UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

IDENTIFICACIÓN DE IMPACTOS EN EL FUSELAJE DE UN AVIÓN UTILIZANDO ALGORITMOS DE APRENDIZAJE DE MÁQUINAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

MATHEUS AUGUSTO PINTO ARRATIA

PROFESORA GUÍA:
VIVIANA MERUANE NARANJO

MIEMBROS DE LA COMISIÓN:
ENRIQUE LÓPEZ DROGUETT
ALEJANDRO ORTIZ BERNARDIN

Este trabajo ha sido parcialmente financiado por FONDECYT:$N°$ 1170535 'Damage assessment in composite sandwich structures based on full-field vibration measurements and automatic image analysis'

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: MATHEUS AUGUSTO PINTO ARRATIA
FECHA: 2019
PROF. GUÍA: VIVIANA MERUANE NARANJO

IDENTIFICACIÓN DE IMPACTOS EN EL FUSELAJE DE UN AVIÓN UTILIZANDO ALGORITMOS DE APRENDIZAJE DE MÁQUINAS

Pese a que la probabilidad de morir en un accidente aéreo es bastante baja (del orden del 0.00053 %), cuando estos accidentes ocurren son catastróficos. Con el fin de prevenir accidentes, los sistemas de monitoreo estructural en tiempo real, que son capaces de identificar automáticamente impactos a medida que ocurren, se han vuelto más atractivos y necesarios para garantizar la seguridad y prevenir accidentes fatales en estructuras aéreas.

Una de las formas de monitoreo estructural, es el análisis de la respuesta vibratoria frente a fuerzas de tipo impacto. En estos algoritmos, la estructura es capaz de "aprender" de los impactos que recibe para reconocer eficientemente los impactos futuros. Así pudiendo detectar, localizar y cuantificar impactos a partir de su respuesta vibratoria.

Para este estudio, la estructura analizada es el fuselaje de un avión, para el cual se realizaron las mediciones, en uno de los aviones comerciales de la empresa SKY Airlines.

La contribución principal de este trabajo es la implementación de siete algoritmos construidos utilizando dos estrategias de identificación de impactos. La primera estrategia consiste en una técnica de reducción de dimensionalidad (aprendizaje no supervisado), aplicada en conjunto a un algoritmo de aprendizaje supervisado, llamado aproximación lineal con máxima entropía (LME). La segunda estrategia que consiste en un modelo de red neuronal convolucional (CNN), aplicado al problema de identificación de impactos. Seis algoritmos fueron elaborados con la primera estrategia y uno con la segunda.

Las técnicas de reducción de dimensionalidad ocupadas fueron: El análisis de componentes principales (PCA), análisis de componentes principales a través de kernels (kernel PCA), escalamiento multidimensional (MDS), mapeo isométrico de características (ISOMAP), análisis de componentes principales probabilístico (PPCA) y autoencoders (AE).

Los algoritmos que obtuvieron mejores resultados fueron, AE+LME y las redes neuronales convolucionales (CNN). El algoritmo AE+LME se caracterizó por ser muy preciso en predecir la localización del impacto, en cambio la CNN fue más precisa en la cuantificación de la fuerza de impacto.

Finalmente, se puede afirmar que los algoritmos desarrollados, cumplen con la función de identificar la ubicación y magnitud de los impactos efectuados en el fuselaje, sin embargo cuatro de estos lo hacen mejor, habilitando su posible aplicación en estructuras aéreas.

Especialmente a mi mamá.

Agradecimientos

Uff durísimo, han sido hartos años de educación, de los cuales me gustaría agradecer a cada una de las personas que formaron parte de mi proceso educativo, tanto como persona y como estudiante.

Por orden cronológico:

Primero que todo agradecer a mis padres, Mamá, Papá, muchas gracias por todo lo que hicieron por mi, todo lo que sacrificaron, todo lo que se esforzaron, por los valores enseñados, solo palabras de agradecimientos para ustedes, no debe haber sido fácil soy el primero y el último, así que, el ensayo y error, no funciono conmigo. Muchas gracias y los amo!.

A mi familia en general, especialmente (espero no se escape ninguno) tía Silvia, tía Flor, tío Roberto, tío Manolo, tía Sole, tío Vicente, tía Malvi, tía Nancy, tía Ana, mis primas Bianca, Vivi, Dani, Valeria y Lore, a la Patita, a mi abuelito Manuel y tío Sergio (a quienes no puede agradecerles en vida) a ustedes, quienes junto a mis padres han sido las personas más importantes en mi formación como persona, y me han apoyado mucho en incontables ocasiones, Muchas gracias!.

Por donde parte el estudiante, el colegio, agradecer mis profesores y compañeros, a todos aquellos que con una conversación de me entregaron una reflexión que me ayudo a ser la persona que soy hoy.

A mis amigos de plan común con los que pase unos buenos primeros años, gracias por las risas y payasadas juntos, ojalá nunca falten. Agradecer también que han sido los mismos desde el día uno de la universidad, nunca han cambiado y siempre manteniendo esa buena onda y ganas de echar la talla, son bacanes.

A “les cons” que fueron un soporte fundamental en esos dos años y medio, si no los hubiese conocido a ustedes, probablemente no hubiese aguantado ni 6 meses, fueron mi familia en un lugar tan lejos, haberlos conocido me hizo crecer mucho como persona y para nunca olvidar los viajes, años nuevos, navidades, cumpleaños,fiestas, desveladas de estudio, caleta juntos, muchas gracias por todo.

A mis amigos de mecánica que sin conocerme mucho me recibieron como si fuera uno más de ellos, y los que me hicieron la raja mi estadía en el dpto de mecánica, son gente bacán, muy buenas personas, no cambien nunca (pd: compadtan la pelota).

A la Fer que fue muy importante en estos meses, me apoyaste en todo, siempre con alguna palabra que te saca una sonrisa y te da energías, fuiste un soporte muy grande para este logró. Eres una polola muy bacán (corazón*3).

Y bueno a mis profesores, las personas más importantes para realizar este trabajo. A la profe Vivi, muchas gracias por aceptarme como su memorista, por ayudarme en todo lo que necesitara y guiarme por los caminos correctos para realizar la memoria, muchas gracias, la admiro mucho, profesionalmente y como persona. Al profesor Enrique López que me ayudó en momentos claves a guiar mi trabajo en una buena dirección, al profesor Alejandro Ortiz que siempre se mostró disponible para resolver cualquier duda que tuviese, también quería agradecer al profe Ruben por su buena onda, sus tallas y ganas de discutir con nosotros en la salita del quinto y que muy amablemente me ayudo a resolver unas dudas de la memoria. Muchas gracias!

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Antecedentes Generales

1.1.1. Identificación de impactos

Las estructuras aéreas están continuamente expuestas a daños superficiales, ya sea durante el vuelo, despegue, aterrizaje, abordaje de pasajeros, durante el mantenimiento, etc. Dentro de este marco, el monitoreo de la integridad estructural ha logrado un gran interés [3]. Detectar, localizar y cuantificar el daño en estructuras de la forma más rápida posible, tiene una gran importancia en las estructuras aeronáuticas. La información oportuna del daño que presentan los diferentes componentes de la estructura, ayuda a incrementar la seguridad en su uso y mejorar su confiabilidad, además de reducir significativamente los costos asociados al mantenimiento, obteniendo mayores beneficios económicos [4].

El principio de un sistema de identificación de impactos es detectar, localizar y cuantificar una fuerza de impacto, con el uso de un sistema pasivo de sensores, los cuales se encuentran distribuidos sobre la superficie de la estructura. De esta manera se detecta en tiempo real, los daños producidos sobre la estructura [3].

En la figura 1.1 se ilustra un montaje experimental típico de detección de impactos, el cual utiliza sensores acelerómetros que detectan las ondas de esfuerzos que se propagan por la superficie. Estas ondas son generadas por una fuerza de impacto, producida con un martillo modal el cual golpea la estructura.

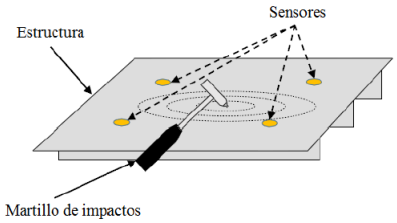

Figura 1.1: Representación de un montaje experimental típico de detección de impactos

1.1.2. Análisis de vibración aplicado a la detección de impactos

El estudio mediante el análisis vibratorio de impactos, se basa en dos principios fundamentales, el primero, corresponde a que cada estructura tiene su forma particular de vibrar, que es única a cada estructura, y el segundo es que al someter a una estructura a un estimulo dado, esta presenta una respuesta vibratoria, que es característica del estimulo y la estructura que se esta analizando.

Esta particularidad permite realizar el proceso inverso, es decir, al conocer cómo cambia la respuesta vibratoria de una estructura, se puede conocer el estimulo que produjo dicho cambio. Conociendo la respuesta vibratoria típica de un sistema, se pueden utilizar algoritmos que permitan determinar características de la excitación inicial, que provocó una determinada respuesta. Para el caso de excitaciones causadas por un impacto, se pueden estudiar modelos que responden a excitaciones de tipo impulso [3].

La figura 1.2 muestra las etapas en los métodos de detección de impacto.

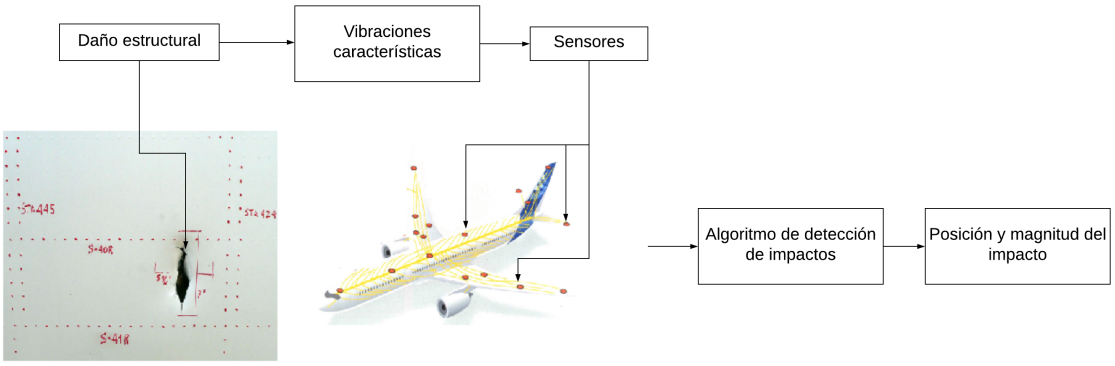

Figura 1.2: Diagrama de detección de impactos mediante vibraciones

1.1.3. Algoritmos utilizados para la detección de impactos

Hoy en día existen distintos algoritmos que pueden ser utilizados para determinar la localización y magnitud de impactos realizados sobre una estructura. Actualmente, se han desarrollado estudios de impactos utilizando algoritmos basados en Redes Neuronales Artificiales (ANN, por sus siglas en inglés), los cuales han obtenido buenos resultados, tanto en estructuras simples como compuestas. Las ANN se destacan por procesar datos de manera rápida, sin embargo, necesita un alto tiempo de aprendizaje y para su aplicación se deben determinar un gran número de parámetros [3].

Otro algoritmo comúnmente utilizado es el de Máquinas de Vectores de Soporte (SVM, por sus siglas en inglés), el cual presenta una ventaja en cuanto a optimización global y mejores capacidades de generalización que las ANN. También se ha desarrollado el algoritmo de Mínimos Cuadrados de Maquinas de Vectores de Soporte (LSSVM, por sus siglas en inglés) que simplifica la regresión a un problema que puede ser resuelto con un grupo de ecuaciones lineales [3].

Un algoritmo que ha obtenido muy buenos resultados consiste en un método de aproximación lineal, el cual está basado en el Principio de Máxima Entropía (PME), y se denota como Aproximación Lineal con Máxima Entropía (LME, por sus siglas en inglés). Este algoritmo se destaca en comparación con otros algoritmos, ya que no necesita de entrenamiento previo [3], además que sólo se necesita determinar un parámetro para su implementación [4]. Además se ha observado que este algoritmo mejora su rendimiento si es usado en conjunto a una técnica de reducción de dimensionalidad como etapa de pre-procesamiento.

Las técnicas de reducción de dimensionalidad, crean una representación de los datos originales, en una dimensión más pequeña a la inicial, estas técnicas, seleccionan los patrones o características más significativas de los datos en alta dimensión.

Métodos de reducción de dimensionalidad como el análisis de componentes principales (PCA) y autoencoders (AE), junto al algoritmo LME, son los que han dado mejores resultados hasta la fecha. Es por esto, que se ha avanzado en el estudio de diferentes técnicas de reducción de dimensionalidad de datos, con el fin de mejorar el algoritmo de detección de impactos.

Por esta razón este trabajo de titulo tiene como finalidad ampliar el estudio de las técnicas de reducción de dimensionalidad, incorporando a la vez los dos métodos PCA y AE, además probando su eficiencia en un caso real.

Además, este trabajo plantea la utilización de una nueva estrategia de identificación de impactos, que utiliza el modelo de una red neuronal convolucional, llamadas CNN por sus siglas en ingles. Las CNN son frecuentemente utilizadas en procesamiento de imágenes, y últimamente se han desarrollado varios enfoques basados en la computación visual en la comunidad de mantenibilidad y confiabilidad para detectar daños de a partir de imágenes [5].

Las CNN's tienen la peculiaridad que extraen características de los datos a través de capas de convolución, estás características son cada vez más abstractas a medida que se aumenta la cantidad (profundidad) de estás capas [5], la información extraídas por este proceso de convolución, son procesadas por una red neuronal ANN "fully connected". La idea de esta nueva estrategia, es adaptar el funcionamiento de las CNN, al problema de identificación de impactos.

1.2. Motivación

Este trabajo se centra principalmente en las fallas producidas por impactos a baja velocidad que pueden causar una cantidad significativa de daño, estas fallas son complejas de detectar ya que generalmente la única indicación de daño es una muesca superficial muy pequeña. Este tipo de daño a menudo se conoce como daño por impacto apenas visible (BVID por sus siglas en ingles), y puede causar una degradación significativa de las propiedades estructurales del fuselaje. Si el laminado dañado está sujeto a una alta carga de compresión, puede producirse una falla por pandeo. En la actualidad se han detectado casos en los que ha ocurrido un desprendimiento del fuselaje mientras el avión se encontraba en vuelo, por causa de este tipo de daño [3].

Por lo tanto, existe la necesidad de desarrollar medios mejorados y más eficientes para detectar tales daños. Actualmente los métodos más usados en la detección de estas fallas son, el uso del ultrasonido y la termografía activa, técnicas que son bastante lentas a implementarse en estructuras de gran superficie, y llegan a ser bastante costosas debido la misma razón, además que están sujetas a errores humanos en el procedimiento [6].

Este trabajo se centra en crear, un sistema de detección de impactos que sea mucho más robusto que las tecnologías empleadas actualmente. Un sistema de detección de impactos que permita localizar puntos que posiblemente hayan sufrido daño, reduciendo el costo y tiempo de las inspecciones. Para aumentar el desempeño de los sistemas de detección de impactos es importante mejorar las estimaciones de localización y magnitud de dichos eventos y acercar este tipo de estudio a un caso real, como lo es el fuselaje de un avión comercial.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo principal de este trabajo es poder detectar,localizar y cuantificar impactos a partir de las mediciones realizadas en el fuselaje de un avión real, utilizando algoritmos que utilizan dos estrategias de identificación de impactos.

1.3.2. Objetivos Específicos

Para poder lograr el objetivo general, se debe deben desarrollar los siguientes objetivos específicos:

1. Diseño de la toma de mediciones experimental.
2. Desarrollar una metodología particular a cada algoritmo, que entregue las mejores estimaciones de posición y magnitud de los impactos, utilizando dos estrategias de identificación de impacto:
   - Reducción de dimensionalidad más aproximación lineal con máxima entropía. Se ocupan junto al algoritmo LME, las siguientes técnicas de reducción de dimensionalidad:
     - Análisis de componentes principales (PCA).
     - Autoencoders (AE).
     - Análisis de componentes principales a través de kernel (Kernel PCA).
     - Mapeo isométrico de características (ISOMAP).
     - Escalamiento multidimensional(MDS).
     - Análisis de componentes principales probabilístico (PPCA)
   - Modelo de una red neuronal convolucional.
3. Evaluación de las metodologías de trabajo en un conjunto de datos de prueba, de los cuales se conoce la magnitud y la posición del impacto.

1.4. Alcances

En este trabajo, se construirán siete algoritmos de identificación de impactos, evaluando el rendimiento de cada uno de ellos en un conjunto de datos prueba.

Lo anterior permitirá realizar un análisis comparativo de los rendimientos de estos algoritmos, de manera de determinar, el o los algoritmos y sus respectivas metodologías que mejor se adapten al problema de identificación de impactos, para la aplicación en fuselajes de aviones.

Capítulo 2

Antecedentes Específicos

En este capitulo se detallan todos los antecedentes teóricos que son necesarios como base para desarrollar los algoritmos.

2.1. Respuesta vibratoria a un impulso

Gran parte de este trabajo, consistirá en el análisis de la respuesta vibratoria a la aplicación de una fuerza de corta duración sobre el fuselaje. Una excitación tipo impulso es una fuerza de impacto, aplicada por un periodo muy breve de tiempo. La respuesta de un sistema a un impulso, la cual se puede describir de acuerdo con la siguiente relación.

$$F(t) = \begin{cases} 0 & t \leq \tau - \varepsilon \\ \frac{\hat{F}}{2\varepsilon} & \tau - \varepsilon \leq t \leq \tau + \varepsilon \\ 0 & t \geq \tau + \varepsilon \end{cases} \tag{2.1}$$

Donde τ corresponde al tiempo en el que se aplica el impulso, la siguiente gráfica muestra el comportamiento de un impulso en el tiempo[7].

Esta representación, también se puede escribir numéricamente de la forma:

$$I(\varepsilon) = \int_{-\infty}^{\infty} F(t)\, dt = \frac{\hat{F}}{2\varepsilon} 2\varepsilon = \hat{F} \tag{2.2}$$

Dado que $\varepsilon \longrightarrow 0$, y mientras no alcance el valor de 0 lo anterior se puede considerar como $I(\varepsilon) = \hat{F}$.

Si la magnitud de $\hat{F}$ es 1 este se puede denominar como impulso unitario $\delta(t)$, también conocido como la función delta de dirac.

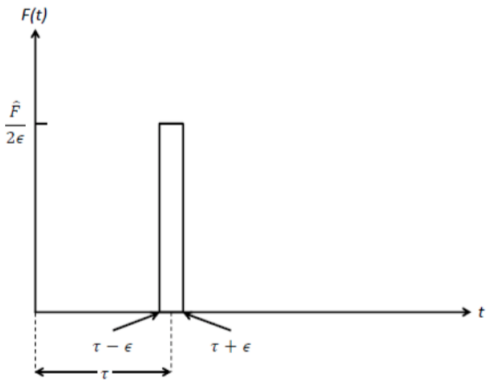

Figura 2.1: Representación de un impulso en el tiempo

La respuesta para un sistema de un grado de libertad, inicialmente en reposo, se puede determinar aprovechando el hecho que un impulso produce un cambio de momentum al cuerpo. Para hacer más simple el análisis consideramos $\tau = 0$, en la definición de impulso. Este instante lo denotaremos por 0^-. Dado que el sistema se encuentra inicialmente en reposo, las condiciones iniciales son ambas cero.

$$\dot{x}\left(0^+\right) = \dot{x}\left(0^-\right) = 0 \tag{2.3}$$

Por lo tanto luego del golpe, el cambio de momentum es:

$$m\dot{x}\left(0^+\right) - m\dot{x}\left(0^-\right) = \quad mv_0 \tag{2.4}$$

Donde :
$\dot{x}\left(0^-\right)$: Velocidad en el instante inmediatamente anterior al impacto. $\dot{x}\left(0^+\right)$: Velocidad en el instante inmediatemente posterior al impacto.

v_0: Velocidad incial del sistema.

De manera que :

$$\hat{F} = \quad F\Delta t \quad = \quad mv_0 - 0 = \quad mv_0 \tag{2.5}$$

Mientras que el desplazamiento inicial se mantiene en cero. Por lo tanto un impulso aplicado a un sistema con un solo grado de libertad es lo mismo que aplicar un condición de borde de $v_0 = \frac{\hat{F}}{m}$.

Para un sistema con amortiguamiento débil $(0 < \xi < 1)$, la respuesta en condiciones iniciales $x_0 = 0, v_0 = \frac{\hat{F}}{m}$ es de la forma:

$$x\left(t\right) = \frac{\hat{F}e^{-\xi\omega_n t}}{m\omega_d} sin\left(\omega_d t\right) \tag{2.6}$$

Es conveniente escribir la ecuación anterior como:

$$x(t) = \hat{F} h(t) \tag{2.7}$$

En la cual $h(t)$ viene dada por :

$$h(t) = \frac{1}{m\omega_d} e^{-\xi\omega_n t} sen(\omega_d t) \tag{2.8}$$

Hay que notar que $h(t)$ es la respuesta en $t = 0$. Si esta respuesta se aplica en un tiempo distinto de cero, esta se escribe de la forma:

$$h(t-\tau) = \frac{1}{m\omega_d} e^{-\xi\omega_n (t-\tau)} sen(\omega_d (t-\tau)) \tag{2.9}$$

Las funciones $h(t-\tau)$ y $h(t)$ se denominan funciones respuesta del impulso.

En la práctica, una fuerza es considerada un impulso si su duración, Δt es muy breve en comparación al periodo $T = \frac{2\pi}{\omega_n}$, asociado a la frecuencia natural del sistema, en este caso la estructura del fuselaje.

En la figura 2.2 se ilustra una respuesta típica a un impulso en un grado de libertad:

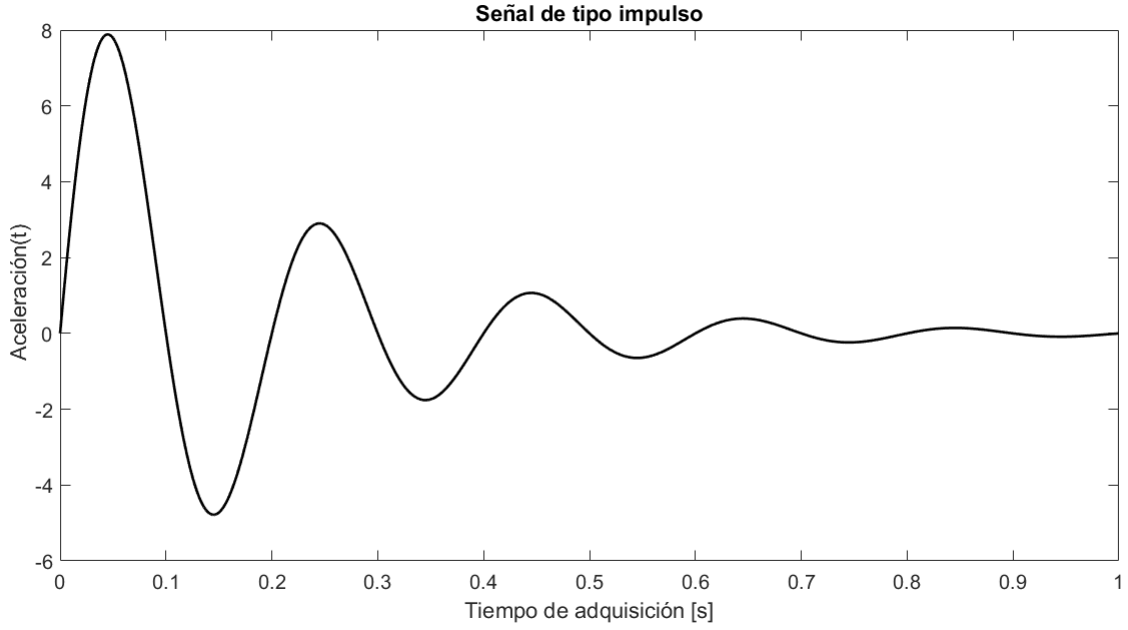

Figura 2.2: Representación de un impulso

2.2. Envolvente de una señal

Según la literatura [3][4], el uso de la envolvente mejora el rendimiento de los algoritmos de detección de impactos, dando buenos resultados en el algoritmo PCA+LME y LME [4]. En estos trabajos se ha probado que el uso de la envolvente, permite identificar de mejor manera las características de una señal impulsiva, por ejemplo, máximos de la señal e identificar el tiempo de desfase entre señales. Es por este motivo que en este trabajo se aplica la envolvente a las señales de tipo impulso detectadas por los sensores.

Para calcular la envolvente de una señal, la transformada de Hilbert debe ser calculada previamente. La transformada de Hilbert de una función real s(t), se calcula de la convolución de las señales s(t) y la función $1/\pi t$ de donde se obtiene la función $\widehat{s}\,(t)$. La transformada de Hilbert se define como muestra la 2.10:

$$\widehat{s}\,(t) = s\,(t) * \frac{1}{\pi t} = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{s\,(t)}{t-\tau} d\tau \tag{2.10}$$

Donde:
$\widehat{s}\,(t)$: Es la transformada de Hilbert de la señal
s(t): Señal original con respecto al tiempo.

