante el entrenamiento. Por ejemplo, si n = 0.5, entonces la mitad de las neuronas de la capa se desactivarán aleatoriamente durante cada iteración de entrenamiento. Un ejemplo de cómo se añade una capa dropout en Keras se muestra en el código 3.2.

Se ha experimentado con distinto número de capas Dropout y tasa de eliminación. En general, los resultados mejoran con respecto a la red anterior. Un ejemplo de predicción con capas de dropout se muestra en la figura 3.4. La red sigue teniendo tendencia a predecir un círculo pero ya no es una circunferencia completa y su tamaño se ajusta en cierta medida a las paredes del ground truth.

Código 3.2 Ejemplo programación capa de dropout.

```
.
.
.
dense2 = Dense(64, activation='relu')(dropout1)
dropout2 = Dropout(0.5)(dense2)
dense3 = Dense(128, activation='relu')(dropout2)
.
.
.
```

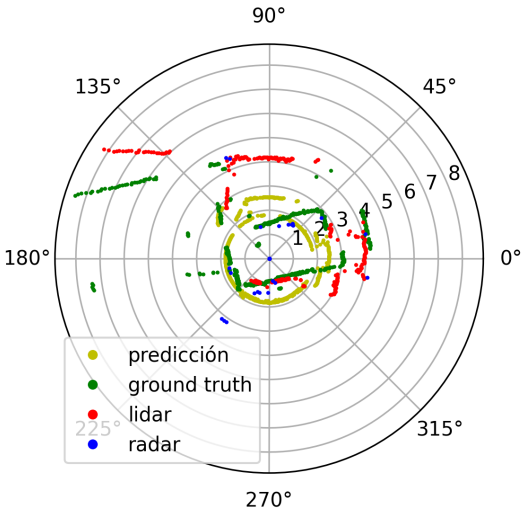


Figura 3.4 Resultado primera mejora. Empleo de capas Dropout.

3.3 Tercera propuesta: Batch Normalization

Se propone añadir capas de Batch Normalization [3] a la estructura anterior. La normalización por lotes "Batch Normalization" es una técnica utilizada para mejorar la estabilidad y el rendimiento de las redes neuronales. La idea detrás de esta estrategia es normalizar las activaciones de las neuronas en una capa para cada mini-lote durante el entrenamiento. Esto ayuda a asegurar que los valores de las activaciones permanezcan dentro de un rango razonable, lo que a su vez ayuda a estabilizar el proceso de entrenamiento y a reducir la posibilidad de sobreajuste. Se ha llevado a cabo la implementación en Keras siguiendo la programación en el código 3.3. Esta es una muestra de las distintas combinaciones de número de capas y parámetros que se han entrenado.

Código 3.3 Red neuronal artificial con capas dropout y batch normalization.

```
input1 = Input(shape=(629,1))
input2 = Input(shape=(629,1))
concat = Concatenate()([input1, input2])
dense1 = Dense(256, activation='relu')(concat)
norm1 = BatchNormalization()(dense1)
dropout1 = Dropout(0.5)(norm1)
dense2 = Dense(512, activation='relu')(dropout1)
norm2 = BatchNormalization()(dense2)
dropout2 = Dropout(0.5)(norm2)
dense3 = Dense(1024, activation='relu')(dropout2)
norm3 = BatchNormalization()(dense3)
dropout3 = Dropout(0.5)(norm3)
dense4 = Dense(2048, activation='relu')(dropout3)
norm4 = BatchNormalization()(dense4)
dropout4 = Dropout(0.5)(norm4)
output = Dense(1)(dropout4)
model = Model(inputs=[input1, input2], outputs=output)
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
model.summary()
```

Tras implementar dicha estrategia y experimentar con distinto número de capas de normalización y dropout, las predicciones no muestran diferencias notables con respecto a la red de la sección anterior. Una muestra de dichas predicciones se observa en la figura 3.5.

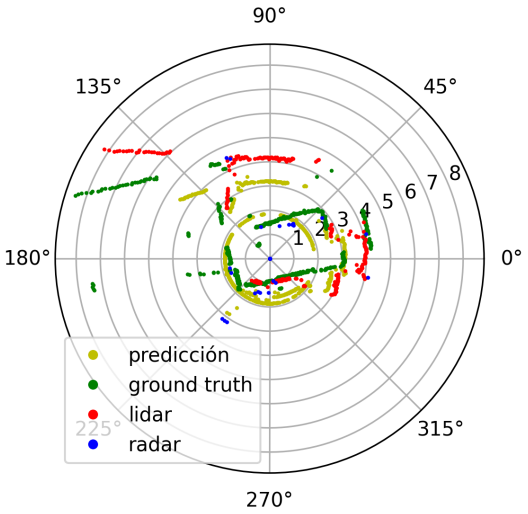


Figura 3.5 Resultado segunda mejora. Combinación dropout + batch normalization.

3.4 Conclusiones

Tras experimentar con distinto número de capas y unidades y añadir los recursos planteados en las secciones 3.2 y 3.3, se llega a la conclusión de que este tipo de estructura de red neuronal no permite resolver el problema ya que no es capaz de capturar las dependencias espaciales de los puntos representados así como la variabilidad entre distintos experimentos (por ejemplo, medidas de radar distribuidas en distintas zonas de la circunferencia).

Para solventar dicha limitación, se propone el empleo de redes convolucionales.

4 Red convolucionales

Una Red Neuronal Convolucional (ConvNet/CNN) es un algoritmo de aprendizaje profundo que puede tomar una imagen de entrada, asignar importancia (pesos y sesgos aprendibles) a varios aspectos/objetos en la imagen y ser capaz de diferenciar unos de otros. El pre-procesamiento requerido en una red convolucional es mucho menor en comparación con otros algoritmos de clasificación. Mientras que en los métodos primitivos los filtros son diseñados manualmente, con suficiente entrenamiento, las redes convolucionales tienen la capacidad de aprender estos filtros/características.

La arquitectura de una red convolucional es análoga a la del patrón de conectividad de las neuronas en el cerebro humano y fue inspirada en la organización del córtex visual. Las neuronas individuales responden a estímulos solo en una región restringida del campo visual conocida como campo receptivo. Una colección de estos campos se superponen para cubrir toda el área visual.

En casos de imágenes binarias extremadamente básicas, las redes neuronales artificiales podrían llegar a obtener un desempeño aceptable en la predicción de clases, pero tendría poca o ninguna precisión cuando se trata de imágenes complejas que tienen dependencias espaciales en todo momento.

