Universidad de Zaragoza

Trabajo Fin de Grado

Facultad de Ciencias

Grado en Física

APLICACIÓN DE ALGORITMOS BASADOS EN REDES NEURONALES PARA EL RECONOCIMIENTO AUTOMÁTICO DE EVENTOS EN UN SISTEMA COTDR (COHERENT OPTICAL TIME DOMAIN REFLECTOMETER)

Curso 2020 - 2021

Autor:
Manuel Romeo Monterde

Director:
Dr. Jesús Subías Domingo

Índice

1. Introducción

Desde hace unos años, hay una corriente tecnológica que se está centrando en explotar las llamadas *técnicas de sensado distribuido*. Una de estas técnicas es el sensado distribuido de vibraciones, basado en fibra óptica y que emplea sistemas COTDR. Estos sistemas permiten monitorizar lo que sucede en entornos de largos tramos de fibra con un único equipo. Muchos de estos tramos están infrautilizados con las denominadas fibras oscuras, que son fibras que no se utilizan para telecomunicaciones.

Esta técnica de monitorización está teniendo gran repercusión en varios sectores industriales porque tiene un amplio abanico de aplicaciones, como por ejemplo la protección de instalaciones, la seguridad perimetral o la monitorización del tráfico. En definitiva, estas técnicas permiten la detección y clasificación de eventos sucediendo en entornos de la fibra. ¿Y qué queremos decir con eventos?

En función de la aplicación, los eventos son diferentes, por ejemplo: a las empresas de red eléctrica les puede interesar medir la velocidad del viento a través del movimiento de sus cables o detectar el arrastre de anclas en las proximidades de sus cables submarinos, y sin embargo, a las compañías ferroviarias les puede interesar controlar la entrada no regulada de personas en las proximidades de las vías o detectar caídas de objetos en las mismas.

A continuación, incluimos una imagen real obtenida en una subestación de Red Eléctrica de España en PLAZA (Zaragoza), cuyo objetivo es detectar intrusiones en el perímetro de la subestación.

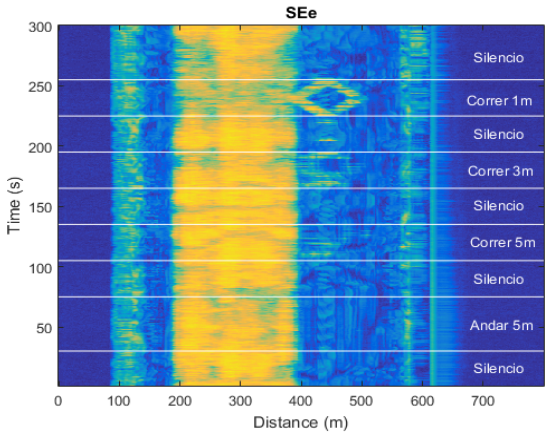


Figura 1: Ejemplo de una imagen real obtenida en PLAZA (Zaragoza).

Por otro lado, los algoritmos de aprendizaje automático basados en redes neuronales, forman parte de lo que se entiende por *Machine Learning*, que es la rama computacional de la inteligencia artificial cuyo objetivo es desarrollar técnicas para que los ordenadores aprendan.

El objeto de este trabajo es unir estos dos campos con el propósito de entrenar a un modelo de inteligencia artificial basado en redes neuronales, con imágenes obtenidas por simulación. Tras el entrenamiento del modelo, este debe ser capaz de, a partir de las imágenes reales como la presentada en la Figura 1, clasificar posibles alarmas o eventos de interés en tiempo real.

2. Objetivos

A partir de un modelo de sistema COTDR creado empleando el lenguaje de programación *Python*, que simule las señales recibidas en un sistema real de este tipo, este trabajo tiene tres objetivos:

1. Proponer una o varias opciones de tratamiento matemático adecuado de las señales, de forma que se registren los cambios en dicha señal de forma ilustrativa y numérica, e implementarlas en el modelo.

2. Proponer y modelar distintas clases de eventos que sucedan en entornos de la fibra, que se manifiestan en forma de estímulos o perturbaciones, y que se propagan hasta generar alteraciones en el estado de la fibra sensora, y obtener gran cantidad de imágenes sintéticas resultantes de implementar estos eventos y sus efectos sobre las señales, en el modelo.

3. A partir de un modelo de aprendizaje automático basado en redes neuronales convolucionales (generado en el lenguaje de programación *Python* utilizando la biblioteca de código abierto *Keras*), conseguir que dicho modelo aprenda a detectar y clasificar los distintos tipos de eventos simulados a partir de conjuntos de imágenes sintéticas que generaremos, y que le proporcionaremos para dicho fin.

3. Aproximación al modelo

Un sistema COTDR consta tanto de interrogador (reflectómetro, que incluye láser y detector), como de la fibra óptica sensora.

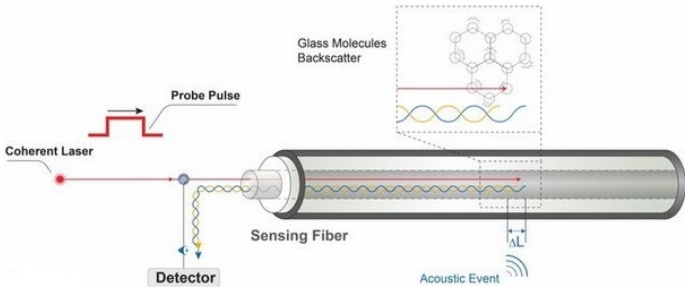


Figura 2: Esquema de un sistema COTDR.

A continuación, introducimos brevemente los aspectos relevantes que debemos conocer acerca del modelo empleado para simular un sistema COTDR.

3.1. Fibra óptica y retrodispersores Rayleigh

La fibra óptica, que va metida dentro de un cable, constituye una guía de ondas dieléctrica, que consta de dos partes: un núcleo central con alto índice de refracción, y un núcleo externo con menor índice de refracción. Estas estructuras permiten un tipo de modo de propagación caracterizado por atenuaciones extremadamente bajas, lo cual permite que el pulso óptico viaje largas distancias a través de la fibra sin apenas pérdidas.

Durante el proceso de fabricación de la fibra óptica, y concretamente en la fase de enfriamiento, tienen lugar variaciones en la densidad y la composición de la red cristalina, que generan irregularidades en la misma que se manifiestan como fluctuaciones del índice de refracción [1]. El conjunto de estas irregularidades es precisamente a lo que llamamos retrodispersores Rayleigh, centros de dispersión o simplemente emisores Rayleigh.

En nuestro modelo, al conjunto de posiciones que ocupan N retrodispersores Rayleigh en la fibra lo llamamos *perfil axial de la fibra óptica* [2]:

$$\{z_1, ..., z_m, ..., z_N\} \tag{3.1.1}$$

La retrodispersión de los pulsos ópticos a causa de la existencia de estos centros de dispersión es la base del modelo COTDR, y nos permitirá detectar eventos sucediendo en puntos en el entorno de la fibra.

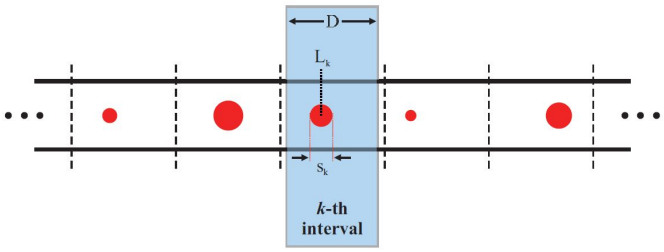


Figura 3: Esquema del perfil axial de una fibra óptica [3].

En nuestro modelo, S_k es el diámetro del emisor, L_k es z_k y el parámetro D que determina la distancia media entre centros de dispersión es lo que llamaremos *longitud de Rayleigh* y denotaremos como L_{Ray}.

3.2. Reflectometría óptico-coherente de dominio de tiempo (COTDR) en un modelo de retrodispersión

La reflectometría óptico-coherente de dominio de tiempo (COTDR) es un método de análisis de fibra óptica, que consiste en enviar pulsos ópticos coherentes a través de la fibra y analizar la señal reflejada [1].