Luego, la señal analítica se construye con la señal original como la parte real y la transformada de Hilbert como la parte compleja, como se muestra en la ecuación 2.11

$$a\,(t) = s\,(t) + i \cdot \widehat{s}(t) \tag{2.11}$$

Donde:
a(t): Señal analítica de s(t).
i: Unidad imaginaría $\sqrt{-1}$

Con esto se calcula la envolvente de la señal, como se muestra en 2.12. Esta función entrega el valor de los extremos de la amplitud de la señal en función del tiempo.

$$e\,(t) = \sqrt{s\,(t)^2 + \widehat{s}\,(t)^2} \tag{2.12}$$

Donde:
e(t): Envolvente de la señal s(t).

La figura 2.3 muestra una señal de tipo impulso y su envolvente por arriba.

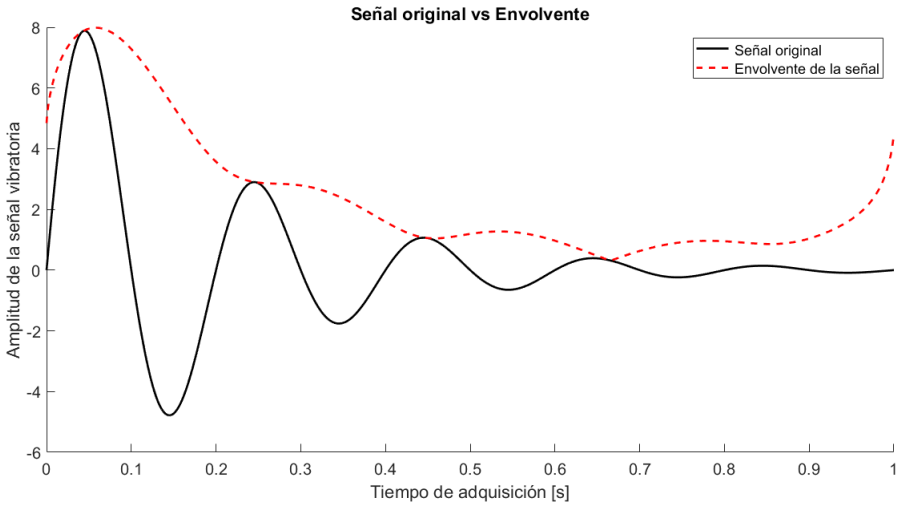

Figura 2.3: Envolvente de una señal de tipo impulso

2.3. Técnicas de reducción de dimensionalidad

2.3.1. Análisis de componentes principales (PCA)

El análisis de componentes principales (PCA), es una técnica de análisis de datos, utilizada para la reducción de dimensionalidad. El método radica en la selección de características y patrones más representativos de un conjunto de datos, con el fin simplificar el problema. Así describiendo el mismo problema en una menor cantidad de datos, los cuales describen un porcentaje de la varianza de los datos originales. Esta técnica es comúnmente utilizada en computación en la etapa de pre-procesamiento de los datos [4].

La principal arista del método PCA consiste en una transformación ortogonal que convierte los datos posiblemente relacionados, en un conjunto de datos no correlacionados entre sí, llamados componentes principales. Los componentes se ordenan en torno a la cantidad de varianza que describen, seleccionando aquellos que describen la mayor cantidad en un nuevo sistema de coordenadas [1].

Existen dos formas de aplicar PCA, la primera es a partir de la matriz de correlación de los datos, está forma es comúnmente utilizada para datos que nos son dimensionalmente homogéneos y el orden de magnitud de variables aleatorias medido no es el mismo. La otra forma de aplicar PCA, la cual es utilizada en este trabajo es a partir de la matriz de covarianza, que se usa cuando los datos son dimensionalmente homgeneos y las variables aleatorias de estos se encuentran en el mismo orden de magnitud [1].

El algoritmo de selección de componentes principales se describe matemáticamente de la siguiente manera:

Se tiene una matriz X de datos, de la cual se extraen una serie de componentes principales a través de la matriz de covarianzas, que se calcula de la siguiente forma:

$$R = \frac{X \cdot X'}{1 - n} \tag{2.13}$$

Luego se resuelve la ecuación:

$$Rq = \lambda q \tag{2.14}$$

Donde q corresponde a los vectores propios de R y λ a los valores propios de R asociado al respectivo vector propio. Se denota de la forma λ_j, para j=1,2,...m los valores propios asociados al respectivo vector propio q_j. De esta forma la transformación ortogonal T= [q_1,q_2,q_3,..., q_m] se utiliza para calcular la proyección A de la matriz X en la dirección principal de la siguiente forma [8]:

$$A = \quad (XT)^T \tag{2.15}$$

$$A = [X^T q_1, X^T q_2, X^T q_3, ..., X^T q_m] = [a_1, a_2, a_3, ..., a_m] \tag{2.16}$$

Donde los componentes a_j representan la proyección de X en los vectores q_j . En resumen, el método de PCA consta en los siguientes pasos. A partir de una matriz de datos X se obtiene una transformación ortogonal T a partir de la matriz de covarianzas. La transformación es utilizada para calcular una matriz A que es la proyección de los datos en un nuevo sistema de coordenadas. Esta proyección es producida en orden descendiente de la varianza en las direcciones principales, de esta forma eligiendo se reduce la dimensión de los datos [8]. En la figura 2.4 se puede ver la proyección de las componentes principales de un conjunto de datos bi-dimensional.

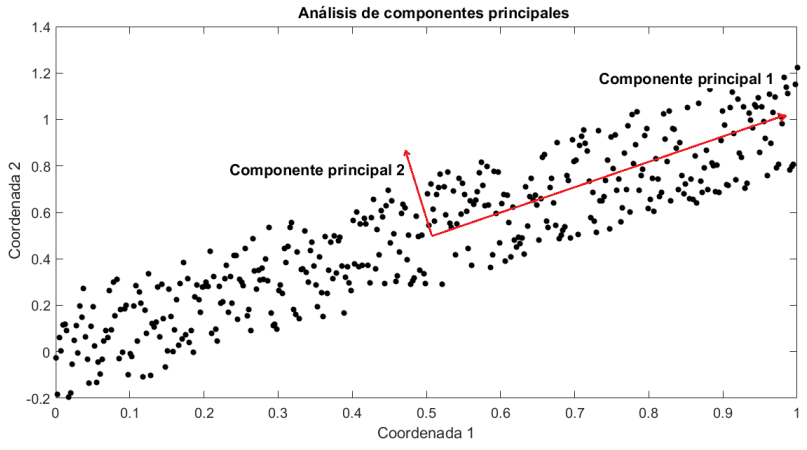

Figura 2.4: Análisis de componentes principales a un conjunto bidimensional

2.3.2. Autoencoders (AE)

Los autoencoders son redes neuronales artificiales, pertenecientes a los algoritmos de aprendizaje profundo, no supervisados, cuya finalidad es replicar la entrada de la red en la salida. Para ello deben crear una representación de la entrada, que almacene las características más relevantes de esta, llamada espacio latente, y comúnmente posee una dimensión bastante menor que el conjunto de datos original, haciendo de los autoencoders útiles para la reducción de dimensionalidad.

Un autoencoder cuenta con una capa interna $h = f(x)$ que codifica el input (genera una representación de este) llamada encoder y una función que produce la reconstrucción $r = g(h)$, el decoder. Un autoencoder buscará aprender los encoder y decoder tal que $g(f(x)) = x$ para todo x. Como el modelo esta forzado a aprender los atributos más importantes para que pueda efectivamente reproducir el input en su output, este aprenderá en general propiedades útiles de los datos de entrenamiento [9].

El AE tiene como objetivo aprender la función presentada en la ecuación 2.17:

$$Y_{W,p}(X) \quad \approx \quad X \tag{2.17}$$

Es decir que busca una aproximación de función de identidad que permita obtener una salida lo más parecida a la entrada, donde la letra W corresponde a los pesos del AE y p a la dimensión del espacio latente.

La figura 2.5 muestra la arquitectura general de un autoencoder.

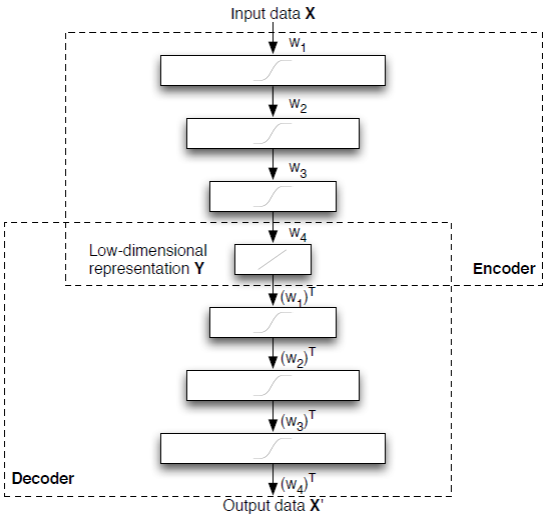

Figura 2.5: Representación de un autoencoder [1]

2.3.2.1. Undercomplete autoencoders

Copiar la entrada a la salida puede sonar una tarea inutil, pero no es la salida de la etapa de decode que sucita la atención. Se espera que entrenando el autoencoder para reproducir la entrada, genere un espacio de dimensión p, que contenga las propiedades más importantes de la entrada [9].

Una forma de obtener características útiles del autoencoder, es restringir la dimensión del espacio latente (p), a tener menor tamaño que la señal de entrada. Un autoencoder cuyo espacio latente es menor que la señal de entrada, es llamado un "undercomplete autoencoder". Las capas interiores, teniendo una dimensión de menor tamaño, fuerza al autoencoder a capturar las características más importantes de los datos de entrenamiento [10].

El proceso de aprendizaje es descrito simplemente como el problema de minimización de la función de perdida.

$$L\left(x, g\left(f(x)\right)\right) \tag{2.18}$$

Donde:
L: Es la función de costo, que penaliza $g(f(x))$, al no ser similar a x, a través del error cuadrático medio.

2.3.2.2. Autoencoders regularizados

Autoencoders con funciones de activación no lineales (etapas de encode y decode) son capaces de aprender poderosas generalizaciones no lineales. Para lograr un procesamiento útil, se debe evitar realizar el procedimiento trivial de reproducir las operaciones de la etapa de reducción (encoding) en la etapa de reconstrucción (decoding), si se cae en este error el espacio latente del autoencoder, será una matriz de identidad (I) del tamaño de la matriz de entrada [10].

Un "autoencoder regularizado" es un autoencoder que contiene una penalidad $\omega(p)$, en adición con el error de reconstrucción, donde p es la dimensión del espacio latente.

$$L\left(x, g\left(f(x)\right)\right) + \omega(p) \tag{2.19}$$

Donde: $g(p)$ es la salida de la etapa de decode, en la que se tiene que $p = f(x)$, salida de la etapa de encode.

Autoencoders regularizados son típicamente usados para aprender características previas para otro trabajo.

Un autoencoder que se ha regularizado, debe responder a características estadísticas únicas del conjunto de datos en el que se ha entrenado, en lugar de actuar simplemente como una función de identidad. De esta forma, al realizar el entrenamiento para produce un modelo que ha aprendido características útiles como un resultado [10].

Podemos pensar en la penalización $\omega(h)$ simplemente como un término de regularización agregado a una red neuronal, cuya tarea principal es copiar la entrada a la salida (objetivo de aprendizaje no supervisado) y posiblemente también realizar alguna tarea supervisada (con un objetivo de aprendizaje supervisado) que depende de las características del entregadas por el autoencoder regularizado [10].

2.3.2.2.1. Regularización L_2

Cuando se entrena un "autoencoder regularizado", es posible reducir el tamaño del regularizador de dispersión, aumentando los valores de los pesos w^l y disminuyendo los valores de $h(w_i{}^{(1)T}x_j + b_i^{(1)})$. Agregar un término de regularización en los pesos a la función de costo, evita que esto suceda [9]. La función de costo de una red neuronal con L_2 es una combinación de un objetivo no regularizado y un término de regularización:

$$L_\lambda(W) = \quad L(W) + \lambda \cdot \|W\|^2 \tag{2.20}$$

Donde:
W: es la matriz de pesos.
λ: Parámetro de regularización.
L: Evaluación de pesos y bias en la función de activación.

2.3.2.2.2. Dropout

El dropout o abandono, es una de las técnicas de regularización más eficaces y más utilizadas para redes neuronales, desarrollado por Geoffinton y sus alumnos en la Universidad de Toronto. El dropout, aplicado a una capa, consiste en abandonar al azar (dejando su salida en cero) una serie de características de salida de la capa durante el entrenamiento. Digamos un una capa dada normalmente devolvería un vector [0.2, 0.5, 1.3, 0.8, 1.1] durante el entrenamiento,para una entrada dada. Después de aplicar el dropout, este vector tendrá algunas entradas cero distribuido al azar: por ejemplo, [0, 0.5, 1.3, 0, 1.1]. La tasa de abandono es la fracción de las características que se ponen a cero; generalmente se establece entre 0.1 y 0.5 [11].

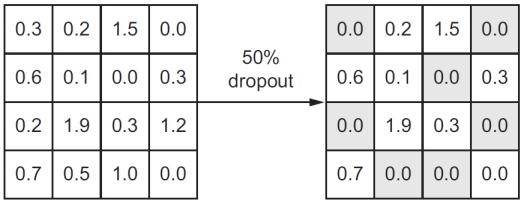

Figura 2.6: Ejemplo de dropout de p=50 % aplicada a una matriz

2.3.2.2.3. Batch Normalization

La normalización es una categoría amplia de métodos que buscan hacer que se vean diferentes datos de entrada más similares entre sí, lo que ayuda al modelo a aprender y generalizar bien a los nuevos datos. La forma más común de normalización de datos es centrar los datos en 0 restando la media de los datos, y dando a los datos una desviación estándar unitaria dividiendo el datos por su desviación estándar. En efecto, esto supone que los datos siguen una distribución normal (o gaussiana) y se asegura de que esta distribución esté centrada y escalado a la unidad varían. Batch Normalization o normalización por lotes, es un tipo de capa (BatchNormalization en Keras), puede normalizar adaptativamente los datos incluso si la media y la varianza cambian en el tiempo [9].

El efecto principal de la normalización por lotes es que ayuda con la propagación del gradiente y por lo tanto permite redes más profundas [9].

2.3.2.3. Selección de parámetros en un autoencoder

Construir el autoencoder que se adapte de mejor manera al problema es un procedimiento que en el cual se debe tomar los siguientes parámetros.

- Dimensión de las capas escondidas: Esto equivale al número de nodos que tendrá cada capa escondida del autoencoder.
- Número de capas: En este procedimiento se debe considerar, la dimensión de los datos de entrada. El número de capas se selecciona a partir de ensayos, prueba y error.
- Funciones de activación: Se deben escoger las funciones de activación de la etapa de encode y decode del autoencoder. Entre estás se encuentran relu, sigmoide, lineal y tangente hiperbólica.
- Función de costo: Se utiliza la función error cuadrático medio.
- Tasa de aprendizaje de la red: También conocida como velocidad de aprendizaje de la red, o paso tamaño del paso en aprendizaje automático, esta determina en que medida la información recién aprendida,anula la antigua, matemáticamente en el proceso de minimización de la función de costos es el paso que se da en cada nueva iteración.
- Tamaño del lote de entrenamiento (batch size): De manera de entrenar la red de la forma más rápida posible, el entrenamiento de esta se desarrolla por lotes de datos,en cada sesión de entrenamiento, se toma una porción de los datos y se entrena la red con estos, así de manera iterativa hasta entrenar con la totalidad de los datos. Esto disminuye considerablemente el tiempo de entrenamiento.
- Número de épocas: Número de iteraciones necesarias para optimizar el proceso, dependiendo del error aceptable, esta cantidad de épocas puede ser más o menos.
- Regularizadores: De peso L_2, dropout o batch normalization.

2.3.3. Kernel PCA

En esta sección se presenta el método de reducción de dimensionalidad basado en kernels para el análisis de componentes principales. El análisis de componentes mediante kernels es una extensión del método PCA en dominios no lineales.

En estadística la transformación de datos es algo natural, cuando se piensa que un fenómeno puede ser explicado considerando nuevas relaciones. En Kernel PCA se hace una transformación de los datos y sobre estos datos transformados se realiza PCA. Haciendo esto, el algoritmo puede detectar no linealidades en la variedad [12].

Es posible sustituir el espacio original de las observaciones χ , por un producto punto $\mathcal{F}$ mapeado (transformado) a través de $\phi : \chi \mapsto \mathcal{F}$. Partiendo de la misma suposición sobre la cual se construyo la matriz de covarianza C, la cual implica que los datos estén centralizados en $\mathcal{F}$, se procede a construir la matriz de covarianza $C_{\mathcal{F}}$ [12]:

$$C_{\mathcal{F}} = \quad \frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T \tag{2.21}$$

En este caso se debe encontrar los valores propios no nulos (λ>0) y los respectivos vectores propios $\nu_{\mathcal{F}}$ que satisfacen la ecuación:

$$\lambda\nu_{\mathcal{F}} = C_T\nu_{\mathcal{F}} \tag{2.22}$$

De la misma forma las soluciones deben estar dentro del espacio generado por $\{\phi(x_1), \phi(x_2), ..., \phi(x_n)\}$. Entonces,

$$\lambda\phi(x_k), \nu_{\mathcal{F}} = \phi(x_k)C_{\mathcal{F}}\nu_{\mathcal{F}} \tag{2.23}$$

Además es posible definir los vectores propios en términos de los datos mapeados en $\mathcal{F}$, debido a que la solución de $\upsilon_{\mathcal{F}}$ vive en el espacio $\phi(x_1)$, $\phi(x_2)$,..,$\phi(x_n)$, lo que es representado en la ecuación 2.24:

$$\upsilon_{\mathcal{F}} = \sum_{i=1}^{n}\alpha_i\phi(x_i) \tag{2.24}$$

Se define la matriz K como $k_{ij} = \langle\phi(x_k), \phi(x_j)\rangle$, se obtiene:

$$n\lambda K\alpha = K^2\alpha \tag{2.25}$$

Donde α denota el vector columna que sintetiza la representación de $\nu_{\mathcal{F}}$ dada en la ecuación 2.24, a través del conjunto de observaciones mapeadas por ϕ. Debido a la simetria de K, sus vectores propios generan el espacio completo, lo que implica que:

$$n\lambda\alpha = K\alpha \tag{2.26}$$

genera las soluciones de la ecuación 2.25. De esta manera, los valores propios asociados a α corresponde a n λ, en consecuencia cada uno de los $\nu_{\mathcal{F}}$ corresponde al mismo reordenamiento de los α. Es necesario trasladar la restricción $\|\upsilon_{\mathcal{F}}\|$ a los correspondientes vectores propios de K [12]:

$$\sum_{i,j}^{n} \alpha_i \alpha_j \left\langle \phi(x_k), \phi(x_j) \right\rangle = \lambda \left\langle \alpha, \alpha \right\rangle = 1 \tag{2.27}$$

Para la extracción de componentes principales, deben proyectarse los datos mapeados en $\mathcal{F}$ sobre los respectivos vectores propios seleccionados. Podemos hacer uso de:

$$\left\langle \nu_{\mathcal{F}}, \phi(x) \right\rangle = \sum_{i=1}^{n} \alpha_i \left\langle \phi(x_k), \phi(x) \right\rangle \tag{2.28}$$

Lo que es equivalente matemáticamente a:

$$S = K\left(x\right)\tilde{V} \tag{2.29}$$

que es la proyección de la matriz K en los vectores propios de encontrados a partir de la ecuación 2.26 [12].

Algunos de los kernels comúnmente utilizados son:

Kernel lineal: Corresponde al producto punto en el espacio de entrada (la utilización de este kernel, es el equivalente a aplicar PCA a los datos):

$$k(x, x') = \left\langle x, x' \right\rangle \tag{2.30}$$

Kernel polinomial: Representa la expansión a todas las combinaciones de monomios de orden d, y está dado por:

$$k(x, x') = \left\langle x, x' \right\rangle^{d} \tag{2.31}$$

Kernel gaussiano: Esta contenido dentro de las funciones de base radial, de parámetro σ:

$$k(x, x') = \quad exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{2.32}$$

Kernel tangente hiperbólica : El menos utilizado asociado a las funciones de activación de redes neuronales (tanh), los parametros son ξ y b:

$$k(x, x') = tanh\left(\xi \left\langle x.x' \right\rangle + b\right) \tag{2.33}$$

2.3.4. Escalamiento multidimensional (MDS)

MDS en su forma más clásica, es una técnica estadística que utiliza como entrada una matriz de disimilitud entre los datos, generalmente la distacia euclidiana entre puntos. Mediante la minimización de cierta función de costos, se busca generar las coordenadas de salida del sistema, buscando la configuración que mejor represente las distancias(disimilitudes) [13].

La entrada en el escalamiento multidimensional es, como la entrada en la mayoría de las otras técnicas de reducción de dimensionalidad. Una matriz de distancia Euclidiana por pares D cuyas entradas d_{ij} que representan la distancia euclidiana entre los puntos de datos de alta dimensión x_i y x_j. El escalamiento clásico o multidimensional encuentra el mapeo lineal M que minimiza la función de costo [14]:

$$\phi\left(Y\right) = \sum_{ij}^{N} \left(d_{ij}^2 - \left\|y_i - y_j\right\|^2\right) \tag{2.34}$$

Donde:

$\|y_i - y_j\|$: Distancia euclidiana entre los puntos y_i

El mínimo de esta función de costo esta dado por la descomposición de valores propios de la matriz de gram $K = MM^T$ del espacio de alta dimensión. Las entradas de la matriz de gram pueden ser encontradas calculando la matriz del cuadrado de distancias euclidianas dadas por [14]:

$$k_{ij} = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}\sum_{l} d_{il}^2 - \frac{1}{n}\sum_{l} {d_{lm}}^2 + \frac{1}{n^2}\sum_{lm} {d_{lm}}^2\right) \tag{2.35}$$

El mínimo de la función de costo en la Ecuación 2.34 ahora se puede obtener multiplicando los vectores propios principales de la matriz de Gram con la raíz cuadrada de sus valores propios correspondientes.

2.3.5. Mapeo isométrico de características (Isomap)

Isomap es una técnica de reducción de dimensión no lineal. Técnicas como el mapeo clásico de MDS retiene las distancias euclidianas por pares, no tiene en cuenta la distribución de puntos de datos en la variedad. Si los datos de alta dimensión se encuentran en una curva , el escalamiento multidimensional (MDS), puede interpretar dos puntos de datos como "estar juntos", mientras que su distancia sobre la variedad es más grande. Isomap resuelve este problema considerando la distancia geodésica por pares, que es la distancia entre un par de puntos medidos sobre la variedad [14].

El principal objetivo de este método es preservar la geometría intrínseca de los datos de entrada reflejada a partir de las distancias geodésicas de la variedad. La clave es encontrar una forma eficiente de calcular la verdadera distancia geodésica entre las observaciones, dada solamente la distancia euclidiana del espacio de alta dimensión en el que se trabaja. En Isomap se asume que los datos se encuentran en una variedad S desconocida en un espacio de alta dimensión. Se busca realizar un escalamiento del espacio original a un espacio de menor dimensión, que preserve lo mejor posible la estructura intrínseca de las observaciones [14].

El algoritmo de Isomap se compone de 6 pasos fundamentales. En primera instancia se determinan que puntos son vecinos en una variedad S. Con este propósito es posible utilizar dos criterios simples: el punto i se considera vecino con el punto j si y solo si este pertenece a los k vecinos más cercanos de j o si se encuentra a una distancia menor de un cierto radio ρ.

A partir de los vecindarios establecidos se construye un grafo G sobre todos los datos de entrada, y se establecen las longitudes de las aristas como $d_x(i,j)$ (distancia euclidiana) entre puntos vecinos. En tercer lugar se realiza el cálculo de los caminos más cortos, estimando las distancias geodésicas $d_s(i,j)$ entre todos los pares de puntos en S y calculando las distancias de los caminos o trayectorias más cortas $d_g(i,j)$ en el grafo G [13].

En el caso de que i y j se encuentren conectados por una arista $d_g(i,j) = d_x(i,j)$, en el otro caso en el que no se encuentran conectados $d_g(i,j) = \infty$. La matriz $D = d_g(i,j)$ contiene las distancias entre los caminos más cortos entre todos los pares de puntos en el grafo G [13].