Una red convolucional es capaz de capturar con éxito las dependencias espaciales y temporales en una imagen mediante la aplicación de filtros entrenados. Esta arquitectura realiza un mejor ajuste al conjunto de datos de imágenes debido a la reducción en el número de parámetros involucrados y la reutilización de pesos. En otras palabras, la red puede ser entrenada para entender mejor la sofisticación de la imagen.

4.1 Redes convolucionales 2D

Para llevar a cabo una implementación convolucional 2D del problema, se realiza una transformación de los datos. Dicha transformación puede describirse como un cambio de coordenadas de base polar a base cartesiana. Como resultado, las distancias representadas en vectores de 629 coordenadas se proyectan en matrices 2D formando lo que podría denominarse "imágenes". De esta forma, los datos de entrada y de salida de la red no son vectores unidimensionales sino matrices con dos dimensiones. Un ejemplo de dicha transformación se muestra en la figura 4.2. Las matrices resultado de dicha proyección tienen tamaño 128x128 y la distancia se ha normalizado empleando 20 metros como valor límite.

4.1.1 Unet 2D

La estructura de red convolucional 2D propuesta es la denominada Unet [12]. Dicha estructura está muy extendida en el desarrollo de redes neuronales cuyos datos de entrada son matrices 2D. La estructura Unet fue desarrollada por Olaf Ronneberger y otros para la segmentación de imágenes

2

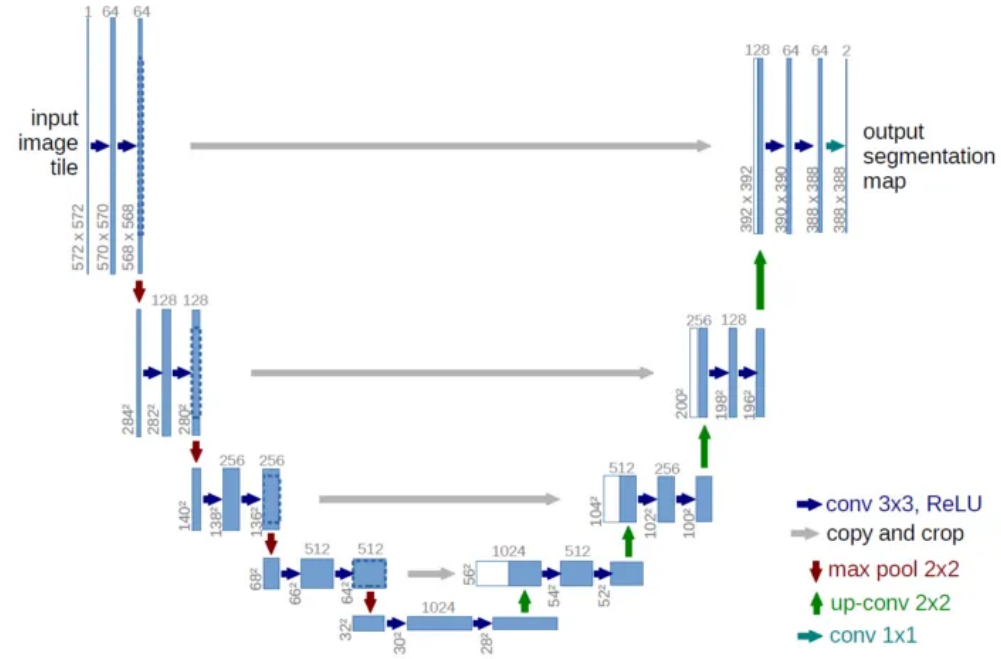


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Figura 4.1 Estructura de Unet propuesta en el paper original.

biomédicas. La arquitectura contiene dos caminos. El primer camino es el camino de contracción (también llamado codificador), que se utiliza para capturar el contexto en la imagen. El codificador es solo una pila tradicional de capas de convolución y agrupación máxima. El segundo camino es el camino simétrico de expansión (también llamado decodificador), que se utiliza para permitir la localización precisa mediante convoluciones transpuestas. Por lo tanto, es una red completamente convolucional de extremo a extremo (FCN), es decir, solo contiene capas de convolución y no contiene ninguna capa densa.

La implementación en Keras se ha realizado mediante bloques, en primer lugar descendentes y, posteriormente, ascendentes. En el código 4.1 se muestra un ejemplo de ambos bloques. La red completa se ha implementado concatenando 5 bloques descendentes y 4 ascendentes como se dibuja en la figura 4.1.

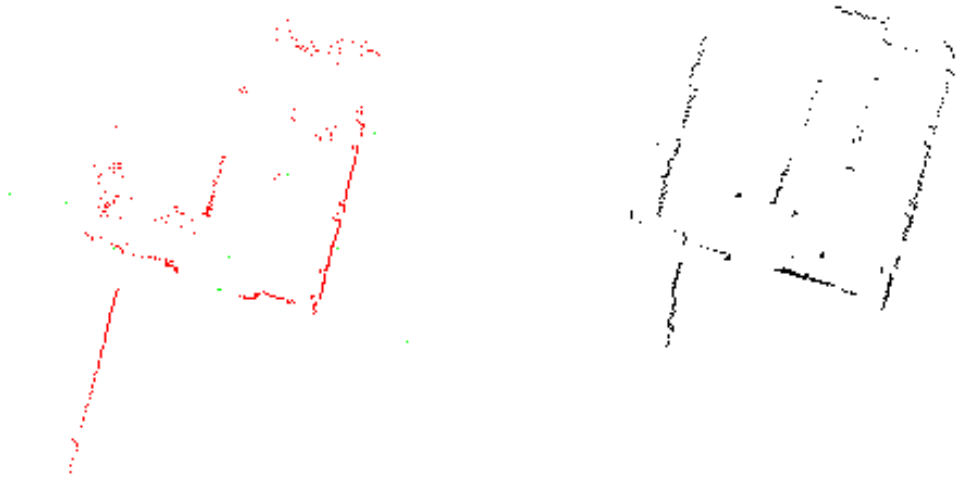


Figura 4.2 Representación gráfica de la transformación de los datos.

Código 4.1 Bloques red Unet 2D.

```
#Bloque descendente
conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_
    initializer = 'he_normal')(inputs)
conv1 = BatchNormalization()(conv1)
conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_
    initializer = 'he_normal')(conv1)
conv1 = BatchNormalization()(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
.
.
.
#Bloque ascendente
up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_
    initializer = 'he_normal')(UpSampling2D(size = (2,2))(drop5))
merge6 = concatenate([drop4,up6], axis = 3)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_
    initializer = 'he_normal')(merge6)
conv6 = BatchNormalization()(conv6)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_
    initializer = 'he_normal')(conv6)
conv6 = BatchNormalization()(conv6)
```

Se ha llevado a cabo el entrenamiento con un 80 % de las muestras dedicadas a entrenar y el 20 % restante a validar. Resultado de dicho experimento se muestra en la figura 4.3. En dicha figura se observa que los resultados no son los esperados.

