UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Supervisión de fallos con OTDR en redes ópticas pasivas mediante técnicas de aprendizaje automático

Autor:
Rubén Urraca Torices
Tutor:
Dña. Noemí Merayo Álvarez
D. Juan Carlos Aguado Manzano

TÍTULO: **Supervisión de fallos con OTDR en redes ópticas pasivas mediante técnicas de aprendizaje automático**
AUTOR: **Rubén Urraca Torices**
TUTOR: **Dña. Noemí Merayo Álvarez y D. Juan Carlos Aguado Manzano**
DEPARTAMENTO: **Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL
PRESIDENTE: Ignacio de Miguel Jiménez
SECRETARIO: Noemí Merayo Álvarez
VOCAL: J. Carlos Aguado Manzano
SUPLENTE: Ramón J. Durán Barroso
SUPLENTE: Rubén M. Lorenzo Toledo

FECHA:
CALIFICACIÓN:

Resumen

En este Trabajo Fin de Grado (TFG), se ha llevado a cabo un estudio de investigación centrado en la monitorización de fallos en redes de acceso ópticas pasivas (*Passive Optical Networks*, PON) y su validación utilizando reflectometría óptica en el dominio del tiempo (OTDR, *Optical Time Domain Reflectometer*).

Para ello se ha hecho uso del OTDR cuyo modelo es PROLITE-50, proporcionado por la compañía PROMAX, así como su software, que permite analizar las trazas tomadas con dicho instrumento.

Con el objetivo de desarrollar un sistema de Inteligencia Artificial (IA), se creó una base de datos apropiada que contenía los datos de las medidas obtenidas anteriormente, la cual utilizará la IA para su análisis, definiendo así nuestro modelo de red neuronal basado en técnicas de Machine Learning (ML).

Por último, se llevó a cabo el diseño y programación de un sistema de IA basado en redes neuronales y que hará uso de la base de datos creada, que nos dará los resultados deseados para nuestro análisis; es decir, que detecte correctamente el fallo en una de las ramas de una red PON, dando como resultado una identificación acertada de la localización del fallo ocurrido.

Palabras clave

PON (Red Óptica Pasiva), Python, Red neuronal, OTDR (Reflectometría Óptica en el Dominio del Tiempo), Trazas, Base de Datos, Modelo, IA (Inteligencia Artificial), ML (Machine Learning)

Abstract

In this Final Degree Project, a research study has been carried out, focused on fault monitoring in Passive Optical Network (PON) systems and their validation using Optical Time Domain Reflectometer (OTDR).

For this purpose, the model PROLITE-50 was the OTDR used, provided by PROMAX, along with its software, allowing the analysis of traces taken with this instrument.

Chasing the goal of developing a program focused on Artificial Intelligence (AI), an appropriate database was created, with the data from the measurements obtained previously. The AI will use this database for the analysis, thus defining our neural network model based on Machine Learning (ML) techniques.

Finally, an AI system based on neural networks and which will use the created database was designed and programmed to provide the desired results for our analysis; that is, to correctly detect the fault in the corresponding branch, resulting in accurate identification of the fault location.

Keywords

PON (*Passive Optical Network*), Python, Neural Network, OTDR (Optical Time Domain Reflectometer), Traces, Database, Model, AI (Artificial Intelligence), ML (Machine Learning)

Agradecimientos

A mi familia por apoyarme, permitiéndome dedicarle al trabajo todo el tiempo que ha requerido, lo cual, en caso contrario, no hubiera sido posible.

A mis tutores, Noemí y Juan Carlos, por la ayuda, consejos, enseñanzas y dedicación que notablemente me han aportado a lo largo de toda la realización de este trabajo.

A mis amigos David y Daniel, y mi pareja Andrea, por haber estado siempre dispuestos a ayudarme y animarme cuando lo he necesitado.

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades, la Agencia Estatal de Investigación y por FEDER/UE (proyecto PID2023-148104OB-C41 financiado por MICIU/AEI/10.13039/501100011033 y por FEDER/UE).

This work has been funded by Ministerio de Ciencia, Innovación y Universidades, Agencia Estatal de Investigación and by ERFD/EU (grant PID2023-148104OB-C41 funded by MICIU/AEI/10.13039/501100011033 and by ERFD/EU).

Índice

Índice de figuras

Índice de tablas

1

Introducción

1.1 Motivación

Las Redes Ópticas Pasivas (PON, *Passive Optical Networks*) han sufrido una gran evolución los últimos años para poder satisfacer una demanda de servicios cada vez mayor, por lo que se han convertido en una solución eficiente y fundamental que, sin embargo, no está exenta de fallos. Son especialmente vulnerables a cortes de fibra o a defectos en los transmisores/receptores de las unidades de red óptica (ONU, *Optical Network Units*); fallos que no solo afectan al usuario final, sino que también pueden generar importantes pérdidas económicas para los operadores de red.

Uno de los principales desafíos es la identificación del origen del fallo de forma rápida y precisa, lo cual no es trivial, pues a medida que las redes se expanden, también lo hace su complejidad, dificultando la monitorización y reduciendo la fiabilidad de los sistemas tradicionales de supervisión. Por ello, surge la necesidad de implementar soluciones más avanzadas capaces de adaptarse a las características actuales.

En este contexto, el aprendizaje automático o ML (Machine Learning) representa una herramienta prometedora por su gran capacidad de manejar grandes volúmenes de datos y detectar patrones complejos. En este trabajo se propone precisamente un enfoque basado en ML, utilizando datos experimentales de reflectometría óptica en el dominio del tiempo (OTDR, *Optical Time Domain Reflectometer*) para poder evaluar el estado de la fibra.

Así pues, este proyecto se motiva por la necesidad de hacer más fiables y eficientes los sistemas de monitorización en redes ópticas, apostando por la innovación tecnológica como vía para optimizar el servicio y reducir costes operativos.

1.2 Objetivos

1.2.1 Objetivos Generales

Los objetivos generales de este trabajo son dos. En primer lugar, se pretende analizar los diferentes resultados obtenidos con datos experimentales para comprender con mayor facilidad la complejidad de los sistemas PON, así como los eventos de reflexión y patrones de retrodispersión que tienen lugar en la fibra.

En segundo lugar, se realizará el diseño e implementación de un sistema de inteligencia artificial capaz de identificar posibles fallos en la fibra óptica dentro del contexto de una red PON; todo ello en el lenguaje de programación Python [1] y mediante ML, en concreto redes neuronales.

1.2.2 Objetivos específicos

Se describen, a continuación, los objetivos específicos necesarios para la consecución de los objetivos generales ya comentados.

1. Estudiar el funcionamiento y las capacidades de la reflectometría óptica en el dominio del tiempo (OTDR) como técnica para la recolección de datos en redes ópticas.

2. Analizar los diferentes tipos de fallos que afectan a las redes PON, especialmente aquellos relacionados con la fibra óptica.

3. Realizar una base de datos con las medidas tomadas, habiéndonos cerciorado previamente de su validez y correlacionándolas de una manera conocida.

4. Diseñar un programa capaz de generar trazas de carácter sintético para añadir a las medidas reales y crear así una base de datos más grande y robusta.

5. Diseñar un modelo de clasificación basado en una red neuronal que utilizará la base de datos previa capaz de detectar fallos en redes PON.

6. Entrenar, validar y probar el modelo propuesto con los datos experimentales reales y sintéticos, evaluando su precisión, fiabilidad y capacidad de generalización.

1.3 Fases y metodología

Se procede a detallar las fases seguidas y la metodología llevada a cabo en cada una de ellas durante la realización de este trabajo.

1.3.1 Fase de iniciación

En esta primera fase se pretende afinar los conocimientos iniciales necesarios para un desarrollo adecuado de este Trabajo Fin de Grado:

- Estudio de la topología y principales características de redes PON.
- Estudio de los principales efectos negativos que afectan a la fibra óptica.
- Documentación y estudio del manual del OTDR [2] para conocer su funcionamiento, así como del software utilizado.
- Familiarización con el lenguaje de programación Python.
- Familiarización con técnicas de inteligencia artificial, especialmente en sistemas ML basados en redes neuronales.

1.3.2 Fase de adquisición

Durante esta fase se obtendrán todas las trazas y medidas necesarias para su posterior análisis y utilización. Es la fase que más tiempo conlleva, pero, a su vez, es absolutamente crucial para manejar con precisión los datos reales experimentales.

En ella, se toman las trazas de la red PON con el OTDR, se transfieren al software de gestión de este dispositivo, el cual las muestra y exporta, y se grafican para su análisis. Bajo los resultados del susodicho proceso, se llevan a cabo correcciones en las medidas, permitiéndonos, tras cierto número de ajustes, comprender mejor los elementos utilizados en la red y obtener unos resultados exentos de errores.

1.3.3 Fase de integración de medidas

En esta fase, se agruparon todas las medidas finales en un único archivo. A partir de este, se desarrolló un programa en Matlab [3] que permitió generar trazas sintéticas basadas en las originales, incorporando diferentes niveles de ruido. De este modo, se obtuvo la base de datos final deseada, la cual contiene un número relativamente grande de

muestras, adecuado para su uso en aplicaciones de inteligencia artificial.

1.3.4 Fase de Implementación

En esta última fase, se pretende llevar a cabo la implementación del código de la IA basado en redes neuronales que utilizará la base de datos para ser entrenada, validada y probada, aportándonos tras su finalización unos resultados que se verán sometidos a un análisis para la detección de fallos en redes de acceso PON.

1.4 Estructura de la memoria del TFG

En el Capítulo 1 se ha llevado a cabo una introducción general sobre este Trabajo Fin de Grado.

En el Capítulo 2 se realiza el análisis de artículos publicados y ya existentes, relacionados con la materia a tratar.

El Capítulo 3 describe las herramientas de trabajo que serán empleadas a lo largo del mismo, así como los programas utilizados.

En el Capítulo 4 se hace una descripción más detallada del OTDR, además de una minuciosa explicación de cómo se realizó la configuración, toma de medidas y extracción de las trazas.

El Capítulo 5 trata el dataset de medidas; es decir, el procedimiento de realización de la base de datos a partir de las trazas reales y las sintéticas realizadas con un programa de Matlab.

En el Capítulo 6 se explica el diseño de la red neuronal utilizada para la detección de fallos con el programa de la IA realizado.

El Capítulo 7 recoge las conclusiones generales de este Trabajo Fin de Grado y las líneas futuras que podrían suceder a este estudio.

Por último, en el Capítulo 8 se encuentran las referencias bibliográficas que han servido de apoyo para la realización de este trabajo.

2

Análisis del estado del arte

2.1 Introducción

Este capítulo se va a centrar en primer lugar en describir en líneas generales los principios fundamentales de la IA y el aprendizaje automático, por ser las herramientas que se utilizarán en este TFG. A continuación, se describirá qué es una red de acceso óptica pasiva, y se hará un análisis del estado del arte de otras investigaciones centradas en la detección de fallos con técnicas de IA en este tipo de redes. En concreto, se analizará la relación y similitudes que tiene este proyecto con otros artículos de investigación ya publicados, así como la importancia que aportan al trabajo. Así, se describirán los principales artículos encontrados más cercanos a esta investigación y recientes. En concreto, los artículos analizados son del 2022, 2023 y 2024 [4], [5], [6] y [7], y estarán debidamente referenciados en el capítulo final de bibliografía.

2.2 Introducción a la Inteligencia Artificial y Machine Learning

Recientemente [16], las técnicas de IA y ML han adquirido un papel clave en numerosos sectores, incluyendo el de las telecomunicaciones. La IA se refiere a sistemas que tienen la capacidad de realizar ciertas tareas las cuales normalmente requieren inteligencia humana, como reconocer patrones, aprender de datos o tomar decisiones. A una rama dentro de ella se le llama Machine Learning o aprendizaje automático (ML), la cual permite a los algoritmos mejorar su rendimiento y eficacia mediante el uso de la experiencia; es decir, a medida que van procesando más datos. Las redes neuronales son

una de las técnicas más utilizadas dentro del ML, que transforman los datos de entrada en una salida deseada. Destacan por su capacidad para analizar grandes volúmenes de datos, detectar patrones complejos y adaptarse a condiciones variables, lo que las hace especialmente adecuadas para tareas de diagnóstico, predicción y automatización.

Para este trabajo, resulta particularmente relevante el uso de redes neuronales, especialmente los modelos multicapa (Multilayer Perceptron, MLP) y sus versiones recurrentes, como *Recurrent Neural Network* (RNN), *Long Short-Term Memory* (LSTM) o *Gated Recurrent Unit* (GRU). Estas arquitecturas han demostrado ser altamente eficaces en tareas como la clasificación de secuencias y el análisis de señales temporales. Su utilidad se puede aplicar de manera efectiva en nuestro caso de investigación, es decir en el análisis y procesamiento de trazas extraídas de una red PON a través de un OTDR, que pueden interpretarse como señales con gran cantidad de información distribuida en el dominio temporal o espacial.

Por otro lado, los métodos de entrenamiento, test y validación de algoritmos de ML han evolucionado notablemente, incorporando estrategias como la validación cruzada y el uso de métricas específicas (precisión, recall, F1-score). Asimismo, se han introducido técnicas como el Grid Search para ajustar los hiperparámetros del modelo. Estas métricas y técnicas se ven implementadas en este TFG como se verá en capítulos sucesores, lo que permite afinar su comportamiento y mejorar su rendimiento general de forma estructurada.

En definitiva, este proyecto se enmarca dentro de esa tendencia, aplicando modelos de ML a la identificación de fallos en redes PON, y demostrando la efectividad de combinar inteligencia artificial con ingeniería de telecomunicación.

2.3 Estado del arte de la monitorización de fallos en redes PON

Una red PON (*Passive Optical Network*) es una arquitectura de red de acceso pasiva que utiliza fibra óptica y componentes pasivos para ofrecer servicios de banda ancha. Su arquitectura punto a multipunto permite que una sola fibra desde la central se divida mediante divisores ópticos hasta múltiples usuarios finales. No requiere componentes activos intermedios, lo que reduce costes de mantenimiento y energía. Se compone principalmente de un OLT (*Optical Line Terminal*) localizado en las dependencias del

operado/proveedor de servicios, divisores ópticos y múltiples ONT/ONU (*Optical Network Terminal/Unit*) en el lado de la casa del usuario final. Las PON son eficientes para despliegues FTTH (*Fiber to the Home*), ofreciendo altas velocidades y ancho de banda. Entre sus variantes actualmente más desplegadas destacan GPON (basada en Gigabit), EPON (basada en Ethernet) y XGS-PON (10G Symmetric PON). La investigación de este TFG se ha realizado sobre una arquitectura GPON.