En cuarto lugar, se construye la matriz de Gram a través de un centrado de la matriz de distancias de la siguiente forma:

$$\widetilde{B} = -\frac{1}{2} J D^2 J \tag{2.36}$$

Donde:
$J = I_n - \frac{1}{n}(1_n 1_n^T)$

Una vez calculada la matriz de Gram se calculan los valores y vectores propios de la misma λ_i y ν_i. Esto permite encontrar el i-esimo vector del espacio reducido como $s_i = \sqrt{\lambda_i}\nu_i$.

Una variedad es el objeto geométrico estándar que generaliza la noción intuitiva de la curva (1D- Variedad) y superficie (2D-Variedad) a cualquier dimensión y sobre cuerpos diversos (no necesariamentente el de los reales)

2.3.6. PCA probabilístico

La derivación más común de PCA es en términos de una proyección lineal estandarizada, que maximiza la varianza en el espacio proyectado.

Una propiedad complementaria de PCA, es que al ser una proyección ortogonal $x_n = W^T(z_n - \overline{z})$ las proyecciones minimizan el error de reconstrucción cuadrático $\sum_n \|z_n - \hat{z}_n\|^2$. Donde la reconstrucción lineal óptima z_n es $\hat{z}_n = Wx_n + \bar{z}$. Sin embargo, una característica notable de estas definiciones de PCA es la ausencia de un modelo probabilístico asociado para los datos observados. El objetivo,por lo tanto, de PCA probabilístico tratar esta limitación demostrando que PCA puede ser derivado dentro de un marco de estimación de una densidad de probabilidad [2].

Para esto, PPCA asigna una distribución de probabilidad a la proyección de los datos en baja dimensión, tal como lo muestra la figura 2.7, les asigna una distribución de probabilidad normal, la cual esta sujeta a la probabilidad condicionada de los datos originales [2]:

$$p(x_i|z_i,\theta) = (Wz_i + \bar{z}) \tag{2.37}$$

En donde los parámetros de la distribución de probabilidad se encuentran resolviendo el problema máxima verosimilitud.

$$\max_{\theta} \quad p(x|\theta) = \prod_{i=1}^{n} p(x_i|\theta) \tag{2.38}$$

$$\max_{\theta} \quad p(x|\theta) = \prod_{i=1}^{n} \int p(x_i|z_i,\theta) p(z_i)\, dt_i \tag{2.39}$$

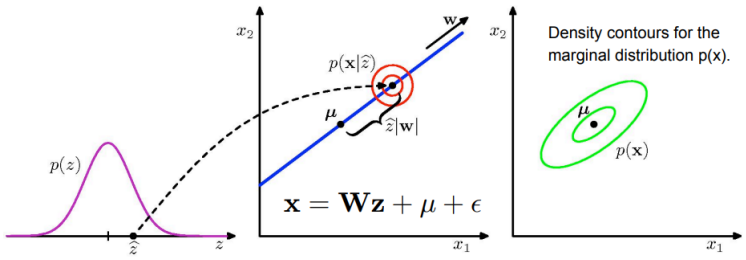

Figura 2.7: Proyección de los datos y asignación de la distribución de probabilidad [2]

2.3.7. Relación entre los métodos de reducción de dimensionalidad

Primero, PCA tradicional es lo mismo que la realización de Kernel PCA con un núcleo lineal, debido a la relación entre los vectores propios de la matriz de covarianza y el doble centrado de la matriz matriz de distancia euclidiana al cuadrado, que a su vez es igual a la matriz de Gram. Autoencoders que emplean funciones de activación lineales (encode y decode), tienen resultados muy similares a PCA [1].

En segundo lugar, realizar una escalamiento multidimensional en una matriz de distancia geodésica, es idéntico a realizar Isomap.

En tercer lugar, las técnicas espectrales Kernel PCA e Isomap pueden ser visto como casos especiales del problema más general del aprendizaje de 'eigenfunctions' [1]. Como resultado, Isomap, Kernel PCA que usa una función de kernel.

Por último PCA probabilístico, es la proyección de los datos en una dimensión más pequeña, tal como lo hace PCA pero no usando la matriz de covarianzas y ordenándolos por varianza adquirida, sino atribuyéndoles una distribución de probabilidad normal, encontrando a través de iteraciones los parámetros σ y μ [2].

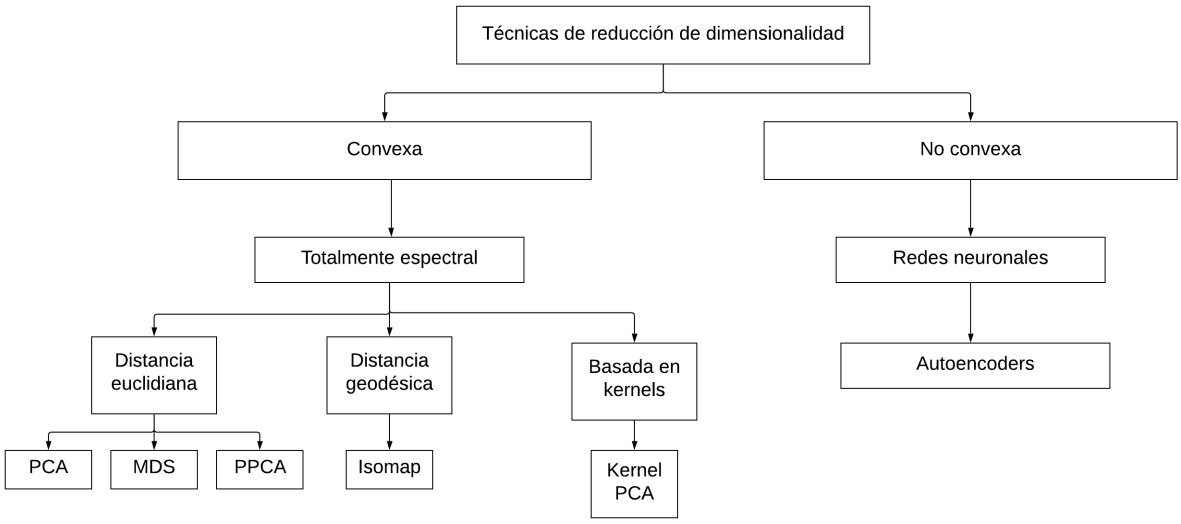

Figura 2.8: Taxonomia de las técnicas de reducción de dimensionalidad presentadas en este trabajo

Las técnicas convexas para la reducción de dimensionalidad optimizan una función de costo que no contiene óptimos locales, es decir, el espacio de solución es convexo [15]. Ocurriendo todo lo contrario en la situación no convexa (sin embargo los autoencoders pueden trabajar con la función de costo RMSE error cuadratico medio, la cual es una función convexa). Existe la clasificación totalmente espectral y escasamente espectral, la primera se refiere a que el método resuelve el problema de vectores y valores propios de una matriz objetivo. En la segunda el método utiliza la minimización de una función objetivo, que relaciona el espacio de alta y baja dimensión.

2.4. Algoritmo LME: Aproximación lineal con máxima entropía

2.4.1. Principio de máxima entropía (PME)

En ciencias de la información, el concepto de entropía es explicado como una medida de incertidumbre de los datos. El principio de máxima entropía sostiene que la distribución de probabilidad menos sesgada que se le puede atribuir a un sistema es aquella que maximiza la entropía o incertidumbre, es decir, aquel momento en que la desinformación es máxima [3].

Si se considera N eventos discretos $\{x_1, x_2, x_3....x_n\}$, donde la probabilidad de cada evento esta dada por $p_a = p\left(x_a\right)\varepsilon \quad [0,1]$, donde la incertidumbre asociada a cada p_a es $-ln\left(p_a\right)$. La entropía de Shannon, es la cantidad de incertidumbre representada por $\{p_1, p_2, p_3, ...p_n\}$. La distribución de probabilidad menos sesgada y la que tiene mayor probabilidad de ocurrir es obtenida a partir del problema de optimización representado en la siguiente ecuación [3]:

$$H\left(p\right) = -\sum_{i=1}^{N} p_i ln\left(p_i\right) \tag{2.40}$$

$$max_{p\varepsilon\Re_+^N}\left[H\left(p\right) = -\sum_{i=1}^{N} p_i ln\left(p_i\right)\right] \tag{2.41}$$

Bajo las siguientes restricciones:

$$\sum_{i=1}^{N} p_i = 1 \tag{2.42}$$

$$\sum_{i=1}^{N} p_i \cdot g_r\left(x_i\right) = \langle g_r\left(x_i\right)\rangle \tag{2.43}$$

Donde:
R_+^N: Reales no negativos
$\langle g_r\left(x_i\right)\rangle$: Valor esperado para la función densidad $g_r(r = 0, 1, 2, ...m)$

2.4.2. Método de aproximación lineal basado en PME

La estrategia de detección de impactos se resuelve como una regresión lineal del principio de máxima entropía. Para lograr esto se debe crear una base de datos, la cual contenga los datos de observación y las características de cada impacto.

El vector de observaciones se define como:

$$Y^j = \left\{ y_1^j, y_2^j, y_3^j \right\} \tag{2.44}$$

Donde el vector y_1^j, y_2^j son las coordenadas x,y del impacto en coordenadas cartesianas y y_3^j representa la magnitud en Newton del impacto. No se considera la variación en el eje z de la estructura debido a que las características fueron adquiridas sin considerar este eje [3].

Luego se define un vector X^j el cual tendrá un grupo de parámetros que están asociados a las respuestas asociadas a los impactos Y^j, estos parámetros serán extraídos según las diferentes técnicas de reducción de dimensionalidad.

$$X^j = \left\{ x_1^j, x_2^j, x_3^j, ..., x_m^j \right\} \tag{2.45}$$

El objetivo de la identificación de impactos es estimar el impacto Y en base a al vector de características observadas X. Para lograr este objetivo se construye la base de datos de J impactos. formada por las siguientes parejas $\{X^1, Y^1\}, \{X^2, Y^2\}, \{X^3, Y^3\}, ..., \{X^j, Y^j\}$. Luego el vector X se representa como una combinación lineal de los N vecinos X^j más cercanos en la base de datos, ponderados con un vector w con elementos w_j que cumplen con la siguiente condición [3].

$$X = \sum_{i=1}^{N} w_j \left(X \right) X^j \tag{2.46}$$

Sujeto a la condición:

$$\sum_{i=1}^{N} w_j \left(X \right) = 1 \tag{2.47}$$

Donde:

- $w_1(X), w_2(X), ..., w_N(X)$: Son las funciones de ponderación o pesos.
- $X^1, X^2, ..., X^N$: Vectores de características dentro de la base de datos.

Esto puede ser escrito de manera matricial y expresarse de la siguiente forma:

$$Ax = b \tag{2.48}$$

$$A = \begin{pmatrix} x_1^1 & x_2^1 & ..... & x_1^N \\ x_1^2 & x_2^2 & ..... & x_2^N \\ x_1^n & x_2^n & ..... & x_n^N \\ 1 & 1 & ..... & 1 \end{pmatrix}_{(n+1)\cdot N} \tag{2.49}$$

$$b = \begin{pmatrix} X_1 \\ X_2 \\ . \\ . \\ . \\ X_n \\ 1 \end{pmatrix}_{(n+1)\cdot 1} \tag{2.50}$$

$$w = \begin{pmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_n \\ 1 \end{pmatrix}_{N\cdot 1} \tag{2.51}$$

Obteniendo el vector w (pesos de la combinación lineal), de la ecuación anterior, podemos estimar los Y, esta estimación se representa en la siguiente ecuación:

$$\hat{Y} = \sum_{j=1}^{N} w_j\,(X) \cdot Y \quad ^{j} \tag{2.52}$$

Este sistema de ecuaciones es indeterminado, ya que N solo considera los valores más cercanos a la vecindad de X. Sin embargo este se puede resolver aplicando el principio de máxima entropía para obtener los valores del vector w.

Retomando el problema de optimización lineal que se presenta en la ecuación 2.41 sujeto a las restricciones 2.42 y 2.44. Se observa que es posible aplicar LME al sistema con una distribución de probabilidad previo, lo que permite reducir la incertidumbre con respecto a la probabilidad p_i. Suponiendo que de esta forma que esta toma una distribución previa de m_i, esta reduce la incertidumbre a $-ln(p_i) + ln(m_i) = -ln\left(\frac{p_i}{m_i}\right)$, lo que modifica el problema de optimización a lo siguiente [3]:

$$max_{p\varepsilon\Re_+^N}\left[H\left(p\right) = -\sum_{i=1}^{N} p_i ln\left(\frac{p_i}{m_i}\right)\right] \tag{2.53}$$

Con las mismas condiciones anteriores.

Gracias a los anterior, se reemplaza la probabilidad p_i por la función ponderación w_i de vector w, proveniente de la misma combinación:

$$max_{w\varepsilon\Re_+^N}\left[H\left(p\right) = -\sum_{i=1}^{N} p_i ln\left(\frac{w_i}{m_i}\right)\right] \tag{2.54}$$

Bajo las restricciones:

$$\sum_{i=1}^{N} w_i = 1 \tag{2.55}$$

$$\sum_{i=1}^{N} w_i \cdot \left(X - X^i\right) = 0 \tag{2.56}$$

Para m_i se utiliza normalmente la distribución de gauss.

$$m_i = \quad exp\left(-\beta_i \left\|X^i - X\right\|^2\right) \tag{2.57}$$

$$\beta_i = \frac{\gamma}{h_i^2} \tag{2.58}$$

Donde:
β_i: Depende de γ
γ : Parámetro que controla la curva de aproximación gaussiana.
h_i: Distancia euclidiana n-dimensional característica entre los vecinos de la base de datos, para cada X^i.

Finalmente las matrices X e Y para el caso LME sin procesamiento previo, son presentadas en las ecuaciones.

$$X = \begin{pmatrix} A_1^1 & ... & A_1^n \\ T_1^1 & ... & T_1^n \\ t_1^1 & ... & t_1^n \\ . & ... & . \\ . & ... & . \\ A_s^1 & ... & A_s^n \\ T_s^1 & ... & T_s^n \\ t_s^1 & ... & t_s^n \end{pmatrix}_{nxs} \tag{2.59}$$

$$Y = \begin{pmatrix} x_1 & ... & x_n \\ y_1 & ... & y_n \\ F_1 & ... & F_n \end{pmatrix} \tag{2.60}$$

2.4.3. Multiplicadores de Lagrange

Para determinar los valores correspondientes a los pesos w_i se utiliza el método de optimización de los multiplicadores de Lagrange, el cual nos permite encontrar mínimos o máximos de una función sujeto a las condiciones de lagrange [3].

En este caso, se trata de un problema de maximización de la ecuación 2.54 aplicando las siguientes condiciones:

$$\sum_{i=1}^{N} w_j = 1 \tag{2.61}$$

$$\sum_{i=1}^{N} w_j X^i - X = 0 \tag{2.62}$$

$$\sum_{i=1}^{N} w_j Y^i - Y = 0 \tag{2.63}$$

Luego se define el lagrangeano de la ecuación 2.61 como sigue:

$$L\left(x, \lambda\right) = \quad f\left(x\right) + \sum_{k=1}^{N} \lambda_k g_k\left(x\right) \tag{2.64}$$

Donde:
f(x): Función objetivo a maximizar
$g_k(x)$: Restricciones aplicadas.

Para encontrar el máximo se aplica el gradiente a la función lagrangeano como sigue de la forma:

$$\nabla L\left(x, \lambda\right) = 0 \tag{2.65}$$

En el caso particular de la ecuación 2.54 el termino del lagrangeano de la ecuación 2.65 donde los multiplicadores correspondientes están asociados a cada una de las restricciones del problema. Luego se aplica la derivada parcial con respecto a w y se obtiene[3]:

$$\delta\left[-\sum_{i=1}^{N} w_j ln\left(\frac{w_j}{m_j}\right) + \lambda_0\left(1 - \sum_{i=1}^{N} w_j\right) + \lambda_1\left(X - \sum_{i=1}^{N} w_j X^j\right) + \lambda_2\left(Y - \sum_{i=1}^{N} w_j Y^j\right)\right] = 0 \tag{2.66}$$

La ecuación 2.67 representa el resultado de la expresión:

$$-1 - ln\left(\frac{w_j}{m_j}\right) - \lambda_0 - \lambda_1 X^j - \lambda_2 Y^j \tag{2.67}$$

A continuación $\lambda_0 = ln(Z) - 1$ donde Z corresponde a la función partición. De este modo la ecuación puede ser descrita como se presenta a continuación.

$$ln\left(\frac{w_j}{m_j}\right) + ln\left(Z\right) = -\lambda_1 X^j - \lambda_2 Y^j \tag{2.68}$$

$$w_j = \frac{e^{-\lambda_1 X^j - \lambda_2 Y^j}}{Z} \tag{2.69}$$

Tomando restricción $\sum_{i=1}^{N} w_j = 1$

$$Z\left(\lambda_1, \lambda_2\right) = \sum_{j=1}^{n} e^{-\lambda_1 X^j - \lambda_2 Y^j} \tag{2.70}$$

Esta expresión puede ser descrita en función del peso:

$$w_j = \frac{e^{-\lambda_1 X^j - \lambda_2 Y^j}}{\sum_{j=1}^{n} e^{-\lambda_1 X^j - \lambda_2 Y^j}} = \quad \frac{Z_j}{\sum_{j=1}^{N} Z_j} \tag{2.71}$$

Al aplicar las restricciones 2.62 y 2.63 se obtiene un sistema de ecuaciones no lineales el cual se describe a continuación:

$$\sum_{j=1}^{n} e^{-\lambda_1 X^j - \lambda_2 Y^j} X^j - X = 0 \tag{2.72}$$

$$\sum_{j=1}^{n} e^{-\lambda_1 X^j - \lambda_2 Y^j} Y^j - Y = 0 \tag{2.73}$$

Problema con el cual es posible obtener los parámetros λ_1 y λ_2, con los que se resuelve el problema. Este método se resuelve de manera iterativa.

2.5. Redes neuronales convolucionales

Las redes neuronales convolucionales (o CNNs) son un tipo de redes neuronales que fueron diseñadas para procesar datos con una tipología tipo-grid (grilla). Una serie de tiempo que tiene observaciones en intervalos regulares de tiempo se puede pensar como un grid de 1 dimensión. Las imagenes se pueden pensar como grillas de 2-D pixeles. El nombre de esta arquitectura hace referencia a la operación matemática conocida como convolución [10]. La operación convolución se define como:

$$s\left(t\right) = \left(x * w\right)\left(t\right) = \int x\left(a\right) w\left(t - a\right) da \tag{2.74}$$

En el contexto de CNNs, el primer argumento a convolucionar x, es el input, y el segundo argumento w, se conoce como kernel. El output, s(t) se conoce como feature map, En aplicaciones de aprendizaje de máquinas, el input será un arreglo multidimensional de datos, y el kernel un arreglo multidimensional de parámetros que se buscarán aprender.Usualmente, al trabajar con datos en un computador el tiempo se considera discreto. por lo que resulta conveniente definir la operación de convolución discreta [10]:

$$s\left(t\right) = \left(x * w\right)\left(t\right) = \sum_{a=-\infty}^{a=\infty} x\left(a\right)\left(t - a\right) \tag{2.75}$$

Se asumirá que las funciones son 0 en todo su dominio excepto en el set finito de puntos para el cual se guardan valores, permitiendo realizar estas sumatorias infinitas. Las librerías de redes neuronales implementan la función cross-correlation y la llaman convolución. Para una imagen I de 2 dimensiones y un kernel K de 2 dimensiones, esto es:

$$S\left(i, j\right) = \left(I * K\right)\left(i, j\right) = \sum_{m} \sum_{n} I\left(i + m, j + n\right) K\left(m, n\right) \tag{2.76}$$

Esto es equivalente a la operación de convolución discreta con la diferencia que el operador no es conmutativo, $S(i,j) = (I*K)(i,j) \neq (K*I)(i,j)$, lo cual no es importante para implementaciones de redes neuronales. La motivación por usar convoluciones surge de 3 importantes propiedades: interacciones sparse, parameter sharing y representaciones equivariantes. Interacciones sparse, se refiere a que hay menos conexiones fully connected, esto mediante el uso de kernels de menor dimensionalidad que el input, lo cual implica una eficiencia en términos de memoria por tener que almacenar menos parámetros, y eficiencia computacional por realizar menos operaciones. Parameter sharing se refiere a usar los mismos parámetros para distintas funciones dentro del modelo. Esto implica usar los pesos aprendidos en múltiples partes del input (como el caso de aplicaciones en imagenes, reconocer bordes por ejemplo). Que una función sea equivariante, significa que si el input cambia, el output cambia de la misma forma. Esto permite que por ejemplo en la aplicación de imagenes la convolución cree un map 2-D dónde ciertos atributos aparecen en el input.
Una capa típica de una red convolucional costa de 3 etapas: primero, aplicar varias convoluciones para producir un set de activaciones lineales. Luego, aplicar una activación no lineal (detector

stage). Finalmente una función de pooling para modificar aún más el output de la capa (ver 2.9). Una función de pooling reemplaza el output de la red en alguna locación por estadísticos de los outputs cercanos.

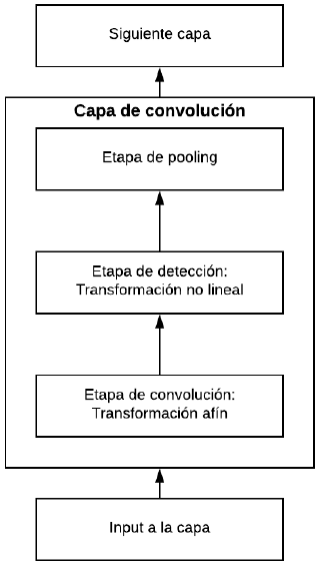

Figura 2.9: Capas de una red convolucional

Maxpooling retorna el máxima de un output en una variedad rectangular. Las operaciones de pooling permiten que la red sea invariante a pequeñas transformaciones en el input. Pooling también es escencial para procesar inputs de tamaño variable(por ejemplo inputs de distinto tamaño). Otras diferencias con respecto a la operación de convolución en el contexto de redes neuronales son, por ejemplo, el aplicar múltiples convoluciones en paralelo, esto permite extraer distintos tipos de atributos en vez de uno solo. Por otro lado el stride hace referencia a cuantas características' se quieren convolucionar en cada dirección en el output. En la figura se muestra el ejemplo de una convolución con stride. Esto operación permite reducir nuevamente el costo computacional. Esto también implica que el output disminuye su tamaño en cada capa. El uso de padding puede revertir esto. Padding se refiere agrandar el input con ceros para hacerlo más amplio. Una convolución en la que no se usa zero-padding se conoce como valid. Una convolución que mantiene el tamaño desde el input al output se conoce como same (ver figura 2.10). En la práctica, las capas de una red convolucional usan operaciones entre una convolución valid y same [10].

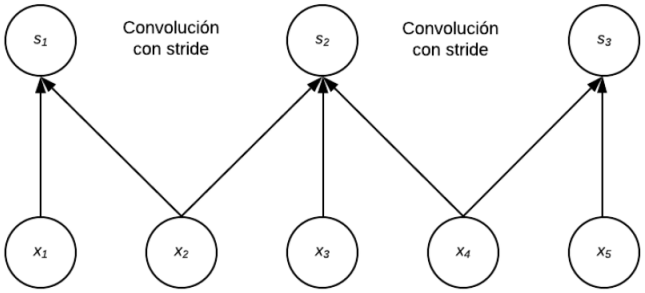

Figura 2.10: Convolución con un stride igual a 2

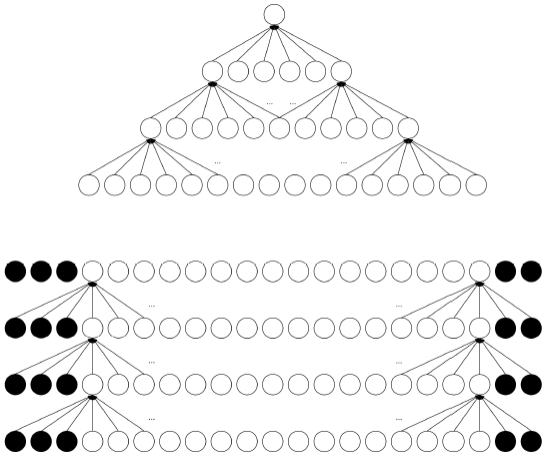

Figura 2.11: Efecto de usar zero padding en una red convolucional en cuanto al tamaño de la red

2.6. Evaluación de métodos de detección de impactos

Para comparar los distintos métodos de detección de impactos se definen las siguientes funciones que determinan el error de estos métodos al estimar la posición y magnitud del impacto.

$$E_x = \frac{1}{n}\sum_{j=1}^{N}\left|\widehat{Y}_1^j - Y_1^j\right| \tag{2.77}$$

$$E_y = \frac{1}{n}\sum_{j=1}^{N}\left|\widehat{Y}_2^j - Y_2^j\right| \tag{2.78}$$

$$E_F = \frac{1}{n}\sum_{j=1}^{N}\frac{\left|\widehat{Y}_3^j - Y_3^j\right|}{Y_3^j}\cdot 100 \tag{2.79}$$

$$E_A = \frac{E_x\,[cm]\cdot E_y\,[cm]}{A\,[cm^2]}\cdot 100 = \frac{\sum_{j=1}^{N}\left|\widehat{Y}_1^j - Y_1^j\right|\cdot\sum_{j=1}^{N}\left|\widehat{Y}_2^j - Y_2^j\right|}{n^2 A}\cdot 100 \tag{2.80}$$

-Donde:
n: Número de elementos en la base de datos de prueba.
A: Área de las mediciones sobre el fuselaje.
Y_1^j: Coordenada X real del j-ésimo impacto [cm].
$\widehat{Y}_1^j$: Coordenada X estimada del j-ésimo impacto [cm].
Y_2^j: Coordenada Y real del j-ésimo impacto [cm].
$\widehat{Y}_2^j$: Coordenada Y estimada del j-ésimo impacto [cm].
Y_3^j: Fuerza experimental del j-ésimo impacto
$\widehat{Y}_3^j$: Fuerza estimada del j-ésimo impacto

E_x: Error promedio de la estimación de impacto en la coordenada X
E_y: Error promedio de la estimación de impacto en la coordenada Y
E_F: Porcentaje de error en la estimación de la magnitud de la fuerza [%]
E_A: Porcentaje de error en la estimación del área del impacto [%]

Se define el error promedio en la localización de los impactos en coordenadas x e y como las funciones presentadas en las ecuaciones mostradas en 2.77 y 2.78, las cuales comparan la posición real del impacto, y luego se obtienen un promedio de esta diferencia para n impactos.