La baja precisión en los resultados puede deberse a dos factores: reducido número de muestras de entrenamiento o baja resolución de las mismas. El primer problema podría atajarse empleando un proceso de data augmentation [14]. Dicho proceso toma las muestras de los experimentos originales y les realiza traslaciones y/o rotaciones para conseguir un dataset más extenso. Se ha descartado esta opción por poder incurrir en error final de predicción. El segundo problema se ha atajado repitiendo

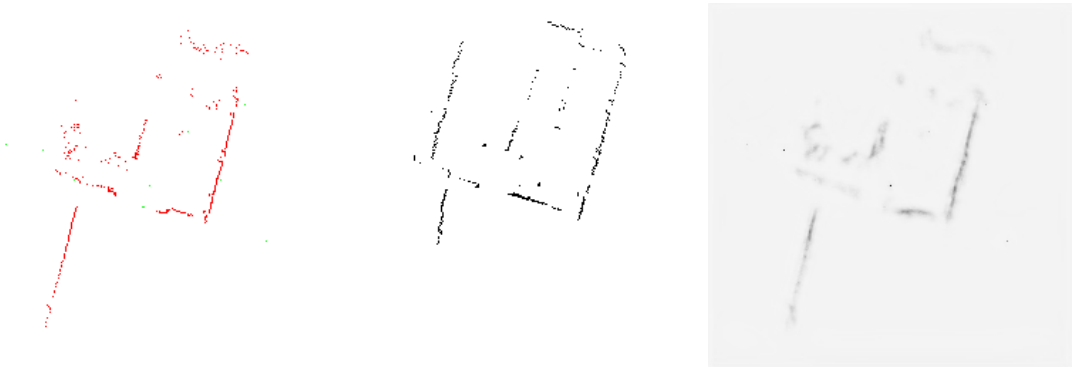


Figura 4.3 De izquierda a derecha: matriz de entrada (lidar+radar), ground truth y predicción.

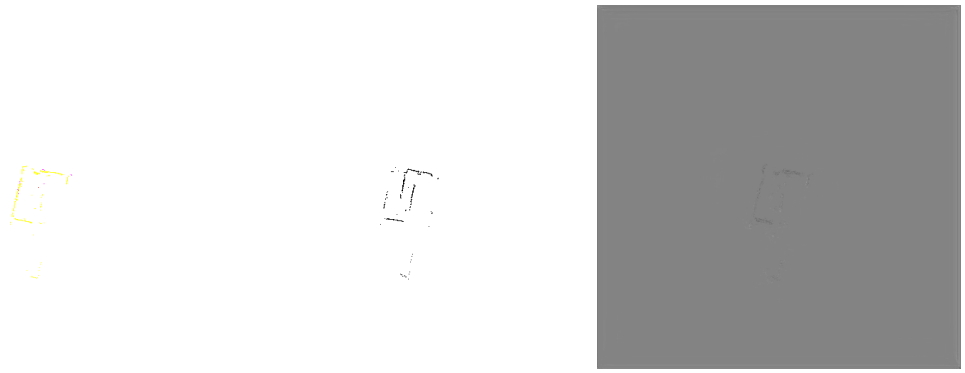


Figura 4.4 De izquierda a derecha: matriz de entrada (lidar+radar), ground truth y predicción.

el experimento con una resolución mayor de las matrices 2D, en este caso 512 x 512.

El resultado de dicha repetición se muestra en la figura 4.4. Como se puede observar, un aumento de la resolución ha tenido como consecuencia un resultado más desfavorable ya que el número de muestras no ha aumentado en consecuencia. Esto ha generado una red más compleja, que en principio necesitaría un mayor número de muestras para entrenarse, de las cuales no se dispone.

4.2 Redes convolucionales 1D

En esta sección se opta por modificar la estrategia anterior. Se emplean convoluciones 1D en lugar de 2D.

4.2.1 Primera aproximación

Se ha realizado una primera implementación de red convolucional 1D [7] con la estructura que se muestra en la figura 4.5. En el código 4.2 se muestra la implementación en Keras de dicha red. Este código define un modelo de red neuronal convolucional en Keras utilizando la clase Sequential. El modelo consta de seis capas, las cuatro primeras son capas convolucionales de una dimensión (Conv1D) que utilizan la función de activación relu. La primera capa convolucional tiene 300 filtros de tamaño 30 y recibe una entrada con forma de (629, 2), mientras que las siguientes dos capas convolucionales tienen 800 filtros de tamaño 30 cada una. Después de cada capa convolucional, se agrega una capa de MaxPooling1D de tamaño 3 para reducir la dimensión temporal de la salida. La quinta capa es una capa de GlobalAveragePooling1D que reduce la dimensión espacial a un solo valor. La sexta y última capa es una capa de Dropout con una tasa de 0.5 que se utiliza para prevenir el sobreajuste. Finalmente, se agrega una capa densa con 629 unidades y activación relu y se guarda el modelo en un archivo llamado "modeloConv1D.h5". Los datos de entrada son las coordenadas

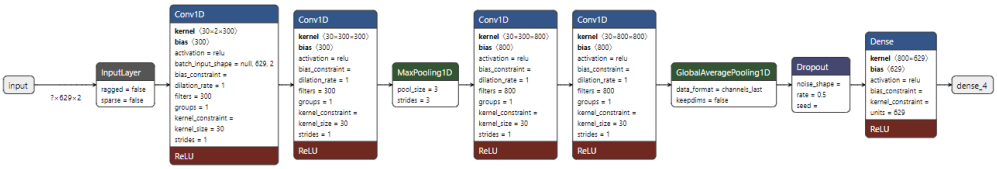


Figura 4.5 Esquema primera red convolucional propuesta.

polares de lidar y radar (matriz 629x2) y los de salida las coordenadas polares del ground truth (matriz 629x1).

Tras un primer entrenamiento los resultados no son los esperados. La red no es capaz de detectar las características principales de los datos de entrada y como resultado se obtiene una red que predice fundamentalmente círculos. El círculo en coordenadas polares tiende a minimizar el error pero no es el efecto buscado. Se muestra dicho resultado en la figura 4.6.