Dado que estas redes transportan datos a altas velocidades para millones de usuarios, garantizar la seguridad y confiabilidad de la comunicación es crucial. Las fibras ópticas pueden presentar anomalías por fallos físicos, como cortes de fibra, o por ataques maliciosos, como la interceptación óptica. Estos fallos pueden comprometer la confidencialidad y disponibilidad del servicio, generando grandes pérdidas económicas y de información. Por ello, la detección temprana y precisa de anomalías es fundamental. En este contexto, el aprendizaje automático (ML) surge como una herramienta crucial para identificar, diagnosticar y localizar fallos en tiempo real, mejorando la resiliencia y fiabilidad de las redes PON. Así pues, el contexto de esta investigación se focaliza en el uso de ML para la detección de fallos en la fibra óptica de una red GPON. A continuación, se describen los principales artículos de investigación más recientes que tienen una relación directa con nuestra investigación.

El primer artículo analizado y en el que inicialmente se ha basado el proyecto [4], ha constituido la principal base metodológica del TFG. En dicho artículo, los autores plantean el uso de redes neuronales con capas intermedias GRU para poder clasificar fallos en redes ópticas pasivas y utilizando trazas de un OTDR. La relación que tiene con este TFG es fácilmente visible, pues es un proyecto muy similar. El TFG adopta un sistema muy parecido, utilizando datos reales sacados de un OTDR. Sin embargo, dada la escala del proyecto, el volumen de datos y la arquitectura neuronal utilizados en el TFG son más limitados. Este artículo, por tanto, tiene gran importancia, pues valida la utilización de técnicas ML para la clasificación precisa de los fallos, justificando de esta manera la viabilidad técnica del TFG.

Abdelli et al. (2022) proponen en [5] un sistema de detección de anomalías en la monitorización de fibras ópticas basado en técnicas de aprendizaje automático, demostrando su eficacia para identificar eventos irregulares en entornos reales. Este artículo complementa al artículo anterior, centrándose ahora en la detección de anomalías

en las trazas utilizando un enfoque híbrido que consta de autoencoders para la detección inicial y GRU bidireccionales para la clasificación y localización precisa de los fallos. Los autoencoders son redes neuronales que se entrenan con datos sin fallos y aprenden una representación comprimida de ellos. Cuando se presenta una anomalía, no son capaces de reconstruir bien los datos y deducen que ha ocurrido un fallo. En el TFG se toma la idea de usar una arquitectura simple sin necesidad de autoencoders primero y GRU bidireccionales después que clasifiquen el fallo detectado por los autoencoders. Por tanto, tiene en común con el artículo el principio de clasificación en presencia de ruido. En resumen, no se implementa una estructura tan compleja como en el artículo, la IA implementada en el TFG clasifica directamente los fallos ya que es entrenada con datos que pueden tener fallos o no, a diferencia del artículo donde los encoders primero detectan si ha habido fallo o no, y luego se clasifica. Se refuerza así la aplicabilidad del sistema diseñado en el TFG en este enfoque, reafirmando que las técnicas de ML tienen un gran potencial ante una variedad de fallos complejos.

En la línea de trabajos previos, Straub et al. (2024) proponen el uso de enfoques basados en aprendizaje automático para el diagnóstico mediante OTDR en redes PON, abordando tanto la detección como la clasificación de eventos [6]. El trabajo introduce un sistema de ML para el diagnóstico que, además de detectar eventos en trazas del OTDR, los asigna a ciertas ramas específicas del árbol ODN (*Optical Distribution Network*) utilizando datos como RTT (*Round Trip Time*) o mapas de infraestructura. Esto quiere decir que el sistema utilizado no solo analiza las trazas del OTDR, sino que además añade otros datos de la red a esa información, como el RTT o tiempo que tarda una señal en ir y volver por la fibra (lo que resulta útil para estimar distancias y ubicar fallos de forma precisa); o los mapas de infraestructura, que son planos o registros de la distribución de la red. Este artículo aporta un enfoque más integral del diagnóstico al incorporar datos de infraestructura de red para mejorar la precisión en la detección. Aunque esta integración de datos topológicos no se implementa en el TFG, el artículo aporta una valiosa visión sobre cómo escalonar el sistema hacia entornos reales más complejos; combinando la información de las trazas con conocimientos de red para mejorar el rendimiento global. Así pues, no carece de importancia ya que introduce la idea de combinar ML con previo conocimiento de la red, abriendo así posibilidades de líneas futuras para el desarrollo del trabajo.

En otro trabajo relevante, Usman et al. (2022) plantean la monitorización de fallos en redes ópticas pasivas (PON) a través de la integración de sensores de fibra óptica con algoritmos de aprendizaje automático [7]. Por lo tanto, los autores proponen una alternativa a la detección de fallos usando sensores FBG (*Fiber Bragg Grating*) como fuentes de datos independientes y un clasificador SVM (*Support Vector Machine*) entrenado con señales reflejadas. Al igual que el TFG, ambos supervisan un conjunto de datos para garantizar la calidad del servicio, aunque se diferencian en la obtención de los datos. En este TFG se obtienen mediante un OTDR, y en el artículo mediante los sensores FBG. Además, este TFG se centra en técnicas basadas en redes neuronales, que pueden superar a los clasificadores SVM en la modelación de relaciones no lineales y el manejo de grandes volúmenes de datos. Por tanto, este artículo es útil para dar a conocer que hay otras opciones para obtener las medidas de una fibra, dando un enfoque alternativo a la monitorización independiente del tráfico. Útil, por tanto, para líneas futuras para aumentar la precisión y transparencia al sistema.

3

Entorno de trabajo y herramientas software utilizadas

3.1 Introducción

Este capítulo recoge información sobre las herramientas empleadas a lo largo de la realización de este trabajo.

En primer lugar, se procede a describir los principales elementos que se han utilizado para configurar nuestra red PON, de donde tomaremos las trazas.

A continuación, se indican las principales características del OTDR en cuestión.

Después, se describen las funcionalidades del software para el manejo de las trazas, denominado OTDR Trace Manager.

El capítulo continúa con unas breves menciones a las herramientas Python y Matlab que han sido utilizadas para la realización del trabajo.

Por último, el capítulo concluye con una breve conclusión sobre estas herramientas de las que se hace uso.

3.2 Descripción de las Redes de Acceso Ópticas Pasivas

Las redes de acceso PON son sistemas de distribución de fibra óptica diseñados de manera que conectan una central (OLT) con múltiples usuarios finales (ONUs) a través de

una infraestructura compartida. Esta arquitectura hace honor a su nombre ya que utiliza elementos pasivos como *splitters* ópticos (para dividir la señal).

La más destacada característica de estas redes es que eliminan la necesidad de utilizar componentes activos o alimentación eléctrica, lo que reduce enormemente los costes y mejora la fiabilidad del sistema. En el sentido descendente la información se transmite a los usuarios (ONTs/ONUs), mientras que en el sentido ascendente va hacia la OLT y se utiliza multiplexación por división en el tiempo para evitar interferencias entre las señales. Las redes GPON ofrecen tasas de transmisión asimétricas de hasta 2,5 Gbit/s en sentido descendente (OLT hacia ONU) y 1,25 Gbit/s en sentido ascendente (ONU hacia OLT), empleando difusión en bajada y acceso múltiple mediante TDMA con asignación dinámica de ancho de banda en subida. El plano de longitudes de onda se organiza en tres ventanas ópticas bien definidas: alrededor de 1490 nm para downstream, 1310 nm para upstream, y opcionalmente 1550 nm para servicios de vídeo broadcast, integrándose mediante multiplexación en longitud de onda (WDM). Estas características permiten que con una única fibra y elementos pasivos se pueda llegar a divisiones de hasta 1:64 (e incluso 1:128 en escenarios específicos) y distancias de hasta 20 km, manteniendo un coste de despliegue reducido frente a arquitecturas punto a punto, sin perder flexibilidad para absorber picos de tráfico gracias a la multiplexación estadística [15].

Este tipo de redes se ha adoptado como una solución eficiente para la entrega de servicios de telecomunicaciones de alta velocidad, gracias a su bajo mantenimiento, su gran escalabilidad y su capacidad para adaptarse a diferentes topologías.

En nuestro caso de análisis, vamos a diseñar una configuración de red PON (en concreto en el contexto de una red GPON - Gigabit Passive Optical Network) con pocos kilómetros donde los usuarios están muy cerca entre sí, y de este modo emular eventos que estén muy cercano entre sí. El montaje real realizado se puede observar en la foto de la Figura 1. Todos los componentes PON de esta red han sido adquiridos al fabricante español Telnet Redes Inteligentes (empresa actualmente cerrada).

Figura 1: Montaje final de la red experimental

Así pues, la configuración final consistirá en que el OLT está conectado a un tramo de fibra óptica troncal de 500m que desemboca en un primer *splitter* 1:4; luego una de las 4 ramas va al segundo *splitter* 1:4 tras 100m de fibra; y una de las ramas del segundo *splitter* entra en el tercero 1:8 tras otros 101m de fibra, dejando así 3 ramas libres del primero, otras 3 libres del segundo y las 8 del *splitter* final. El haz de luz será proporcionado por el OTDR durante la toma de las trazas al comienzo del primer tramo de fibra, simulando así la señal proveniente del nodo central o punto de inicio de la red conocido como OLT. Esta configuración se describirá en detalle en el siguiente capítulo de la memoria, donde se podrá ver de forma gráfica en la Figura 11.

3.3 OTDR (*Optical Time Domain Reflectometer*)

El OTDR utilizado es el modelo PROLITE-50, aportado por la empresa PROMAX; un reflectómetro óptico monomodo diseñado para medir atenuación, buscar eventos en la fibra y evaluar su calidad desde un único extremo. Opera en longitudes de onda de 1310 nm y 1550 nm. Tiene un rango dinámico de unos 24 dB y es adecuado para analizar enlaces de hasta 120 km, según condiciones específicas de configuración y estado del enlace óptico. El manual especifica que el OTDR puede utilizar pulsos de 30 ns, 100 ns, 275 ns, 1 µs y 2,5 µs. Sin embargo, se ha podido comprobar que solo están disponibles los dos primeros (30 ns y 100 ns). Parece que es una limitación de fabricación y no un mal funcionamiento del equipo. El manual del mismo se puede consultar en español en [2] y se hablará más en detalle de sus características en el siguiente capítulo. El OTDR tiene la capacidad de almacenar las trazas de los test que se realizan y posteriormente pueden ser pasadas al software proporcionado. Aparte de las limitaciones que impone el ancho de pulso en la detección de eventos (a mayor ancho de pulso mayor separación entre eventos se necesita para poder detectarlos), el OTDR tiene una zona muerta de aproximadamente 2m. Esto significa que incluso aunque el ancho del pulso permitiera distinguir eventos con una separación inferior a 2m, debido a la ceguera temporal del sensor después de detectar un evento, eventos a distancias inferiores a 2 m serán vistos como uno solo.

El resto de parámetros se configurará por defecto inicialmente y, si se cambian a lo largo del trabajo, se indicará en su debido momento.

3.4 OTDR Trace Manager

Es el software que proporciona la empresa para el manejo de las trazas tomadas con el OTDR. Permite abrir un archivo de traza de tipo .sor y visualizarla en una gráfica cuyo eje X es la distancia de la fibra óptica y el eje Y es la potencia en dB del pulso [8].

Con este software podremos exportar las trazas a código ASCII en un archivo .txt para usarlo más tarde. También permite visualizar información relevante de la traza y sus parámetros, así como ver los eventos que ocurren en la fibra que han sido identificados por el OTDR durante la toma de la traza.

3.5 Matlab

Matlab [3] es un entorno de programación de alto nivel ampliamente utilizado en ingeniería que consiste en el tratamiento y visualización de datos numéricos, así como la facilidad que aporta para desarrollar algoritmos complejos de forma estructural y eficiente.

Ya que aporta una gran capacidad para trabajar con grandes volúmenes de datos y automatizar procesos mediante bucles y funciones, se ha utilizado en este proyecto para realizar un programa que utilice la información de las trazas en su conjunto y cree una base de datos con repeticiones de las mismas trazas, pero con niveles de ruido diferente en cada una de ellas para incrementar artificialmente la variabilidad de las muestras.

3.6 Python

Python [1] es un lenguaje de programación de propósito general que recientemente ha ganado mucha popularidad por su sintaxis clara y su gran versatilidad. En el ámbito de la IA y ML, Python destaca sin lugar a dudas porque consta de amplias librerías especializadas como Tensorflow o Keras, que facilitan el desarrollo de modelos neuronales.

Por su compatibilidad con múltiples entornos, su sencillez y sus capacidades gráficas se ha utilizado para llevar a cabo la implementación del programa de IA, que usará la base de datos creada por el programa de Matlab mencionado anteriormente

3.7 Conclusiones

En este primer capítulo de la memoria se ha establecido el marco de trabajo sobre el que se va a desarrollar el resto del estudio, donde se ha llevado a cabo una descripción superficial de las herramientas empleadas.

El uso de todas estas herramientas hace que el proyecto se centre en un entorno real y completamente práctico, por lo que los resultados de este proyecto podrían generalizarse a redes de diferentes tamaños sin perder su validez.

4

Análisis y adquisición de trazas OTDR para la caracterización de fallos en redes PON

4.1 Introducción

En este capítulo se profundiza en el uso del OTDR como herramienta principal para el análisis de la red de acceso GPON desplegada, detallando de forma meticulosa el procedimiento seguido para la configuración del mismo, así como los pasos seguidos para la toma de medidas y la extracción de las trazas ópticas.

El uso de este instrumento busca garantizar la correcta instalación de la red y obtener datos precisos que permitan evaluar su rendimiento y fiabilidad.