De forma similar se obtiene el error porcentual de área con la función presentada en la ecuación . Este se calcula al multiplicar los errores de cada coordenada y dividir por el área total de la superficie estudiada, obteniendo finalmente el error de la localización del impacto como un error de área porcentual.

Para el caso del error en la estimación de la fuerza del impacto, se realiza un procedimiento análogo, utilizando la diferencia entre el valor estimado por el algoritmo y el valor experimental medido por el martillo modal. Sin embargo, en este caso se divide por la fuerza experimental, como se muestra en la ecuación 2.80, para obtener el error porcentual.

Se define el error de identificación de impacto normalizado como sigue:

$$E_I = E_A \cdot E_F \tag{2.81}$$

El algoritmo de identificación de impactos se evalúa para diferentes valores de nv (número de vecinos) hasta que finalmente se selecciona, eligiendo los valores nv* que entreguen el menor error de identificación de impacto.

Capítulo 3

Metodología

La metodología empleada utiliza dos estrategias de estimación de impacto, la primera es constituida por la aplicación de una técnica de reducción de dimensionalidad (aprendizaje no supervisado) junto a una aproximación lineal del principio de máxima entropía (LME, aprendizaje supervisado), mientras que la otra estrategia es un modelo de una red neuronal convolucional (aprendizaje supervisado) para estimar los impactos.

Ambas estrategias seleccionan las características más relevantes de una base de datos de entrenamiento, para luego a partir de ellas, estimar nuevos impactos. La estrategia rd+LME, selecciona las características relevantes de los impactos de entrenamiento por medio de una técnica de reducción de dimensionalidad y estima impactos futuros con el LME, mientras que para la estrategia de redes convolucionales, se seleccionan características relevantes por medio de bloques de convolución y se ocupan estás características en una regresión de un red neuronal artificial (fully connected).

La metodología general consta de cuatro partes principales, que son comunes a las dos estrategias: montaje experimental, construcción de los conjuntos de datos, selección de parámetros y evaluación de los métodos de identificación de impacto.

Las figuras 2.9 y 3.1 muestran un diagrama de flujo del funcionamiento de las dos estrategias de identificación de impactos.

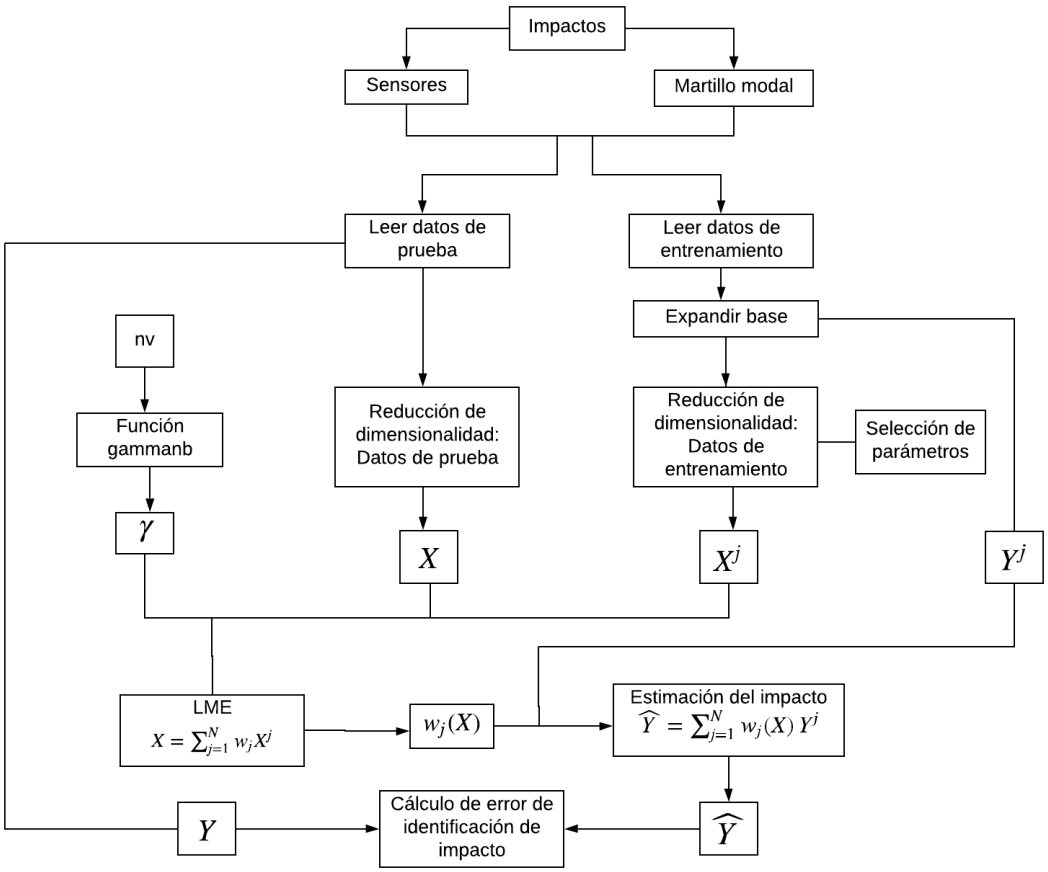

Figura 3.1: Estrategia de identificación de impactos: Técnica de reducción de dimensionalidad + LME

En lo que se refiere a reducción de dimensionalidad de los datos se utilizarán las siguientes técnicas de reducción de dimensionalidad:

1. Análisis de componentes principales (PCA).
2. Autoencoders (AE).
3. Análisis de componentes principales mediante kernels (Kernel PCA).
4. Mapeo isométrico de características (ISOMAP).
5. Escalamiento multidimensional (MDS).
6. PCA probabilístico (PPCA).

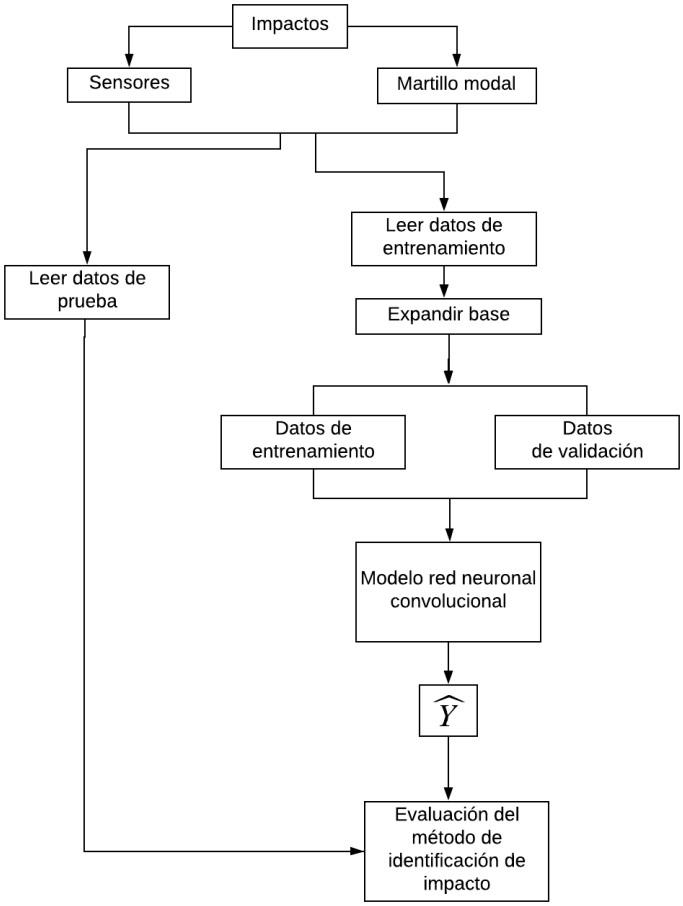

Figura 3.2: Estrategia de identificación de impactos: Modelo de red neuronal convolucional

3.1. Montaje experimental

Dentro de los requerimientos fundamentales de este trabajo es el poder adquirir una base de datos de entrenamiento que describa de la mejor manera posible, que este uniformemente distribuida sobre la superficie del fuselaje del avión. Además de esta base de datos de entrenamiento se requiere una base de datos de prueba que se distribuya aleatoriamente en la estructura, la cual nos permitirá testear la calidad de nuestros algoritmos. Debido a esto, es fundamental el diseño de un montaje experimental, el cual permita obtener mediciones representativas del comportamiento vibratorio del fuselaje de un avión. Para la realización de este trabajo la empresa SKY Airlines facilitó un avión, el cual se encontraba en mantenimiento y en el cual se realizaron estas mediciones.

La finalidad del montaje experimental es obtener a partir de ensayos, las respuestas vibratorias a impactos en una parte del fuselaje de un avión. Las respuestas entregadas por el sistema deben ser medidas por sensores (acelerómetros) y ser digitalizadas para su posterior procesamiento. En la presente sección se presentan todas las herramientas y sistemas necesarios para poder adquirir estás mediciones.

1. Seleccionar la superficie de trabajo en el fuselaje, aquella en la cual se harán las mediciones,

la dimensión de trabajo es de 90 [cm] en vertical y 135 [cm] en horizontal.

2. 6 sensores de tipo acelerómetros que son adheridos a la superficie de la estructura.
3. Martillo modal, para medir directamente la fuerza que fue aplicada en cada impacto.
4. Tarjeta de adquisición para la digitalización de los datos.
5. Computador personal con software de adquisición de datos, además de los software Matlab y Python.

A continuación se presentara el detalle de todos los sistemas y elementos para realizar estas mediciones.

3.1.1. Superficie de trabajo

Para este trabajo se selecciona una superficie de trabajo en costado derecho del avión (vista frontal), el área de trabajo tiene 90 [cm] en la dirección vertical y 135 [cm] en la dirección horizontal. La figura 3.3 muestra la superficie de trabajo en el avión.

Figura 3.3: Mediciones realizadas en la empresa SKY Airlines

Se divide la superficie de la estructura en tramos de 10 [cm] en al dirección vertical y en 15 [cm] en la dirección horizontal. De esta forma se forma un mallado con 81 secciones y 100 puntos (vertices de cada sección), en los cuales se aplicarán los impactos de entrenamiento.

Para los impactos de prueba se golpea en 25 puntos aleatorios de la superficie, en los cuales se mide su posición en x e y, se mide la fuerza detectada por el dinamómetro del martillo modal. La figura 3.5 muestra el posicionamiento de estos impactos en el fuselaje.

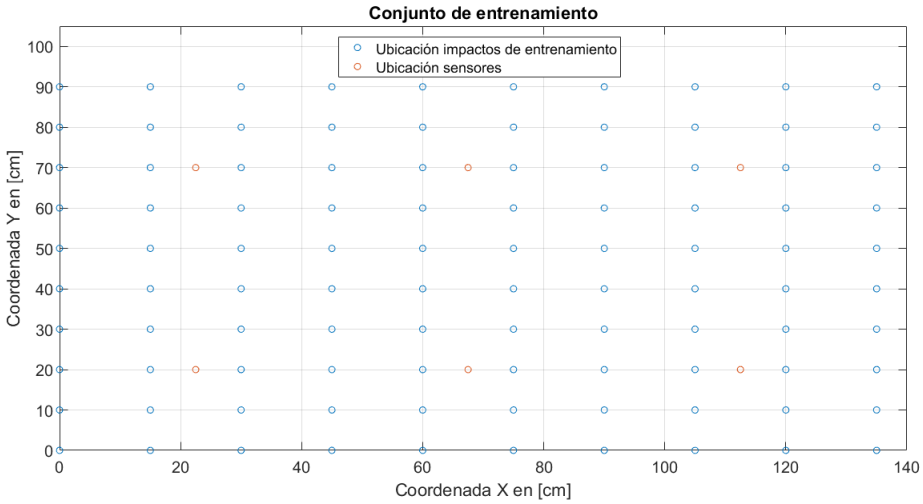

Figura 3.4: Posicionamiento de impactos de entrenamiento y de los sensores

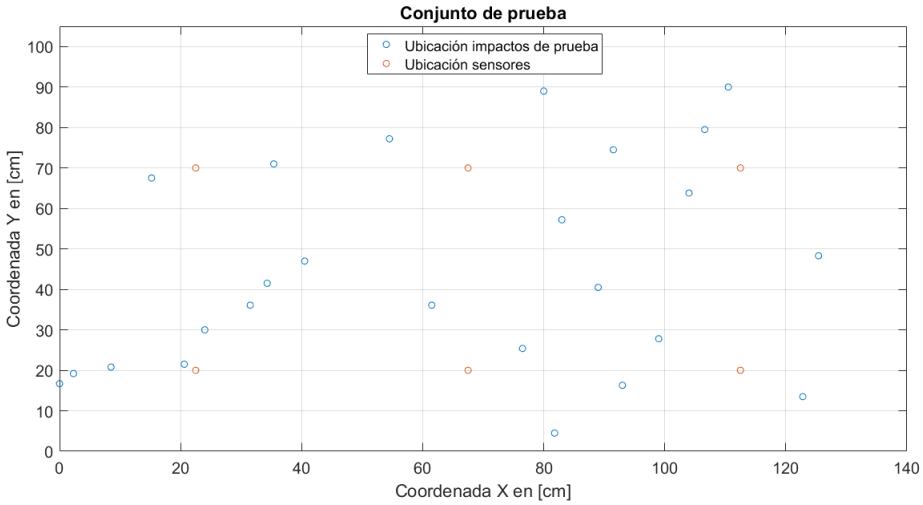

Figura 3.5: Posicionamiento de impactos de prueba y de los sensores

3.1.2. Sistema de adquisición de datos

La adquisición de datos es comúnmente conocida con las siglas DAQ por su significado en inglés, el cual es “data acquisition”, el cual se refiere al proceso de medir con una computadora un fenómeno físico o eléctrico. En este trabajo el sistema de adquisición de datos se compone de 4 elementos principales: el sensor, el martillo modal, la tarjeta de adquisición de datos (hardware) y una computadora (software programable).

3.1.2.1. Acelerómetros

Los acelerómetros piezoeléctricos de METRA, también conocidos como acelerómetros ICP comercialmente, son acelerómetros pensados para la medida de vibraciones a medias y altas frecuencias. Por su característica física, constan de un material piezoeléctrico que al ser manipulado mecánicamente proporciona una tensión muy pequeña, esta señal es amplificada por un amplificador interno que entrega una tensión proporcional a ese movimiento.

En la actualidad, la calidad de los acelerómetros ICP, su reducido tamaño, su peso y el hecho de que prácticamente todos los analizadores y colectores de datos portátiles, disponen de entradas adecuadas para ICP, hacen que este tipo de transductor sea el más utilizado por técnicos de mantenimiento para el análisis de vibraciones.

En este trabajo se ocupan 6 acelerómetros ICP. Las características técnicas de cada acelerómetro se muestran en la tabla 3.1 y los sensores son mostrados en las figuras 3.6a y 3.6b.

(a) Sensor modelo: 352A24 (b) Sensor modelo: 352B10

Figura 3.6: Sensores acelerómetros

Funcionamiento	Modelo: 352A24	Modelo: 352B10
Sensibilidad (± 10 %)	10.2 mV / (m /s^2)	1,02 mV / (m /s^2)
Rango de medicion	± 490 m / s^2 pk	± 4905 m / s^2pk
Rango de frecuencia (± 5 %)	1.0 a 8000 Hz	2 a 10000 Hz
Rango de frecuencia (± 10 %)	0.8 a 10000 Hz	1 a 17000 Hz
Frecuencia de resonancia	≥ 30 kHz	≥65 kHz
Resolución de banda ancha (1 a 10000 Hz)	0.002 m / s^2 rms	0.03 m / s^2 rms
No linealidad	≤1 %	≤1 %
Sensibilidad Transversal	≤5 %	≤5 %
Electrico		
Voltaje de excitación	18 a 30 VDC	18 a 30 VDC
Excitación de corriente constante	2 a 20 mA	2 a 20 mA
Impedancia de salida	≤300 Ohm	≤200 Ohm
Voltaje de polarización de salida	8 a 12 VDC	7 a 12 VDC
Constante de tiempo de descarga	0.4 a 1.5 seg	0.3 a 1.0 seg
Tiempo de establecimiento (dentro del 10 % del sesgo)	≤8 seg	≤3 seg
Ruido espectral (1 Hz)	785 (μm /s^2) / $\sqrt{Hz}$	9810 (μm /s^2) / $\sqrt{Hz}$
Ruido espectral (10 Hz)	147 (μm / s^2) / $\sqrt{Hz}$	2943 (μm /s^2) / $\sqrt{Hz}$
Ruido espectral (100 Hz)	39 (μm /s^2) / $\sqrt{Hz}$	785 (μm / s^2) / $\sqrt{Hz}$
Ruido espectral (1 kHz)	9.8 (μm / s^2) /$\sqrt{Hz}$	308 (μm /s^2) / $\sqrt{Hz}$
Aislamiento Eléctrico (Base)	≥100000000 Ohm	≥100000000 Ohm
Físico		
Tamaño - Diámetro	-	6.1 mm
Tamaño - Altura	4.8 mm	8.1 mm
Tamaño - Longitud	12.2 mm	-
Tamaño - Ancho	7.1 mm	-
Peso	0.8 gm	0.7 gm
Elemento de detección	Cerámico	Cerámico
Geometría sensora	Cortar	Cortar
Material de la carcasa	Aluminio anodizado	Titanio
Sellado	Epoxy	Hermético
Conector eléctrico	Jack coaxial 3-56	Pasadores de soldadura con cable adjunto
Posición de conexión eléctrica	Lado	Parte superior
Montaje	Adhesivo	Adhesivo

Tabla 3.1: Características de los sensores utilizados.

3.1.2.2. Martillo modal

El utilizado es un LC-04A que es especial para impactos, cuyas características son mostradas en la tabla 3.2:

Sensibilidad [PC/N]	4
Máximo peak de fuerza [kN]	60
Diametro de la cabeza [mm]	30
Masa de la cabeza [g]	300
Masa del cuerpo [g]	150
Largo [mm]	300
Transconductor de fuerza	CL-YD-305

Tabla 3.2: Características del martillo utilizado en las mediciones.

Se utiliza el martillo modal presentado en la figura 3.7 para realizar los impactos sobre la superficie de la estructura. El martillo posee internamente un dinamómetro que permite medir la magnitud de la fuerza de impacto, el cual esta conectado con la tarjeta de adquisición.

Figura 3.7: Martillo modal usado para provocar los impactos en el fuselaje

3.1.2.3. Tarjeta de adquisición

Todo esto se puede registrar gracias a las tarjetas de adquisición de datos, las cuales sirven para obtener una muestra de una variable física (en este caso voltaje) es decir, toman una señal de un sensor (sistema analógico) y después la adecuan para transformarla en un dato que pueda ser reconocido y registrado por un sistema digital, con el fin de que la pueda leer una computadora y realizar una tarea en específico mediante un software específico. Para este estudio, la tarjeta de adquisición utilizada, es el modelo NI 9232 que podemos ver en la figura 3.8

Figura 3.8: Tarjeta de adquisición modelo NI9332

3.1.2.4. Software

El software utilizado fue Flexlogger, este es capaz de construir sistemas de registro de datos flexibles y escalables con hardware de adquisición de datos NI (de la cual es parte la tarjeta de adquisición), sin necesidad de programación.

3.2. Construcción del conjunto de datos

3.2.1. Lectura de datos

Los impactos que son realizados sobre la superficie son tomados por los sensores y el martillo modal y posteriormente enviados a la tarjeta de adquisición, para ser transformados en señales de aceleración y de fuerza respecto al tiempo. Luego, estas son tomadas por el software de procesamiento de señales (Flexlogger), para finalmente ser usadas por el algoritmo de identificación de impactos que se desea evaluar.

Para la detección de la ubicación y magnitud de los impactos es importante considerar dos conjuntos de datos, uno que sea usado para el entrenamiento de los diferentes algoritmos de detección de impactos y el cual caracteriza la respuesta del sistema y otro set de datos que prueba, con el cual se mide la efectividad de los diferentes algoritmos de detección. Las figuras 3.4 y 3.5 muestran la posición de los impactos para los dos conjuntos de datos.

La etapa de lectura de datos concluye al obtener los vectores X^i e X_P^i con las aceleraciones, correspondiente a cada uno de los 6 sensores. Además de esto se crean las matrices Y^i Y^t que corresponden a las etiquetas de posición y fuerza de las matrices de entrenamiento y de prueba, respectivamente.El código de esta etapa se muestra en la sección de anexos.

3.2.2. Expansión de la base de entrenamiento

Dado el comportamiento lineal que presenta la respuesta vibratoria de la estructura, es posible ampliar la base de entrenamiento, añadiéndole respuestas frente a impactos de diferentes magnitudes. Esto es posible, al normalizar la señal vibratoria de cada impacto de entrenamiento, por la fuerza con la que este se produjo, al hacer esto se obtiene la respuesta vibratoria de cada punto a una fuerza unitaria, (se hace esto para los 100 puntos de entrenamiento). Con este conjunto de datos construido, es posible expandir el conjunto de datos de entrenamiento para cualquier fuerza (asumiendo un comportamiento lineal de la estructura).

La expansión de datos se obtiene de la ecuación 3.1:

$$X^t = X^j \frac{Y_3^t}{Y_3^j} \tag{3.1}$$

Donde:
Y_3^t: Fuerza definida para la expansión de los datos.
Y_3^j: Fuerza medida por el martillo modal para el impacto j.
X^j: Datos de respuesta obtenidos por un sensor para un impacto j.
X^t: Respuesta expandida para un sensor en un impacto j.

Las dos estrategias de identificación de impacto, requieren dos bases de datos distintas, cuya

construcción es mostrada a continuación.

3.2.2.1. Conjunto de datos: Reducción de dimensionalidad + LME

Para esta estrategia de identificación de impactos, se utiliza un vector de expansión de fuerzas igual a Y_3^t=[50, 60, 70, 80, 90, 100, 120, 140, 160, 180, 200, 220, 240, 260], 14 fuerzas, en el rango de 50 a 260 [N]. Para cada una de estas fuerzas se obtiene la respuesta vibratoria, en los 100 puntos de entrenamiento (ver 3.4). De esta forma por cada sensor (6 sensores) se obtiene la siguiente matriz de entrenamiento, cada columna de la matriz es la respuesta vibratoria a 1 impacto en una posición determinada.

$$Xi_{expandida} = \begin{pmatrix} a_{1,1} & . & . & . & a_{1,1400} \\ a_{2,1} & . & & & . \\ . & & . & & . \\ . & & & . & . \\ a_{P,1} & . & . & . & a_{P,1400} \end{pmatrix}_{Px1400} \tag{3.2}$$

Donde:
P: Dimensión de los datos, la que inicialmente es igual a 2000.
$a_{i,j}$: Envolvente de las aceleraciones, medidas por los sensores, (i: Paso temporal, j: Número del impacto).

Para aplicar LME (aprendizaje supervisado) todas las columnas de la matriz $Xi_{expandida}$ tienen asociado un vector etiqueta, el cual contiene posición y magnitud del impacto representado en la ecuación 3.4.