Código 4.2 Red convolucional 1D en Keras.

```
model_m = Sequential()
model_m.add(Conv1D(300, 30, activation='relu', input_shape=(629, 2)))
model_m.add(Conv1D(300, 30, activation='relu'))
model_m.add(MaxPooling1D(3))
model_m.add(Conv1D(800, 30, activation='relu'))
model_m.add(Conv1D(800, 30, activation='relu'))
model_m.add(GlobalAveragePooling1D())
model_m.add(Dropout(0.5)) #
model_m.add(Dense(629, activation='relu'))
print(model_m.summary())
model_m.save('modeloConv1D.h5')
```

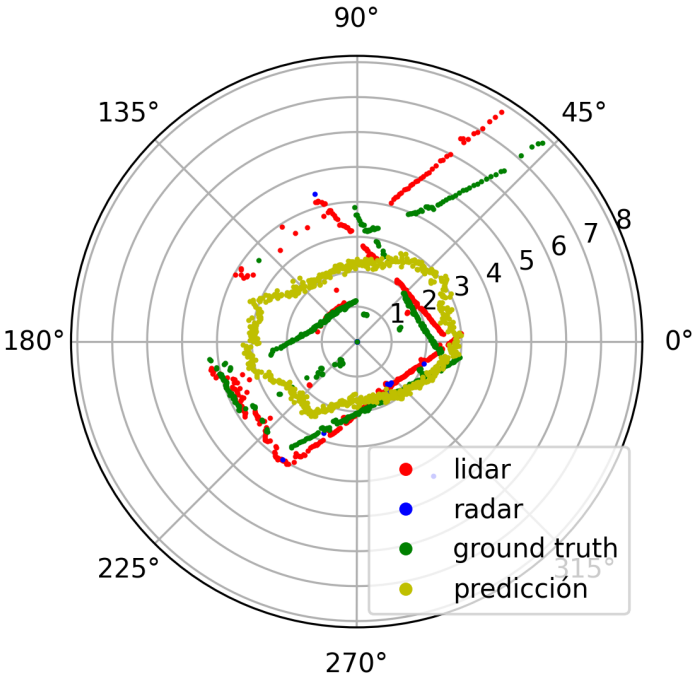


Figura 4.6 Predicción primera red convolucional 1D propuesta.

4.2.2 Estudio de hiperparámetros

Para llevar a cabo la toma de decisiones en parámetros configurables de la red anteriormente propuesta, se ha codificado un entrenamiento automatizado basado en hiperparámetros (código 4.3). Este ejercicio consiste en realizar entrenamientos consecutivos variando diferentes parámetros de la red automáticamente, para estudiar a posteriori cuál es la mejor combinación de los mismos. En este caso se han tomado como hiperparámetros:

- num_filters: es el número de filtros por capa. Cada filtro se desliza a través de la matriz y produce un mapa de activación que representa la presencia de un patrón específico en los datos de entrada.
- kernel_size: es el tamaño del filtro que se desliza a través de la matriz de entrada. Un núcleo más grande se encargará de detectar patrones más grandes, mientras que un núcleo más pequeño detectará patrones más detallados.

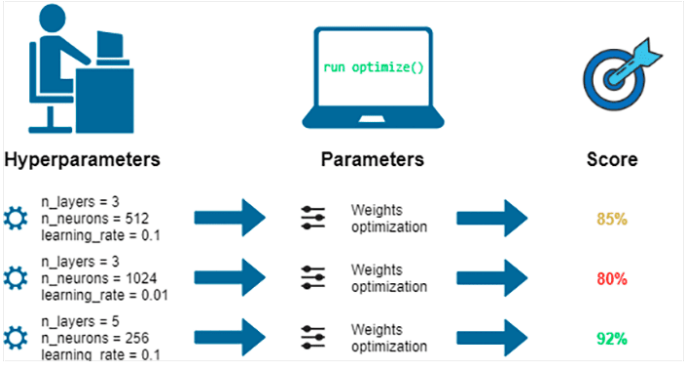


Figura 4.7 Esquema estudio hiperparámetros.

- activation: función de activación.
- pooling: La operación de pooling se aplica a los mapas de activación producidos por los filtros de convolución y se reduce su tamaño. Esto mantiene las características importantes y permite a la red neuronal generalizar mejor.
- dropout: ya se ha comentado el dropout en el capítulo anterior.

Código 4.3 Estructura red convolucional 1D con hiperparámetros.

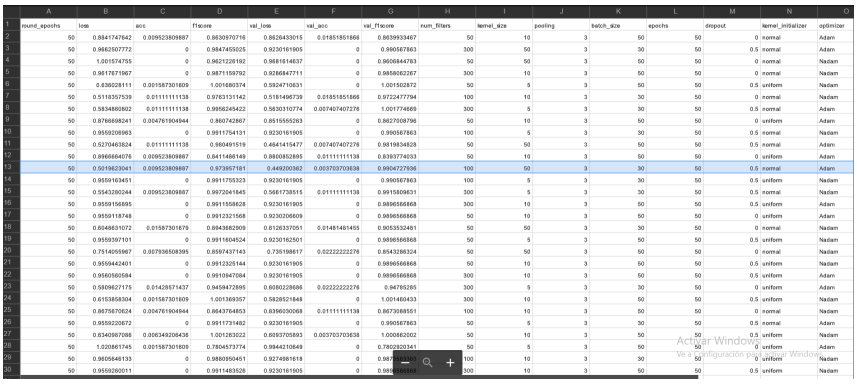
```
model = Sequential()
model.add(Conv1D(params['num_filters'], params['kernel_size'],
    activation=params['activation'], input_shape=(629, 2)))
model.add(Conv1D(params['num_filters'], params['kernel_size'],
    activation=params['activation']))

model.add(MaxPooling1D(params['pooling']))

model.add(Conv1D(params['num_filters'], params['kernel_size'],
    activation=params['activation']))
model.add(Conv1D(params['num_filters'], params['kernel_size'],
    activation=params['activation']))

model.add(GlobalAveragePooling1D())
model.add(Dropout(params['dropout']))
model.add(Dense(629, activation=params['last_activation']))
```