4.2 Descripción y detalles del OTDR

El OTDR, como se ha explicado anteriormente, es un instrumento esencial para la caracterización y supervisión de redes de fibra óptica PON. Utiliza un láser monomodo transversal para transmitir un pulso de luz hacia la fibra óptica y analiza la señal reflejada permitiendo así obtener información sobre el estado de la fibra. A partir de este análisis, el OTDR puede detectar eventos como empalmes, pérdidas por atenuación o por conectores defectuosos, etc., así como estimar la distancia a la que ocurren los mismos. El OTDR devuelve como datos una traza. En una traza típica, los eventos siempre aparecen como caídas y algunos pueden venir acompañados de picos previos a las caídas.

El OTDR es un instrumento que resulta imprescindible para obtener las medidas, dado que permite evaluar toda la red actuando desde un único extremo sin necesidad de acceder físicamente a toda la fibra.

La fibra y el equipo utilizados en el proyecto están optimizados para trabajar en condiciones donde los efectos de dispersión modal o cromática no afecten significativamente a las mediciones, ya que se tratan de enlaces de corta distancia y condiciones de operación compatibles con el OTDR, el cual se puede apreciar en la foto de la Figura 2 que aparece a continuación.

Figura 2: OTDR utilizado para el TFG

Cabe resaltar que las malas conexiones generan pérdidas adicionales y reflexiones. Esto es muy importante, pues nuestra red consta de una cantidad de adaptadores y conexiones que no es irrelevante, por lo que será crucial asegurarnos de que todos ellos estén correctamente conectados en relación a la validez de las medidas tomadas.

En cuanto a la configuración del aparato, se ha ido ajustando a lo necesitado según las trazas tomadas se iban analizando, pues en algunos casos ocurrió que fue necesario cambiar la configuración debido a diversos factores, como se explicará más tarde.

4.3 Procedimiento de captura de trazas OTDR

Durante el desarrollo del proyecto, se empleó el OTDR junto con el *software* proporcionado para llevar a cabo la toma de trazas y su posterior análisis. El experimento consistirá en montar una configuración de red GPON con los elementos disponibles, la cual sea parecida a la utilizada en el artículo base [4] y sobre ella, tomar las trazas correspondientes obteniendo así las medidas experimentales. Para ello, se disponen de suficientes tramos de fibra óptica de diferentes longitudes (desde *pigtails* de 1m de longitud hasta fibras de 500m), bobinas con longitudes de 5km y 10km que nos servirán para el tramo inicial de la red, un *splitter* óptico de 8 salidas y otros 2 *splitters* de 4 salidas; además de suficientes conectores y adaptadores ópticos para permitir una total y correcta conectividad sobre la red GPON. Por supuesto, se dispone del OTDR para realizar las medidas, el software que permitirá analizarlas más adelante, y lápices limpiadores ópticos que permitirán mantener en buen estado continuamente la red óptica montada. Además, se utilizarán atenuadores que se colocarán al final de las ramas y servirán para simular que ha ocurrido un fallo en la misma. Concretamente, se dispone de atenuadores de 1dB, 3dB, 5dB y 10dB. El procedimiento se desarrolló en diversas fases progresivas que se describen en los siguientes apartados.

4.3.1 Instalación y familiarización

Inicialmente, se realizaron tareas básicas de configuración, como la instalación del software del OTDR. Tras ello, se procedió a la lectura del manual del OTDR y a la elaboración de una guía práctica muy completa y accesible en [9] para facilitar el uso del equipo y del software asociado.

4.3.2 Captura preliminar de trazas OTDR

La primera serie de mediciones se realizó sobre una red GPON muy sencilla compuesta por tres tramos de fibra óptica de 5 km, 10 km y 5 km, sumando así un total de 20 km. Estas mediciones tenían como objetivo estudiar la influencia de los parámetros modificables en el OTDR en cuanto a la resolución espacial se refiere; así como ver el número de muestras generadas por el reflectómetro. Para ello, se tomaron cinco trazas con

diferentes combinaciones de rango de medida y ancho de pulso (por ejemplo, 40 km con 1000 ns; 80 km con 1000 ns; 40 km con 300 ns, etc.). Algunos ejemplos se pueden ver en la Figura 3, Figura 4 y Figura 5.

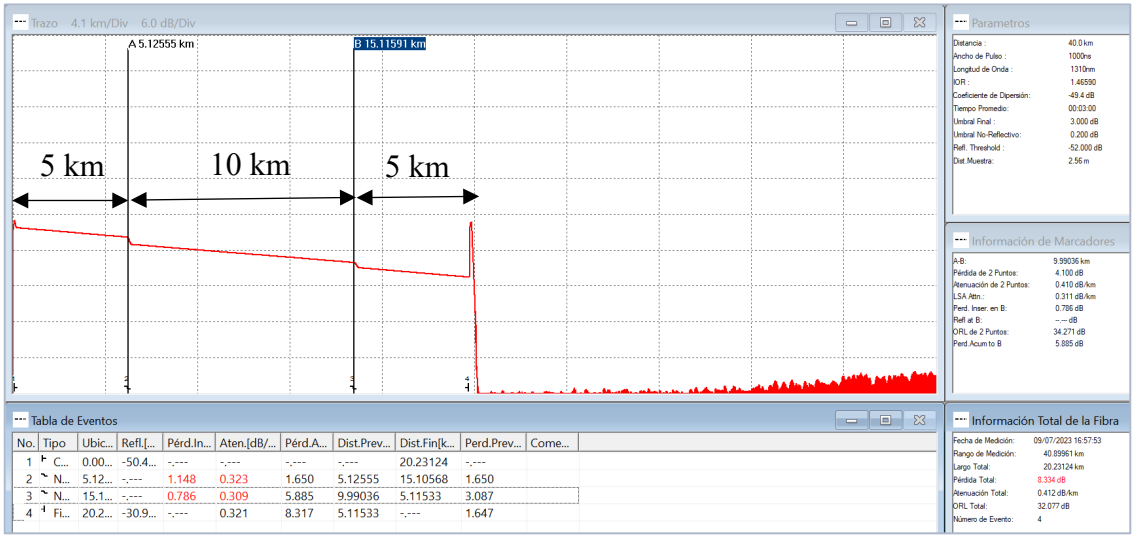

Figura 3: Traza del OTDR con 40 km de rango y 1000 ns de ancho de pulso

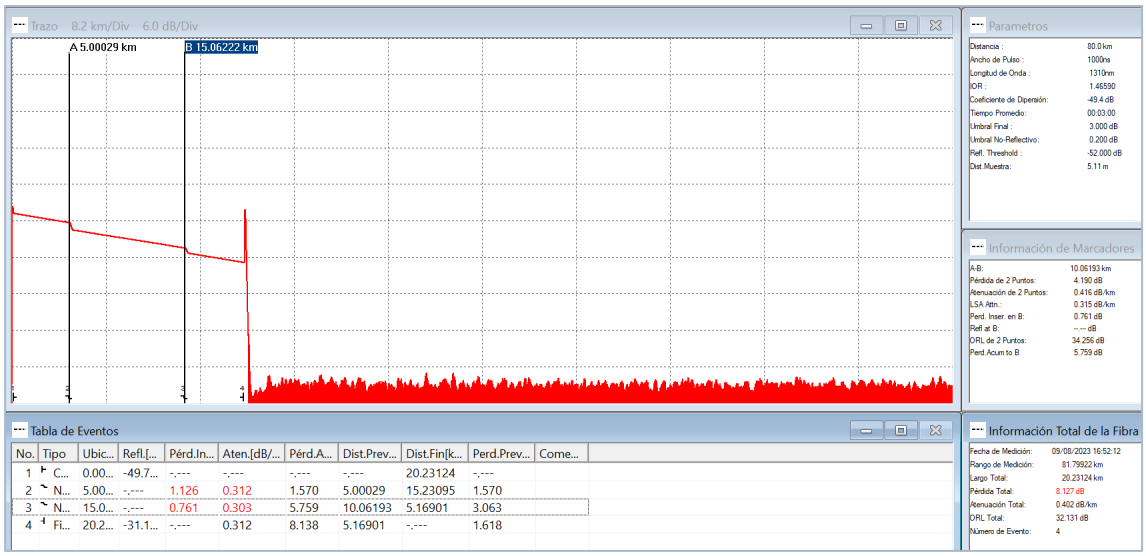

Figura 4: Traza del OTDR con 80 km de rango y 1000 ns de ancho de pulso

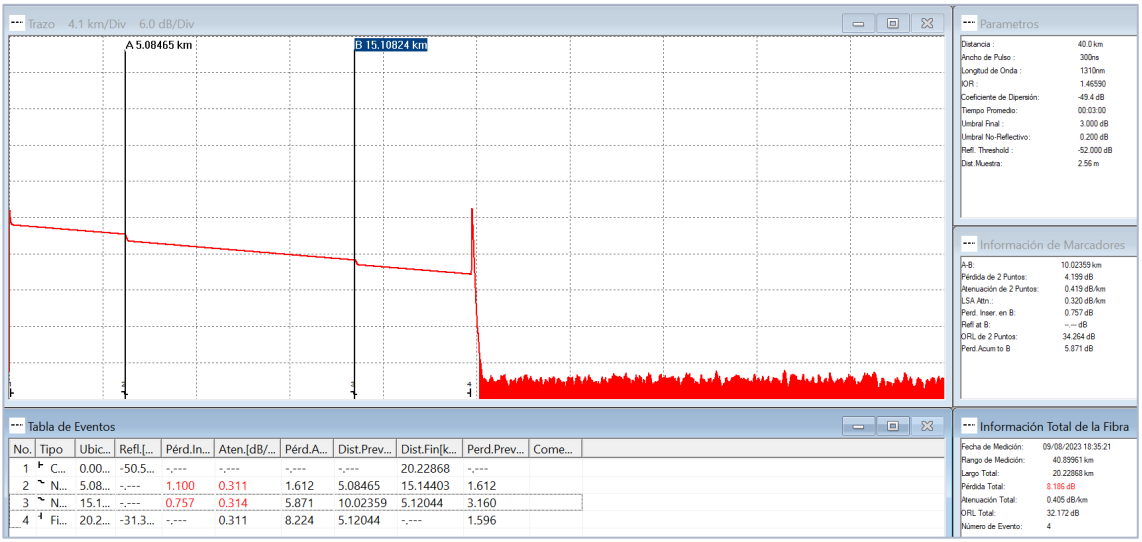

Figura 5: Traza del OTDR con 40 km de rango y 300 ns de ancho de pulso

Para determinar cuántas muestras se obtenían en cada configuración, las trazas se exportaron en formato ASCII (.txt), y se contaron las líneas de cada archivo, siendo cada línea una muestra. En total se generaron 34 trazas para esta caracterización sistemática.

Con la adquisición de estas trazas se busca responder a dos preguntas. Por un lado, estamos interesados en saber cuántas muestras tienen las trazas con respecto a la longitud de medida. Un cálculo sencillo nos dirá la distancia espacial teórica entre muestras, y de alguna forma anticipará la capacidad de resolución de eventos.

Por otro lado, para usar los datos recopilados con la IA, será necesario conocer a ciencia cierta el número exacto de muestras que obtenemos, dado que la IA es inflexible en cuanto al formato de las entradas.

Los resultados del número de muestras para cada configuración de distancia y tamaño de pulso se puede ver en la Tabla 1. En dicha tabla se puede ver que el número de muestras es diferente en cada caso, además de la distancia entre muestras, lo cual hay que considerar. Esto se tendrá muy en cuenta a la hora de realizar el conjunto de medidas agrupadas en Excel antes de realizar la base de datos.

Muestras	300m	1.3km	2.5km	5km	10km	20km
5ns	3008	12784	12288	10080	-	-
10ns	3008	12784	12288	10080	-	-
30ns	3008	12784	24064	10080	10000	8000
100ns	3008	12784	5120	10080	20000	20000
300ns	-	-	1024	2016	4000	8000
1us	-	-	-	10080	4000	8000
2.5us	-	-	-	-	4000	8000
Distancia extra km	0.32711	1.39055	2.61751	5.15316	10.22299	20.44853

Tabla 1: Tabla del número de muestras para cada distancia de fibra en función del ancho del pulso

4.3.3 Captura de trazas OTDR con inserción de divisores ópticos

Posteriormente, se montaron diversas configuraciones de red, pero ahora incorporando varios *splitters* ópticos (1:4, 1:8) y se insertaron atenuadores en distintas ramas que servirían para simular el fallo en la rama donde estén puestos. La configuración de prueba que seguimos se ve reflejada en el esquema mostrado en la Figura 6. Primero se parte de una bobina de 5 km; luego va un *splitter* 1:4 del que salen 3 ramas de 1m y una de 5m; la cual va a otro *splitter* 1:4 con ramas de 1m, 3m, 1m y 10m; la cual va a el *splitter* 1:8 con las 8 ramas finales de diferentes longitudes todas ellas. Dado que se tiene poco más de 5 km de fibra, se decidió tomar la medida con un rango de 10 km y un ancho de pulso para comenzar de 30 ns, que es el más pequeño posible cuando el rango es de 10 km para obtener una mejor resolución.