En un reflectómetro COTDR, se mide una señal eléctrica en un fotodetector, proporcional a la potencia óptica que recibe. Dicha potencia proviene de la señal óptica reflejada que llega en tiempos de sampleo [2]. La corriente en el fotodiodo es proporcional al módulo al cuadrado del campo reflejado total E_R, tal que:

$$I_{detectada} \propto |E_R(t)|^2 \tag{3.2.1}$$

Lo que llega al detector es la suma interferencial de las reflexiones que en cada momento t son no nulas, y a la figura interferencial que genera cada pulso óptico en el tiempo que tarda en ir y volver la llamaremos traza interferencial. Como podemos observar en la figura a continuación, dicha traza se puede representar frente al espacio conociendo la velocidad de propagación del pulso:

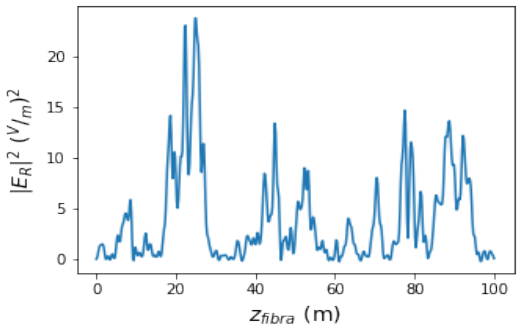


Figura 4: Ejemplo de traza interferencial.

Si los centros de dispersión no cambian de posición, todos los pulsos producirán la misma

traza interferencial. Sin embargo, cuando uno o varios centros de dispersión cambian de posición, introducen cambios de fase en la señal retrodispersada que llega al reflectómetro. Esto se traduce en cambios en las trazas interferenciales, y un adecuado tratamiento de las señales nos permitirá localizar dónde se están produciendo los cambios en las posiciones de los emisores.

El campo eléctrico de la onda óptica reflejada por los centros de dispersión a lo largo de la fibra es, salvo una fase global debida a la reflexión:

$$E_R = E_0 e^{j(kx-wt)} \tag{3.2.2}$$

donde $k = \frac{2\pi}{\lambda_{pulso}}$ es el número de ondas del láser y $c = \frac{w}{k}$ es la velocidad del pulso óptico a través de la fibra.

Sea un centro de dispersión m, definido por una posición en la fibra z_m que refleja un pulso óptico, llamamos tiempo de llegada al tiempo que tarda la onda reflejada en llegar al detector, que es el punto de partida:

$$t_{R_m} = \frac{2z_m}{c} \tag{3.2.3}$$

y que es dos veces el tiempo que le cuesta llegar al emisor. Ahora, definimos el intervalo de tiempo durante el cual se detectará campo eléctrico E_{Rm} a causa de la reflexión del centro de dispersión m durante el tiempo Δt que dura el pulso como:

$$(t_{Rm}, t_{Rm} + \Delta t) \tag{3.2.4}$$

Suponemos que tenemos un conjunto de N centros de dispersión, con posiciones $\{z_1, ..., z_N\}$. Estos emisores reflejarán luz durante un intervalo de tiempo Δt, y llegarán al detector en la siguiente lista de tiempos, que llamamos *lista de tiempos de activación de los emisores*:

$$\{(t_{R1}, t_{R1} + \Delta t), ..., (t_{RN}, t_{RN} + \Delta t)\} \tag{3.2.5}$$

Sea $N_{act}(t)$ el número de emisores activados en un tiempo t, el campo eléctrico total reflejado que llega al detector es igual a la suma de las contribuciones de todos los centros de dispersión que estén activados en dicho tiempo t, es decir:

$$E_R = \sum_{m=1}^{N_{act}(t)} E_{Rm}(t) = \sum_{m=1}^{N_{act}(t)} a_m e^{j(kz_m - wt)} \tag{3.2.6}$$

3.3. Matriz de interferencia M: *"fast time"* y *"slow time"*

En un sistema COTDR, lidiamos con la detección de ondas reflejadas empleando frecuencias de muestreo del orden de 1 GHz [3], fruto de inyectar pulsos ópticos en la fibra con frecuencias de repetición muy inferiores. Por tanto, es muy conveniente distinguir las dos escalas de tiempo presentes en el problema: las escalas *"fast time"* y *"slow time"*. Puesto que las simulaciones son numéricas, ambas escalas de tiempo son discretas.

La escala *"fast time"*, que también llamamos t_{fast}, o más convenientemente τ, es la que rige cuando tratamos una única traza interferométrica, es decir, la reflexión de un único pulso.

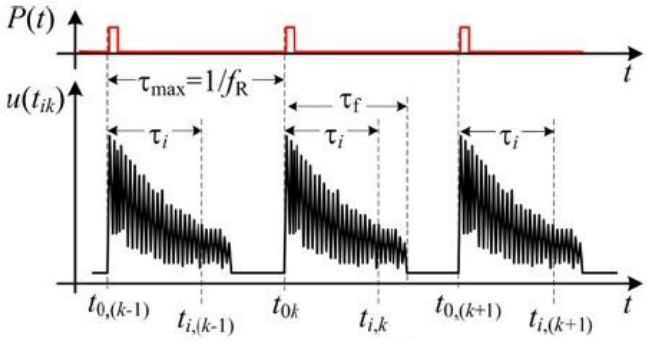


Figura 5: Escala de tiempo *"fast time"* [2].

La discretización de esta escala depende de la frecuencia de sampleo ADC del detector del sistema COTDR, a la que llamaremos f_d. Por tanto, dividiremos τ en steps i, de forma que:

$$\tau_i = \frac{i}{f_d} \ , \ con \ [\tau_i] = s \tag{3.3.1}$$

El tiempo total que tarda un pulso óptico en recorrer ida y vuelta una fibra de longitud L a una velocidad c es $\tau_{tot} = \frac{2L}{c}$ segundos, lo cual corresponde a dividir τ_{tot} en

$$\tau_{tot} = \frac{2L}{c} = \frac{i_{tot}}{f_d} \Rightarrow i_{tot} = \tau_{tot} f_d = \frac{2L}{c} f_d \ , \ con \ [i_{tot}] = \ steps \tag{3.3.2}$$

steps temporales. Es importante remarcar que tanto i como i_{tot} son pasos temporales y no tienen unidades, mientras que τ_i y τ_{tot} son tiempos y se miden en segundos.

Podemos relacionar esta escala temporal con una escala espacial, a través de la velocidad de la luz en la fibra c, de forma que,

$$z_i = c\tau_i = c\frac{i}{f_d} \ , \ con \ [z_i] = m \tag{3.3.3}$$

lo cual nos permite ubicar los valores de la traza en el espacio.

Por otro lado, a la escala de tiempo que rige cuando tratamos el tiempo entre pulsos se le llama *"slow time"*, que también llamamos t_{slow}, o más convenientemente t_k.

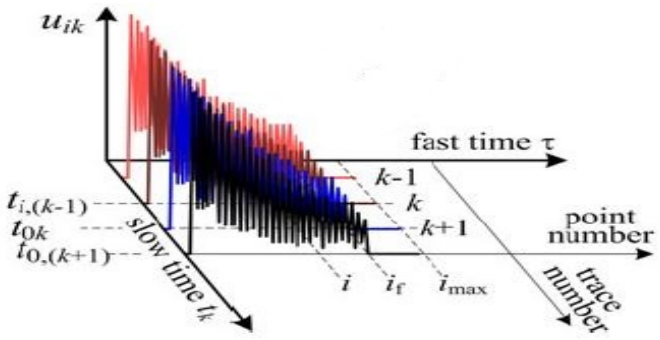


Figura 6: Escala de tiempo *"slow time"* [2].

Sea un pulso k, la discretización de esta escala está determinada por la frecuencia de repetición de pulso del láser, que llamamos f_R, según la siguiente relación:

$$t_k = \frac{k}{f_R} \tag{3.3.4}$$

donde el valor de la traza interferencial correspondiente al pulso k en el tiempo τ_i será $M_k(\tau_i)$.

Juntando ambas escalas, podemos describir el tiempo como $t_{k,i} = t_k + \tau_i$. La forma de tratar el problema numéricamente pasa por esta idea, y para ello es de vital importancia construir la matriz de interferencia M.