En la figura 4.8 se muestra una preview de los resultados obtenidos tras el entrenamiento automatizado variando los hiperparámetros. Se ha implementado la red con menor valor de "loss". Una muestra de los resultados obtenidos tras entrenar esa red se muestran en la figura 4.9. Se aprecia que los resultados son similares a los de la red anterior. No ha habido mejora considerable por lo que se opta por otra estructura de red.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	round_epochs	loss	acc	f1score	val_loss	val_acc	val_f1score	num_filters	kernel_size	pooling	batch_size	epochs	dropout	kernel_initializer	optimizer
2	50	0.8841747642	0.009523809867	0.8630970716	0.8626433015	0.01851851866	0.8639933467	50	10	3	50	50	0	normal	Adam
3	50	0.9662507772	0	0.9847455025	0.9230161905	0	0.990567863	300	50	3	50	50	0.5	normal	Adam
4	50	1.001574755	0	0.9621228192	0.9681614637	0	0.9606844783	50	50	3	50	50	0	normal	Nadam
5	50	0.9617571987	0	0.9871159792	0.9268447711	0	0.9858062267	300	10	3	50	50	0	normal	Nadam
6	50	0.836028111	0.007587301609	1.001680374	0.5924710631	0	1.001502872	50	5	3	50	50	0.5	uniform	Nadam
7	50	0.5118357539	0.01111111138	0.9763131142	0.5181496739	0.01851851866	0.9722477794	100	10	3	30	50	0	normal	Nadam
8	50	0.5834660802	0.01111111138	0.9956245422	0.5630310774	0.007407407276	1.001774669	300	5	3	30	50	0.5	normal	Adam
9	50	0.8766808241	0.004761904944	0.860742867	0.8515555283	0	0.8627008796	50	10	3	50	50	0	uniform	Adam
10	50	0.9559208963	0	0.9911754131	0.9230161905	0	0.990567863	100	5	3	30	50	0.5	normal	Nadam
11	50	0.5270463624	0.01111111138	0.980491519	0.4641415477	0.007407407276	0.9819834828	50	50	3	50	50	0.5	normal	Adam
12	50	0.8966864076	0.009523809867	0.9411466149	0.9800852895	0.01111111138	0.8393774033	50	10	3	50	50	0	uniform	Adam
13	50	0.5019623041	0.009523809867	0.973957191	0.449200362	0.003703703638	0.9904727936	100	50	3	30	50	0.5	normal	Adam
14	50	0.9559163451	0	0.9911755323	0.9230161905	0	0.990567863	100	5	3	30	50	0.5	uniform	Nadam
15	50	0.5543290244	0.009523809867	0.9972041845	0.5661738515	0.01111111138	0.9915809631	300	5	3	30	50	0.5	normal	Nadam
16	50	0.9559156895	0	0.9911558628	0.9230161905	0	0.9896566868	300	10	3	50	50	0.5	uniform	Nadam
17	50	0.9559119748	0	0.9912321568	0.9230206809	0	0.9896566868	50	10	3	50	50	0	uniform	Adam
18	50	0.6048631072	0.01587301679	0.8943982809	0.6126337051	0.01481481455	0.9053532481	50	50	3	30	50	0	normal	Nadam
19	50	0.9559397101	0	0.9911604524	0.9230162501	0	0.9896566868	50	5	3	50	50	0.5	uniform	Nadam
20	50	0.7514055967	0.007936508395	0.8597437143	0.735198617	0.02222222278	0.8543286324	50	50	3	50	50	0	normal	Nadam
21	50	0.9559442401	0	0.9912325144	0.9230161905	0	0.9896566868	50	10	3	50	50	0.5	uniform	Nadam
22	50	0.9560560584	0	0.9910947084	0.9230161905	0	0.9896566868	300	10	3	50	50	0.5	uniform	Adam
23	50	0.5809627175	0.01428571437	0.9459473895	0.6080228686	0.02222222278	0.94785285	300	5	3	30	50	0	uniform	Adam
24	50	0.6153658304	0.007587301609	1.001369357	0.5828521648	0	1.001460433	300	10	3	30	50	0.5	uniform	Nadam
25	50	0.8573570524	0.004761904944	0.8643764853	0.8396030068	0.01111111138	0.8673088551	100	10	3	30	50	0	normal	Nadam
26	50	0.9559220672	0	0.9911731482	0.9230161905	0	0.990567863	50	5	3	30	50	0.5	normal	Adam
27	50	0.6340987086	0.006349206436	1.001263022	0.6091705893	0.003703703638	1.000862002	50	10	3	50	50	0.5	uniform	Nadam
28	50	1.020861745	0.007587301609	0.7804573774	0.9944210649	0	0.7802920341	50	5	3	50	50	0	uniform	Adam
29	50	0.9605646133	0	0.9860950451	0.9274981618	0	0.987	100	10	3	30	50	0	uniform	Nadam
30	50	0.9559260011	0	0.9911483526	0.9230161905	0	0.989	300	10	3	50	50	0.5	uniform	Nadam

Figura 4.8 Resultados entrenamiento automatizado con hiperparámetros.

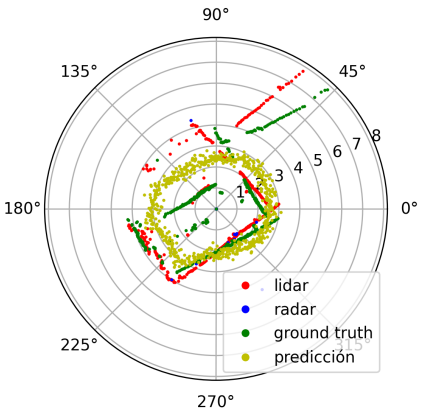


Figura 4.9 Resultados red con menor loss.

4.2.3 Unet 1D

Dados los resultados obtenidos en la sección anterior, se propone un cambio en la estructura de la red. En concreto, se opta por una red de tipo Unet 1D [5]

Unet 1D funciona igual que la versión 2D, pero es aplicable a secuencias unidimensionales, como señales de audio, secuencias de tiempo y otros datos unidimensionales. La arquitectura de Unet 1D consta de dos partes principales: una parte de codificación y una parte de decodificación. La parte de codificación se encarga de comprimir la información de la entrada y extraer características importantes, mientras que la parte de decodificación se encarga de ampliar las características comprimidas y producir la salida.

Unet 1D es una arquitectura efectiva para tareas de segmentación de señales unidimensionales, ya que permite a la red neuronal aprender patrones complejos en los datos de entrada y producir resultados precisos.

En concreto se ha propuesto una red Unet 1D con cuatro etapas de codificación y cuatro de decodificación. Se muestra dicha estructura en la figura 4.12. La implementación en Keras se ha realizado mediante bloques descendentes y bloques ascendentes tal y como se muestra en el código 4.4.