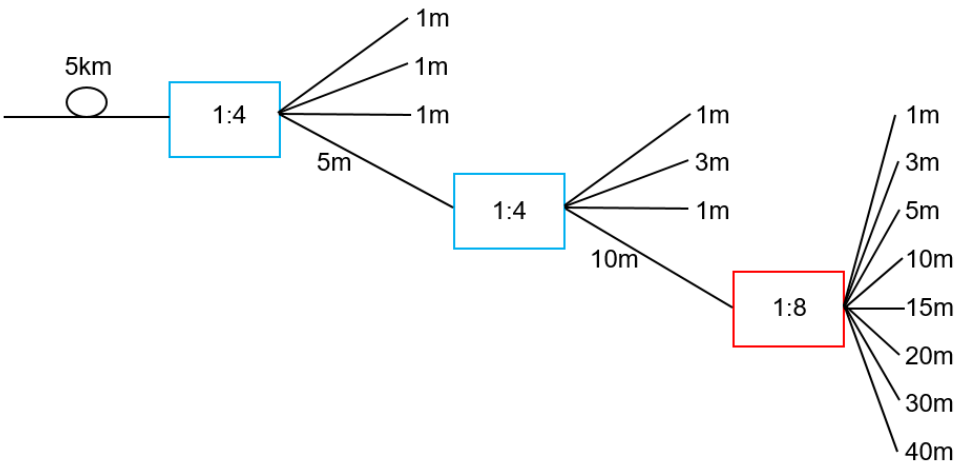

Figura 6: Esquema de red inicial analizado

Las medidas que se tomaron fueron varias: una sin atenuadores (ideal), y luego una con un atenuador en cada una de las ramas finales del tercer *splitter* y también con atenuadores entre los 3 *splitters*. Todas ellas repetidas para los 4 atenuadores (con

atenuaciones de 1dB, 3dB, 5dB y 10dB) con los que se contaba. Esto permitiría ver diferencias de las trazas con atenuadores en diferentes ramas, pues aquella atenuada tendría el pico de reflexión más bajo que las demás. Sin embargo, a la hora de tomar las trazas todas parecían prácticamente iguales a simple vista, aunque tenían diferencias prácticamente imperceptibles en los últimos metros de fibra. Un ejemplo de cómo se ven los eventos concentrados al final de la fibra se puede ver en la Figura 7.

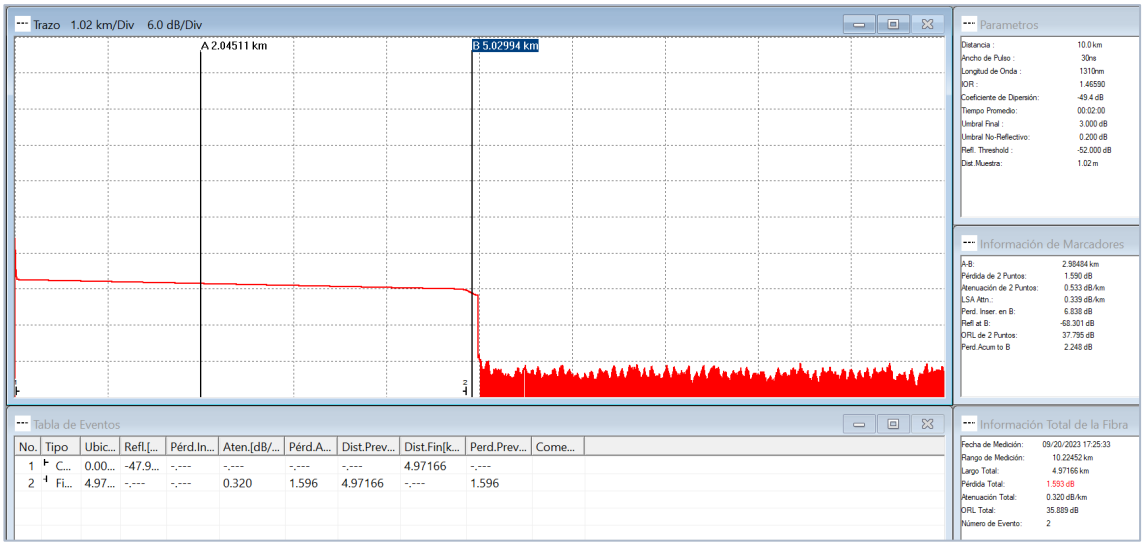

Figura 7: Con atenuador de 10 dB en la rama de 40 m

Aparte del hecho de que, en escala, los últimos 55m, que es la parte de la red en la que estamos interesados, es mucho más pequeña que los 5km de lanzamiento que hemos introducido, la combinación de la propia distancia entre muestras a partir de la traza (1 m entre muestras), la zona muerta del equipo y la resolución que el propio tamaño del pulso ofrece, hace que los eventos en los que estamos interesados no se diferencien, al menos a simple vista. Así pues, se realizaron 2 configuraciones de prueba más sencillas que estaban más enfocadas a poder distinguir los eventos a simple vista de tal manera que pudiéramos conocer el comportamiento de los *splitters*, tal y como se muestra en la Figura 8 y la Figura 9. En concreto, en estos esquemas se aplicaban distancias más grandes entre los *splitters* seguidos, en concreto 100 metros.

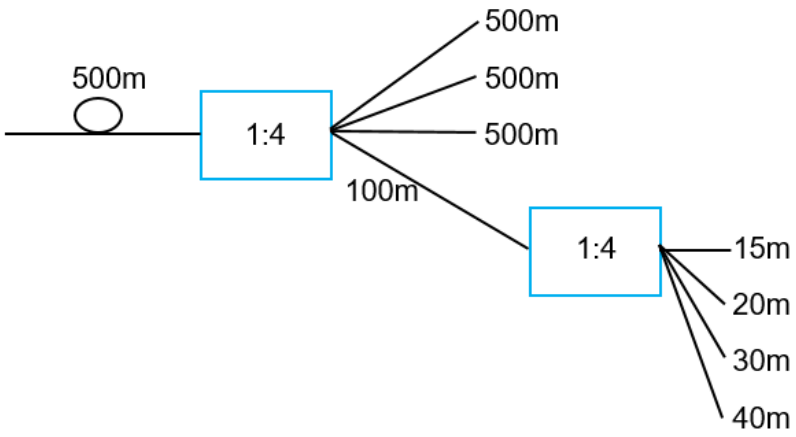

Figura 8: Esquema de red con más distancia de fibra entre splitters (prueba 1)

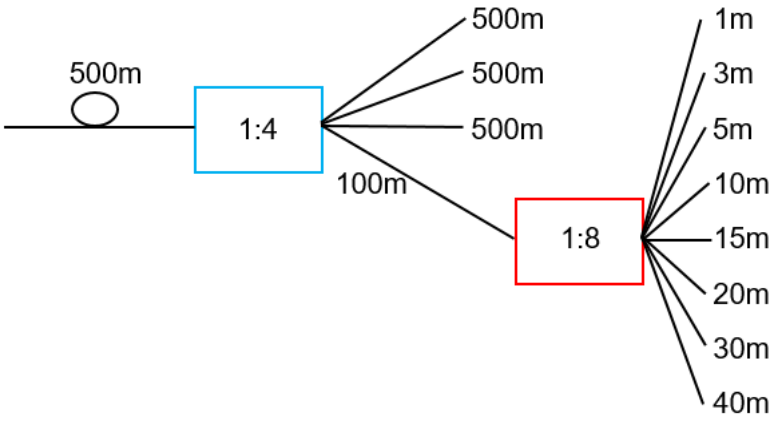

Figura 9: Esquema de red con más distancia entre splitters (prueba 2)

El resultado de tomar medidas en estas 2 nuevas pruebas fue que, como se sospechaba, las distancias entre *splitters* no eran lo suficientemente grandes como para que el OTDR pudiera separar los eventos. Ahora que la distancia es de 100 m, se pueden diferenciar con claridad cómo se puede observar en la Figura 10.

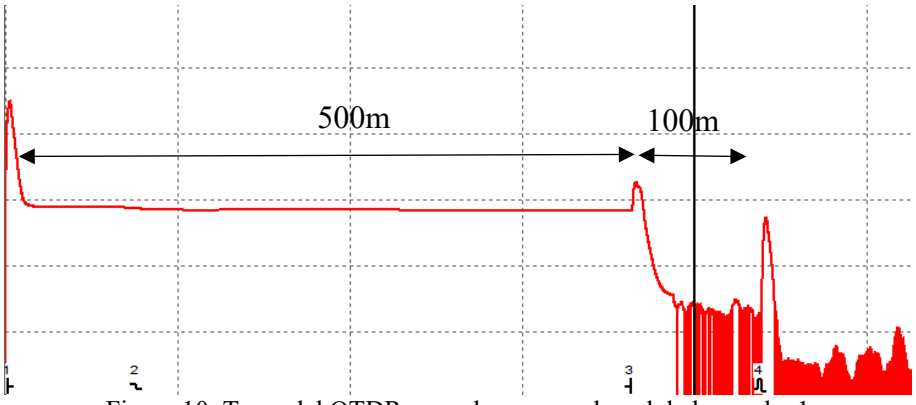

Figura 10: Traza del OTDR para el esquema de red de la prueba 1

Sin embargo, también fue necesario tener en cuenta el rango dinámico del OTDR

utilizado. Con tantos conectores y *splitters*, las pérdidas por inserción eran demasiado altas, de tal manera que detrás de los *splitters* no se puede ver cómo se va incrementando la atenuación del pulso. Esto también implicaba que no se podía ver el final de fibra, cuestión importante para poder determinar si una rama está correcta o no. En el artículo [4], se vio que al final de las fibras se incluían unos reflectores, es más, se buscaron estándares de testeo de redes PON y se encontró que el testeo y certificación de redes de este tipo se tiene que hacer con reflectores antes de su puesta en funcionamiento. De esta forma fue posible que el OTDR detectara un pico de reflexión justo al final de las fibras.

Es importante señalar también la longitud de las fibras al final del primer *splitter*, que se puede ver es de 500m. Esta longitud se introdujo para mejorar la respuesta del OTDR y que fuera más fácil observar los eventos del segundo *splitter*. El razonamiento es el siguiente: el OTDR es incapaz de analizar ninguna señal que cae por debajo de un cierto valor de potencia. En el caso de nuestro OTDR la sensibilidad es de -50 dBm. Cuando se introduce un *splitter* se tienen dos fuentes de pérdidas, por un lado, los dB que se pierden en el sentido de subida y los dBs que se pierden en el sentido de bajada. Así un *splitter* de 4 salidas introduce 6 dBs de pérdida en cada sentido y uno de 8,9dBs. Dos *splitters* de 4 salidas introducen unas pérdidas de 24 dB ida y vuelta, que es prácticamente el rango dinámico de nuestro OTDR. Los enlaces de 500m suponen una solución a este problema, dado que la potencia reflejada por estos enlaces se suma a la vuelta de los *splitters*, con lo que la potencia total recibida por el OTDR es suficiente para ver el final de fibra.

4.3.4 Etapa final de configuración y captura de trazas OTDR

Tras la realización de múltiples pruebas y configuraciones, se diseñó y configuró un esquema final. Este esquema supone un equilibrio entre poder detectar eventos de final de fibra gracias a los reflectores y paso por *splitters* gracias a las pérdidas que introducen, frente a no poder ver cómo va aumentando progresivamente la atenuación en los tramos de fibra por superar el rango dinámico del OTDR. La configuración se puede ver en la Figura 11. Este modelo mantiene una distancia entre *splitters* lo suficientemente grande como para diferenciarlos tal y como vimos anteriormente en la Figura 10, pero utilizando los 3 *splitter* como se quería hacer al principio.

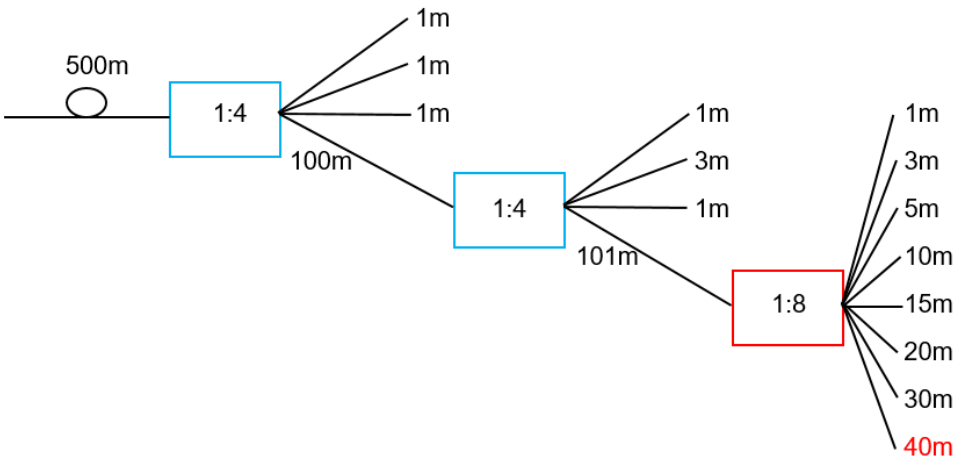

Figura 11: Configuración final de red GPON para las mediciones finales

Con respecto a los eventos relacionados con el final de cada una de las ramas, en los dos primeros *splitters* no hay prácticamente diferencia de longitud entre ellas, con lo que se detectarían como un único evento en cada *splitter*. Para el tercer *splitter* podría haber problemas para distinguir los eventos correspondientes a las ramas de 1m, 3m y 5m, puesto que la diferencia de longitud es de exactamente 2m que corresponde a la longitud de la zona muerta del OTDR. En la Figura 12 se ve un ejemplo de lo que se ha comentado. En el primer *splitter* se observan dos picos, que pueden deberse a los diferentes conectores y/o a las ramas. En el segundo *splitter* solo se observa un único pico, que podría ser lo que se debería ver por defecto, siendo lo que se ve en el *splitter* 1 (dos picos) lo excepcional. En cualquier caso, hay problemas con los conectores que se resolverán más adelante. Finalmente, en el *splitter* tres, como tiene más ramas, deberíamos diferenciar varios picos y no solo uno, debido a tener más distancia en ellas. Esta configuración es la mejor que hemos podido desarrollar, pues es un buen equilibrio entre la configuración dada por el artículo base [4], a la que se asemeja bastante, y, por otro lado, se adapta mejor a las posibilidades del OTDR utilizado en este TFG, que tiene características algo inferiores al utilizado en el artículo (por ejemplo, no soporta el ancho de pulso de 1ns, de tal manera que está en inferioridad a la hora de detectar eventos demasiado juntos).

Figura 12: Configuración final con 5, 10 y 30 ns de pulso

Para comprobar la idoneidad de esta configuración para la creación de una base de datos de trazas, inicialmente se tomaron las medidas sin atenuadores y con diferentes anchos de pulso. Aparte de comprobar la efectividad de esta configuración también se pudo comprobar el efecto de los *splitters* en las trazas, como la de la Figura 12, tal y como se ha explicado anteriormente. En cuanto a la posibilidad de distinguir eventos cercanos, depende mucho del ancho del pulso utilizado y del rango dinámico disponible; si se quisiera distinguir todas las ramas de los *splitters* de forma mucho más clara, aunque las distancias sean pequeñas, se debería tener un ancho de pulso mucho más pequeño, pero el dispositivo no lo da, por lo que esta configuración es, de nuevo, la más adecuada para los recursos utilizados. En concreto, la configuración final utiliza como parámetros una distancia de medida de 1.3 km, anchos de pulso de 30 o 100 ns, tiempo de medida de 3 minutos, longitud de onda de 1550 nm, IOR (Índice de Refracción) de 1.4856 y coeficiente de dispersión de -51.8 dB.