La matriz de interferencia M se construye registrando los valores de las trazas $M_k(\tau_i)$, para $k = 1, ..., K$ pulsos lanzados e $i = 1, ..., i_{tot}$ pasos temporales. Los sucesivos pulsos k generan sucesivas filas de la matriz M. Si lanzamos un número total de pulsos K, podemos expresar la matriz de interferencia M como:

$$M_{ki} = \begin{pmatrix} M_1(\tau_1) & M_1(\tau_2) & \cdots & M_1(\tau_{i_{tot}}) \\ M_2(\tau_1) & M_2(\tau_2) & \cdots & M_2(\tau_{i_{tot}}) \\ \vdots & \vdots & \ddots & \vdots \\ M_{tot}(\tau_1) & M_{tot}(\tau_2) & \cdots & M_{tot}(\tau_{i_{tot}}) \end{pmatrix} \tag{3.3.5}$$

3.4. Implementación y parámetros

Para modelar un sistema COTDR en Python, se crean clases de los objetos que conforman el sistema COTDR, es decir: láser, fibra, detector y reflectómetro COTDR. Uno de los métodos que incluye la clase COTDR es *ondaSuma(t)*, que recibe la lista de tiempos de activación de los emisores, calcula los desfases correspondientes a recorrer la fibra ida y vuelta y finalmente calcula la interferencia de dichas ondas en cada tiempo τ_i, devolviendo el valor de la traza interferencial en ese tiempo [4], es decir:

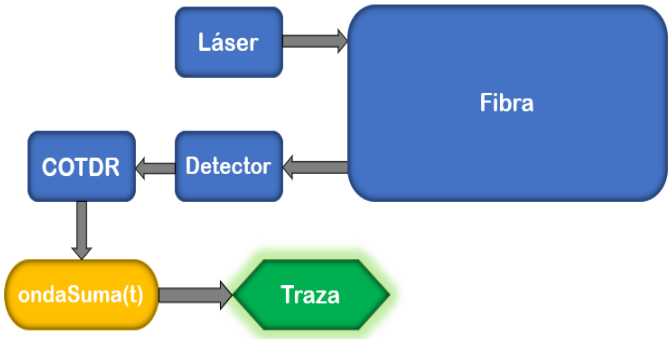


Figura 7: Esquema del modelo del sistema COTDR.

Al instanciar las clases mencionadas (cuadrados azules en la figura anterior) y para futuras simulaciones, hacemos la siguiente elección de parámetros, basada en las características de los equipos comerciales: una velocidad de los pulsos en la fibra de $c = 2 \cdot 10^8 \, \frac{m}{s}$, longitud de Rayleigh $L_{Ray} = 1 \, m$, y un modelo de atenuación constante en la fibra, de amplitud 0,15 $\frac{dB}{km}$, dependiente

del tiempo de forma que $atenuacion(t) = 10^{-\frac{0{,}15 \cdot ct}{1000 \cdot 20}}$. Asumimos que el láser genera un pulso óptico monocromático de longitud de onda central $\lambda = 1550\ nm$ y duración $\Delta t = 200\ ns$, y que el detector opera a una frecuencia de muestreo $f_d = 1\ GHz$, con frecuencia de corte $f_{corte} = 0{,}1\ GHz$.

Realizamos las simulaciones para una fibra de longitud $L = 100\ m$, y el número de puntos espaciales en los que se discretiza la fibra viene determinado por la expresión (3.3.4), y es igual a 500, que se traduce en 500 columnas en la matriz de interferencia. Simularemos tiempos de 60 segundos, discretizaremos el tiempo largo en pasos temporales de 0.12 segundos. De esta forma, obtenemos 500 trazas interferométricas, lo cual corresponde a 500 filas en la matriz de interferencia.

4. Tratamiento matemático de la matriz de interferencia

4.1. Construcción del *waterfall* W

Asumiendo que se producirán cambios en las K trazas interferenciales, debemos transformar la matriz M en otra matriz que permita evaluar las variaciones de los elementos de cada columna i de una forma ilustrativa y numérica. La nueva matriz que construiremos a partir de la matriz de interferencia M se denomina *waterfall*, y nos referiremos a ella como W. Nuestra propuesta para la construcción de esta nueva matriz se basa en el análisis corrido por bloques de la matriz M. ¿Y qué quiere decir esto?

Por un lado, el análisis por bloques consistirá en aplicar un algoritmo de análisis "estadístico" a cada bloque de la matriz M. ¿Y qué son los bloques? Los bloques serán submatrices de dimensión (N, i_{tot}) en las que dividiremos a la matriz de interferencia, y el algoritmo trabajará sobre todos los vectores columna que lo conforman. El resultado de aplicar el algoritmo a las submatrices será un vector de dimensión $(1, i_{tot})$, y dicho vector será una nueva fila del *waterfall*. Se pretende ilustrar este proceso con el siguiente ejemplo, en el que vemos lo que le sucede a una columna i de 5 filas, seleccionando bloques de tamaño $N = 3$.

$$\begin{pmatrix} M_1(\tau_1) & \cdots & M_1(\tau_i) & \cdots & M_1(\tau_{i_{tot}}) \\ M_2(\tau_1) & \cdots & M_2(\tau_i) & \cdots & M_2(\tau_{i_{tot}}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ M_5(\tau_1) & \cdots & M_5(\tau_i) & \cdots & M_5(\tau_{i_{tot}}) \end{pmatrix} \rightsquigarrow \begin{pmatrix} M_{1i} \\ M_{2i} \\ M_{3i} \\ M_{4i} \\ M_{5i} \end{pmatrix} \rightsquigarrow \begin{pmatrix} W_{1i} = algoritmo(bloque1) \\ W_{2i} = algoritmo(bloque2) \\ W_{3i} = algoritmo(bloque3) \end{pmatrix} \quad (4.1.1)$$

Por otro lado, el concepto de análisis corrido también se entiende con el ejemplo anterior: la columna i tiene 5 filas, y elegimos un tamaño $N = 3$ para los bloques. El bloque 1 incorpora los datos: $\{M_{1i}, M_{2i}, M_{2i}\}$. Hacer un análisis corrido por bloques implica que el bloque 2 deseche el primer elemento del bloque 1 e incorpore el elemento de la fila siguiente, es decir: $\{M_{2i}, M_{3i}, M_{4i}\}$, y así hasta llegar a la última fila.

El proceso continuará hasta que, de forma general, pasemos de la matriz de interferencia M_{ki} de dimensión (K, i_{tot}) a la matriz *waterfall* W_{fi} de dimensión $(K - N, i_{tot})$, de forma que:

$$dim(M_{ki}) = (K, i_{tot}) \longrightarrow dim(W_{fi}) = (K - N, i_{tot}) \tag{4.1.2}$$

donde recordemos que K es el número total de pulsos lanzados, es decir el número total de trazas interferométricas.

4.2. Tipos de algoritmos

Teniendo claro cómo se construye el *waterfall*, podemos centrarnos en la forma de los algoritmos empleados para analizar las variaciones de las trazas. Proponemos el estudio de tres opciones de análisis: varianza, ajuste funcional y análisis espectral por transformada de Fourier.

4.2.1. Varianza

La idea es calcular la media de cada columna i del bloque, y su correspondiente varianza, es decir:

$$W_{(k-N),i} = \frac{1}{N}\sum_{j=1}^{N}(M_{(k-j),i} - \bar{M}_{(k-N),i})^2 \ \forall \ k \ \epsilon \ [N, K] \tag{4.2.1}$$

donde $\bar{M}_{(k-N),i}$ es la media aritmética de los elementos de la columna i del bloque $(k-N)$ del *waterfall*.

El algoritmo creado consiste en realizar la convolución de dos matrices: un vector de unos de tamaño N con el bloque correspondiente. La media del bloque corresponde a dividir la matriz resultado por la longitud del vector de "unos", es decir N. En las simulaciones, empleamos bloques de tamaño $N = 20$.

4.2.2. Ajuste funcional

Realizamos un ajuste polinómico de orden *deg* por el método de mínimos cuadrados del bloque de la columna i. Entonces, calculamos la diferencia al cuadrado entre el valor que se obtendría por extrapolación para el elemento en la fila inmediatamente posterior al final del bloque y dicho elemento, es decir:

$$W_{(k-N),i} = (M_{(k+1),i} - y_{(k-N),i}(M_{(k+1),i}))^2 \ \forall \ k \ \epsilon \ [N, K] \tag{4.2.2}$$

donde $y_{(k-N),i}(M_{(k+1),i})$ corresponde al valor extrapolado según el ajuste del bloque $(k-N)$.