Código 4.4 Estructura red convolucional Unet1D.

```
#Bloque descendente
conv2 = Conv1D(64, kernel_size, activation="relu", padding="same")(pool
    1)
conv2 = Conv1D(64, kernel_size, activation="relu", padding="same")(conv
    2)
pool2 = MaxPooling1D(1)(conv2)
pool2 = Dropout(0.5)(pool2)
.
.
.
#Bloque ascendente
deconv4 = Conv1DTranspose(512, kernel_size, padding="same")(convm)
uconv4 = concatenate([deconv4, conv4])
uconv4 = Dropout(0.5)(uconv4)
```

```
uconv4 = Conv1D(512, kernel_size, activation="relu", padding="same")(
    uconv4)
uconv4 = Conv1D(512, kernel_size, activation="relu", padding="same")(
    uconv4)
```

Una muestra de los resultados tras el entrenamiento se muestran en la figura 4.10.

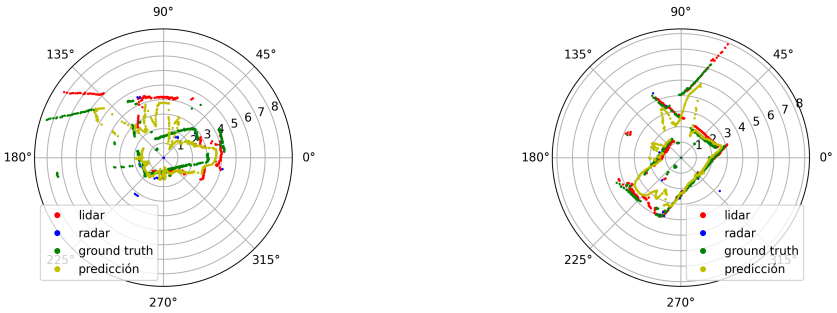


Figura 4.10 Resultados red Unet 1D.

Esta estructura de red es la que mejores resultados ha aportado, teniendo mejor rendimiento en los experimentos en los que los datos de radar presentan mayor similitud con el ground truth.

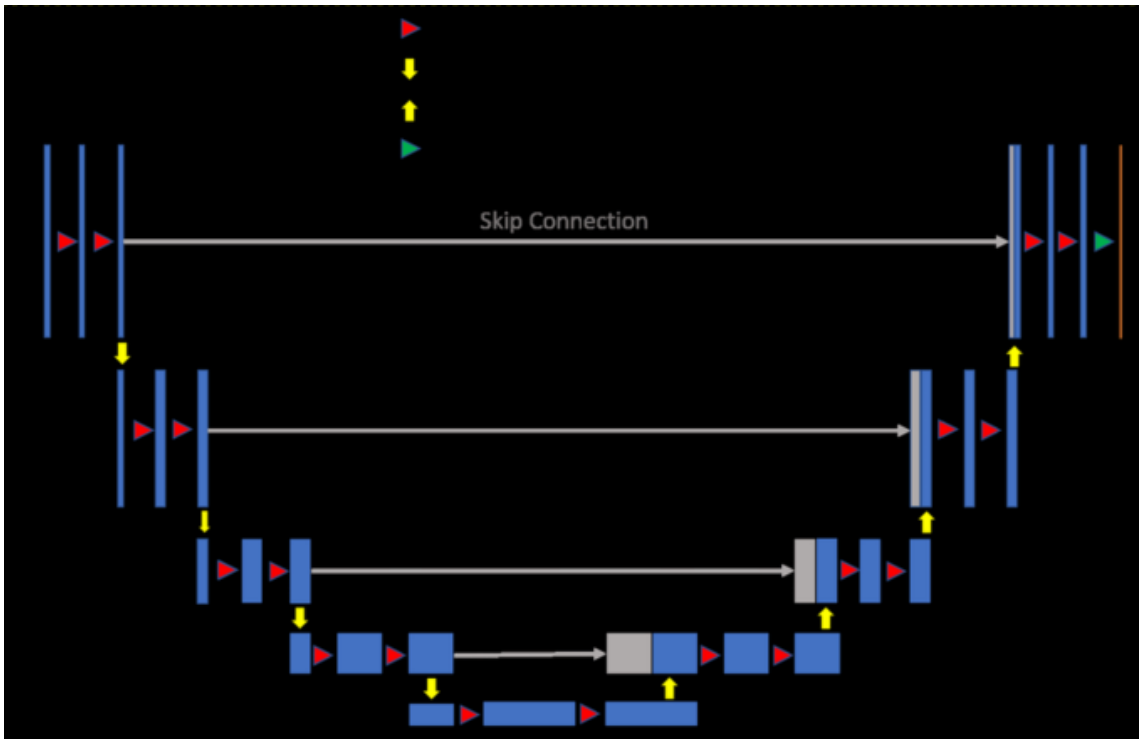


Figura 4.11 Estructura red Unet 1D.

4.2.4 Unet ++

Dado los resultados prometedores que ha arrojado la estructura Unet 1D se ha optado por dar el salto a Unet++. Unet++ es una versión evolucionada de la red neuronal Unet. Tiene tres mejoras en comparación con Unet:

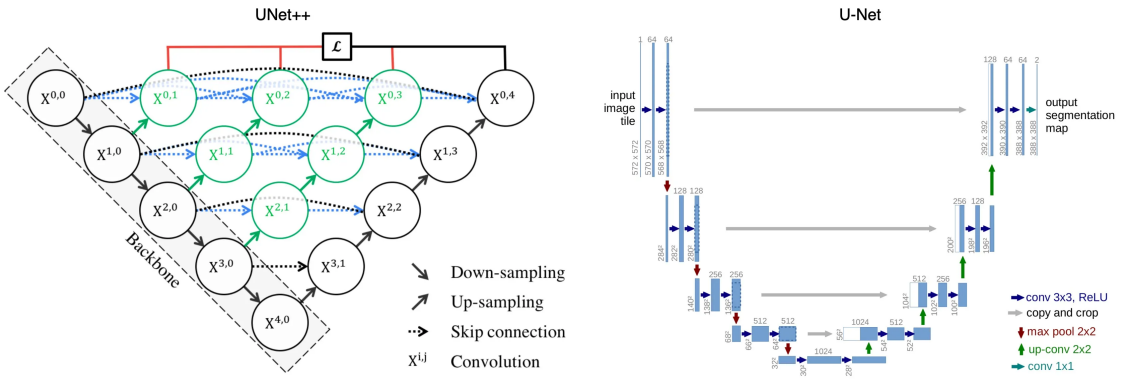


Figura 4.12 Estructura red Unet++ 1D.