Más adelante, nos dimos cuenta de que no se veían bien todos los picos de reflexión por problemas de conectores, pero la idea de la configuración final de la red será esa, que ajustando mejor las conexiones obtendremos la traza correcta.

Resultaba importante, en esta fase de la experimentación, asegurarnos que la introducción de atenuadores modificaba las tramas a simple vista, al menos para el caso de mayor atenuación. La razón para esto es doble. Por un lado, nos necesitamos asegurar que

los picos que vemos en la traza original sin fallos de red corresponden a cada una de las ramas. Por otro lado, nos queremos asegurar de que la introducción de los atenuadores modifica la traza. No nos importa que esta modificación no sea apreciable para valores de atenuación más pequeños, puesto que eso es justamente lo que tiene que resolver la IA, sino simplemente que al menos podemos confirmar que para el caso de mayor atenuación se aprecia la modificación. Por lo tanto, se colocó un atenuador de 10 dB en la rama final de 40 m y se tomaron medidas, como la mostrada a en la Figura 13. Cabe mencionar que, aunque en la figura ponga 10 ns, realmente el ancho utilizado ahí es de 30 ns, pues se tomó antes de ver que solo se tenían 2 anchos de pulso. Efectivamente, el pico de reflexión correspondiente desaparecía como era de esperar, pues el haz en esa rama no se refleja y el OTDR detectaría menos potencia en esa rama, tanta menos potencia como atenuación se coloque. Esto se realizó en cada una de las ramas para asegurarnos que todas ellas funcionaban de manera correcta.

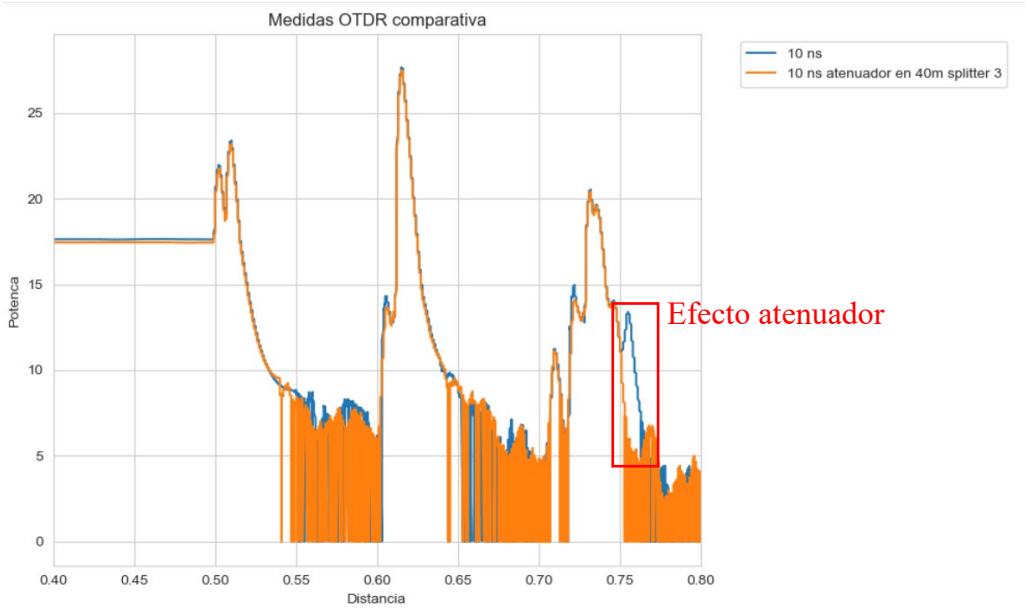

Figura 13: Configuración final con 10 ns de pulso, con y sin el atenuador en el tercer splitter

4.3.5 Adquisición final de trazas OTDR para la generación del dataset

Teniendo ya claro el funcionamiento y efectos de los *splitters*, atenuadores y reflectores, solo queda tomar las medidas finales con las que se elaborará posteriormente una base de datos; no sin antes revisar todas las conexiones como se dijo anteriormente.

Tras ello, y con la red final correctamente conexionada, se obtuvieron un total de 90 trazas, repetidas una segunda ronda, haciendo así 180 medidas experimentales. Esas 90

trazas vienen de colocar el atenuador en los diferentes sitios donde puede ocurrir el fallo: al final de las 8 ramas del tercer *splitter* y antes de cada *splitter*, lo que hacen 11 trazas por atenuador. Como se dijo en capítulos anteriores, se dispone de 4 atenuadores diferentes (1dB, 3dB, 5dB y 10dB), por lo que ya son 11x4=44 trazas + 1 traza ideal sin atenuador. Esas 45 trazas se toman con 30ns y con 100ns, haciendo así las 90 trazas mencionadas. Todas ellas son trazas diferentes, variando la localización del fallo emulado por un atenuador de diferente grado de atenuación y en diferentes ramas. Las trazas tienen en común el rango de medida, que es de 1.3 km. Además, realizar una segunda tanda de medidas ayuda a mejorar la variabilidad del modelo, pues existen cambios, por pequeños que sean, al tomar exactamente la misma traza una segunda vez.

Para visualizarlo, la gráfica de la Figura 14 muestra la traza ideal, sin atenuadores en ninguna rama, que se tomará como referencia para identificar los fallos en las ramas.

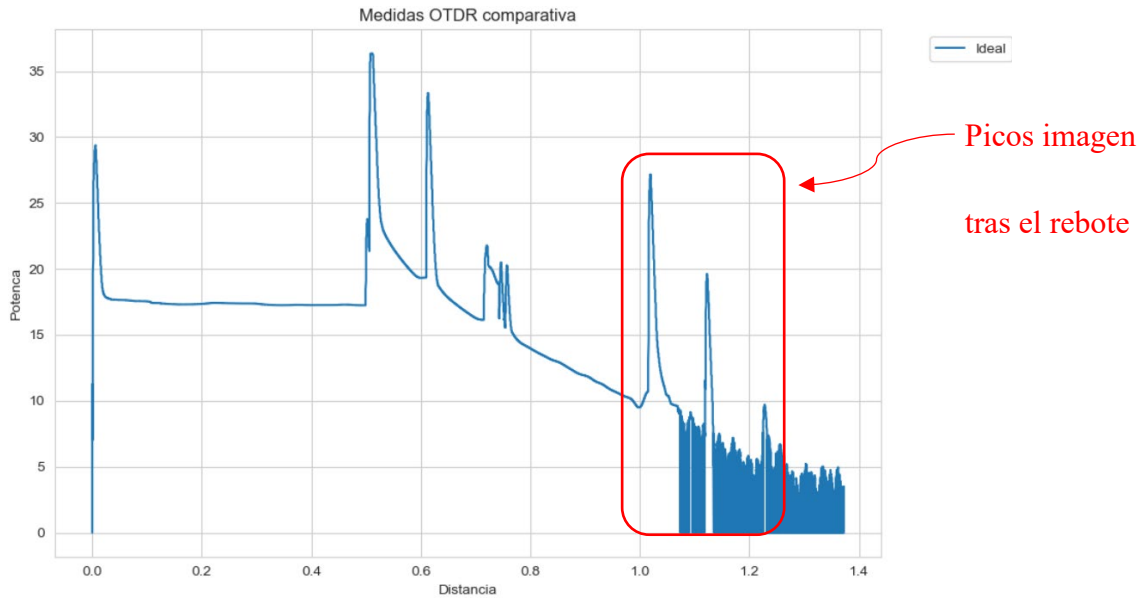

Figura 14: Traza ideal (sin atenuadores) con 30 ns

Es interesante observar que aparecen picos imagen en aproximadamente 1000m, 1100, y 1200m, tal y como se indica en la Figura 14. Estos picos no deberían aparecer en la traza, dado que por las dimensiones de nuestra red experimental el pico más lejano estaría a unos 740m, al final de la última rama del tercer *splitter*. En este caso, creemos que esto se debe a que el OTDR es capaz de detectar la misma traza una segunda vez después de verse reflejada en los conectores del OTDR. Sin embargo, este hecho carece de importancia, pues basta con cortar la traza y quedarnos con la parte que verdaderamente nos interesa, que será desde el comienzo del primer *splitter* (desde unos 495m) hasta el

final del tercero si queremos abarcar toda la red (hasta unos 760m). Esto, además, será necesario más tarde, pues para la IA es mucho más complicado manipular trazas con muchas muestras. Reducir el número de muestras al eliminar aquellas innecesarias ayudará en la eficiencia del modelo.

Finalmente, también cabe destacar que en las trazas tomadas solo se permite un fallo en la red. Por lo tanto, el entrenamiento que se haga posteriormente a la IA será para detectar red sin fallo o red con un fallo en una de las ramas o de los *splitters*. Para detectar casos con más de un fallo, sería necesario realizar nuevas capturas de trazas.

4.4 Conclusiones

En este capítulo se recoge una mayor comprensión del funcionamiento del OTDR y su comportamiento en diferentes escenarios de red, señalando la importancia de una configuración inicial necesaria para la toma de medidas experimentales.

Además, se ha detallado todo el procedimiento de obtención de medidas y de modificación de la red GPON de prueba utilizadas, lo cual nos ha permitido conformar una gran cantidad de muestras que utilizaremos en nuestra base de datos en capítulos posteriores.

5

Elaboración del dataset a partir de trazas OTDR y trazas sintéticas

5.1 Introducción

En este capítulo se describe el proceso completo de elaboración del dataset, con especial atención a la combinación de trazas OTDR obtenidas experimentalmente y señales sintéticas generadas mediante un programa en MATLAB. Se comienza hablando de cómo se agruparon los datos de las trazas en un Excel [10] siendo todas ellas exportadas antes a un archivo formato ASCII (.txt) como etapa preliminar.

A continuación, se detalla el diseño y programación del programa de Matlab que recoge el conjunto de datos y crea la base de datos final, en formato .txt y añadiendo repeticiones con ruido para mayor versatilidad en el modelo de IA que se aplicará después.

5.2 Etapa preliminar de preparación de datos

En este apartado se describe, antes de ejecutar el programa de Matlab, la preparación y organización de los datos extraídos de las trazas del OTDR.

En primer lugar, se pasaron las trazas previamente guardadas en el OTDR al ordenador y, mediante el software proporcionado, las trazas se exportaron una a una a formato .txt. Cada archivo contiene la información de las trazas y los eventos que ocurren en ellas en columnas separadas. Las muestras aparecen como la medida de la potencia en números decimales y la distancia a la que se encuentra cada muestra.

Luego, como es deseable que cada traza sea una fila y no una columna (más adecuado para la IA) al momento de agrupar todas las trazas en un Excel, se trasponen, dejando así las trazas en filas y cada columna es una muestra de las trazas. Como se

mencionó con anterioridad, el número de muestras es muy grande, por lo que se hizo necesario cortar la traza y quedarnos con la zona de interés. De esta forma, el archivo Excel final con las trazas experimentales [11] quedó conformado por una tabla de 180 filas (una por cada traza) y 2897 columnas correspondientes a las muestras de cada traza, además de 11 columnas adicionales de codificación binaria que se agregaron al final para identificar los distintos puntos donde aparece un fallo. Eso último se hace para poder identificar los fallos en el enlace de fibra correspondiente. Una codificación compuesta por 11 ceros representa una traza ideal, sin fallos. Si hay un solo '1' en la codificación (es decir, 10 ceros y un '1'), indica la presencia de un fallo en una ubicación específica. En total, se pueden representar hasta 11 fallos distintos: 8 correspondientes a las ramas finales del tercer divisor (*splitter*) y 3 más que indican posibles fallos antes de cada uno de los tres divisores de la red GPON. Por lo tanto, en este TFG tan solo se considera la detección de un único fallo en una rama o enlace de fibra óptica, no se considera la detección de fallos en más de una rama.

Finalmente, si multiplicamos las filas por las columnas, como curiosidad, se trata de una tabla con 523440 datos. Las correspondencias entre codificación y fallo en nuestra arquitectura de red son las siguientes:

- Codificación 00000000000: Traza ideal
- Codificación 00000000001: Fallo en la rama de 1m (*splitter* 3)
- Codificación 00000000010: Fallo en la rama de 3m (*splitter* 3)
- Codificación 00000000100: Fallo en la rama de 5m (*splitter* 3)
- Codificación 00000001000: Fallo en la rama de 10m (*splitter* 3)
- Codificación 00000010000: Fallo en la rama de 15m (*splitter* 3)
- Codificación 00000100000: Fallo en la rama de 20m (*splitter* 3)
- Codificación 00001000000: Fallo en la rama de 30m (*splitter* 3)
- Codificación 00010000000: Fallo en la rama de 40m (*splitter* 3)
- Codificación 00100000000: Fallo antes del *splitter* 1
- Codificación 01000000000: Fallo antes del *splitter* 2
- Codificación 10000000000: Fallo antes del *splitter* 3

5.3 Diseño del programa en Matlab para la generación de trazas sintéticas

Para complementar la base de datos en Excel obtenida por la agrupación de las trazas experimentales, se ha desarrollado un programa de Matlab [12] cuyo objetivo principal es aumentar la diversidad y variabilidad del conjunto de datos, cosa que se

consigue mediante la simulación de nuevas trazas con diferentes niveles de ruido. El programa, estructurado en diversos bloques diferenciados, realiza los siguientes pasos.