El algoritmo creado incluye un método de la librería NumPy de Python, llamado *polyfit* [5], que realiza el ajuste a un polinomio de grado *deg* de una serie de datos, calculando el vector de coeficientes que minimizan el error al cuadrado, en el orden $deg, deg-1, ..,0$. Con dichos coeficientes se calculan los valores extrapolados con otra función de Numpy llamada *polyval* [5]. En las simulaciones, empleamos bloques de tamaño $N = 10$, y ajustes lineales, es decir $deg = 1$.

4.2.3. Análisis espectral por transformada de Fourier

Calculamos la transformada de Fourier discreta del bloque de tamaño N de cada columna i. Dicha operación nos devuelve N coeficientes de la transformada de Fourier. Como ningún evento corresponde a frecuencia nula, eliminamos el coeficiente correspondiente a la misma y almacenamos el coeficiente con mayor módulo.

El algoritmo creado corresponde a realizar la transformada de Fourier discreta, tomando como entrada un vector columna de $M_{k,i}$ de dimensión $(N, 1)$, es decir:

$$Y_K = \sum_{n=0}^{N-1} M_{(k-n),i} e^{-\frac{2\pi j}{N} Kn} \ , \ K = 0, ..., N-1 \tag{4.2.3}$$

donde Y_K es el coeficiente frecuencial correspondiente a la frecuencia dada $f = \frac{K}{n}$. El algoritmo que calcula todos los coeficientes y devuelve un vector columna con todos los coeficientes frecuenciales es un método perteneciente a la librería SciPy, y llamado *fft* [6]. De entre los coeficientes, eliminamos el correspondiente a frecuencia nula y calculamos el módulo de todos ellos. Finalmente almacenamos el cofieciente de módulo máximo en la posición del *waterfall* $W_{(k-N),i}$. En las simulaciones, empleamos bloques de tamaño $N = 75$.

4.3. Implementación

Dentro de la clase COTDR, creamos tres métodos distintos, que llamaremos: *generaWaterfallVarianza*, *generaWaterfallAjuste* y *generaWaterfallFourier*. Todos estos métodos reciben como parámetro de entrada la matriz de interferencia M y el tamaño de los bloques N, y devuelven la matriz *waterfall* W. Esquemáticamente esto se traduce en:

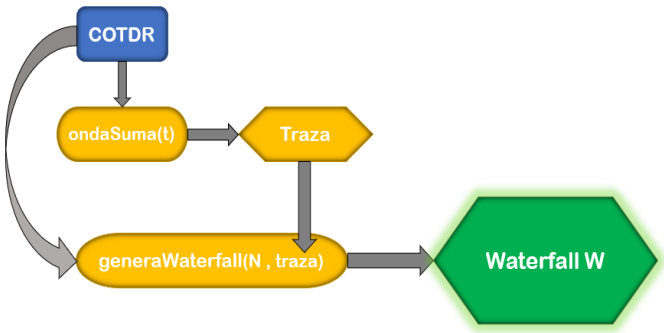


Figura 8: Esquema de implementación del *waterfall*

.

De ahora en adelante, hablaremos tanto de *waterfall* como de *imagen*, pues el *waterfall* no deja de ser una imagen en dos dimensiones, donde en cada posición de la matriz tenemos un valor. Representaremos W como un mapa de color de valores arbitrarios (en nuestro caso el mapa de color *plasma*, de Matplotlib), lo que dará lugar a imágenes donde en zonas donde se estén produciendo muchos cambios el *waterfall* tendrá un valor alto y el color amarillo asociado, y las zonas donde no sucedan cambios se ilustrarán en color morado.

5. Simulación de eventos

Hasta ahora, hemos presentado el principio de funcionamiento de un sistema COTDR, y las posibilidades de construcción del *waterfall* para interpretar las trazas interferenciales a lo largo del eje de tiempos largos t_k.

En el apartado 1 remarcamos que dichas variaciones son fruto de cambios de posición de los centros de dispersión. Estos desplazamientos pueden ser motivados por muchos tipos de estímulos, que se traducen en cambios de presión, temperatura, etc. en la fibra. A estos estímulos es a lo que nosotros llamamos *eventos*.

A continuación, exponemos las consideraciones tomadas para modelar tanto los eventos en sí, como su propagación por un medio material hasta la fibra. Además, presentamos las cuatro clases de eventos que proponemos estudiar, y veremos qué efectos tienen en las trazas a través de los tres tipos de *waterfall* W.

5.1. Consideraciones para simulación de eventos

Para modelar los eventos, tendremos en cuenta los siguientes puntos:

i. Los estímulos que den lugar a eventos, se propagarán como ondas de presión en frentes de onda semiesféricos a una velocidad de propagación determinada por el medio material, que llamaremos v_{prop}.

ii. La velocidad de propagación de un pulso óptico es mucho más grande que la velocidad de propagación de las ondas de presión en el medio, es decir $c \ggg v_{prop}$. Por tanto, podemos despreciar la propagación de las ondas de presión a lo largo del eje de tiempos rápidos τ.

iii. No consideraremos efectos dispersivos en la propagación de las ondas de presión, por lo que la perturbación conservará su forma durante la propagación.

iv. Sí consideraremos atenuación y el tiempo de vuelo de la onda de presión que transporta la perturbación hasta la posición de cada emisor m en la fibra, que llamaremos $t_{vuelo,m}$.

v. Los eventos tendrán lugar en el plano de la superficie, a una distancia d_m de cada emisor que dependerá posición del mismo en el espacio, $\vec{r}_{cdd,m} = (z_m, y_{fibra}, z_{fibra})$, y de la posición del evento, $\vec{r}_{evento} = (x_{ev}, y_{ev}, z_{sup})$. Además, el punto donde se localiza el evento se denomina *foco*, y el estímulo se considera puntual.

vi. Definimos una función temporal a la que llamamos *función paso* $P(t)$. Dicha función describe la variación de la presión en el punto de la superficie en el que se produce el evento. Tiene una amplitud máxima igual a la unidad, es adimensional y se caracteriza por dos parámetros: el tiempo de subida y bajada, t_{subida}, y el tiempo de pisada t_{pisada}.

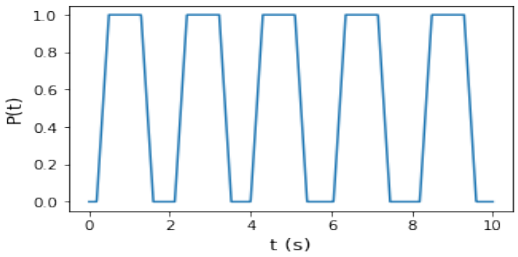


Figura 9: Ejemplo de la función *paso*.

vii. Sea un punto de la fibra cualquiera a una distancia d del foco del estímulo, la atenuación puede ser representada como un factor con la siguiente forma:

$$P_0 \frac{1}{d} e^{-\alpha d} \tag{5.1.1}$$

donde P_0 es la amplitud de la perturbación, α es el coeficiente de absorción en amplitud, y $\frac{1}{d}$ el factor debido a la expansión de los frentes de onda semiesféricos.

La evolución temporal vendrá dada por la función *paso* definida anteriormente, retrasada en el tiempo:

$$P(t - t_0 - t_{vuelo}) \tag{5.1.2}$$

donde t_0 es el tiempo en el que comienza el estímulo en el foco, y $t_{vuelo} = \frac{d}{v_{prop}}$ es el tiempo que tarda en llegar el frente de onda de presión desde el foco hasta el punto de la fibra.

Finalmente, a cada punto de la fibra z_m donde tenemos un centro de dispersión le llega una presión:

$$P_m(d_m, t) = P_0 \frac{1}{d_m} e^{(-\alpha d_m)} P(t - t_0 - t_{vuelo}) \tag{5.1.3}$$

viii. Asumimos que la dilatación que se le asigna a cada punto de la fibra z_m donde hay emisor es proporcional a P_m en cada punto temporal del eje de tiempos largos t_k. La dilatación estará normalizada según el factor $\frac{L_{Ray}}{10}$, donde L_{Ray} corresponde a la longitud de Rayleigh, y es la distancia media entre centros de dispersión. Esto se hace para asegurar que los emisores no se lleguen a tocar. Por tanto, cada centro de dispersión m sufrirá una dilatación:

$$A_{0,m} = \frac{L_{Ray}}{10} P_m(d_m, t) \tag{5.1.4}$$

ix. La fibra se considera unidimensional, y que todos los desplazamientos debido a dilatación tendrán el mismo sentido.