- Rediseño de las vías de omisión para reducir la brecha semántica entre el codificador y el decodificador.
- Conexiones de omisión densas para mejorar la precisión de la segmentación y el flujo de gradientes.
- Supervisión profunda para mejorar la calidad de la segmentación en diferentes niveles.

En Unet, las vías de omisión conectan directamente los mapas de características entre el codificador y el decodificador, lo que puede fusionar mapas de características semánticamente diferentes. En Unet++, las vías de omisión están rediseñadas para reducir esta brecha semántica.

Las conexiones de omisión densas en Unet++ aseguran que todas las características anteriores se acumulen y lleguen al nodo actual. Esto mejora la precisión de la segmentación y el flujo de gradientes.

La supervisión profunda en Unet++ utiliza múltiples niveles de segmentación en lugar de uno solo. Esto mejora la calidad de la segmentación a diferentes niveles de detalle en la imagen.

La implementación del modelo se ha realizado en Keras siguiendo el esquema propuesto en la figura 4.12. Para ello, se emplea la función AvgPool1D() para implementar los caminos "Down-sampling", la función Conv1DTranspose() para implementar los caminos "Up-sampling" y la función concatenate() para implementar los caminos "Skip connection". Las convoluciones son el conjunto Conv1D() y BatchNormalization().

Código 4.5 Implementación Unet++.

```
conv0_0 = bloque_conv_relu(inputs, filtro=filt[0])
pool1 = AvgPool1D((2, 2), strides=(2, 2))(conv0_0)

conv1_0 = bloque_conv_relu(pool1, filtro=filt[1])
pool2 = AvgPool1D((2, 2), strides=(2, 2))(conv1_0)

up0_1 = Conv1DTranspose(nb_filter[0], (2, 2), strides=(2, 2),padding='
    same')(conv1_0)
conv0_1 = concatenate([up0_1, conv0_0], axis=bn_axis)
conv0_1 = bloque_conv_relu(conv0_1, filtro=filt[0])

conv2_0 = bloque_conv_relu(pool2, filtro=filt[2])
pool3 = AvgPool1D((2, 2), strides=(2, 2))(conv2_0)
```

```
up1_1 = Conv1DTranspose(nb_filter[1], (2, 2), strides=(2, 2), padding='
    same')(conv2_0)
conv1_1 = concatenate([up1_1, conv1_0], axis=bn_axis)
conv1_1 = bloque_conv_relu(conv1_1, filtro=filt[1])

up0_2 = Conv1DTranspose(nb_filter[0], (2, 2), strides=(2, 2), padding='
    same')(conv1_1)
conv0_2 = concatenate([up0_2, conv0_0, conv0_1], axis=bn_axis)
conv0_2 = bloque_conv_relu(conv0_2, filtro=filt[0])

conv3_0 = bloque_conv_relu(pool3, filtro=filt[3])
pool4 = AvgPool1D((2, 2), strides=(2, 2))(conv3_0)

up2_1 = Conv1DTranspose(nb_filter[2], (2, 2), strides=(2, 2), padding='
    same')(conv3_0)
conv2_1 = concatenate([up2_1, conv2_0], axis=bn_axis)
conv2_1 = bloque_conv_relu(conv2_1, filtro=filt[2])

up1_2 = Conv1DTranspose(nb_filter[1], (2, 2), strides=(2, 2), padding='
    same')(conv2_1)
conv1_2 = concatenate([up1_2, conv1_0, conv1_1], axis=bn_axis)
conv1_2 = bloque_conv_relu(conv1_2, filtro=filt[1])

up0_3 = Conv1DTranspose(nb_filter[0], (2, 2), strides=(2, 2), padding='
    same')(conv1_2)
conv0_3 = concatenate([up0_3, conv0_0, conv0_1, conv0_2], axis=bn_axis)
conv0_3 = bloque_conv_relu(conv0_3, filtro=filt[0])

conv4_0 = bloque_conv_relu(pool4, filtro=filt[4])

up3_1 = Conv1DTranspose(nb_filter[3], (2, 2), strides=(2, 2), padding='
    same')(conv4_0)
conv3_1 = concatenate([up3_1, conv3_0], axis=bn_axis)
conv3_1 = bloque_conv_relu(conv3_1, filtro=filt[3])

up2_2 = Conv1DTranspose(nb_filter[2], (2, 2), strides=(2, 2), padding='
    same')(conv3_1)
conv2_2 = concatenate([up2_2, conv2_0, conv2_1], axis=bn_axis)
conv2_2 = bloque_conv_relu(conv2_2, filtro=filt[2])

up1_3 = Conv1DTranspose(nb_filter[1], (2, 2), strides=(2, 2), padding='
    same')(conv2_2)
conv1_3 = concatenate([up1_3, conv1_0, conv1_1, conv1_2], axis=bn_axis)
conv1_3 = bloque_conv_relu(conv1_3, filtro=filt[1])

up0_4 = Conv1DTranspose(nb_filter[0], (2, 2), strides=(2, 2), padding='
    same')(conv1_3)
conv0_4 = concatenate([up0_4, conv0_0, conv0_1, conv0_2, conv0_3], axis=
    bn_axis)
conv0_4 = bloque_conv_relu(conv0_4, filtro=filt[0])
```

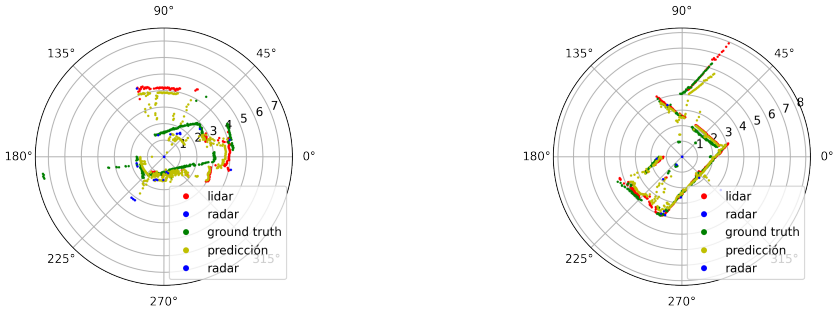


Figura 4.13 Resultados red Unet++ 1D.

Se ha realizado un entrenamiento de la red con el dataset de entrenamiento. Una muestra de las predicciones obtenidas se muestran en la figura 4.13.

Como se aprecia en los resultados mostrados, la red hace predicciones menos erráticas que con la anterior propuesta. Sin embargo, sigue presentando dificultades cuando los datos de lidar distan demasiado del ground truth.