Primero, es necesario hacer una carga y separación de los datos. El primer bloque lee el archivo Excel que contiene las trazas y separa por un lado las muestras (datos) y por otro las codificaciones (etiquetas). Esto se hace con las siguientes líneas de código, las cuales están debidamente documentadas y comentadas en el archivo referenciado:

```
data_bd = readmatrix('BASE_DE_DATOS_FINAL.xlsx');
longitud_codificacion=11;
data=data_bd(:,1:end-longitud_codificacion);
codificacion=data_bd(:,end-longitud_codificacion+1:end);
[num_trazas, num_muestras] = size(data);
```

Además, se definen los diferentes niveles de SNR con los que generaremos el ruido que, en este caso, serán de 5dB a 30dB en pasos de 1dB. También, dado que las trazas están expresadas en dB, se convierten a escala lineal para poder simular de forma precisa el efecto del ruido blanco gaussiano que se añadirá. Este proceso se observa en las siguientes líneas de código:

```
SNR_range = 5:1:30;
signal_power_nat=10^(signal_power/10);
```

Para cada nivel de SNR, se repite cada traza 100 veces, generando ruido distinto en cada repetición. Luego, para cada traza y cada nivel de SNR definido, el programa calcula la potencia del ruido y genera una señal aleatoria con esa potencia, siguiendo una distribución gaussiana. Este ruido se suma a la traza original en escala lineal y luego se convierte nuevamente a dB. De este modo, se obtienen versiones realistas de lo que serían las trazas afectadas por ruido. Todo esto se realiza con los bucles que se presentan en el siguiente extracto de código:

```
for i = 1:num_trazas
    traza_original = 10.^(data(i, :)/10);
    traza_original=repmat(traza_original,[num_repeticiones 1]);
    fprintf(file_id, [repmat('%6.2f ', 1,
    num_muestras+longitud_codificacion) '\r\n'],
    data(i,:),codificacion(i,:));
    for j = 1:num_SNR
        SNR_dB = SNR_range(j);
        SNR_linear = 10^(SNR_dB / 10);
        noise_power = signal_power_nat / SNR_linear;
        noise = noise_power * randn(num_repeticiones, num_muestras);
        traza_noisy = 10*log10(abs(traza_original + noise));
        for k = 1:num_repeticiones
```

```
                fprintf(file_id, [repmat('%6.2f ', 1,
                num_muestras+longitud_codificacion) '\r\n'], traza_noisy(k,
                :),codificacion(i,:));
            end
            fila_actual=fila_actual+1
        end
    end
```

Por último, todas las trazas, junto con las originales sin ruido, se guardan en un archivo de texto que constituirá la base de datos final. Cada fila contiene la secuencia de muestras y su correspondiente codificación binaria. Este archivo resultante es el que se utilizará como entrada para los programas de IA implementados en Python.

5.4 Configuración final del Dataset

Con el conjunto de datos, esto es trazas finales experimentales recogidas con el OTDR, y ya tratadas y organizadas en Excel, se procede a la creación de la base de datos final mediante un programa de Matlab [12] que automatiza el proceso usando bucles.

El programa cargará los datos del Excel y los plasmará en un archivo de salida .txt, pero en ese archivo no solo aparecerán los datos recogidos. Además, el programa añadirá, para cada traza, 26 trazas más que tendrán ruido blanco gaussiano añadido. El ruido se verá comprendido entre 5 y 30 dB de relación señal-ruido (de ahí los 26 niveles de SNR), descrito anteriormente. Además, tal y como se ha comentado en el apartado anterior, para cada nivel de SNR se realizan 100 repeticiones, lo que aporta una mayor variabilidad al conjunto, siendo así útil para el entrenamiento o validación de la IA al tener un mayor número de datos.

Por lo tanto, nuestra base de datos final (dataset) estará compuesta por 468,000 filas (trazas), resultantes de multiplicar las 180 trazas originales por 26 niveles de ruido y 100 desplazamientos por ventana, a las que se suman las 180 trazas originales. Esto hace un dataset más de 10 veces del tamaño del dataset utilizado en el artículo base. Manteniendo el mismo número de columnas que el archivo Excel, el dataset total contiene aproximadamente 1,361,467,440 datos. Con esta base de datos y diseñada, se finaliza esta etapa y se da paso a la creación del programa de la IA. Para visualizar esta adición de ruido con el programa, se puede observar en la Figura 15 un ejemplo de una traza cualquiera

superpuesta con otras dos que son la misma traza, solo que con diferente nivel de ruido.

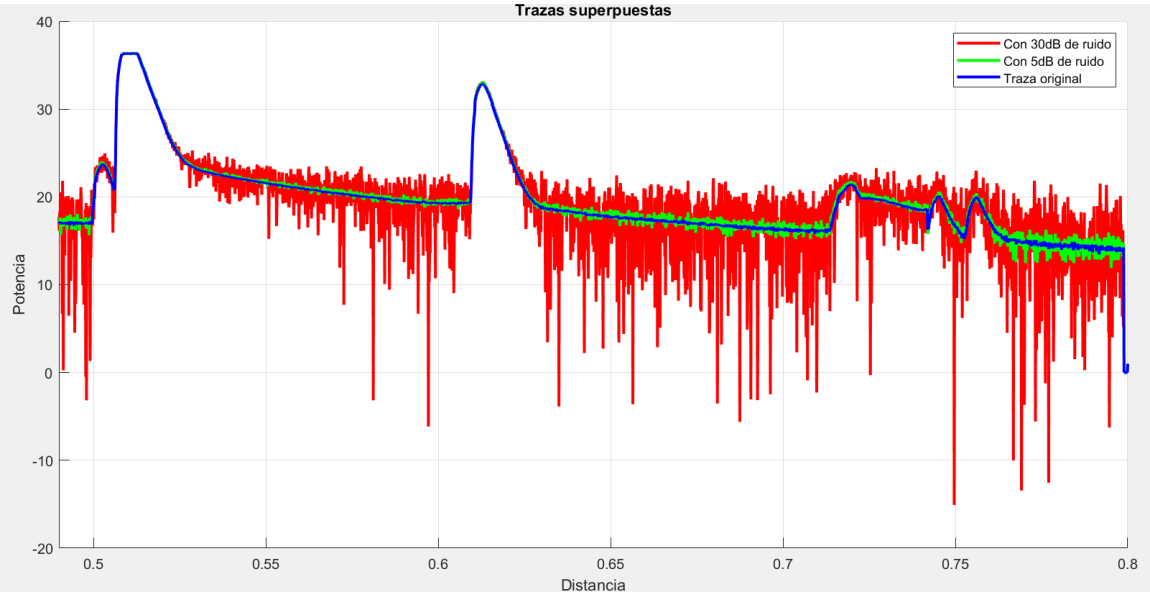

Figura 15: Comparativa de la misma traza superpuesta con ruido

5.5 Conclusiones

En este capítulo se ha documentado en detalle el proceso completo de construcción del dataset de medidas, elemento clave para el desarrollo de los modelos de IA. Desde la agrupación de datos en un Excel hasta la generación automatizada de la base de datos final, todos los pasos seguidos en este apartado se han diseñado para garantizar un conjunto de datos estructurado, ordenado y versátil.

La base de datos, además de recoger fielmente los datos obtenidos, añade variabilidad al conjunto mediante adición de ruido blanco gaussiano, lo que ayudará a mejorar la capacidad del modelo de IA a la hora de detectar e identificar los fallos, al tener mayor número de muestras disponibles. Para realizar esta generación de muestras sintéticas se ha diseñado y programado un programa en Matlab, cuya descripción se ha llevado a cabo a lo largo de este capítulo.

6

Arquitectura y diseño de un modelo de clasificación basado en redes neuronales

6.1 Introducción

En este capítulo se detalla el diseño e implementación de un sistema de inteligencia artificial en el proyecto, generando así un sistema para la detección automática de los fallos en nuestra configuración GPON. El objetivo principal es explicar cómo se estructura la red neuronal y cómo esta se ha implementado en un programa funcional capaz de clasificar y detectar fallos en nuestra configuración de red GPON a partir de trazas del OTDR.

Así pues, primero se describe la arquitectura de red neuronal utilizada, y a continuación la creación del programa que lo implementará, así como los resultados obtenidos tras las ejecuciones del mismo.

6.2 Arquitectura del modelo basado en redes neuronales

El modelo diseñado y que se encuentra accesible en [13] es utilizado para la identificación y detección de los fallos y se basa en una red neuronal construida con la biblioteca Keras. Es una arquitectura efectiva y simple, que se organiza de forma secuencial y diseñada específicamente para clasificación multiclase, pues lo que interesa es saber cuál de las ramas es donde hubo fallo; dicho de otra manera, cuál es la codificación que corresponde al fallo.

Como se muestra en la Figura 16, dado que se parte de 11 posibles clases de fallo, la capa de salida del modelo estará compuesta por 11 neuronas con función de activación

softmax. La capa de entrada tendrá tantas neuronas como muestras contenga cada traza, lo que permite al modelo evaluar la probabilidad de que una traza pertenezca a una de las clases de salida. Inicialmente, se considera una única capa oculta con 200 neuronas y activación *ReLU*; sin embargo, posteriormente se utilizará un procedimiento de *Grid Search* para analizar y determinar no solo el número óptimo de neuronas en esta capa, sino también el número adecuado de capas ocultas, con el fin de optimizar el rendimiento del modelo.

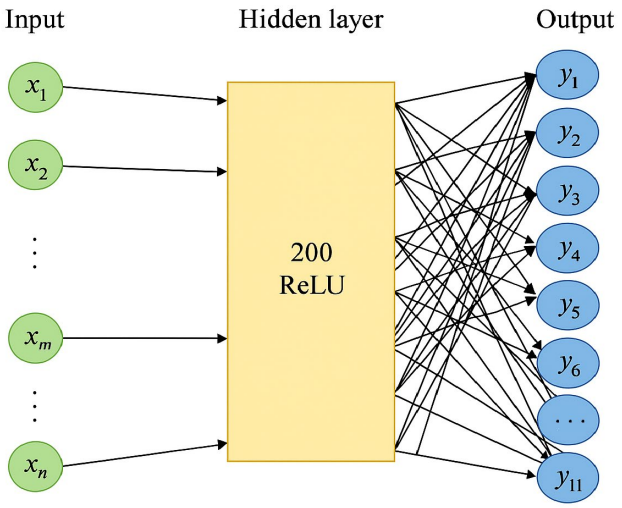

Figura 16: Representación modelo red neuronal

Para la compilación, se utiliza el optimizador Adam con una tasa de aprendizaje de 0.001 y la función de pérdida categorical_cossentropy, haciéndolo apto para salidas codificadas en formato one-hot. Además, se implementa un mecanismo de EarlyStopping que detiene el entrenamiento si no se observa una mejora significativa tras cinco épocas consecutivas, lo que evita el sobreentrenamiento. En conjunto, este modelo ofrece un gran equilibrio entre eficiencia computacional y precisión. Se utiliza este modelo basado en redes neuronales ya que es la aproximación utilizada en el artículo de partida, en cuyo caso usaban capas ocultas GRU, lo cual sirve para seguir un punto de partida común.

6.3 Implementación del modelo de red neuronal en Python

Una vez diseñada la configuración del modelo de red neuronal, resta llevar a cabo las tareas de entrenamiento, test y validación sobre el código. Además, se realizó otro

código para encargarse de las tareas de Grid Search, que sirve para optimizar los hiperparámetros y mejorar la precisión. A continuación, se describe todo este proceso.

6.3.1 Descripción del modelo implementado en Python

Como ya se ha mencionado, se ha utilizado el lenguaje de programación Python para desarrollar el modelo de red neuronal descrito [13]. En él se utilizan librerías como Keras y Scikit-learn. El modelo tiene como finalidad detectar y localizar los fallos en la red óptica PON a partir del análisis de trazas del OTDR que se encuentran en la base de datos creada.

Primero, y tras cargar las librerías necesarias, se cargan los datos de la base de datos final generada por el programa de Matlab y se separan las entradas de las salidas. En el Excel, cuando se recortó la traza para eliminar muestras innecesarias y mantener la parte que interesa de la traza, se puede ver que el número de muestras por cada traza tras el corte es de 2897, y a continuación las 11 columnas finales de codificación. Esa información nos sirve para separar los datos (entradas) de las etiquetas (salidas) de la forma que se ve en las siguientes líneas de código.

```
data = np.loadtxt("MATLAB/database_ruido_final.txt")
X = data[:, :2897]
X = X.astype(np.float32)
y = data[:, -11:]
```

A continuación, se procede a escalar los datos para que tengan media 0 y desviación estándar 1, lo que ayuda a entrenar más eficientemente el modelo. Además, se crean las 12 clases de fallo diferentes que el modelo tendrá que ser capaz de clasificar y se convierten a formato one-hot. Esto se ve en las siguientes líneas de código

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
num_clases = 11
num_muestras = X.shape[0]
y_data = to_categorical(np.argmax(y, axis=1), num_classes=num_clases)
```

A continuación, se divide el conjunto de datos en entrenamiento 80%, validación 10% y test 10%; y se define la red neuronal descrita en el apartado anterior antes de compilarla con el optimizador Adam. Todo ello se observa en las siguientes líneas de

código que, aunque aparezca una única capa oculta inicialmente, en el código final del Grid Search este parámetro se convierte en dinámico.

```
x_train, x_temp, y_train, y_temp = train_test_split(X, y_data,
test_size=0.2, random_state=42) # 80% entrenamiento, 20% lo demás
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp,
test_size=0.5, random_state=42) # del 20%, 10% validación y 10% test
model = models.Sequential()
model.add(layers.Dense(200, activation='relu',
input_shape=(X.shape[1],)))  # Capa de entrada con 2000 neuronas
model.add(layers.Dense(num_clases, activation='softmax'))  # Capa de
salida
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
```

Después, se define el callback del early stopping para evitar el sobreentrenamiento; luego se entrena y finalmente se evalúa el modelo, obteniendo unos resultados que se imprimirán por pantalla. Todo este proceso se muestra en las siguientes líneas de código.