5.2. Clases de eventos

Las consideraciones anteriores conciernen especialmente al modelo de propagación de las perturbaciones generadas por los eventos, su forma y sus efectos en la fibra sensora. El siguiente paso es definir los eventos que simularemos, para generar imágenes aplicando todo lo anterior.

5.2.1. Ruido térmico

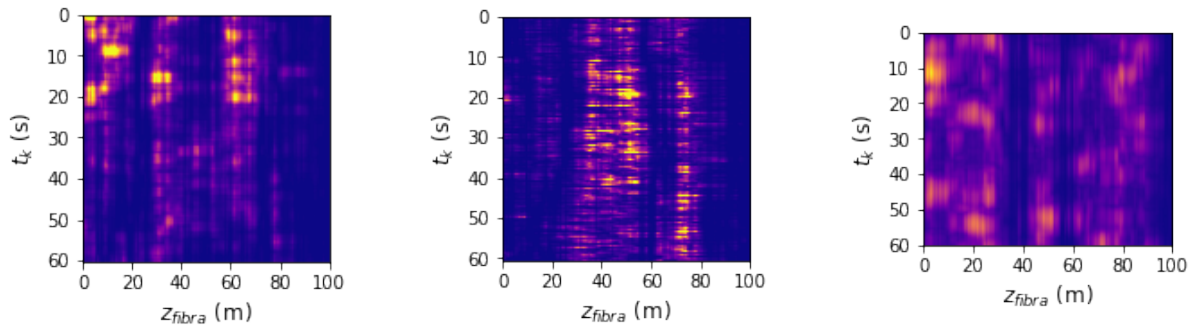
Proponemos modelar el ruido térmico como una variación aleatoria de las posiciones de los emisores Rayleigh, según una distribución normal [7].

Determinamos una intensidad de ruido constante, que posteriormente en las simulaciones variará dentro de un rango para hacerlas más realistas. Dicha intensidad estará normalizada a $\frac{L_{Ray}}{10}$ para evitar la superposición de emisores, y la amplitud de dilatación debida a ruido térmico será un valor aleatorio perteneciente a la distribución normal

$$A_{ruido}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.2.1}$$

donde $\mu = 0$ y $\sigma = \frac{L_{Ray}}{10}$.

Implementando este evento en el modelo, obtenemos imágenes como las siguientes por los tres métodos:



(a) Varianza (b) Ajuste (c) Fourier

Figura 10: Ejemplos de imágenes de ruido térmico.

5.2.2. Evento fijo

El evento fijo se caracteriza porque la posición del foco es constante y no depende del tiempo, es decir

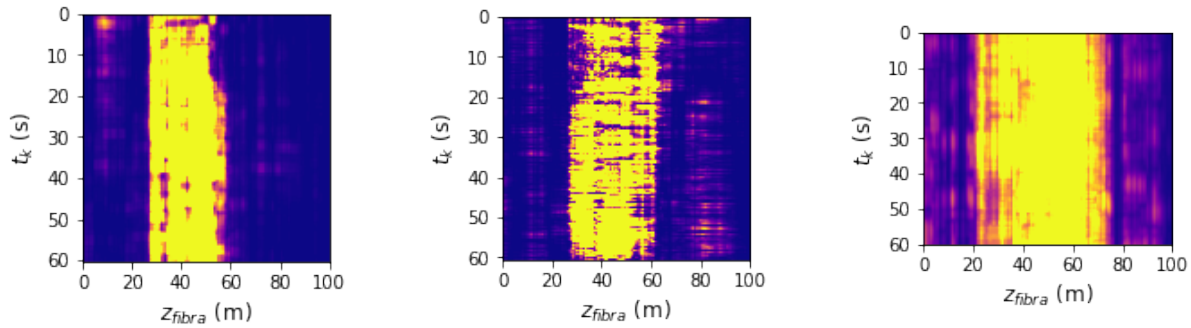
$$\vec{r}_{evento} \neq \vec{r}_{evento}(t) \tag{5.2.2}$$

Puede corresponder a varios tipos de eventos reales que pueden suceder en entornos de la fibra, como por ejemplo la apertura de una canaleta, forzar una verja, un taladro, etc.

Para simular de forma realista estos eventos, la frecuencia de los pulsos de presión es variada ligeramente, con la intención de reproducir golpes con una frecuencia algo indefinida.

La función genérica planteada necesita las distancias del foco a los emisores, y los tiempos de vuelo.

Implementando este evento en el modelo, obtenemos imágenes como las siguientes por los tres métodos:



(a) Varianza (b) Ajuste (c) Fourier

Figura 11: Ejemplos de imágenes de evento fijo situado en $z_{fibra} = 50$ con ruido de fondo bajo.

5.2.3. Evento móvil

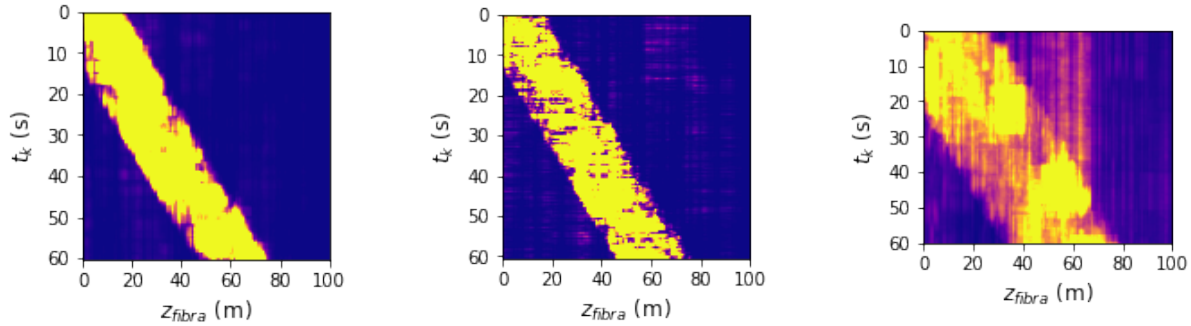
El evento móvil se caracteriza por que la posición del foco no es constante y depende del tiempo, es decir

$$\vec{r}_{evento} = \vec{r}_{evento}(t) \tag{5.2.3}$$

Los eventos móviles corresponden al desplazamiento de personas, vehículos, etc. en entornos de la fibra, a una velocidad determinada. Asumiremos desplazamientos paralelos a la fibra, dentro de un rango no superior a 3 metros de alejamiento transversal a la fibra, en la superficie, y también variaremos ligeramente la frecuencia de los pulsos para que la simulación sea más realista.

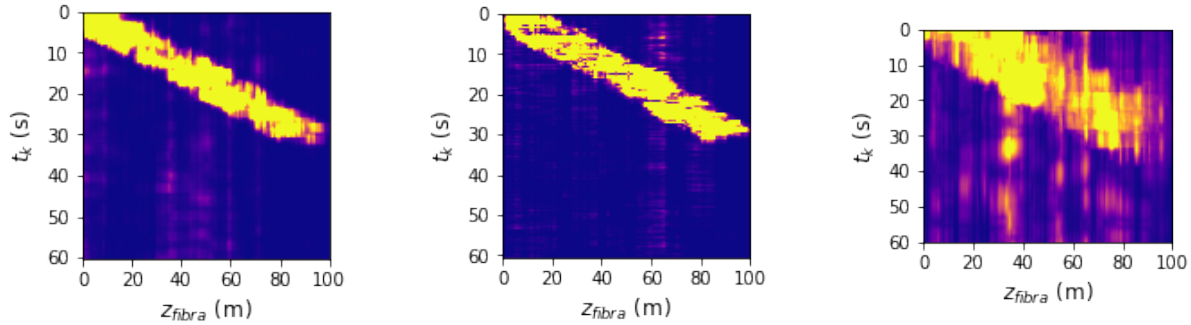
Dividiremos el evento móvil en dos clases de eventos: *andar* y *correr*. La única diferencia entre estos eventos será la velocidad, siendo evidente que $v_{andar} < v_{correr}$. Esto se traducirá en un cambio en la pendiente en las imágenes cuando representemos el tiempo largo frente a los puntos de la fibra.