3.2.2.2. Conjunto de datos: Red neuronal convolucional

Para esta estrategia, se necesita una estructura de la base de datos en forma de tensores de 3 dimensiones. Cada impacto, será caracterizado por una matriz bidimensional que contiene la respuesta de impulso de los 6 sensores. Es decir, por cada impacto se tendrá una matriz de la forma:

$$x_{train}{}^{j} = \begin{pmatrix} a_{1,1}{}^{1} & a_{1,1}{}^{2} & . & . & a_{1,1}{}^{6} \\ a_{2,1}{}^{1} & . & & & a_{2,1}{}^{6} \\ . & & . & & . \\ . & & & . & . \\ a_{P,1}{}^{1} & . & . & . & a_{P,1}{}^{6} \end{pmatrix}_{Px6} \tag{3.3}$$

Donde:
$x_{train}{}^{j}$: Matriz de respuesta del j-ésimo impacto.
$a_{i,j}$: Envolvente de las aceleraciones medidos por los sensores.

Además cada uno de los impactos tiene asociado un vector respuesta o etiqueta ${y_{train}}^j$, que es de la forma de 3.4.

$$ {y_{train}}^j = \left[x^j, y^j, F^j\right] \tag{3.4} $$

La base de datos de esta estrategia esta compuesta por 1400 matrices, cada una de ellas con su respectiva etiqueta, la cual contiene la información de la posición y magnitud del impacto. Además de este conjunto de datos es dividido en un 90 % datos entrenamiento y un 10 % para datos de validación.

De esta forma, se obtiene el tensores 3.5 y su etiqueta 3.6:

$$ {X_{train}}^1 = \quad \left[{x_{train}}^1, \quad {x_{train}}^2, \quad ..., \quad {x_{train}}^{1400}\right] \tag{3.5} $$

$$ {Y_{train}}^1 = \quad \left[{y_{train}}^1, \quad {y_{train}}^2, \quad ..., \quad {y_{train}}^{1400}\right] \tag{3.6} $$

Las redes convolucionales trabajan comúnmente con imágenes, la figura A.6, en anexos, muestra como se ve un impacto transformado a una imagen en blanco y negro.

3.2.3. Normalización de los conjuntos de datos

La normalización utilizada para ambos conjuntos de datos es la normalización 'scaling', mostrada en la ecuación 3.7

$$ X_n = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.7} $$

3.3. Selección de parámetros

Cada algoritmo de identificación de impactos requiere hacer una selección de parámetros, de modo de escoger aquellos que tengan mejor rendimiento. En la presente sección se detalla la metodología usada para la selección de parámetros de cada uno de los algoritmos de identificación de impactos.

3.3.1. Aplicación de LME

En el caso de la estrategia de identificación de impactos reducción de dimensionalidad + LME, cada técnica de reducción de dimensionalidad debe ser evaluada junto al algoritmo LME.

Este algoritmo utiliza funciones del software " Maxent basis functions for matlab", desarrollado por el profesor integrante de esta memoria, Alejandro Ortiz Bernardin, profesor del departamento de ingeniería mecánica de la Universidad de Chile. Este programa permite ocupar el algoritmo, aproximación lineal con máxima entropía (LME) en el entorno de matlab.

Como se puede observar en el diagrama que representa la estrategia de identificación, reducción de dimensionalidad + LME (ver 3.1), el único parámetro que hay que seleccionar para el funcionamiento del algoritmo LME, es el número de vecinos (n_v), este parámetro permite el cálculo de otro parámetro, γ el cual se calcula a partir de la función Gammanb (perteneciente al software " Maxent basis Functions for matlab"). El parámetro γ controla la influencia que poseen los elementos de la base, al realizar la estimación de los valores de $\widehat{Y}$ (matriz resultante de impactos identificados por el algoritmo). Un valor elevado de γ implica que la $\|X^j - X\|$ tiene poca influencia, es decir, los elementos de la base que se encuentran a mayor distancia euclidiana aportan cada vez menos.

El número de vecinos del algoritmo LME que mejor se adapte a cada técnica de reducción de dimensionalidad se busca en un rango de n_v=[5:5:15 20:20:1400], es decir un vector que va de 5 en 5 hasta 20 vecinos y de 20 en 20 hasta 1400 (siendo 1400 la totalidad de vecinos disponibles en la base de datos). Se selecciona el número de vecinos que obtiene menor error de identificación de impactos.

3.3.2. Parámetros en las técnicas de reducción de dimensionalidad

La figura 3.3 muestra los parámetros a definir en cada una de las técnicas de reducción de dimensionalidad. Se seleccionan aquellos que en conjunto con el algoritmo LME, entreguen menor error de identificación de impactos.

Tabla 3.3: Parámetros asociados a las técnicas de reducción de dimensionalidad.

Método	**Parámetros**
PCA	Ninguno
Kernel PCA	σ (kernel gaussiano) y d(grado del kernel polinomial)
Isomap	h: Número de vecinos
MDS	Ninguno
ProbPCA	Ninguno
Autoencoders	Arquitectura y selección de hiperparámetros

3.3.2.1. Metodología: Análisis de componentes principales (PCA)

El método PCA proyecta los datos de entrada en un nuevos sistema de coordenadas, llamadas "componentes principales", las cuales están ordenadas por orden decreciente de varianza. Cada una de estás "componentes", describe un porcentaje de la varianza de la base de datos. Para este trabajo se seleccionan el número de componentes principales que retienen un 99.99 % de la varianza total.

Este método de reducción de dimensionalidad, es aplicado las bases de datos, que contienen las primeras 100, 200, 400, 800, 1600 y 2000 observaciones (time steps) de la señal vibratoria.

3.3.2.2. Selección de dimensión intrínseca (p) y cantidad de pasos temporales (P)

Posterior a la aplicación de la técnica de reducción de dimensionalidad PCA se determina la dimensión intrínseca, p, de los datos, la cual corresponde a la dimensión a la cual fueron reducidas los datos originales, conservando un 99.99 % de la varianza.

Esta dimensión intrínseca de la señal vibratoria, es usada para las siguientes técnicas de reducción de dimensionalidad:

- Autoencoders.
- Kernel PCA.
- Isomap.
- PCA probabilístico
- Escalamiento multidimensional.

3.3.2.3. Metodología: Análisis de componentes principales a través de kernels (Kernel PCA)

Kernel PCA es una extensión del método PCA a variedades no lineales, en la cual se aplica una transformación en los datos "método de kernel" con el fin de determinar las componentes principales, en este caso no lineales, que mejor representan los datos originales. En este trabajo se usan "kernels" de tipo polinomial y gaussiano.

Los kernel polinomiales se definen a partir del grado que poseen (ver 2.33), para encontrar el grado del polinomio que mejor se adapte al problema, se busca de una forma creciente, comenzando de un polinomio de grado 2, y aumentando el grado al polinomio si los resultados obtenidos van mejorando, llegando hasta un polinomio de grado 5 como máximo.

Los kernel gaussianos son definidos a partir de la ecuación 2.32. Se selecciona el parámetro σ con dos búsquedas por cuadrilla ("grid search"), en primera instancia en un rango amplio ([1,80] por ejemplo) y después acotar la búsqueda en el rango de valores que entreguen mejores resultados.

Se seleccionan los parámetros que en conjunto con el algoritmo LME entreguen un menor error de identificación de impactos.

3.3.2.4. Metodología: Mapeo isométrico de características (Isomap)

El principal objetivo de este método es preservar la geometría intrínseca de los datos de entrada reflejada a partir de las distancias geodésicas de la variedad. Para ello, el algoritmo necesita trabajar con un número fijo de vecinos, los cuales permiten describir un punto en función de sus vecinos. De trabajos anteriores similares a este[16], el número de vecinos para el algoritmo Isomap, se obtuvo en un rango entre $[15, 30]$. Para este trabajo se decide ampliar el rango de vecinos a un rango de $[12, 80]$. Eligiendo el número de vecinos que reduzca el error de identificación de impactos.

3.3.2.5. Metodología: Autoencoders (AE)

Para la construcción de los modelos de autoencoders, se utilizan dos espacios de trabajo, el primero es la bibliotecas de código abierto Keras y Tensorflow en el ambiente de programación Python, junto a el software Matlab.

La elección de los parámetros del autoencoder con el cual se realizará la reducción de dimensionalidad de los datos, se descompone en cuatro etapas:

- Elegir optimizador, número de épocas y tasa de aprendizaje.
- Elegir arquitectura del autoencoder.
- Funciones de activación de las capas de encode y decode.
- Regularizadores: Dropout, norma L_2 o normalización por lotes (Batch Normalization).

3.3.2.5.1. Optimizador, número de épocas y tasa de aprendizaje

Luego de la etapa de pre-procesamiento, se definen hiperparámetros de entrenamiento, como, la tasa de aprendizaje, el número de épocas y el optimizador. La función de costo utilizada para entrenar los modelos, es el error cuadrático medio (RMSE por sus siglas en ingles). Las funciones de activación del autoencoder, serán seleccionadas posterior a la elección de la arquitectura, usando como función de activación ReLU (Rectifier Linear Unit), para la elección estos tres hiperparámetros. El número de neuronas de las capas profundas, fueron escogidos para que este. disminuya de manera lineal en función del número de capas, hasta llegar a la dimensión intrínseca de los datos p (dimensión del espacio latente).

3.3.2.5.2. Arquitectura y pasos temporales

Para definir el número de capas ocultas del autoencoder y el número de pasos temporales que guarda la información más relevante de la señal impulsiva, se hacen pruebas en autoencoders entrenados con bases de datos que contienen P=100, 200, 400, 800, 1600 y 2000 pasos temporales, utilizando diferentes arquitecturas, dependiendo de la base de datos.

La tabla 3.4 detalla las arquitecturas que se prueban, para cada una de las bases de datos, donde horizontalmente se ven las bases de datos y verticalmente el número de capas ocultas que posee la arquitectura del autoencoder.

La figura 3.9 muestra la metodología empleada para elegir el número de capas ocultas del autoencoder y la cantidad de pasos temporales que mejor representa la señal de tipo impulso.

Tabla 3.4: Arquitecturas probadas en las bases de datos

Selección de la arquitectura y número de pasos temporales	**Cantidad de pasos temporales en la base de datos**					
Número de capas ocultas de la arquitectura	100	200	400	800	1600	2000
1	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓
3	✓		✓	✓	✓	✓
4	✓		✓	✓	✓	✓
5					✓	✓
6						✓
7						✓

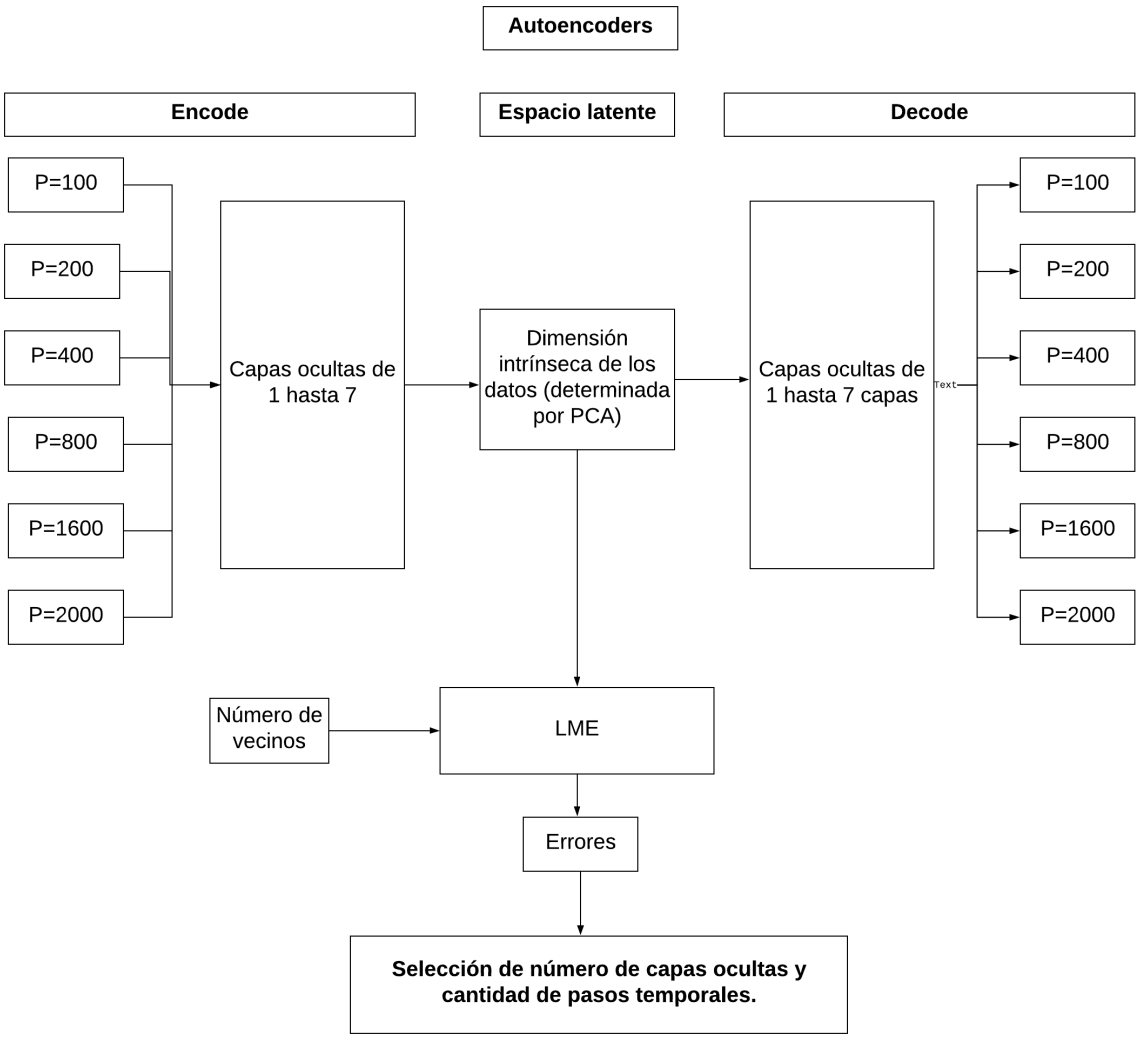

Figura 3.9: Metodología empleada para seleccionar el número de capas ocultas y paso temporal con la información más relevante de la señal impulsiva

Como se explica en el esquema 3.9, se selecciona la arquitectura y pasos temporales, que entregan un menor error de identificación de impactos.

3.3.2.5.3. Funciones de activación

Con la arquitectura y el número de pasos temporales seleccionado. Se selecciona las funciones de activación de la etapa de encode y decode del autoencoder. Para ello se prueban, las siguientes configuraciones, mostradas en la figura 3.10, teniendo un total de 16 combinaciones de funciones de activación. Se elige la o las funciones de activación que entreguen menor error de identificación de impactos.

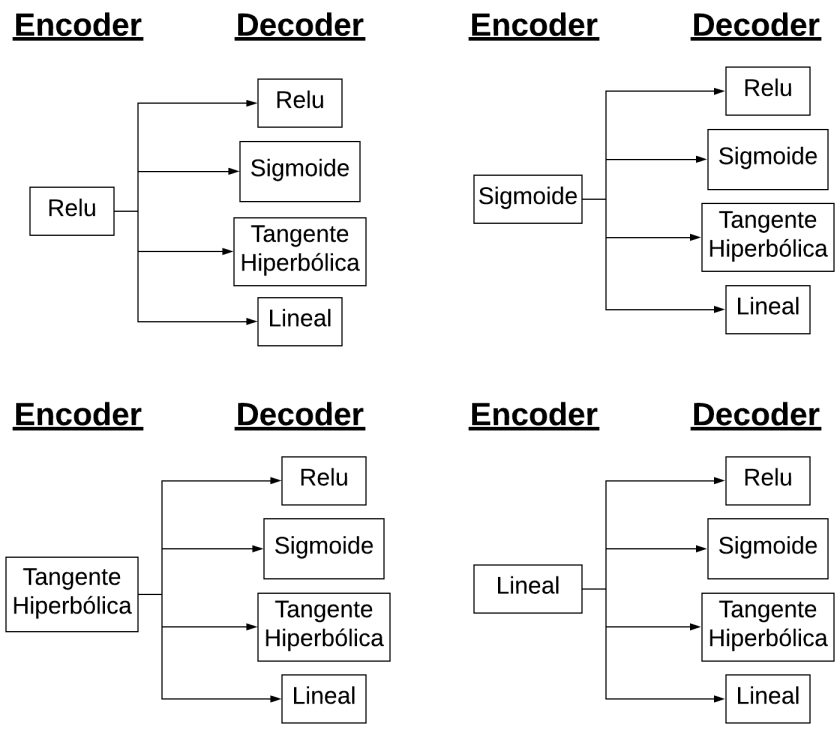

Figura 3.10: Pruebas de funciones de activación, en las etapas de encode y decode.

3.3.2.5.4. Regularizadores

Se ocupan tres métodos de regularización:

- Regularizadores L_2: Variando el parámetro λ en el intervalo de $[10^{-9}, 10^{-1}]$
- Dropout: Ocupando tasa de probabilidad de d=0.1, d=0.2, d=0.3, d=0.4.
- Batch Normalization: Se ocupa la capa de BatchNormalization() de la librería keras.

Se elige la técnica de regularización reduzca el error de impacto de identificación de impacto normalizado.

3.3.3. Metodología red neuronal convolucional

Esta estrategia busca construir un modelo de una red neuronal convolucional. La idea es que este modelo, reciba la respuesta vibratoria de los 6 sensores, y a partir de estas, encuentre la localización y cuantifique el impacto a partir de una regresión.

El funcionamiento para esta estrategia de identificación de impactos se puede ver en la figura 3.11:

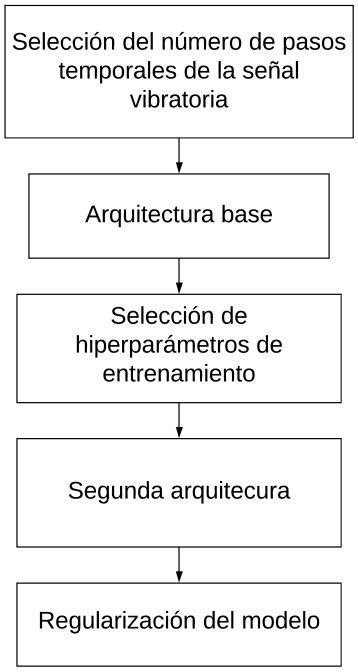

Figura 3.11: Metodología empleada para la elección del modelo

Se seleccionan aquellos parámetros que entreguen menor error de identificación de impactos en los datos de prueba.

3.3.3.1. Arquitectura base

Se selecciona una arquitectura inicial con la cual se seleccionan los siguientes parámetros:

- Optimizador
- Número de épocas máximo a entrenar
- Tasa de aprendizaje
- Condiciones del callback “Early Stopping”

3.3.3.2. Segunda arquitectura

Se selecciona una nueva arquitectura variando los siguientes parámetros:

- Número de capas convolucionales: 1,2,3,4,5,6 capas convolucionales.
- Tamaño de los kernels: k=(2,1), k=(2,2), k=(3,1), k=(3,3), k=(5,1), manteniendo kernels constantes en función de las capas convolucionales, y variando su tamaño a través de ellas.
- Número de filtros, agregando más filtros en función de la profundidad de las capas, tomando valores de 8,16,32,64,128.

3.3.3.3. Regularización del modelo

Se utilizan tres técnicas de regularización:

- Normalización por lotes, la cual es aplicada desde la selección de la arquitectura base.
- Dropout, con tasas de deserción de d=0.1, d=02, d=0.3 y d=0.4.
- Regularización por pesos, norma L_2 con parámetro $\lambda = 10^{-4}$ y $\lambda = 10^{-5}$

3.4. Evaluación de las estrategias de identificación de impacto

Para la determinación de los errores en los impactos, se utilizan las ecuaciones 2.77, 2.78, 2.79, 2.80. Lo que entrega los errores promedio tanto en la localización, como en la magnitud de los impactos. Además de la determinación del mínimo error de identificación de impactos normalizado, usando la ecuación 2.81, esta última convirtiéndose en la métrica de evaluación de cada uno de los algoritmos utilizados.

Capítulo 4

Resultados

4.1. Algoritmo PCA+LME

4.1.1. Dimensión intrínseca de los datos

Corresponde a la dimensión a la cual fueron reducidos los datos a través de la técnica de reducción de dimensionalidad PCA, conservando un 99.99 % de la varianza de estos. La dimensión intrínseca es usada en próximas técnicas de reducción de dimensionalidad como el tamaño del espacio latente.

Tabla 4.1: Dimensión intrínseca de los datos

Dimensión original (cantidad de pasos temporales, P / por sensor)	**Dimensión intrínseca (para cada sensor)**	**Dimensión del conjunto de datos global (con los 6 sensores)**
100	**49**	294
200	**68**	408
400	**81**	486
800	**90**	540
1600	**93**	558
2000	**93**	558

De la tabla 4.1 se puede observar que a partir de 800 pasos temporales, el valor de la dimensión intrínseca casi no aumenta, por lo que a partir de P=800, no se esta agregando información tan relevante, como para pasos temporales menores.

4.1.2. Evaluación de PCA+LME, en función del número de vecinos del algoritmo LME

Los gráficos mostrados a continuación, muestran los errores promedio en la localización de impactos, coordenada X e Y, E_X e E_Y , el error porcentual de área E_A, el error porcentual de cuantificación de la fuerza E_F y el error de identificación de impacto normalizado E_I, para conjuntos de datos que contienen cantidades de pasos temporales (P) diferentes. Estos errores están graficados en función del número de vecinos del algoritmo LME.

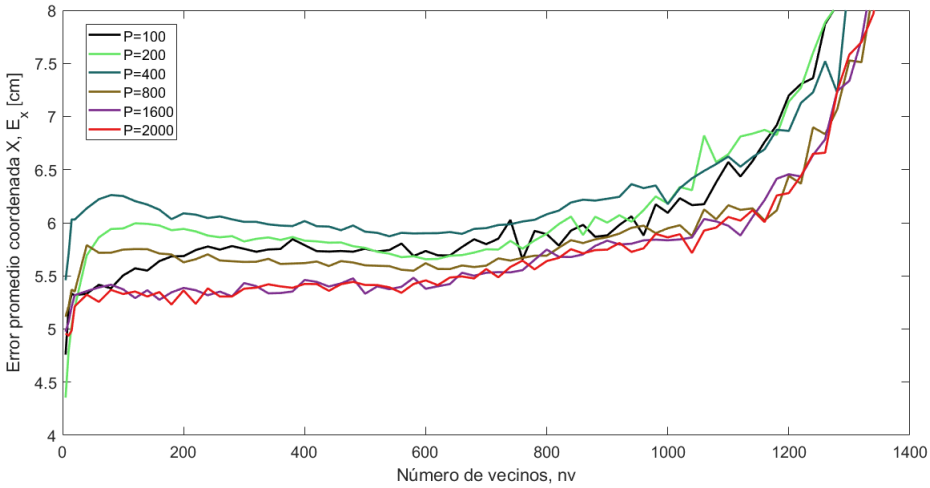

Figura 4.1: Errores promedio en la coordenada que X en función del número de vecinos seleccionados por el algoritmo LME, para pasos temporales de: P=100, P=200, P=400, P=800, P=1600, P=2000.

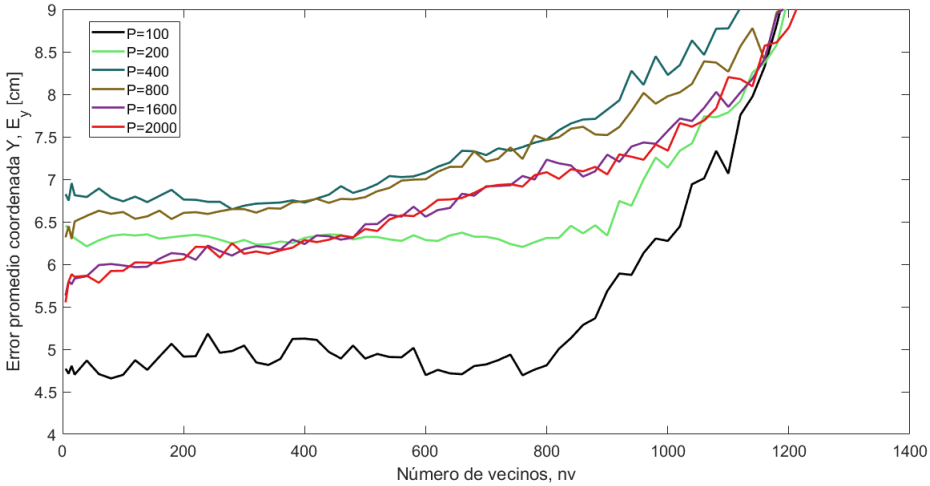

Figura 4.2: Errores promedio en la coordenada que Y en función del número de vecinos seleccionados por el algoritmo LME, para pasos temporales de: P=100, P=200, P=400, P=800, P=1600, P=2000.