```
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=5,
                               restore_best_weights=True,
                               verbose=1)
history = model.fit(x_train, y_train, epochs=100, batch_size=512,
validation_data=(x_val, y_val),callbacks=[early_stopping])
results = model.evaluate(x_test, y_test)
print("Resultados en el conjunto de test:", results)
```

Los resultados obtenidos se pueden graficar como está hecho en el código. Cabe mencionar que el programa está debidamente documentado y con explicaciones comentadas para resolver cualquier duda que pueda surgir tras la lectura. Finalmente, se realizan las predicciones del modelo y se obtienen las métricas como se muestra en el siguiente extracto del código (precisión, recall, F1-score). De forma más concreta, el valor de precisión o accuracy mide la proporción de predicciones correctas sobre el total de predicciones positivas realizadas; es decir, cuantos de los fallos detectados lo eran realmente. El valor de recall indica la capacidad del modelo para detectar todos los fallos reales. Es la proporción de verdaderos positivos sobre el total de fallos reales. Por último,

el valor de F1-score es la media armónica entre los dos anteriores y se suele utilizar cuando se busca un equilibrio entre ambos.

```
 predictions = model.predict(x_test)
print("Predicciones del modelo en el conjunto de test:")
for i in range(10):
    print(f"Muestra {i+1}: Predicción = {np.argmax(predictions[i])}, Real
= {np.argmax(y_test[i])}")
if predictions.dtype != int:
    predictions = (predictions > 0.5).astype(int)
print(classification_report(y_test, predictions))
print("F1 Score:", f1_score(y_test, predictions, average='macro'))
print("Recall:", recall_score(y_test, predictions, average='macro'))
print("Accuracy:", accuracy_score(y_test, predictions))
```

6.3.2 Entrenamiento y evaluación del modelo

Utilizando los datos de la base de datos final creada, se entrena el modelo neuronal. Como se dijo en apartados anteriores, primero se dividen los datos, cargando y separando las entradas (muestras de las trazas) de las salidas (codificaciones de los fallos), normalizando los datos de entrada y convirtiendo las salidas a formato one-hot para permitir una clasificación multiclase.

Por supuesto, es necesario dividir el conjunto de datos en 3 subconjuntos. Uno se utilizará para el entrenamiento, que en este caso consta el 80% de los datos; luego, un 10% para la validación y otro 10% para el test. Tras el optimizador Adam, el modelo se entrena con un máximo de 100 épocas, aunque el mecanismo de Early Stopping lo detendrá antes si es necesario y de forma automática.

Tras evaluar el rendimiento sobre el conjunto de test se generan gráficas para visualizar el resultado obtenido en cuanto a pérdidas y precisión se refiere a lo largo de las épocas. Se observa a continuación, en la Figura 18 y Figura 19. También se generan predicciones y se calculan métricas como recall y F1-score. Analizando todos estos resultados, las gráficas muestran una disminución progresiva de la training loss, lo que indica que el modelo aprende adecuadamente con cada época. Además, la *validation loss*, aunque inicialmente baja, se estabiliza conforme pasan las épocas e incluso tiende a aumentar ligeramente a partir de la época 6, lo que podría sugerir un inicio de *overfitting*. Aun así, las pérdidas se mantienen bastante bajas en general, por lo que el modelo muestra

un rendimiento razonable. Como se dijo antes, se pueden ver 12 *épocas* porque hay *early stopping*; esto implica que el modelo no mejora de manera significativa a partir de la época 12, por lo que el resto de épocas se tendrá una precisión y pérdidas algo superiores, aunque similares, a la época 12.

```
Epoch 1/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 138s 161ms/step - accuracy: 0.6805 - loss: 1.2344 - val_accuracy: 0.9198 - val_loss: 0.3220
Epoch 2/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 18s 25ms/step - accuracy: 0.9289 - loss: 0.2655 - val_accuracy: 0.9311 - val_loss: 0.2396
Epoch 3/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 18s 25ms/step - accuracy: 0.9496 - loss: 0.1825 - val_accuracy: 0.9380 - val_loss: 0.2292
Epoch 4/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 17s 23ms/step - accuracy: 0.9644 - loss: 0.1220 - val_accuracy: 0.9409 - val_loss: 0.2161
Epoch 5/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 17s 23ms/step - accuracy: 0.9753 - loss: 0.0866 - val_accuracy: 0.9457 - val_loss: 0.2103
Epoch 6/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 16s 22ms/step - accuracy: 0.9739 - loss: 0.0931 - val_accuracy: 0.9429 - val_loss: 0.2252
Epoch 7/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 17s 23ms/step - accuracy: 0.9855 - loss: 0.0508 - val_accuracy: 0.9521 - val_loss: 0.2057
Epoch 8/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 17s 23ms/step - accuracy: 0.9831 - loss: 0.0576 - val_accuracy: 0.9457 - val_loss: 0.2281
Epoch 9/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 17s 23ms/step - accuracy: 0.9784 - loss: 0.0846 - val_accuracy: 0.9470 - val_loss: 0.2252
Epoch 10/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 16s 22ms/step - accuracy: 0.9931 - loss: 0.0284 - val_accuracy: 0.9518 - val_loss: 0.2239
Epoch 11/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 17s 24ms/step - accuracy: 0.9929 - loss: 0.0271 - val_accuracy: 0.9507 - val_loss: 0.2387
Epoch 12/100
732/732 ━━━━━━━━━━━━━━━━━━━━ 18s 24ms/step - accuracy: 0.9943 - loss: 0.0219 - val_accuracy: 0.9383 - val_loss: 0.2991
Epoch 12: early stopping
Restoring model weights from the end of the best epoch: 7.
1464/1464 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9500 - loss: 0.2065
Resultados en el conjunto de test: [0.2029462456703186, 0.9516211748123169]
```

Figura 17: Épocas antes del Early Stopping

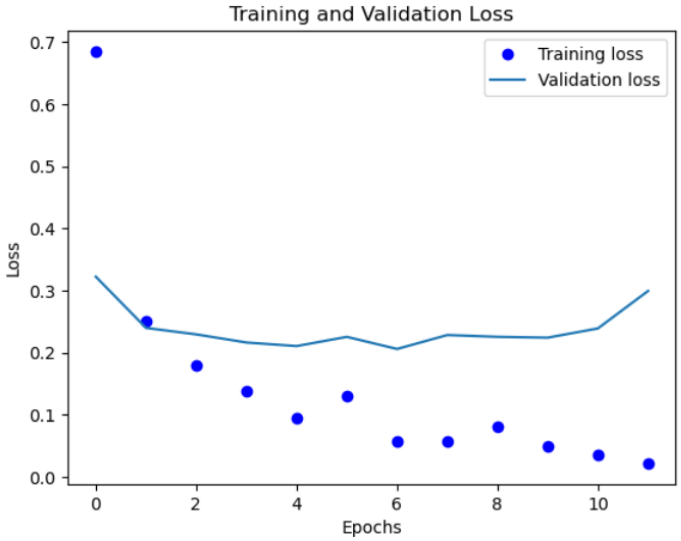

Figura 18: Gráfica de la evolución de las pérdidas a lo largo de las épocas

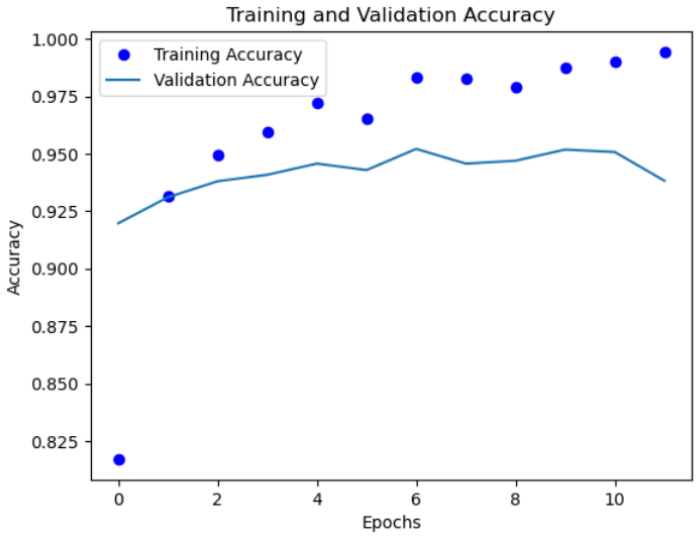

Figura 19: Gráfica de la evoluación de la precisión a lo largo de las épocas

Las métricas tomadas de este modelo inicial (sin optimización de hiperparámetros) se muestran en la Tabla 2. En dicha tabla se se observan unas métricas de F1-score, recall y precisión bastante elevados, tanto en la detección de fallos en cada una de las ramas como en el global, por lo que se puede determinar que la precisión total del modelo (o probabilidad de detectar correctamente un fallo en una traza) es aproximadamente del 95%. Sin embargo, estas medidas son del modelo inicial sin optimización de hiperparámetros, ya que a eso se procede en el siguiente apartado. La columna Fallo/rama se refiere a las 11 clases de fallo mencionadas en el Apartado 5.2, siguiendo el mismo orden, pero sin tener en cuenta la ideal; es decir, 0 => Fallo en la rama de 1m (3º splitter), 1 => Fallo en la rama de 3m (3º splitter), y así sucesivamente.

Fallo/rama	precision	recall	F1-score	Support
0	0.97	0.97	0.97	5281
1	0.99	0.99	0.99	4163
2	0.99	0.98	0.98	4041
3	0.97	0.96	0.97	4221
4	0.97	0.96	0.97	4190
5	0.94	0.94	0.94	4167
6	0.95	0.90	0.92	4090
7	0.93	0.92	0.93	4186
8	0.94	0.92	0.93	4238
9	0.96	0.95	0.95	4105

10	0.94	0.95	0.94	4136
macro avg	0.96	0.95	0.95	46818
weighted avg	0.96	0.95	0.95	46818
samples avg	0.95	0.95	0.95	46818

Tabla 2: Métricas sobre el modelo inicial de red neuronal

6.3.3 Optimización del modelo de red neuronal mediante Grid Search

Pese a las métricas obtenidas en el apartado anterior, el segundo programa diseñado y que se encuentra accesible en [14] está orientado a la optimización del modelo mediante la técnica de Grid Search.

Tras cargar y preprocesar los datos, se define una función para crear un modelo con parámetros configurables, que se pueden modificar. Los hiperparámetros elegidos en nuestro caso serán la tasa de aprendizaje, el número de capas, el número de neuronas ocultas en capas intermedias y la función de activación. Este modelo será de tipo MLP, utilizando así la librería torch y en las siguientes líneas de código se pueden ver los parámetros que se van a optimizar.

```
param_grid = {
    'batch_size': [64, 128],
    'epochs': [20, 30, 40],
    'learning_rate': [0.001, 0.0001, 0.00001],
    'hidden_layer_sizes': [[64],[128],[200],(128, 64), (200, 100), (256,
128),      (64, 32)],
    'activation': [nn.ReLU]
```

Así pues, se define un espacio de búsqueda donde se incluyen diversas combinaciones de hiperparámetros. En concreto, en nuestro caso se realiza el Grid Search con un total de 126 combinaciones, tal y como se observa en el código previo. Tras dicho proceso de búsqueda, los mejores hiperparámetros obtenidos (Figura 20) son:

- Función de activación del modelo: ReLU
- Batch size: 64
- Número de épocas: 30
- Tasa de aprendizaje: 0.001

- Número de capas ocultas: 2
- Número de neuronas: 256 neuronas en la primera capa oculta y 128 neuronas en la segunda capa.

```
 Mejores hiperparámetros encontrados: {'activation': <class
'torch.nn.modules.activation.ReLU'>, 'batch_size': 64, 'epochs': 30,
'hidden_layer_sizes': (256, 128), 'learning_rate': 0.001}
 Mejor score en validación: 0.9731406723909607