Dicho cambio se puede apreciar en los siguientes ejemplos de imágenes obtenidas tras la implementación de los dos tipos de eventos en el modelo:



(a) Varianza (b) Ajuste (c) Fourier

Figura 12: Ejemplos de imágenes de evento móvil *andar* con $v = 1$ y ruido de fondo bajo.



(a) Varianza (b) Ajuste (c) Fourier

Figura 13: Ejemplos de imágenes de evento móvil *correr* con $v = 3$ y ruido de fondo bajo.

5.3. Implementación y parámetros

Dentro de la clase Fibra, creamos tres métodos distintos: uno es *impulsoEscalon*, y los otros dos son *añadeRuidoTermico* y *añadeDilatacionesLocales*.

Para generar los eventos debemos crear un método que cree la función paso P(t), de forma que devuelva una lista con el valor del pulso en cada momento temporal que se simula. El método creado se incluye dentro de la clase Fibra, y lo llamamos *impulsoEscalon*. Recibe como parámetros la lista de tiempos de simulación, tiempo de inicio y final de los pulsos, tiempo entre pasos y tiempos de subida y bajada de los pulsos. Además cambiaremos el tiempo entre pasos aleatoriamente dentro de un rango pequeño para aproximarnos más a la realidad.

Por otro lado, creamos otros dos métodos dentro de la clase Fibra que se encargarán de actualizar la lista de posiciones de los centros de dispersión z_m en todo momento, en función de los parámetros que definan a los eventos que estén teniendo lugar. Estos métodos reciben los nombres *añadeRuidoTermico* y *añadeDilatacionesLocales* (empleado para evento fijo y evento móvil). Además de actualizar la lista de posiciones de los emisores, actualizan la lista de tiempos de activación de los emisores, que es lo que recibe el método *ondaSuma* para calcular las interferencias.

Esto se traduce en el siguiente esquema:

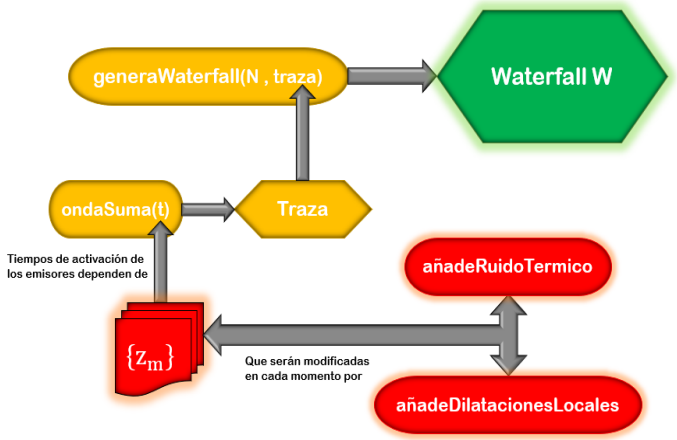


Figura 14: Esquema de implementación de la simulación de eventos en el sistema COTDR.

Para simular los eventos asumimos: una velocidad de propagación de las perturbaciones en el medio $v_{prop} = 1000\frac{m}{s}$. Además, asumimos que la fibra está contenida en un cubo de lado L, y que x_{fibra} es la única coordenada libre, porque asumimos que $y_{fibra} = \frac{L}{2}$ y que $z_{fibra} = 1m$. Por otro lado, el plano de la superficie es un plano XY, ubicado en $z_{plano} = 0$. Por tanto, la fibra está situada a 1 metro de la superficie en todo momento.

Además, en cada simulación situamos el evento en un punto aleatorio según una distribución uniforme, entre $x = 10$ y $x = 90$. Hacemos lo mismo con y, pero en este caso de forma que siempre esté, como máximo, a una distancia de 3 metros de la fibra en dirección transversal, es decir entre $y = 47$ e $y = 53$. Para evento móvil, si x es mayor que L/2, la velocidad paralela a la fibra en x será negativa, y si x es menor que L/2 la velocidad paralela a la fibra en x será positiva.

Para evento móvil, tratamos por separado andar y correr, de forma que:

- Andar: tiempo entre pasos 0.75s , tiempo subida 0.2s y tiempo pisada 0.5s, y una velocidad de 1 m/s
- Correr: tiempo entre pasos 0.5s, tiempo subida 0.15s y tiempo pisada 0.35s, y una velocidad de 3 m/s

El tiempo de comienzo del evento también se asigna aleatoriamente según distribución uniforme entre 0 y 10 segundos, y el de finalización entre 50 y 60 segundos.

La intensidad de ruido térmico también es un valor aleatorio según una distribución normal, entre $4 \cdot 10^{-8}$ y $8 \cdot 10^{-8}$, la amplitud de la onda de presión es $p_0 = 100000\ \frac{N}{m^2}$ y el coeficiente de absorción es $\alpha = 5$.

6. Redes neuronales convolucionales (CNN) de clasificación

6.1. Teoría de redes para clasificación de eventos

Abordaremos brevemente la teoría que subyace en las redes neuronales destinadas a la clasificación de eventos. Gran parte de la información aportada en esta sección de teoría del capítulo está contenida en el libro de Michael A. Nielsen [8].

6.1.1. Red neuronal: concepto

Comenzaremos definiendo lo que es una *neurona* en este contexto, pero antes introducimos un par de conceptos.

En redes de clasificación, dado un set de datos (ejemplo: traza interferométrica) almacenado en un vector llamado *input*, buscamos determinar si dicho set de datos tiene o no una cierta característica. Más concretamente, buscamos determinar la probabilidad de que tenga esa característica. Esta probabilidad se evalúa con lo que llamamos *función de activación*. Hay muchos ejemplos de funciones de activación, y es bastante común emplear la función *sigmoide*, que tiene la siguiente forma:

$$\sigma(a) = \frac{1}{1 + e^{(-a)}} \tag{6.1.1}$$

La función de activación no sólo recibe como parámetros el vector de *input*, sino también un conjunto de *pesos* $\vec{w}$, de la misma longitud que el vector *input*. Concretamente recibe

$$a = w^t \vec{x} + b = \sum_{d=1}^{D} w_d x_d + b \tag{6.1.2}$$

que determinará el tipo de característica que está buscando, y devuelve la probabilidad de que el set de datos de *input* tenga esa característica.

Una *neurona* está definida por ese conjunto de *pesos* $\vec{w}$, de la misma longitud que el vector *input*, y una función de activación, que en nuestro caso será $\sigma(a)$.

En otras palabras: en redes de clasificación, podemos decir que cada neurona evalúa la probabilidad de presencia de una característica en un conjunto de datos de entrada.

Tras pasar los datos por la neurona, arrojará la probabilidad de que el conjunto de datos tenga cierta característica, es decir:

$$P(Y = 1|\vec{x}) = \sigma(a) = \sigma(w^t \vec{x} + b) = \sigma(\sum_{d=1}^{D} w_d x_d + b) \tag{6.1.3}$$

donde $\vec{w}$ es el vector columna que contiene los pesos que caracterizan a la neurona, y D es la longitud del *input*.

En nuestro caso, el conjunto de datos de entrada será el *waterfall*, es decir la matriz W colapsada a un vector de longitud $D = F \times i_f$, con $F = K - N$. Este vector será el *input* $\vec{x}$.

Lo siguiente que definimos es una *capa*. Una capa es un conjunto de neuronas, y está definida por el número de neuronas que contiene. Imaginemos una capa compuesta por tres neuronas,

que recibe como *input* dos datos. Pues bien, el *output* de esta capa arroja un vector con tres datos.

Finalmente, una *red neuronal* es una sucesión de capas. Nosotros trabajamos con una red neuronal de tipo "feedforward", que quiere decir que dentro de la sucesión de capas, el *output* de una capa será el *input* de la siguiente, y así hasta atravesar toda la capa.

6.1.2. Clasificación multiclase

En nuestro problema, queremos ser capaces de distinguir cuatro tipos de eventos diferentes. En el ámbito de las redes neuronales, a cada tipo de evento lo llamaremos *clase*. Por lo tanto, la red neuronal debe ser capaz de distinguir un total de $K = 4$ clases. Esto nos obliga a construir una red neuronal de clasificación multiclase.