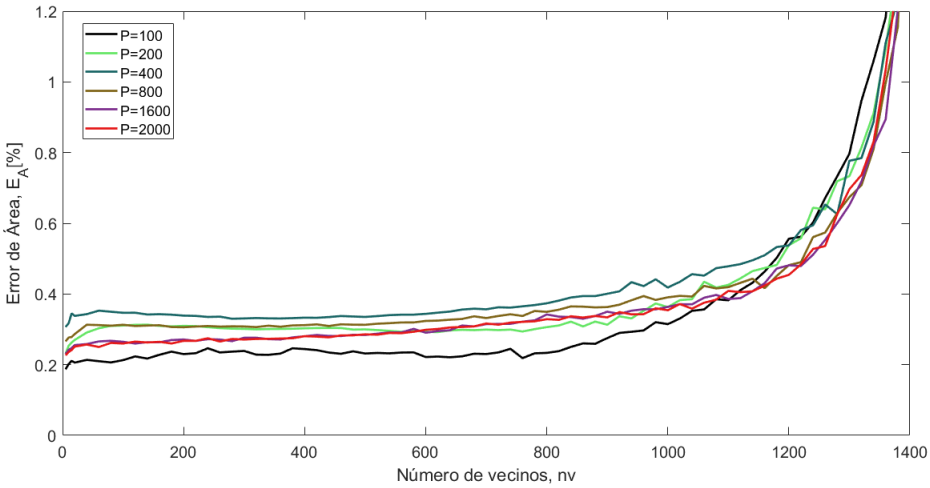

Figura 4.3: Errores porcentuales de área en función del número de vecinos seleccionados por el algoritmo LME, para pasos temporales de: P=100, P=200, P=400, P=800, P=1600, P=2000.

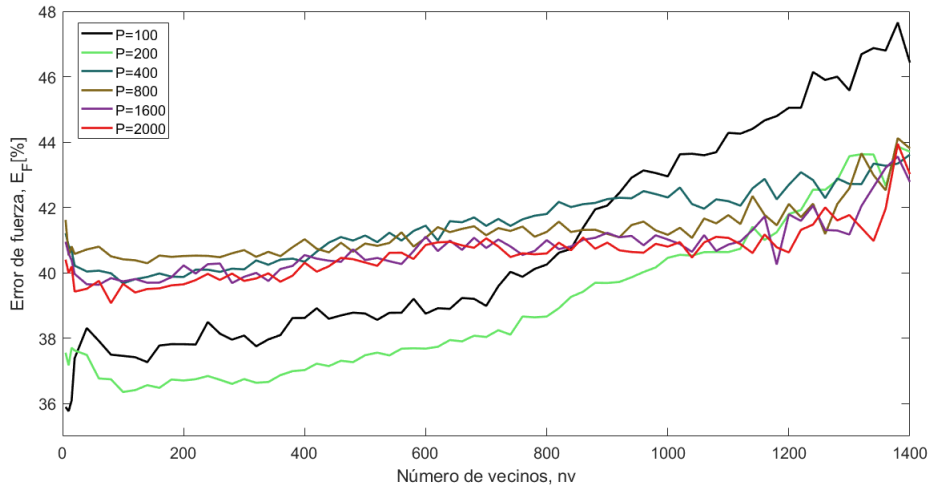

Figura 4.4: Errores porcentuales de cuantificación de la fuerza en función del número de vecinos seleccionados por el algoritmo LME, para pasos temporales de: P=100, P=200, P=400, P=800, P=1600, P=2000.

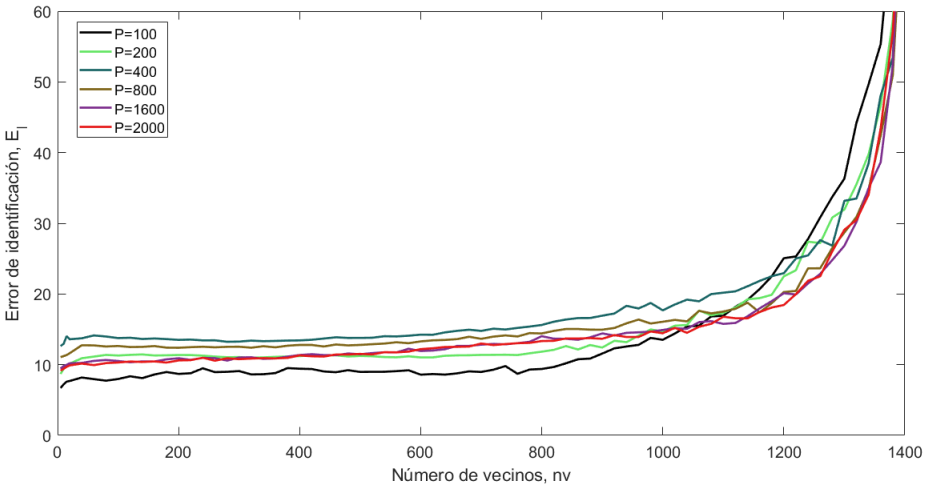

Figura 4.5: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo LME, para pasos temporales de: P=100, P=200, P=400, P=800, P=1600, P=2000.

De las figuras 4.1, 4.2, 4.3, 4.4 y 4.5 se observa menos error para un número de vecinos entre 0 y 200, a pesar que los errores son bastante constante hasta $\approx$ 800 vecinos (creciendo al usar una cantidad mayor a 800). El menor error de identificación se obtuvo con 5 vecinos.

El número de pasos temporales considerados de la señal vibratoria, influye notoriamente en los resultados, encontrando una variación entre el menor y más alto de E_I del orden del 10 % (esto considerando el número de vecinos que minimiza el error para cada paso temporal). A pesar de lo anterior, no se puede decir nada concluyente respecto a la cantidad de pasos temporales a elegir, los mejores resultados se encontraron en orden creciente ([%]) para P=100, P=200, P=2000, P=1600, P=800 y P=400.

La tabla 4.2 muestra los parámetros y resultados del algoritmo PCA+LME con los cuales se obtiene un menor error de identificación de impacto normalizado.

Tabla 4.2: Parámetros y resultados del algoritmo PCA+LME con mejor rendimiento

Parámetros y resultados	**Valor**
Pasos temporales de la señal vibratoria	P=100
Número de vecinos del algoritmo LME	5
Error promedio de localización en la coordenada X	4.75 [cm]
Error promedio de localización en la coordenada Y	4.77 [cm]
Error porcentual de área	0.187 [%]
Error de cuantificación de fuerza	35.89 [%]
Error de identificación de impacto normalizado	6.71 [%]

4.1.3. Evaluación PCA+LME en el conjunto de prueba

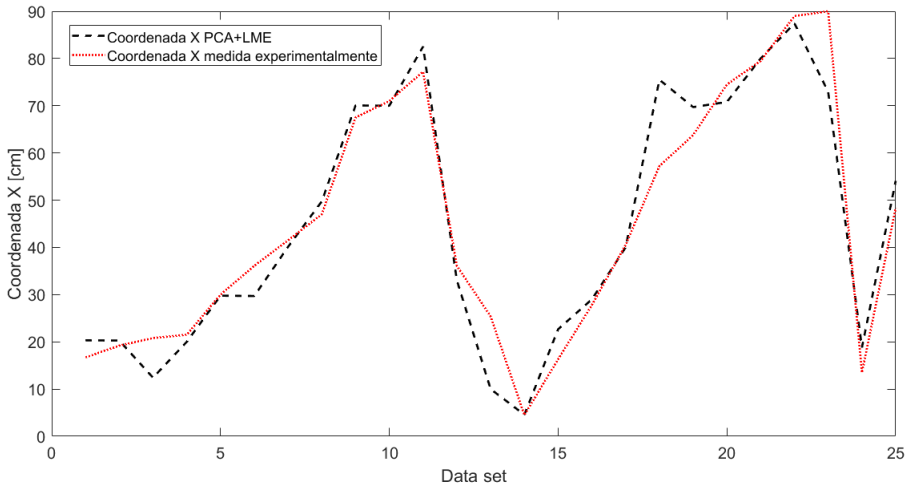

Figura 4.6: Error de localización de la coordenada X, en función de los datos de prueba

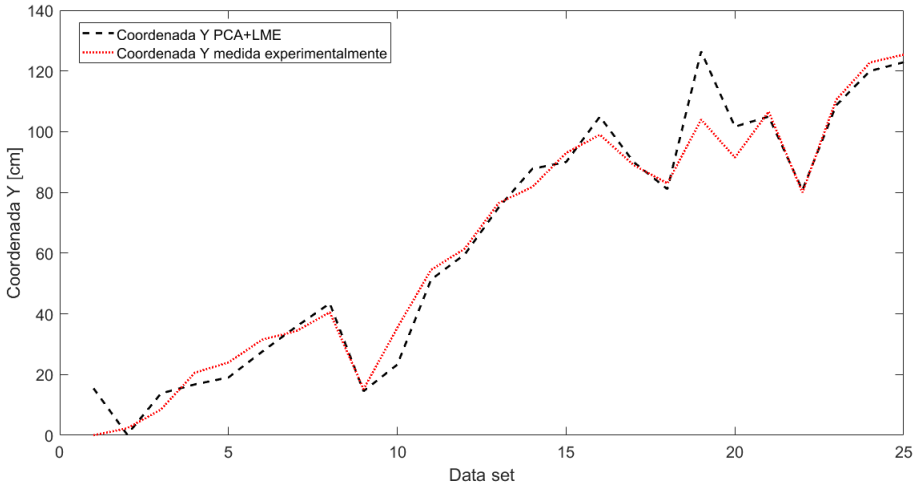

Figura 4.7: Error de localización de la coordenada Y, en función de los datos de prueba

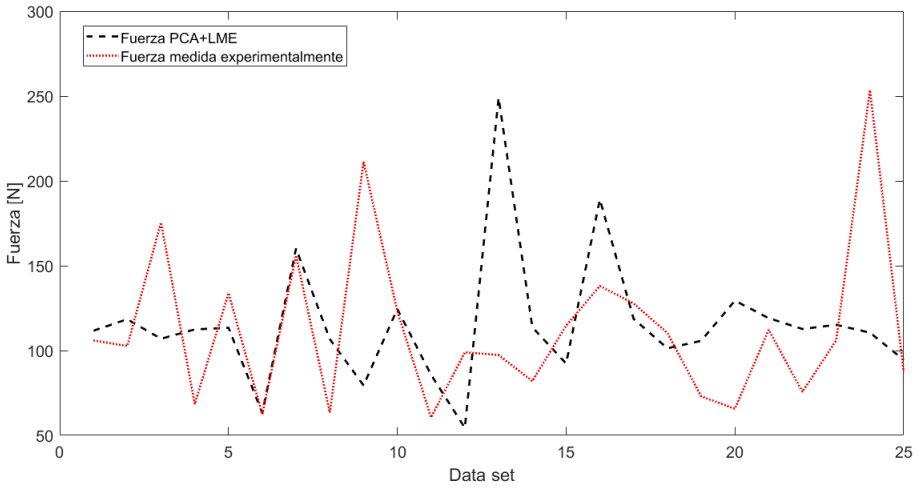

Figura 4.8: Error de localización de la coordenada X, en función de los datos de prueba

En las figuras 4.6, 4.7 y 4.8 se observa que el algoritmo PCA+LME, tiende a ser más preciso prediciendo la localización del impacto, que su magnitud, en la cual se observa gran discordancia con la señal medida experimentalmente.

4.2. Algoritmo AE+LME

4.2.1. Optimizador, número de épocas y tasa de aprendizaje

El optimizador usado en la construcción de los autoencoders fue el ADAM (Adaptative momentum), este optimizador se encuentra en el estado del arte de los optimizadores, guarda las ventajas de los optimizadores AdaGrad y RMSProp, además, hiperparámetros como la tasa de aprendizaje, tienen una interpretación intuitiva y generalmente requieren poca sintonización.

Al tratarse de una prueba de múltiples modelos, con distintos conjuntos de datos (bases de datos con diferente número de time steps: 100, 200, 400, 800, 1600 y 2000) una técnica regularmente usada para determinar el número de épocas, previniendo la situación de overffiting en el modelo, es el EarlyStopping un callback de keras, que permite detener el entrenamiento, una vez que se considere que no hay más progreso en el entrenamiento del AE, esta función trabaja con una variable de monitoreo (en este caso error RMSE de validación), un δ de tolerancia, que es el valor que debe disminuir la variable de monitoreo, para considerarse como "*progreso*", y un número de épocas de paciencia, que corresponde a la cantidad de épocas a esperar sin que se produzca "*progreso*" en el entrenamiento. Además este callback, proporciona la posibilidad de recuperar los pesos que obtuvieron mejor rendimiento en la etapa de entrenamiento.

A pesar del uso de Adam como optimizador y de EarlyStopping, se deben seleccionar los parámetros de la tasa de aprendizaje inicial, número de épocas máximo de entrenamiento y condiciones del EarlyStopping (número de épocas de paciencia y δ). Dado que son seis modelos los que deben ser entrenados bajo las mismas condiciones (un modelo por conjunto de datos, un conjunto de datos por sensor) se eligen los parámetros con los cuales se obtiene un menor error promedio de validación RMSE para los 6 modelos entrenados, sin embargo hay que tener en consideración que el AE es usado como una etapa de previa al algoritmo LME y esta pensado como una técnica de reducción de dimensionalidad para mejorar este algoritmo, por lo tanto si bien para la tasa de aprendizaje, número de épocas máximo y condiciones del EarlyStopping se utiliza este criterio (con el fin de minimizar el número de pruebas) para otros hiperparámetros, como, número de capas ocultas, funciones de activación, tipo de regularización, el criterio de selección de parámetros será la minimización del error de identificación de impacto.

Para la elección de la tasa de aprendizaje inicial, se eligió el error de validación promedio para los 6 sensores, del modelo más simple, un modelo con 1 capa oculta, la función de activación relu, considerando 100 time steps de la señal vibratoria y un número de neuronas de p=49, correspondiente a la dimensión intrínseca de 100 time steps (ver 4.1).

Al analizar el desempeño del AE para variados valores de estos parámetros, se obtiene: Para la elección de la tasa de aprendizaje inicial las figuras 4.9 y 4.10, para el número de épocas, la tabla 4.3.

Tabla 4.3: RMSE de validación en función del número de épocas

Epocas	**RMSE de validación promedio de los 6 sensores**
100	1.460E-02
500	4.200E-03
1000	**9.970E-04**
2000	4.678E-04
3000	3.612E-04
4000	3.097E-04
5000	2.863E-04
10000	2.211E-04
20000	**1.765E-04**
30000	1.585E-04
40000	1.508E-04
50000	1.470E-04

Como se observa en la tabla 4.3, a partir de 1000 épocas de entrenamiento se obtiene un valor de la función de costo del mismo orden de magnitud. A raiz de esto, se seleccionan los hiperparámetros restantes, entrenando los modelos por 1500 épocas como máximo de épocas iniciales (más de 1000), y con las condiciones del EarlyStopping de δ=5 $\cdot 10^{-4}$, el cual corresponde a la disminución del error de validación entre las 1000 y 2000 épocas (ver tabla 4.3). Se ocupa una paciencia de 500 épocas, por lo que los modelos generalmente se detienen entre las 1000 y 1500 épocas de entrenamiento.

A pesar de que el error promedio de validación en los autoencoders para datos de los 6 sensores, siempre disminuye, a partir de $\approx$ 20000 épocas de entrenamiento, el error de validación comienza a subir en dos modelos (el promedio sigue disminuyendo), por lo que hay sobre ajuste en estos modelos. Por esta razón se decide no entrenar más de 20000 épocas. En las etapas finales de elección de parámetros, los modelos serán entrenados por 20000 épocas.

Para la tasa inicial de aprendizaje, se hace una búsqueda por cuadrícula, se busca en primero en un rango amplio que de valores que es $[10^{-7}, 10^{-1}]$, para luego acotar la búsqueda al intervalo de $[5 \cdot 10^{-4}, 10^{-2}]$ que es donde se obtienen mejores resultados. El criterio que se eligió fue aquel que obtenga un menor error promedio de validación en los 6 modelos.

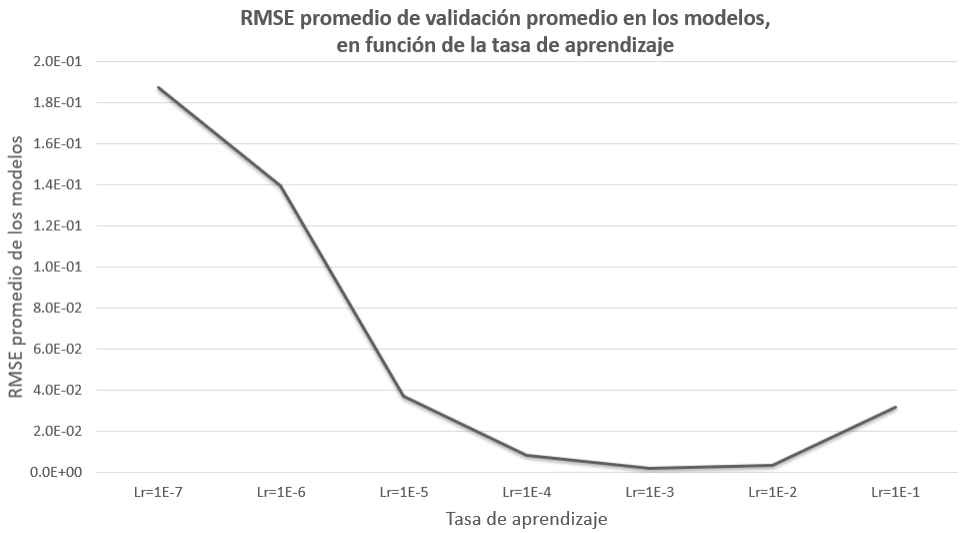

Figura 4.9: RMSE promedio de validación en función de la tasa de aprendizaje en el rango de valores de $[10^{-7}, 10^{-1}]$

Se continua la búsqueda de la mejor tasa de aprendizaje, en el rango de valores de $[5 \cdot 10^{-4}, 10^{-2}]$.

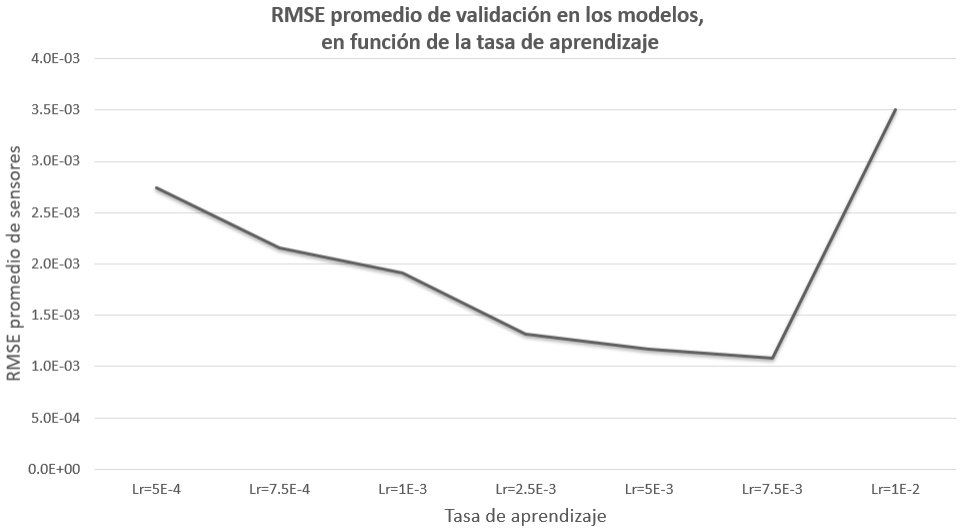

Figura 4.10: RMSE promedio de validación en función de la tasa de aprendizaje en el rango de valores de $[5 \cdot 10^{-4}, 10^{-2}]$

Con esto en se selecciona una tasa de aprendizaje inicial de $7.5 \cdot 10^{-3}$, la cual mejora el rendimiento promedio de los modelos para los 6 sensores.

4.2.2. Número de capas ocultas y pasos temporales con mejores resultados

Se entrenan modelos con diferentes números de capas, con distintas bases de datos (que contienen los diferentes cantidades de time steps), se utiliza la función de activación ReLU en cada uno de ellos. Estas pruebas determinan los pasos temporales que contiene mejor información para el problema de identificación de impactos y el número de capas ocultas que debe tener el autoencoder para que el error de identificación de impactos en el algoritmo LME sea menor. Los resultados son mostrados en la tabla 4.4.

La cantidad de neuronas por capa fue elegida de modo que descienda de manera lineal hasta la dimensión intrínseca de cada conjunto de datos (ver 4.1); Por ejemplo si P=400 (pasos temporales) y entonces p=81 (dimensión intrínseca), y se construye un autoencoder con 3 capas ocultas en la etapa de codificación, este tendrá la arquitectura 400-294-187-81-187-294-400 neuronas por capa.

Tabla 4.4: Mínimo error de identificación de impacto de los algoritmos AE+LME, para arquitecturas de autoencoder distintas y con distintas cantidades de pasos temporales en los conjuntos de datos. El mínimo fue escogido para un número de vecinos del algortimo LME en un rango de valores de 1 a 1400

Error de identificación de impacto [%]	**Pasos temporales**					
Número de capas ocultas(etapa encode)	**100**	**200**	**400**	**800**	**1600**	**2000**
1	**8.75**	22.77	25.47	21.31	29.80	27.45
2	12.04	21.45	28.38	30.62	28.60	27.65
3	29.71	-	27.93	25.65	23.22	20.14
4	27.00	-	25.72	19.47	17.55	15.87
5	-	-	-	-	22.42	106.87
6	-	-	-	-	-	52.24
7	-	-	-	-	-	30.60

De la tabla 4.4 se puede observar que el número de pasos temporales que minimiza el error de identificación de impactos es P=100, este es un resultado bastante concluyente, ya que concuerda con lo obtenido previamente con el algoritmo PCA+LME. Además se encontró que el número de capas ocultas que reduce el error de identificación es 1 capa, arquitectura que se utiliza en las siguientes pruebas para determinar el resto de hiperparámetros.

4.2.3. Funciones de activación

Una vez determinado el número de time steps y el número de capas ocultas del AE, se determinan las funciones de activación de este. La tabla 4.5 muestra las distintas combinaciones de funciones de activación que se probaron, tanto para la etapa de encode como decode del AE.

Tabla 4.5: Mínimo error de identificación de impacto de los algoritmos AE+LME, para distintas funciones de activación y para un número de vecinos del algortimo LME en un rango de valores de 1 a 1400

Mínimo error de identificación de impacto [%]	**Decode**			
Encode	Lineal	Relu	**Sigmoide**	Tangente hiperbólica
Lineal	11.57	10.47	14.82	7.35
Relu	12.35	8.76	6.18	9.82
Sigmoide	8.10	8.75	**5.87**	8.86
Tangente hiperbólica	12.15	11.33	6.23	12.53

De la tabla 4.5 se puede observar que la función de activación que reduce el error de identificación de impacto, y que por lo tanto mejor se adapta a este problema, es la función sigmoide.

4.2.4. Regularizadores

Si bien los regularizadores podrían no ser necesarios ya que el modelo de autoencoder que mejor trabaja junto al algoritmo LME, es bastante simple y esta en una situación de "underffiting", estos podrían llegar a ser útiles, ya que permitirían que la red sea capaz de generalizar mejor y por lo tanto entregar un espacio latente más robusto al algoritmo LME, reduciendo el error de identificación de impactos.

Se ocupan tres técnicas de regularización: Por pesos (norma L_2) dropout y normalización por lotes (capas de BatchNormalization). Las tres técnicas de regularización empleadas, se aplican a cada modelo por separado y se elige aquella que entregue un menor error de identificación de impactos.

4.2.4.1. Regularización L_2

Se hace un grid search para determinar el parámetro λ de la regulación L_2, los cuales son mostrados en la figura 4.11 y en la tabla 4.6.

Figura 4.11: RMSE promedio en los 6 sensores para el parámetros λ de regularización

Tabla 4.6: RMSE de validación promedio en los 6 sensores en función del parámetro λ de la regularización L_2.

Parámetro λ	**RMSE promedio en los 6 sensores**
lambda=1E-9	4.522E-03
lambda=1E-8	3.952E-03
lambda=2.5E-8	4.073E-03
lambda=5E-8	2.650E-03
lambda=7.5E-8	**2.600E-03**
lambda=1E-7	3.283E-03
lambda=1E-6	5.183E-03
lambda=1E-5	1.093E-02
lambda=1E-4	3.322E-02
lambda=1E-3	8.428E-02
lambda=1E-2	3.067E-02
lambda=1E-1	3.018E-02

Del gráfico 4.11 y de la tabla 4.6 se obtiene el parámetro λ=7.5 10^{-8}, con el cual se entrena un AE y se prueba junto al algoritmo LME. La figura 4.12 muestra el error de identificación de impactos para número de vecinos de LME entre 5 y 1400.