5 Conclusiones

Como se ha mostrado en los sucesivos experimentos realizados en el presente trabajo, la fusión de datos de lidar y radar permite obtener una imagen de los obstáculos del entorno de medida más cercana a la realidad que empleando únicamente los sensores por separado. En el caso concreto del dataset de entrenamiento y validación de este proyecto, al tratarse de datos vectoriales, las redes unidimensiones han superado el rendimiento de las bidimensionales. Sin embargo, incluso las aquitecturas que mejores resultados han aportado (Unet 1D y Unet++ 1D) presentan problemas cuando los datos de lidar y radar distan del ground truth.

En una primera aproximación al problema, antes de analizar el dataset, pudiera suponerse que los datos de radar, pese a ser inferiores en número, iban a aportar la precisión que faltaba en algunos casos en los datos de lidar. Sin embargo, se ha comprobado experimentalmente que esto no es así. Una pequeña representación de este inconveniente se muestra en la figura 5.1. Los datos de radar, además de muy inferiores en número, carecen de la precisión esperada.

Pese a este inconveniente, las redes Unet 1D y Unet++ 1D han demostrado que poseen un gran potencial en la fusión de datos de lidar y radar, tal y como se ha mostrado en sus respectivas secciones, por lo que se propone como trabajo futuro repetir dichos entrenamientos con otro dataset en el que los datos de radar provengan de un sensor con mayor preción y número de muestras.

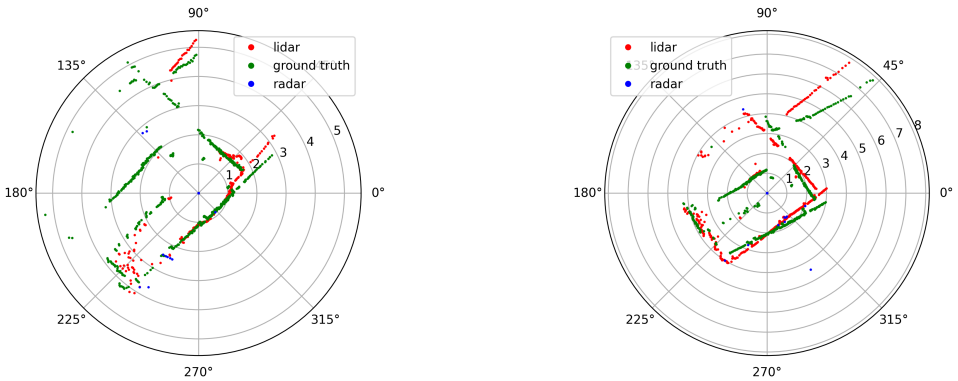


Figura 5.1 Muestras del dataset.

Índice de Figuras

1.1 Representación gráfica del esquema propuesto 4
1.2 Representación gráfica de uno de los experimentos. Rojo: lidar, azul: radar, verde: ground truth 5
1.3 Vista previa de uno de los experimentos 5

2.1 Robot móvil en entorno con fuego y humo. Fuente: [9] 7
2.2 Rendimiento de distintos sensores en entornos con humo. Procede de [13] 8

3.1 Esquema representativo red neuronal artificial 11
3.2 Esquema red inicial 12
3.3 Resultado primera propuesta red neuronal 13
3.4 Resultado primera mejora. Empleo de capas Dropout 14
3.5 Resultado segunda mejora. Combinación dropout + batch normalization 15

4.1 Estructura de Unet propuesta en el paper original 18
4.2 Representación gráfica de la transformación de los datos 19
4.3 De izquierda a derecha: matriz de entrada (lidar+radar), ground truth y predicción 20
4.4 De izquierda a derecha: matriz de entrada (lidar+radar), ground truth y predicción 20
4.5 Esquema primera red convolucional propuesta 21
4.6 Predicción primera red convolucional 1D propuesta 22
4.7 Esquema estudio hiperparámetros 22
4.8 Resultados entrenamiento automatizado con hiperparámetros 23
4.9 Resultados red con menor loss 24
4.10 Resultados red Unet 1D 25
4.11 Estructura red Unet 1D 25
4.12 Estructura red Unet++ 1D 26
4.13 Resultados red Unet++ 1D 28

5.1 Muestras del dataset 29

Bibliografía

[1] *Keras*, *https://keras.io/*, 2015, Último acceso abril 2023.

[2] Pierre Baldi and Peter J Sadowski, *Understanding dropout*, Advances in Neural Information Processing Systems (C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[3] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger, *Understanding batch normalization*, Advances in Neural Information Processing Systems (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[4] Johann Borenstein, Hobart R. Everett, and Liqiang Feng, *Where am i?" sensors and methods for mobile robot positioning*, 1996.

[5] Zhenqin Chen, Mengying Wang, Meiyu Zhang, Wei Huang, Hanjie Gu, and Jinshan Xu, *Post-processing refined ecg delineation based on 1d-unet*, Biomedical Signal Processing and Control **79** (2023), 104106.

[6] Xinxin Du, Marcelo H. Ang, and Daniela Rus, *Car detection for autonomous vehicle: Lidar and vision fusion approach through deep learning framework*, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 749–754.

[7] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman, *1d convolutional neural networks and applications: A survey*, Mechanical Systems and Signal Processing **151** (2021), 107398.

[8] Qiang Li, Ranyang Li, Kaifan Ji, and Wei Dai, *Kalman filter and its application*, 2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), 2015, pp. 74–77.

[9] Sen Li, Chunyong Feng, Yunchen Niu, Long Shi, Zeqi Wu, and Huaitao Song, *A fire reconnaissance robot based on slam position, thermal imaging technologies, and ar display*, Sensors **19** (2019), no. 22.

[10] Faten Hamed Nahhas, Helmi Z. M. Shafri, Maher Ibrahim Sameen, Biswajeet Pradhan, and Shattri Mansor, *Deep learning approach for building detection using lidar–orthophoto fusion*, Journal of Sensors **2018** (2018), 7212307.

[11] Felix Nobis, Maximilian Geisslinger, Markus Weber, Johannes Betz, and Markus Lienkamp, *A deep learning-based radar and camera sensor fusion architecture for object detection*, 2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF), 2019, pp. 1–7.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *U-net: Convolutional networks for biomedical image segmentation*, CoRR **abs/1505.04597** (2015).

[13] Joseph W. Starr and B. Y. Lattimer, *Evaluation of navigation sensors in fire smoke environments*, Fire Technology **50** (2013), no. 6, 1459–1481.

[14] David A van Dyk and Xiao-Li Meng, *The art of data augmentation*, Journal of Computational and Graphical Statistics **10** (2001), no. 1, 1–50.