En una red neuronal de este tipo compuesta por L capas tenemos que a se convierte en un vector $\vec{a}^{(L)}$ de longitud K. Por tanto, necesitamos convertir este vector $\vec{a}^{(L)}$ en un conjunto de probabilidades para cada una de las K clases. Estas probabilidades deben cumplir dos condiciones:

$$P(y = k|\vec{x}) \geq 0 \ \forall k \tag{6.1.4}$$

$$\sum_{k=1}^{K} P(y = k|\vec{x}) = 1 \ \forall k \tag{6.1.5}$$

En redes de clasificación multiclase utilizamos otra función de activación en lugar de la *sigmoide* cuando $K > 2$, que es la función *softmax*:

$$P(y = k|\vec{x}) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}} \tag{6.1.6}$$

6.1.3. Red convolucional

Comenzamos introduciendo el concepto de convolución de matrices en el ámbito de las redes neuronales. La idea de las redes convolucionales es sustituir la multiplicación de matrices convencional, por la convolución, que resulta mucho más eficiente algunos lenguajes de programación (ej: Python).

La convolución de dos matrices A y W corresponde a realizar la siguiente operación:

$$(A * W)_{ij} = \sum_{i'=1}^{K} \sum_{j'=1}^{K} A(i + i', j + j')W(i', j') \tag{6.1.7}$$

donde las matrices A y W corresponden, por ejemplo, a la matriz *waterfall* W_{fi} y un filtro. ¿Qué queremos decir con filtro?

Un filtro es otra matriz, que busca la existencia de una cierta característica en la matriz *input*, y genera una nueva imagen. Por lo tanto, en redes convolucionales las neuronas son filtros, que contienen los pesos que las definen.

Por ejemplo, si aplicamos 10 filtros diferentes a una matriz *input A*, obtenemos 10 imágenes que constan de un valor para cada píxel de la imagen. Si aplicamos N filtros, obtenemos N imágenes. A esas N imágenes resultantes de aplicar N filtros las llamamos *mapas de características*. El número de mapas de características que hay coincide con el número de filtros aplicados, que llamaremos C_2. Inicialmente, podríamos decir que la matriz de *input A* es un mapa de características de dimensión $C_1 = 1$, porque a cada píxel le asocia un solo valor. A este tipo de matrices de *input* las llamamos *Input Image Gray Scale*. Los *waterfalls W* que usaremos para entrenar la red son de este tipo.

La red neuronal que implementamos consiste en una sucesión de capas de convolución, aplicando filtros de un tamaño constante (en nuestro caso, matrices de dimensión 9×9), en la que vamos aumentando el número de mapas de características en cada capa, yendo desde 32 hasta 128 en la última capa.

La sucesión de capas hace que vayamos perdiendo información espacial y ganando información en términos de qué características vamos encontrando.

6.2. Implementación de la red CNN

Para entrenar y validar el funcionamiento de la red CNN de clasificación, generaremos 1000 imágenes por cada uno de los tres métodos presentados. Por cada método, generaremos 250 imágenes de cada una de las cuatro clases de eventos a clasificar.

Dividimos las 1000 imágenes obtenidas por cada método en dos conjuntos: un conjunto de entrenamiento al que llamaremos *train set* compuesto por 670 imágenes, y otro conjunto de imágenes al que llamaremos *test set* compuesto por 330 imágenes. El primer conjunto se empleará para entrenar la red neuronal, y el segundo para evaluar el funcionamiento de la misma.

La red neuronal consistirá inicialmente en tres capas de convolución, llamadas *Conv2D*, con los siguientes parámetros:

- Primer parámetro: número de filtros que queremos que aplique a la imagen *input*. Por tanto, esto determina el número de mapas de características a la salida de la capa. Aumentamos este parámetro en cada capa, asignándole los valores 32, 64 y 128 respectivamente.
- Segundo parámetro: dimensión de los filtros. Lo mantendremos constante, aplicando filtros con forma de matriz cuadrada de dimensiones 9×9.
- Tercer parámetro: el parámetro *strides*. Es equivalente a hacer un *Pool* sobre los mapas de características de salida. Hacer un *Pool* consiste en reducir el tamaño de estas matrices de salida, seleccionando el valor máximo de los encontrados si dividimos el mapa en submatrices de dimensión $strides \times strides$. Fijamos el valor de este parámetro a 2 en todas las capas.
- Cuarto parámetro: la función de activación empleada, que en las capas de convolución será la función $ReLU$.

Entre capas de convolución aplicaremos una capa llamada *Dropout*, cuya función es igualar a 0 un porcentaje determinado de pesos, para evitar tendencias. El único parámetro que recibe esta capa es la fracción de pesos que queremos anular, y lo mantenemos en 0.2 en todas las capas.

Finalmente, aplicamos dos capas consecutivas de tipo *Dense* con una capa *Dropout* en medio. Estas capas hacen el mismo trabajo que las convolucionales pero sin emplear la convolución para la multiplicación de matrices. Esta capa busca valores de los *pesos* w para que la función de pérdida (o *loss function*) tenga el menor valor posible. Dicha función es un parámetro que debemos elegir a la hora de entrenar al modelo, y se especifica su elección más adelante.

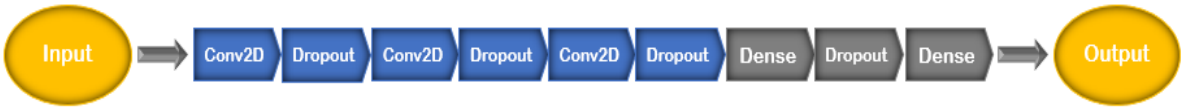


Figura 15: Esquema de implementación de la simulación de eventos en el sistema COTDR.

Finalmente, creamos el objeto *Model* a partir del API funcional de TensorFlow *Keras*, que agrupa las distintas capas obtenidas en un objeto capaz de entrenar a la red.

Una vez se ha creado el modelo, ya podemos entrenarlo con el método *fit*. Para el proceso de entrenamiento, debemos establecer ciertos parámetros en el método *compile*, que son:

- Optimizer: es el algoritmo numérico empleado para minimizar la función de pérdida. Emplearemos *adam*.
- Loss: función de pérdida que queremos emplear. A grandes rasgos, dicha función tiene un valor pequeño si muchas predicciones son correctas, y un valor grande si muchas predicciones son incorrectas. Emplearemos *sparse categorical crossentropy*.
- Metrics: métrica para evaluar el éxito de las predicciones. Emplearemos *accuracy*, que significa *precisión*, y es simplemente el cociente entre las predicciones acertadas sobre las totales.

Los parámetros del método *fit* son:

- Imágenes de entrenamiento
- Imágenes de test
- Epochs: es el número de veces que ejecutamos algoritmo numérico para minimizar la función de pérdida. También lo llamaremos *iteraciones*, y le damos un valor de 12 en los tres modelos (uno por cada método de cálculo del *waterfall*, tras comprobar que es el que mejores resultados da).

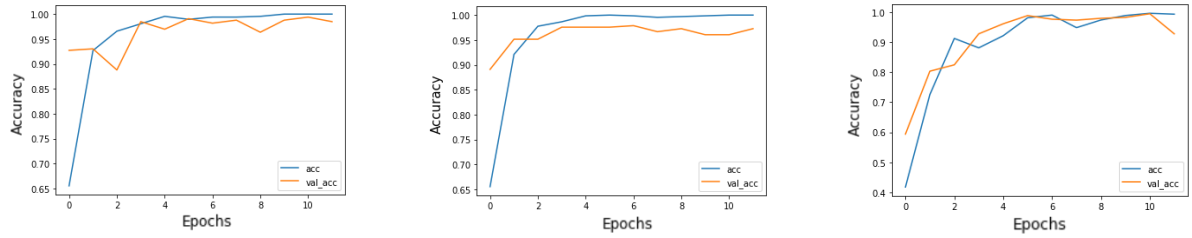
6.3. Resultados numéricos

Al entrenar la red, vamos computando el valor de la función de pérdida y la precisión en cada *epoch*. La representación de estos valores frente a las iteraciones, tanto para el conjunto de entrenamiento como para el conjunto de test, nos da una idea de cómo evoluciona el aprendizaje de la red.