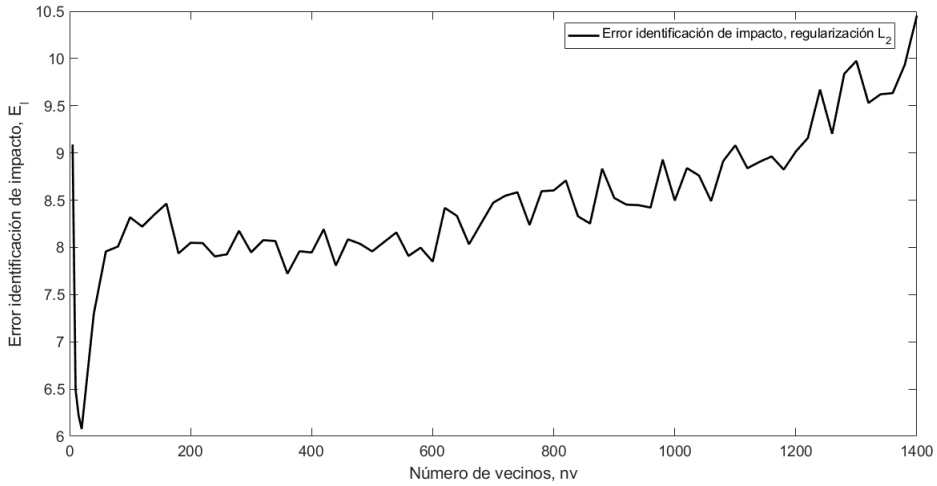

Figura 4.12: Error de identificación en el AE entrenado con regularización L_2 (λ=7.5 10^{-8}), en función del número de vecinos de LME

El mínimo de la figura 4.12 se encuentra con 20 vecinos del algoritmo LME y tiene un error de identificación de impactos de E_I =6.077 %.

4.2.4.2. Dropout

Se entrenan 3 autoencoders con distintas tasas de abandono, probándose junto al algoritmo LME, los resultados son mostrados en la figura 4.13 y sus mínimos mostrados en la tabla 4.7

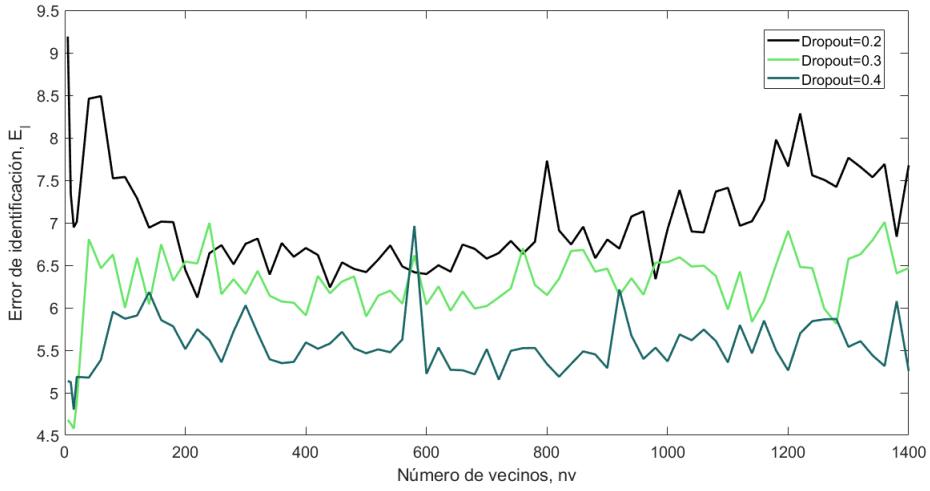

Figura 4.13: Error de identificación en autoencoders entrenados con distintos dropout

Tabla 4.7: Mínimo error de identificación de impacto, utilizando un autoencoder entrenado con regularización de dropout.

Dropout	**Mínimo error de identificación de impacto [%]**
d=0.2	6.12
d=0.3	**4.58**
d=0.4	4.80

Así, de la tabla 4.7 se observa que la tasa de dropout que minimiza el error de identificación de impacto es de d=0.3 con un error de identificación de impactos de 4.58 %

4.2.4.3. Normalización por lotes (BatchNormalization)

La figura 4.14 muestra el error de identificación normalizado en función del número de vecinos del algoritmo LME, encontrando un mínimo para 10 vecinos y un error de 7.82 %.

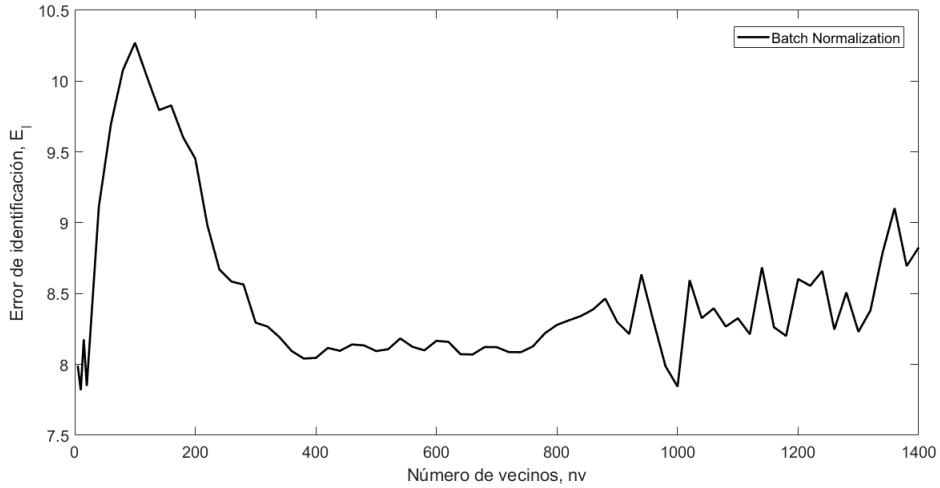

Figura 4.14: Error de identificación en autoencoders entrenados con capas de Batch Normalization

Cabe remarcar que esta técnica tuvo mejor rendimiento en la reducción del RMSE tanto de entrenamiento como de validación con respecto a las anteriores técnicas de regularización, no obstante, estos resultados no condujeron a mejoras en el error de identificación de impactos.

4.2.4.4. Resumen regularización

Tabla 4.8: Tabla resumen con las técnicas de regularización empleadas junto al algoritmo LME.

Técnica de regularización	**Error de identificación de impactos mínimo** E_I **[%]**
Regularización L_2	6.07
Dropout (d=0.2)	6.12
Dropout (d=0.3)	**4.57**
Dropout (d=0.4)	4.80
Batch Normalization	7.82

La tabla 4.8 muestra un resumen de las técnicas de regularización empleadas, se observa que el dropout con tasa de abandono de 0.3 es la técnica de regularización que mejor resultado obtuvo para el problema de identificación de impactos.

4.2.5. Evaluación de AE+LME, en función del número de vecinos del algoritmo LME

Los gráficos mostrados a continuación, muestran los errores promedio en la localización de impactos, coordenada X e Y, E_X e E_Y , el error porcentual de área E_A, el error porcentual de cuantificación de la fuerza E_F y el error de identificación de impacto normalizado E_I, para los los parámetros con los que se obtuvo mejores resultados. Estos errores están graficados en función del número de vecinos del algoritmo LME.

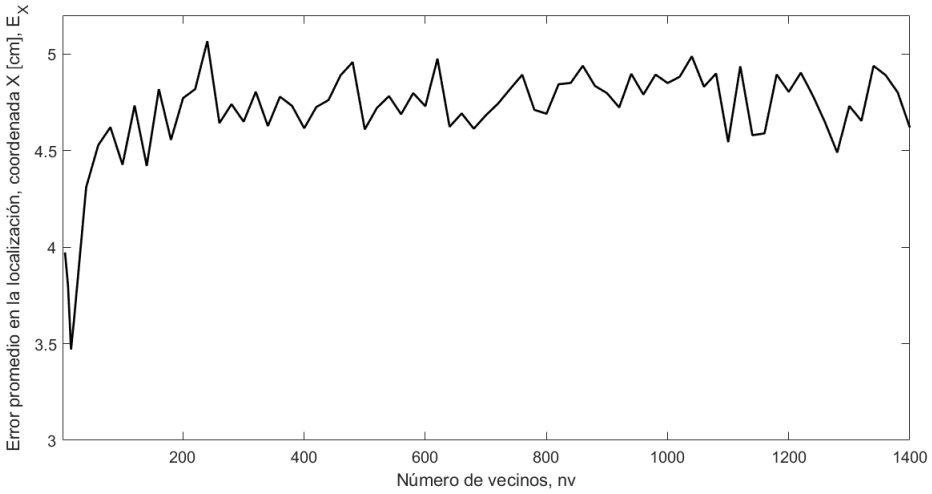

Figura 4.15: Errores promedio en la coordenada que X en función del número de vecinos seleccionados por el algoritmo LME

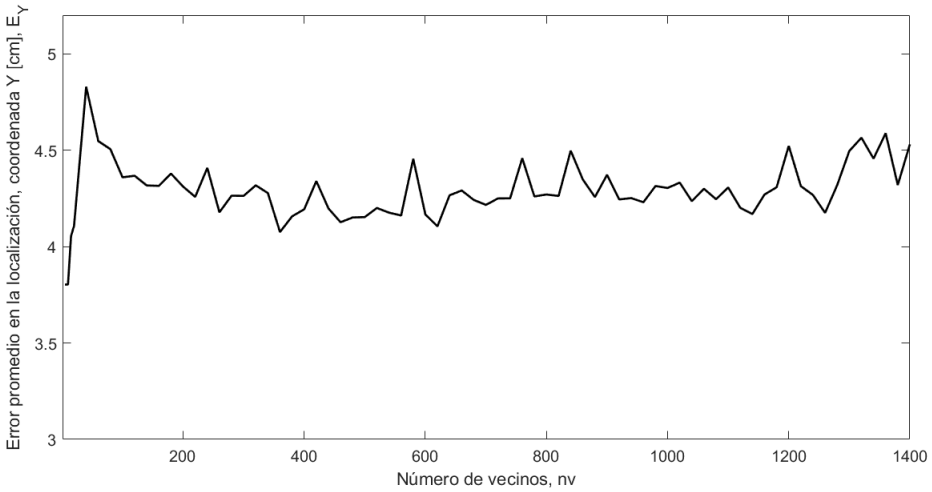

Figura 4.16: Errores promedio en la coordenada que Y en función del número de vecinos seleccionados por el algoritmo LME

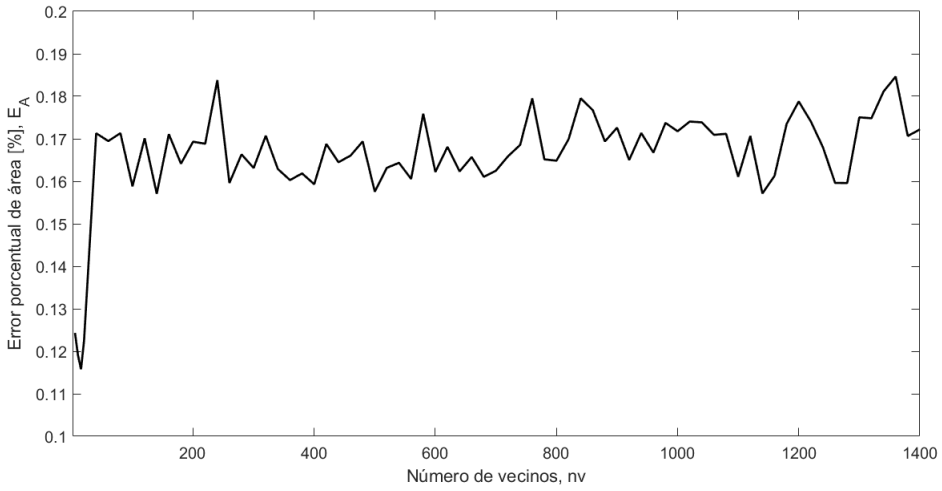

Figura 4.17: Error porcentual de área en función del número de vecinos del el algoritmo LME

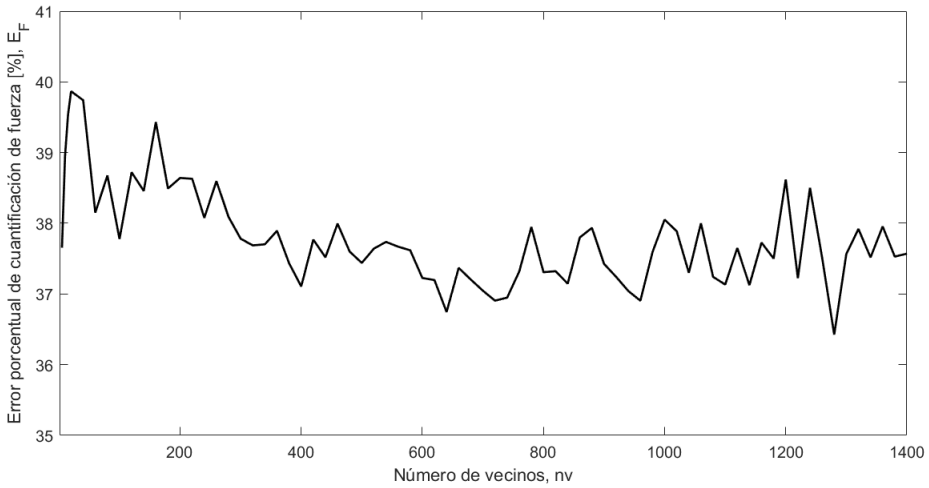

Figura 4.18: Error de cuantificación de la fuerza en función del número de vecinos del el algoritmo LME

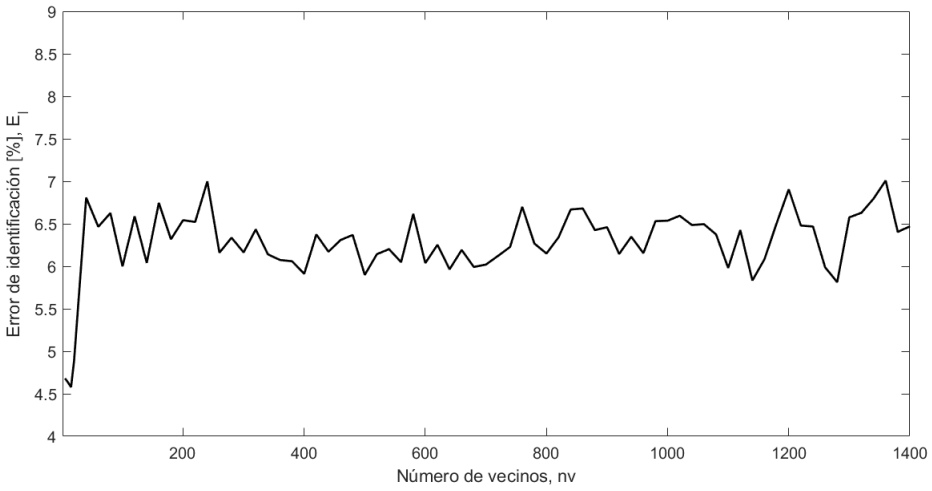

Figura 4.19: Error porcentual de identificación de impacto normalizado en función del número de vecinos del el algoritmo LME

De las figuras 4.15, 4.16, 4.17, 4.18 y 4.19 se observa una variación muy pequeña en los errores en función del número de vecinos, en la cual se obtienen errores más bajos para un número pequeño de vecinos (≤ 20). Se observa un mejor desempeño del algoritmo en la predicción de la posición del impacto que de la fuerza.

4.2.6. Evaluación AE+LME en el conjunto de prueba

- Con respecto al autoencoder con mejor rendimiento:
  - Se entrena con la base de datos que contiene 100 pasos temporales.
  - La dimensión del espacio latente es de p=49.
  - Se entrena con el optimizador ADAM
  - Tasa de aprendizaje inicial de $7.5{\cdot}10^{-3}$.
  - Se entrena por 20000 épocas.
  - Utiliza la función de activación sigmoide en la etapa de codificación como decodificación.
  - Posee un capa de dropout seguido a la capa de entrada con una tasa de abandono de 0.3.
- Se obtiene el mínimo error de identificación para 15 vecinos en el algoritmo LME.

Tabla 4.9: Resultados de los autoencoders de mejor rendimiento

Resultados del algortimo AE+LME	**Valor**
Error promedio en la localización, coordenada X	3.46 [cm]
Error promedio en la localización, coordenada Y	4.05 [cm]
Error porcentual de área	0.11 [%]
Error de cuantificación de fuerza	39.53 [%]
Error de identificación de impacto normalizado	4.57 [%]

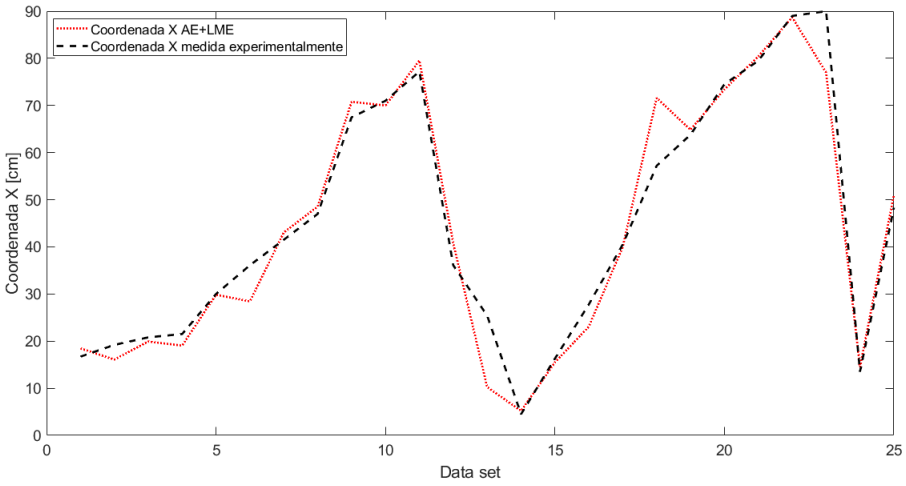

Figura 4.20: Error de localización de la coordenada X, en función de los datos de prueba

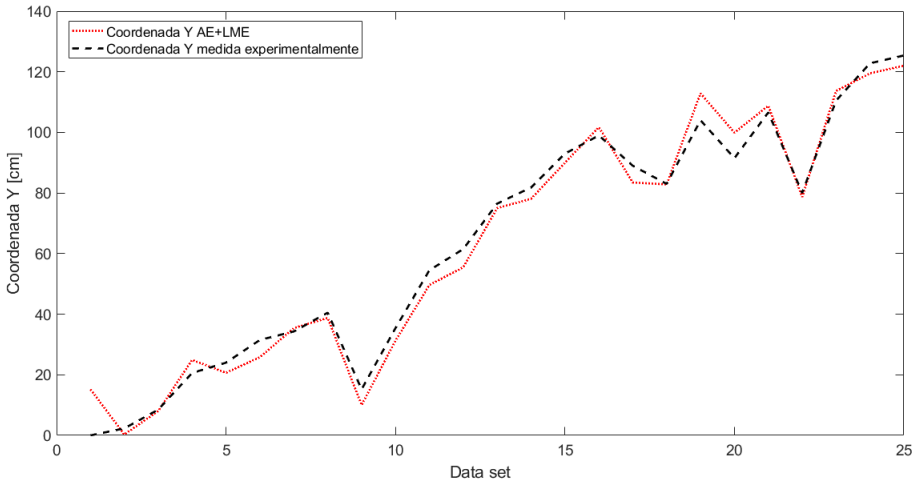

Figura 4.21: Error de localización de la coordenada Y, en función de los datos de prueba

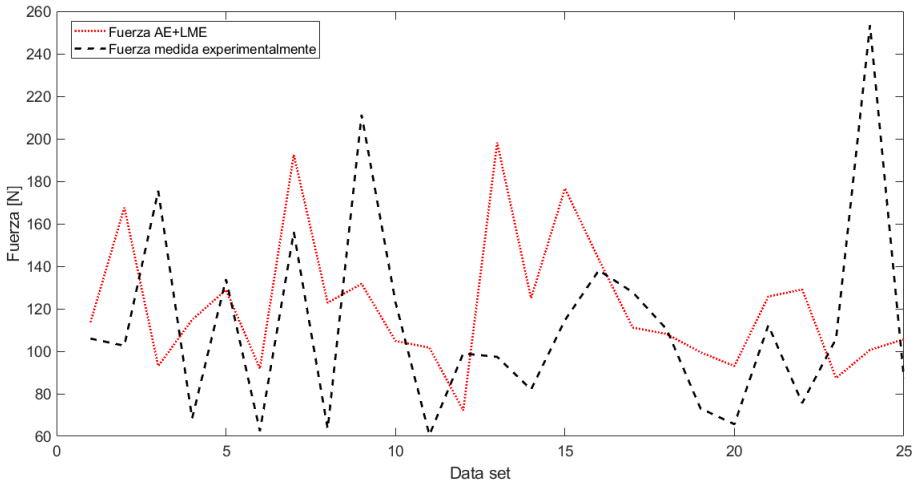

Figura 4.22: Error de cuantificación de la fuerza, en función de los datos de prueba

El algoritmo diseñado muestra gran precisión en la localización del impacto, contrariamente a lo que se ve en la cuantificación del del mismo, discordando en gran medida con los resultados medidos.

4.3. Algoritmo Kernel PCA+LME

Se aplica el método de reducción de dimensionalidad con dos tipos de kernels, kernel gaussiano y kernel polinomial.

4.3.1. Evaluación de Kernel PCA+LME, en función del número de vecinos del algoritmo LME

Los gráficos mostrados a continuación, presentan los errores promedio en la localización de impactos, coordenada X e Y, E_X e E_Y , el error porcentual de área E_A, el error porcentual de cuantificación de la fuerza E_F y el error de identificación de impacto normalizado E_I, para kernel polinomial, en el cual se ve el comportamiento en función del grado del polinomio y kernel gaussiano, en el cual se analiza el comportamiento de la variable σ (ver ecuación 2.32). Estos errores están graficados en función del número de vecinos ocupados por el algoritmo LME.

4.3.1.1. Kernel Polinomial+LME

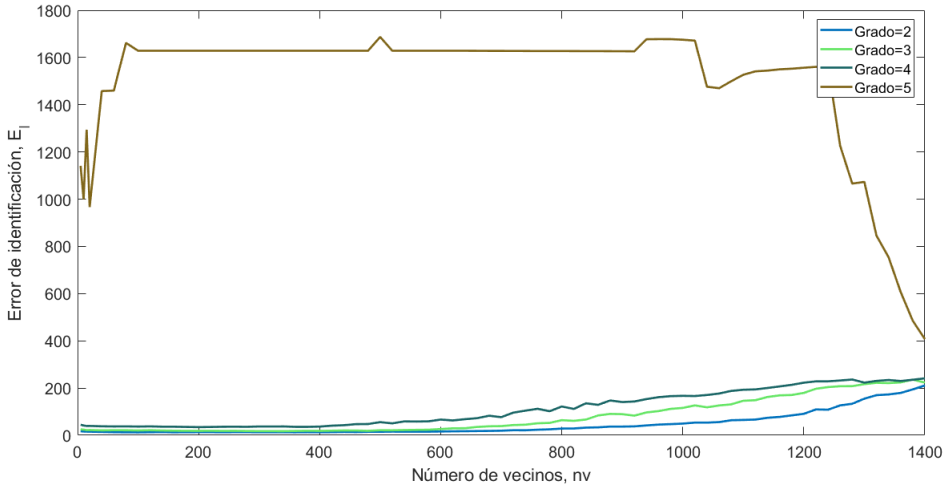

Figura 4.23: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo Kernel PCA polinomial + LME, para distintos grados del polinomio, grado=2,3,4 y 5.

De la figura 4.23 se puede observar que utilizar un kernel de grado 5 no tiene utilidad, ya que dispara el error de identificación, por lo que no se muestra en los próximos gráficos.

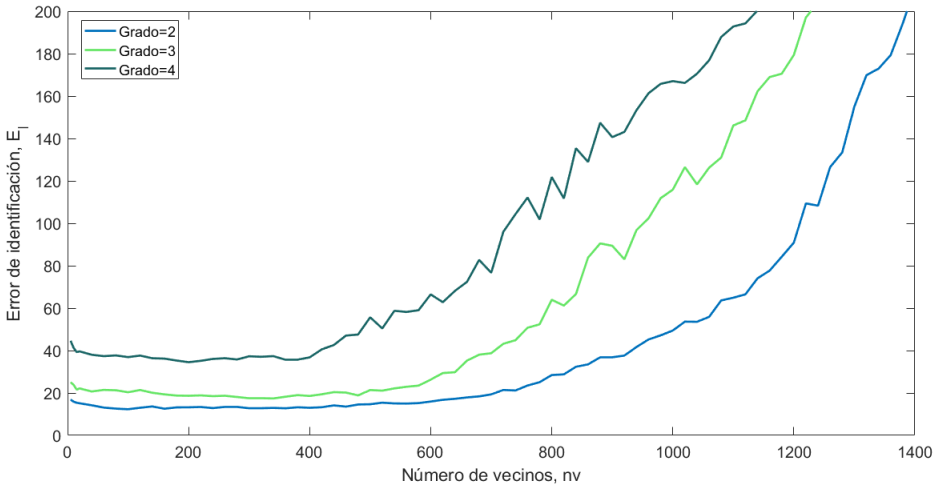

Figura 4.24: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo Kernel PCA polinomial + LME, para distintos grados del polinomio, grado=2,3 y 4.