 Score (test, DATOS NUNCA VISTOS): 0.9733863044128327
```

Figura 20: Resultados de los hiperparámetros encontrados tras el proceso del Grid Search

Tal y como se observa en la Figura 20, el accuracy global es del 97.3%. Esa precisión es mejor que la anterior, ya que la adición de otra capa aumenta la capacidad y precisión del modelo, mejorando los resultados. Además de estos resultados, se obtuvo el siguiente reporte de clasificación rama a rama de la Tabla 3.

Fallo/rama	Precisión	recall	F1-score	Support
0	0.99	0.99	0.99	10404
1	0.99	1	1	8323
2	1	0.99	1	8323
3	0.98	0.98	0.98	8323
4	0.98	0.98	0.98	8324
5	0.96	0.97	0.96	8323
6	0.97	0.94	0.96	8323
7	0.92	0.97	0.94	8324
8	0.96	0.95	0.96	8323
9	0.97	0.97	0.97	8323
10	0.98	0.96	0.97	8323
macro avg	0.97	0.97	0.97	93636
weighted avg	0.97	0.97	0.97	93636
accuracy			0.97	93636

Tabla 3: Métricas sobre el modelo optimizado mediante Grid Search

Este reporte muestra que el modelo optimizado mediante Grid Search ofrece un rendimiento muy alto de forma generalizada, con precisiones y F1-score superiores a 0.96 en la mayoría de las clases. Las ramas que presentan un mejor comportamiento son las

primeras, la 0, 1 y 2, con F1-score prácticamente perfectos (0.99 o 1.00); esto se debe a que, al ser las primeras ramas del *splitter* 3, el efecto de la zona muerta es mínimo (no hay picos de reflexión justo antes, así que no se juntan con ningún otro) y los picos de reflexión se pueden distinguir bastante bien. Eso se visualiza en la Figura 21, donde aparece un zoom de los picos de reflexión del tercer *splitter*.

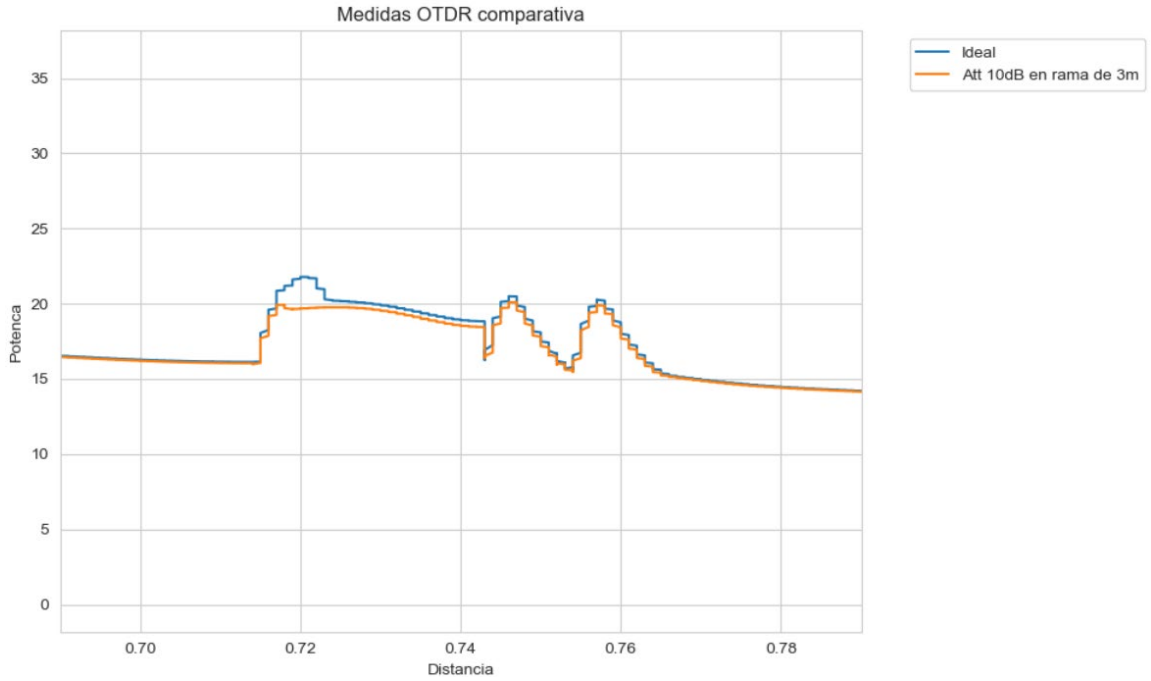

Figura 21: Zoom del tercer splitter donde se aprecian los picos de reflexión

Como dicho efecto se va acumulando conforme hay más ramas juntas en dicho *splitter* (8 ramas en total), se puede apreciar que la rama 7 presenta un F1-score ligeramente inferior, ya que las distancias entre los picos son pequeñas y el efecto de la zona muerta se ve arrastrado por los picos de reflexión anteriores, a diferencia de las primeras ramas. Además, las ramas 8, 9 y 10 vuelven a tener mejores métricas porque los fallos ocurren entre los *splitters* y no entre ramas del mismo, por lo que resulta más fácil de distinguir al haber mayores diferencias de distancia entre ellos (100 metros).

Además, después del modelo de Grid Search se ilustra la matriz de confusión para conocer la veracidad de las predicciones que hace y si son correctas o no. Dicha matriz se puede ver en la siguiente imagen de la Figura 22. Como se puede ver, en la matriz de confusión hay un gran número de casos ya que la base de datos utilizada es muy grande. Cada casilla es el número de veces que el programa dice que ha ocurrido un fallo ahí; y la gran mayoría de estos casos son correctamente predichos, es decir, que el sistema detecta

el fallo en la rama donde realmente ocurrió. Esto confirma la veracidad y aplicabilidad del proyecto, demostrando así su eficacia. Además, observando la matriz de confusión de la Figura 22, se aprecia que las ramas 0, 1, 2 y 3 son aquellas en las que el modelo predice con más claridad y menor confusión ya que concentran la gran mayoría de los aciertos en su diagonal correspondiente y apenas presentan valores significativos en otras celdas, lo que coincide con el análisis hecho anteriormente con el reporte de clasificación obtenido mediante el Grid Search.

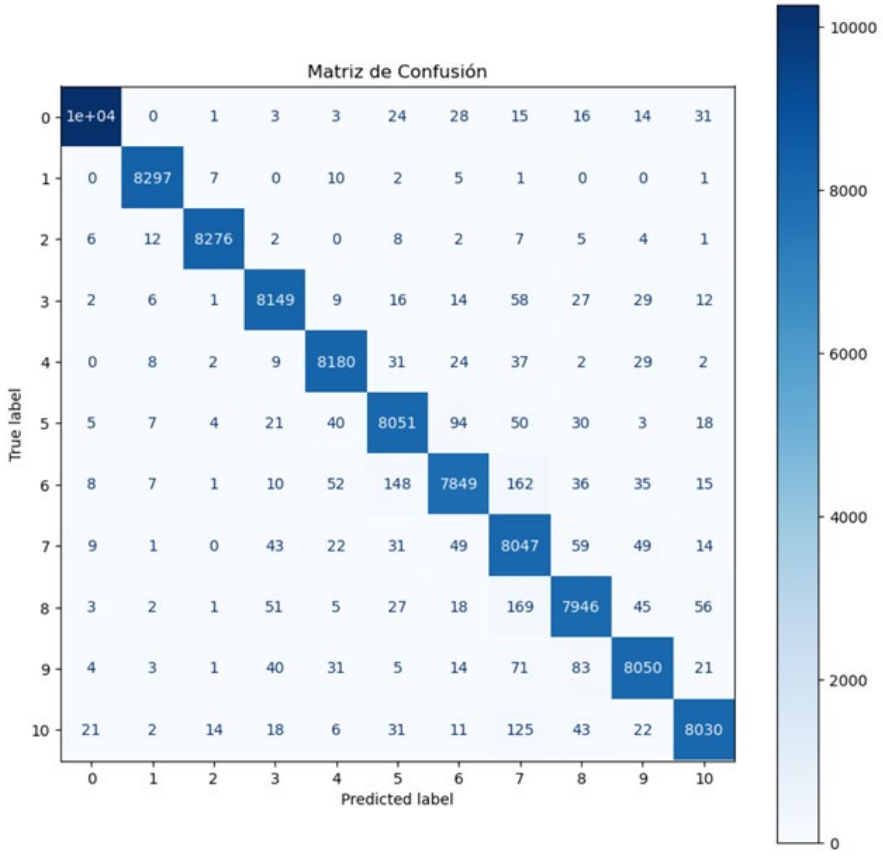

Figura 22: Matriz de confusión

En esta matriz de confusión se observa que los errores más frecuentes se dan en la rama 7 sobre todo. Estas confusiones se pueden deber a que las distancias de cada rama son pequeñas, dificultando así su distinción para el modelo.

6.4 Conclusión

En este capítulo se ha implementado un sistema de IA basado en redes neuronales para la detección y clasificación de fallos en la red de fibra. Se ha demostrado que un modelo simple pero efectivo, con una capa oculta ofrece resultados de cerca del 95% de precisión tras un entrenamiento adecuado.

Además, mediante la técnica del Grid Search se ha optimizado el modelo, observando mejoras significativas al añadir una segunda capa intermedia y alcanzando una precisión del 97.3%. Esto confirma la eficiencia del modelo y su potencial aplicabilidad práctica con recursos suficientes. Para reforzar esta justificación, se mostró la matriz de confusión que ayudó a ver con claridad que el modelo es muy fiable, pues se equivoca una cantidad relativamente pequeña.

7

Conclusiones y líneas futuras

7.1 Conclusiones

Tras el desarrollo de este TFG, se ha conseguido implementar un sistema basado en redes neuronales que emplea técnicas de *Machine Learning* para la detección de fallos en las trazas obtenidas mediante OTDR. Tal como se ha demostrado a lo largo del trabajo, esta metodología se presenta como una solución eficaz y prometedora frente a los métodos tradicionales. Entre las conclusiones que se pueden sacar una vez finalizado este Trabajo Fin de Grado destacan:

- La correcta utilización del OTDR para la obtención de medidas experimentales. Esto aporta realismo al trabajo, pues se podría escalar a cualquier otras arquitecturas y configuraciones GPON con solo tomar medidas en ella.
- Se ha realizado una guía que facilita la comprensión del proceso de toma de medidas, y ayuda a que cualquier persona externa siga los mismos pasos para sus propios fines.
- Se ha creado una base de datos robusta para su posterior manipulación mediante programas que interpretan y transforman datos con un fin determinado.
- Se ha elaborado un modelo de red neuronal que permite clasificar y detectar fallos con alta efectividad. Y el proceso de optimización mediante Grid Search en combinación con las técnicas de validación y test de aprendizaje automático han resultado ser determinantes para obtener unos resultados aptos que refuerzan la utilidad de este tipo de recursos, constituyendo una solución eficaz y escalable.

7.2 Líneas futuras

Tras la realización de este TFG, y a la vista de los resultados obtenidos en esta investigación, se plantean varias posibilidades de desarrollo que podrían mejorar y ampliar este trabajo.

Se podría dar una ampliación del modelo de IA, incluyendo arquitecturas más complejas como redes convolucionales (CNN) o recurrentes (RNN) para demostrar si se obtiene un rendimiento superior, implementando además escenarios más complejos o con ruido controlado o estructurado. También se podría añadir información previa sobre infraestructuras de red al sistema de diagnóstico para mejorar la capacidad de detección tal y como se indicó en el análisis del tercer artículo científico [12].

Además, sería interesante la propuesta de generalizar este modelo para redes GPON de mayor longitud, extendiendo el análisis a longitudes de varios kilómetros y considerando cómo influyen la atenuación acumulada tras tanta distancia y la dispersión en las trazas OTDR. Adicionalmente, podría resultar de interés ampliar la capacidad del sistema para detectar fallos simultáneos en múltiples ramas y no solo en una única ubicación. Una idea para ello sería rediseñar la codificación de salida y emplear enfoques de clasificación multilabel.

Por supuesto, algo que puede resultar de gran ayuda sería el diseño y realización de una interfaz de usuario (GUI) o una aplicación que permita a cualquier técnico cargar trazas y visualizar los resultados o predicciones del modelo, mejorando así su adopción práctica y la eficiencia en el trabajo de dicho técnico.

Finalmente, sería de interés integrar el sistema en tiempo real dentro de redes PON que estén operativas, utilizando servidores para el procesamiento de datos o predicciones.

Este trabajo constituye el comienzo de un sistema completamente automatizado e inteligente para el diagnóstico de redes ópticas de acceso PON. Su futuro desarrollo y ampliación puede suponer una nueva base sólida que permitirá evolucionar la forma de gestionar y mantener infraestructuras que pueden resultar críticas en las telecomunicaciones.

8

Bibliografía

[1] Python. Available: [Último acceso: 29 agosto 2021]

[2] PROMAX Electrónica. *Manual de usuario PROLITE-50*. Available: https://www.promax.es/downloads/manuals/Spanish/PROLITE-5x.pdf [Último acceso: 23 de abril de 2025].

[3] Matlab. Available: [Último acceso: 8 septiembre 2021]

[4] Abdelli, K., Cho, J. Y., Azendorf, F., Griesser, H., Tropschug, C., & Pachnicke, S. (2023). *Fault Monitoring in Passive Optical Networks Using Machine Learning Techniques*. Available: https://doi.org/10.1109/ICTON59386.2023.10207489 [Último acceso: 26 de junio de 2025].

[5] Abdelli, K., Cho, J. Y., Azendorf, F., Griesser, H., Tropschug, C., & Pachnicke, S. (2022). *Machine Learning-based Anomaly Detection in Optical Fiber Monitoring*. Available: https://doi.org/10.1364/JOCN.451289 [Último acceso: 26 de junio de 2025].

[6] Straub, M., Reber, J., Saier, T., Borkowski, R., Shi Li, Khomchenko, D., Richter, A., Färber, M., Käfer, T., & Bonk, R. (2024). *Machine Learning Approaches for OTDR Diagnoses in PONs—Event Detection and Classification*. Disponible en: https://doi.org/10.1364/JOCN.516659 [Último acceso: 26 de junio de 2025].

[7] Usman, A., Zulkifli, N., Salim, M. R., & Khairi, K. (2022). *Fault Monitoring in Passive Optical Network Through the Integration of Machine*

Learning and Fiber Sensors. Disponible en: https://doi.org/10.1002/dac.5134 [Último acceso: 26 de junio de 2025].

[8] PROMAX Electrónica. (s.f.). *OTDR Trace Manager – PROLITE-50 Micro-OTDR [Software]*. Available: https://www.promax.es/esp/descargas/software-y-firmware/PROLITE-50/Micro-OTDR/ [Último acceso: 26 de junio de 2025].

[9] R. Urraca Torices, *Guía OTDR*, documento interno, Universidad de Valladolid, 2025. Available: https://drive.google.com/drive/folders/1UafjMnCqwwOxJeuTa67LmUFrMN2ksPKg?usp=sharing [Último acceso: 26 de junio de 2025].

[10] Microsoft Corporation, *Microsoft Excel*, versión 2305, Microsoft, 2023. [Software]. Available: https://www.microsoft.com/excel [Último acceso: 26 de junio de 2025].

[11] R. Urraca Torices, *Base de datos OTDR*, archivo Excel interno, Universidad de Valladolid, 2025. Available: https://drive.google.com/drive/folders/1UafjMnCqwwOxJeuTa67LmUFrMN2ksPKg?usp=sharing [Último acceso: 26 de junio de 2025].

[12] R. Urraca Torices, Script MATLAB para creación de base de datos, código fuente interno, Universidad de Valladolid, 2025. Available: https://drive.google.com/drive/folders/1UafjMnCqwwOxJeuTa67LmUFrMN2ksPKg?usp=sharing [Último acceso: 26 de junio de 2025].

[13] R. Urraca Torices, Modelo de red neuronal para clasificación OTDR, script Python interno, Universidad de Valladolid, 2025. Available: https://drive.google.com/drive/folders/1UafjMnCqwwOxJeuTa67LmUFrMN2ksPKg?usp=sharing [Último acceso: 26 de junio de 2025].

[14] R. Urraca Torices, Optimización del modelo OTDR mediante Grid Search en PyTorch, script Python interno, Universidad de Valladolid, 2025. Available:

https://drive.google.com/drive/folders/1UafjMnCqwwOxJeuTa67LmUFrMN2ksPKg?usp=sharing [Último acceso: 26 de junio de 2025].

[15] Hood, D., & Trojer, E. (2011). *Gigabit-capable Passive Optical Networks (GPON): A Practical Guide to GPON and Next-Generation PON Systems*. John Wiley & Sons. Available: http://dx.doi.org/10.1002/9781118156070 [Último acceso: 26 de junio de 2025].

[16] Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Prentice Hall. Available: https://aima.cs.berkeley.edu/ [Último acceso: 26 de junio de 2025].