Inicialmente presentamos dos gráficas representativas de la evolución y los resultados del entrenamiento de la red neuronal: evolución de la precisión y del valor de la función de pérdida frente al número de iteraciones.

En ambos casos se representan en la misma gráfica la evolución para los dos conjuntos: el conjunto de imágenes de entrenamiento (línea azul) y el conjunto de imágenes de validación (línea naranja).

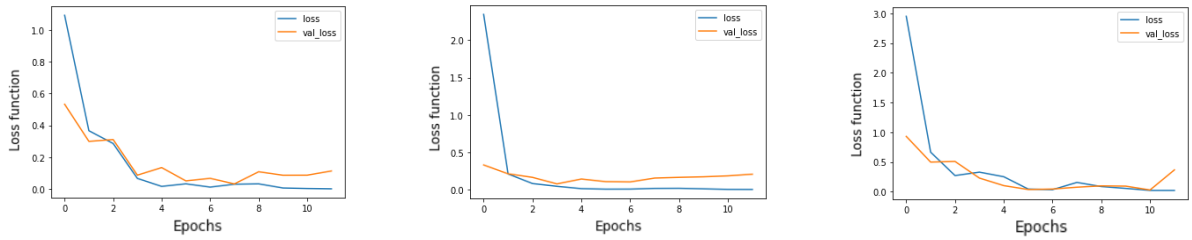
En primer lugar, adjuntamos las evoluciones de la precisión para cada uno de los métodos:



(a) Varianza (b) Ajuste (c) Fourier

Figura 16: Evolución de *accuracy* para los tres métodos.

En segundo lugar, adjuntamos las evoluciones de la función de pérdida para cada uno de los métodos:



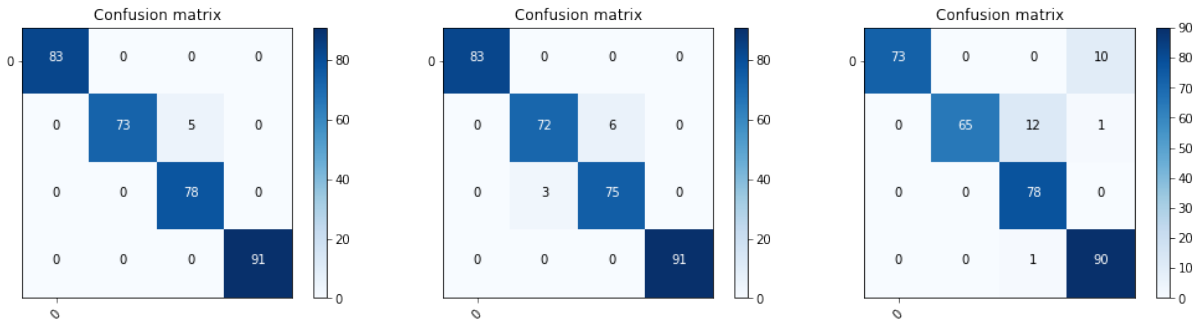
(a) Varianza (b) Ajuste (c) Fourier

Figura 17: Evolución de *función de pérdida* para los tres métodos.

Con estas gráficas nos cercioramos del correcto entrenamiento de la red. Ambas tendencias son las esperadas: la precisión en las predicciones aumenta mientras que la función de pérdida disminuye su valor, en cada iteración. También es coherente que, en el caso de la precisión, el conjunto de validación vaya por debajo del de entrenamiento, y en el caso de la función de pérdida vaya por encima. Esto se debe a que, al ser un conjunto de menos imágenes, obtiene reultados ligeramente peores.

Por último, el resultado más ilustrativo fruto del proceso de entrenamiento de la red, es la construcción de la matriz de confusión. La matriz de confusión es una matriz cuadrada de

lado $K = 4$, es decir, el número de clases a clasificar. Las columnas de la matriz representan el número de predicciones de cada clase, y las filas corresponden a las imágenes que realmente corresponden a cada clase. Dicha matriz permite visualizar qué clases confunde el sistema, y se construye con el conjunto de validación. Los resultados obtenidos con los parámetros y procedimientos citados a lo largo de todo el trabajo son:



(a) Varianza (b) Ajuste (c) Fourier

Figura 18: Matrices de confusión para los tres métodos.

La información que extraemos de estas matrices es que en general, la clasificación es buena para todas las clases por los tres métodos. Sin embargo, nos permite detectar dónde el sistema encuentra problemas, y es evidente que el sistema tiene cierta tendencia a confundir los eventos de tipo fijo con el evento móvil a poca velocidad, que corresponde con andar. Ádemás, también extraemos que el método de Fourier es el menos preciso de los tres, lo cual no nos sorprende pues, como hemos visto en el apartado anterior, genera imágenes más confusas que los otros dos métodos.

7. Conclusiones

En este trabajo hemos planteado dos modelos diferentes en el lenguaje de programación *Python*: un modelo de simulación de imágenes ante la presencia de eventos en un sistema COTDR (Modelo 1), y un modelo de aprendizaje automático basado en redes neuronales convolucionales de clasificación multiclase (Modelo 2). Los *scripts* correspondientes a ambos modelos se incluyen como Anexo del trabajo, y se pueden consultar accediendo al enlace incluido en la bilbiografía que conduce a un repositorio en GitHub [4].

Por un lado, con el primer modelo para generación de imágenes sintéticas hemos demostrado que las aproximaciones y consideraciones tomadas en cuenta para la construcción del mismo, han derivado en la producción de imágenes muy parecidas a las reales. Hemos planteado un modelo que se puede adaptar todo tipo de parámetros para simular imágenes con un amplio abanico características. La calidad de las imágenes obtenidas por simulación es buena, y en caso de tener acceso a imágenes reales, se podrían simular imágenes en base a estas.

Por otro lado, también hemos mostrado que el segundo modelo planteado, el de aprendizaje automático, es capaz de aprender a clasificar los cuatro tipos de eventos por los tres métodos propuestos, a partir del modelo planteado del sistema real. Esto significa que el modelo está preparado para ser puesto a prueba con imágenes reales de características parecidas.

De los tres métodos propuestos para el tratamiento de la matriz de interferencia, el que mejores resultados ha provisto es el de la varianza, y el que peores resultados ha provisto es el de análisis espectral por transformada de Fourier. Sin embargo, los tres métodos proveeen resultados aceptables, y la calidad de los mismos puede cambiar en función tanto de los parámetros del sistema, como de los parámetros de los métodos en sí.

Como trabajo futuro, quedan abiertas varias posibilidades de investigación en esta línea, como por ejemplo: implementación de redes neuronales recurrentes en vez de convolucionales, introducción de pulsos con distintas formas (por ejemplo, gaussiana) o ajustes polinómicos de orden mayor que uno en el caso del método del ajuste funcional.

Otra posibilidad interesante es la de extraer datos sobre el evento a partir de la imagen. Como ya vimos en el caso de eventos móviles, podemos hacer una estimación de la velocidad a la que está teniendo lugar la perturbación en la superficie. Se podría explorar la capacidad de estimación de otros parámetros del evento, como puede ser por ejemplo la magintud del evento en términos de amplitud de presión, para estimar la fuerza que ejerce el mismo sobre la superficie.

8. Bibliografía

Referencias

[1] John M. Senior and M. Yousif Jamro. *Optical Fiber Communications Principles and Practice*. Pearson Education, 2009.

[2] Leonid B. Liokumovich, Nikolai A. Ushakov, Oleg I. Kotov, Mikhail A. Bisyarin, and Arthur H. Hartog. Fundamentals of optical fiber sensing schemes based on coherent optical time domain reflectometry: Signal model under static fiber conditions. *Journal of Lightwave Technology*, 33(17):3660–3671, 2015.

[3] Ali Masoudi and Trevor P. Newson. Analysis of distributed optical fibre acoustic sensors through numerical modelling. *Opt. Express*, 25(25):32021–32040, Dec 2017.

[4] Manuel Romeo. https://github.com/manuelromeo/tfg-manuel-romeo.

[5] NumPy. Numpy v1.21 manual, 2021.

[6] SciPy. Scipy v1.7.1 manual, 2021.

[7] Bruce Carter. In Ron Mancini and Bruce Carter, editors, *Op Amps for Everyone (Third Edition)*, pages 163–188. Newnes, Boston, 2009.

[8] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.