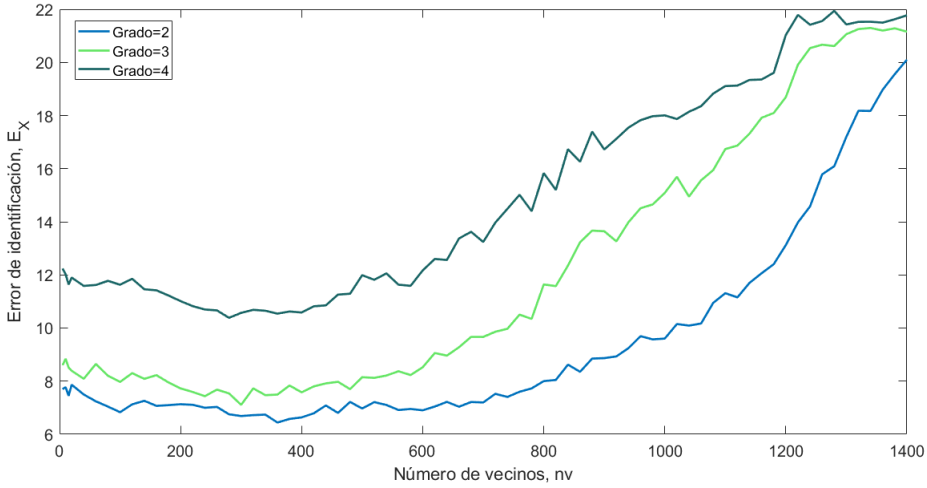

Figura 4.25: Errores promedio en la coordenada que X en función del número de vecinos seleccionados por el algoritmo Kernel PCA polinomial + LME, para distintos grados del polinomio, grado=1,2 y 3.

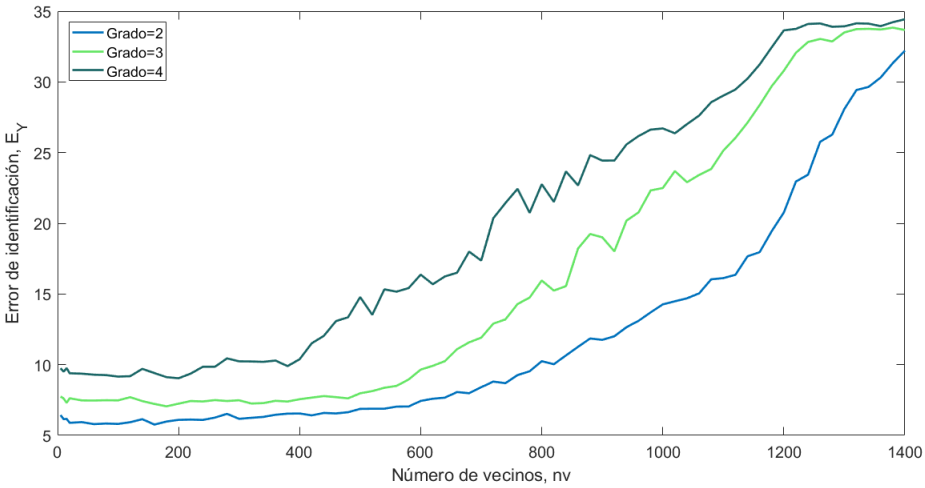

Figura 4.26: Errores promedio en la coordenada que Y en función del número de vecinos seleccionados por el algoritmo Kernel PCA polinomial + LME, para distintos grados del polinomio, grado=1,2 y 3.

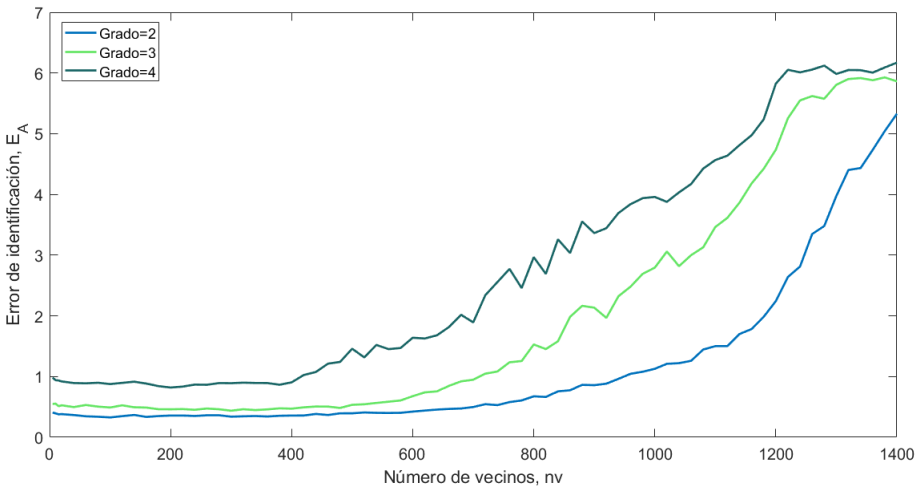

Figura 4.27: Errores porcentuales de área en función del número de vecinos seleccionados por el algoritmo Kernel PCA polinomial + LME, para distintos grados del polinomio, grado=1,2 y 3.

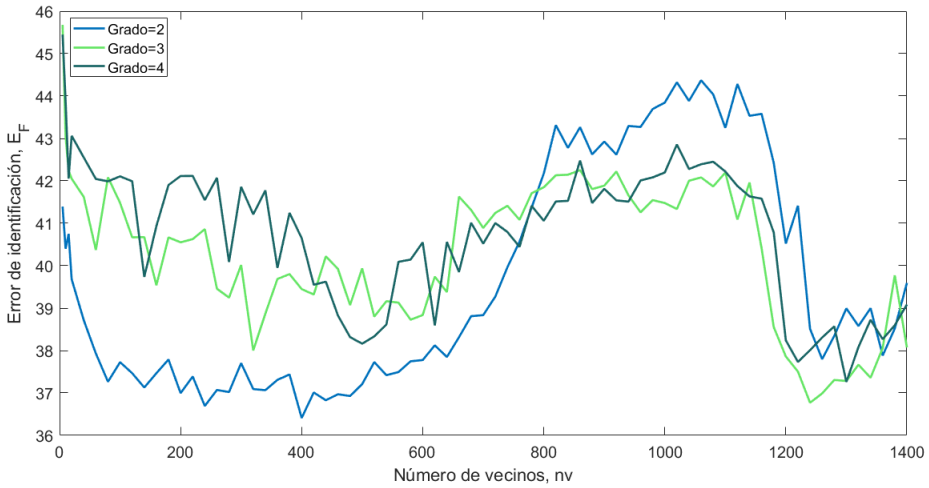

Figura 4.28: Errores porcentuales de cuantificación de la fuerza en función del número de vecinos seleccionados por el algoritmo Kernel PCA polinomial + LME, para distintos grados del polinomio, grado=1,2 y 3.

De los gráficos 4.28,4.27,4.25,4.26 y 4.24, se puede observar que el kernel de grado 2 es el que mejor ajusta los datos y del cual se puede obtener un menor error de identificación de impactos. De manera general, los resultados de este kernel funcionaron de peor manera que que las técnicas de reducción de dimensionalidad precedentes, por lo que se decidió no seguir probando con grados más altos de polinomios. No se uso el polinomio de grado 1, ya que corresponde a la aplicación de PCA.

La tabla 4.10 muestra los resultados y parámetros de del kernel polinomial que obtuvo un mejor rendimiento.

Tabla 4.10: Resultados del algoritmo Kernel PCA polinomial de grado 2+LME

Resultados del algoritmo de detección de impactos del algortimo Kernel PCA polinomial+LME	**Valor**
Número de vecinos del algortimo LME	100
Error promedio de localización en la coordenada X	6.83 [cm]
Error promedio de localización en la coordenada Y	5.81 [cm]
Error porcentual de área	0.32 [%]
Error de cuantificación de fuerza	37.72 [%]
Error de identificación de impacto normalizado	12.32 [%]

4.3.1.2. Kernel gaussiano+LME

Las figuras 4.29, 4.30, 4.31, 4.32 y 4.33 muestran el comportamiento de kernels gaussianos con diferentes valores del parámetro σ, tomando valores en un rango entre 1 y 80.

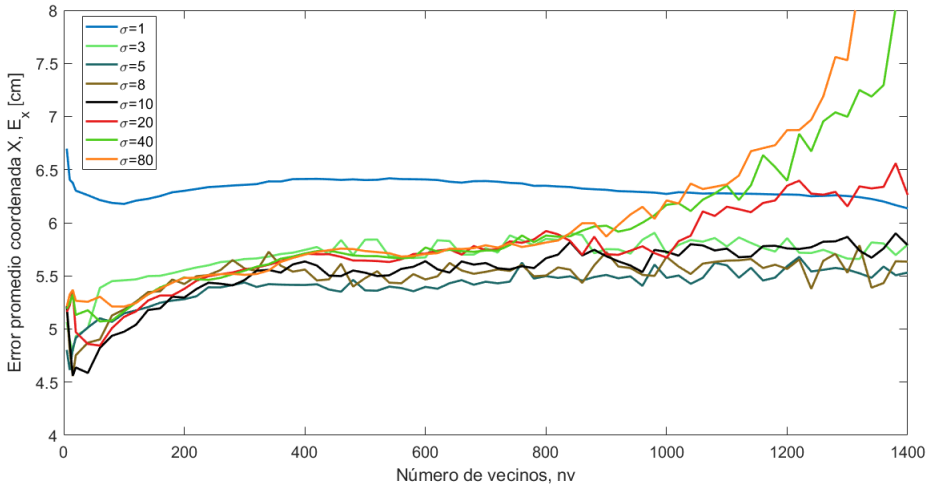

Figura 4.29: Errores promedio en la coordenada que X en función del número de vecinos seleccionados por el algoritmo Kernel PCA gaussiano + LME, para distintos parámetros σ del kernel gaussiano, σ=1,σ=3, σ=5, σ=8, σ=10, σ=20, σ=40, σ=80.

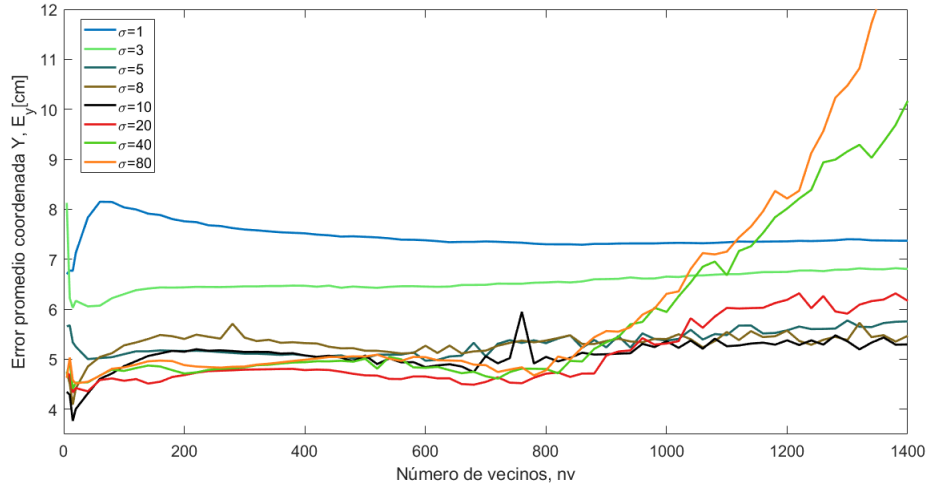

Figura 4.30: Errores promedio en la coordenada que Y en función del número de vecinos seleccionados por el algoritmo Kernel PCA gaussiano + LME, para distintos parámetros σ del kernel gaussiano, σ=1,σ=3, σ=5, σ=8, σ=10, σ=20, σ=40, σ=80.

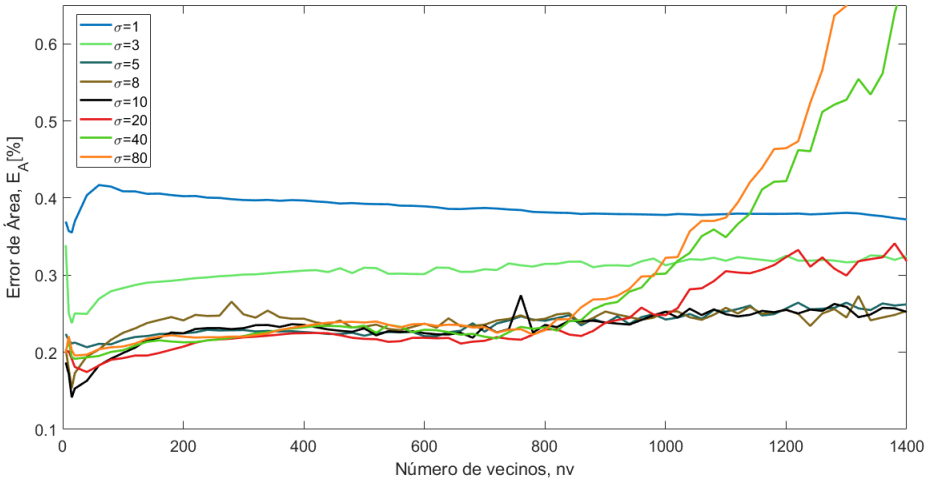

Figura 4.31: Errores porcentuales de área en función del número de vecinos seleccionados por el algoritmo Kernel PCA gaussiano + LME, para distintos parámetros σ del kernel gaussiano, σ=1,σ=3, σ=5, σ=8, σ=10, σ=20, σ=40, σ=80.

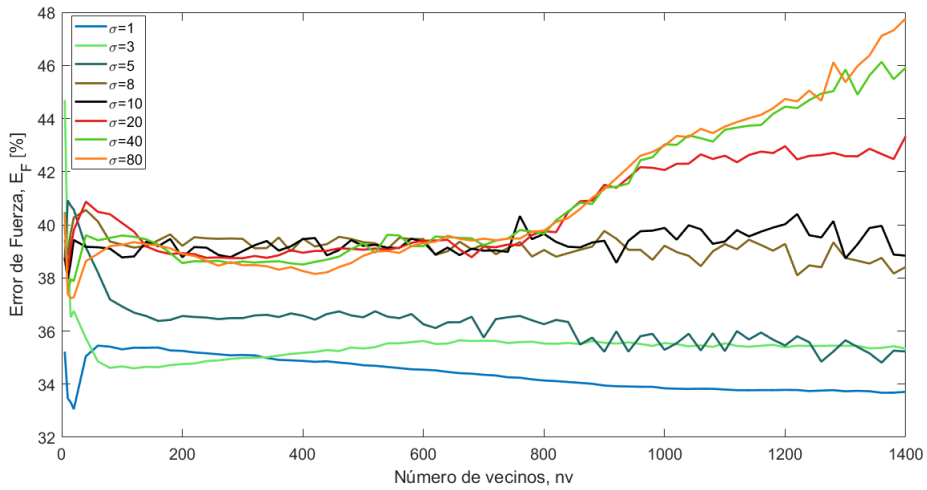

Figura 4.32: Errores porcentuales de cuantificación de la fuerza en función del número de vecinos seleccionados por el algoritmo Kernel PCA gaussiano + LME,para distintos parámetros σ del kernel gaussiano, σ=1,σ=3, σ=5, σ=8, σ=10, σ=20, σ=40, σ=80.

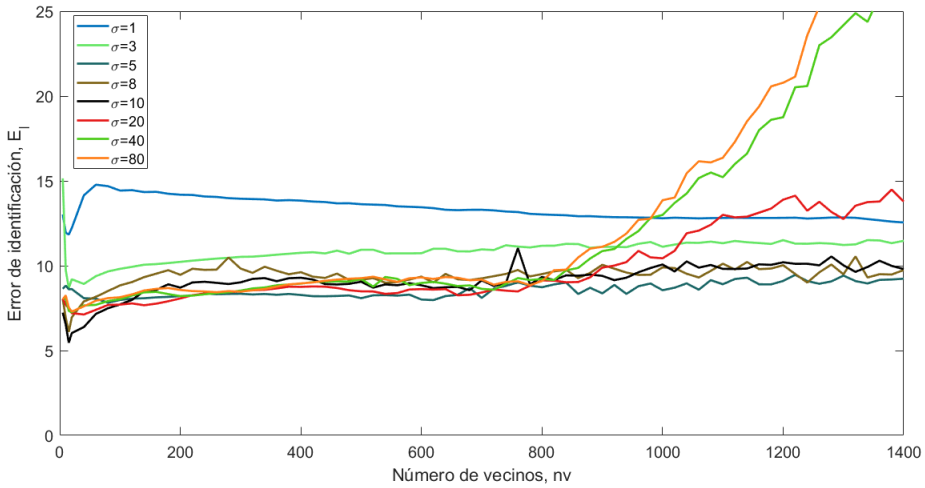

Figura 4.33: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo Kernel PCA gaussiano + LME, para distintos parámetros σ del kernel gaussiano, σ=1,σ=3, σ=5, σ=8, σ=10, σ=20, σ=40, σ=80.

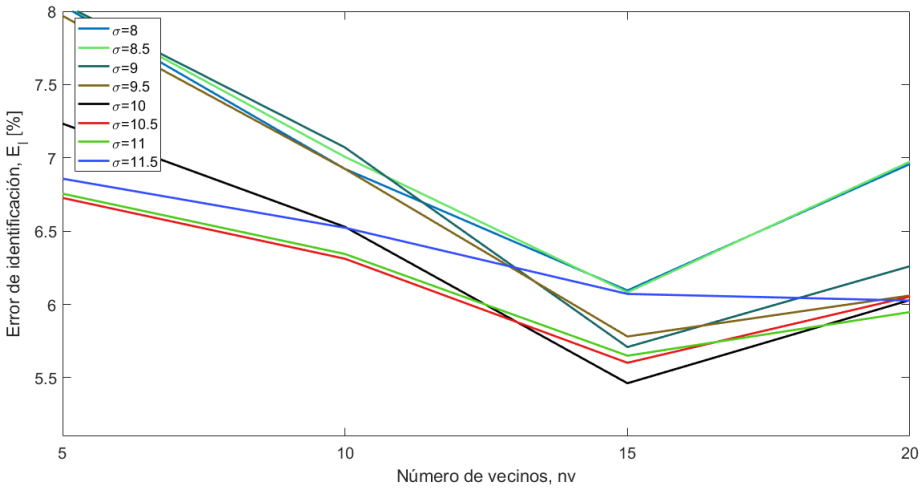

Figura 4.34: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo Kernel PCA gaussiano + LME, para distintos parámetros σ del kernel gaussiano, σ=8,σ=8.5, σ=9, σ=9.5, σ=10, σ=10.5, σ=11, σ=11.5.

Los resultados mostrados en las figuras 4.29, 4.30, 4.31, 4.32, 4.33 y **??** muestran que el kernel gaussiano con mejor resultado es aquel con parámetro σ=10.

De los gráficos 4.31 y 4.32 se puede observar que para σ más pequeño, es decir σ=1, 2 o 5, el algoritmo predice con mayor exactitud la fuerza del impacto, contrariamente a lo ocurrido con la posición, siendo parámetros entre 8 y 20 aquellos que mejor rendimiento. Por ejemplo con el parámetro σ=1, el error porcentual de fuerza es del orden de un 33 %, lo que es $\approx$ 5 % menos en comparación a los 38.6 % encontrados con el parámetro σ=10, en cuanto a la localización del impacto el parámetro σ=10 es solo 0.2 % más preciso que el parámetro σ=1. Esto puede explicarse debido al peso que tiene el error porcentual de área, en la función error de identificación de impactos.

4.3.2. Selección de parámetros y evaluación en el conjunto de prueba

Los parámetros seleccionados en kernel PCA fueron los grados del polinomio en el caso del kernel polinomial y el parámetro σ en el caso gaussiano. Cabe notar que la utilización de un kernel es para saber cual de ellos es el que exacerba mejor las características de los datos y se ajusta mejor a ellos.

Para el caso del kernel polinomial se tomaron polinomios de grado 2, 3, 4 y 5. Se obtuvieron mejores resultados utilizando kernels de grado 2, notando un notable cambio en el rendimiento del algoritmo para grados superiores a 4, aumentando el error de identificación del orden de un 100 % más en comparación a los otros grados, sin embargo kernel polinomial, obtuvo en general, peores resultados que el kernel gaussiano.

Con el kernel gaussiano, encontrar el parámetro σ que se hizo a través de una búsqueda por cuadricula en los siguientes valores: σ=1, 3 ,5, 8, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12, 20, 40, 80. Los mejores resultados se obtuvieron con un kernel gaussiano con $\sigma = 10$, siendo con el que se encontró un menor error de identificación de impactos en el algoritmo kernel PCA+LME.

La tabla 4.11 muestra los parámetros que mejor desempeño tuvieron y resultados en el conjunto de prueba.

Tabla 4.11: Parámetros y resultados del algoritmo Kernel PCA+LME con mejor rendimiento

Parámetros y resultados del algoritmo Kernel PCA+LME	**Valor**
Parámetro σ	10
Número de vecinos del algoritmo LME	15
Error promedio de localización en la coordenada X	4.56 [cm]
Error promedio de localización en la coordenada Y	3.76 [cm]
Error porcentual de área	0.14 [%]
Error de cuantificación de fuerza	38.60 [%]
Error de identificación de impacto normalizado	5.46 [%]

En las figuras 4.35, 4.36 y 4.37 muestran en un mismo gráfico lo predicho por el algoritmo kernel PCA+LME con el conjunto de prueba.

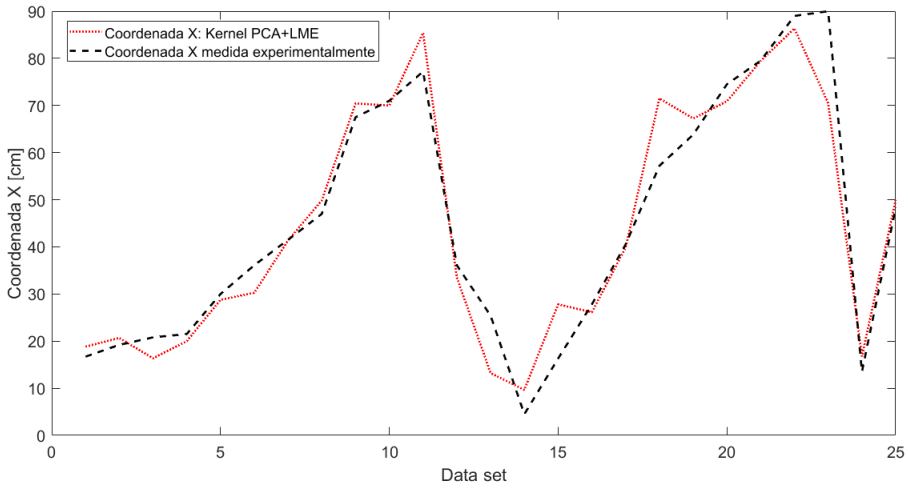

Figura 4.35: Error de localización de la coordenada X, en función de los datos de prueba

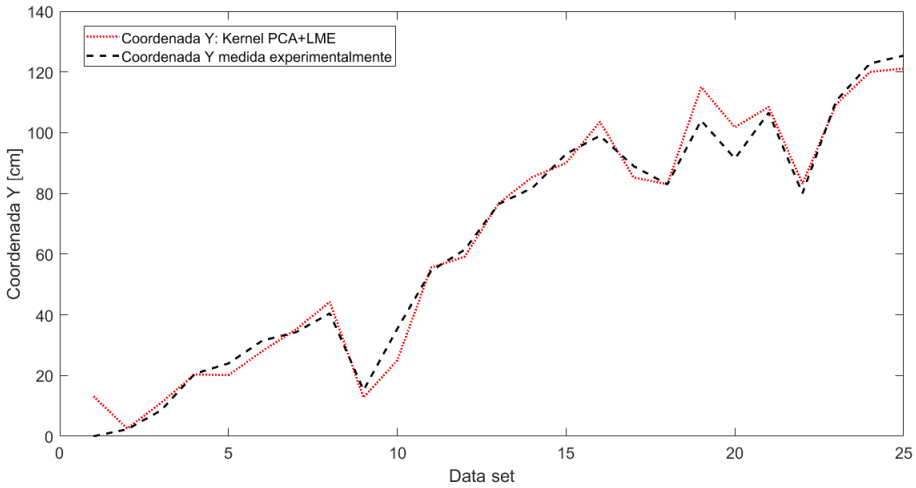

Figura 4.36: Error de localización de la coordenada Y, en función de los datos de prueba

El comportamiento del modelo en el conjunto de prueba, sigue la misma tendencia que las técnicas anteriores, ajustando muy bien los datos a la posición del impacto y siendo más impreciso con la fuerza del impacto.

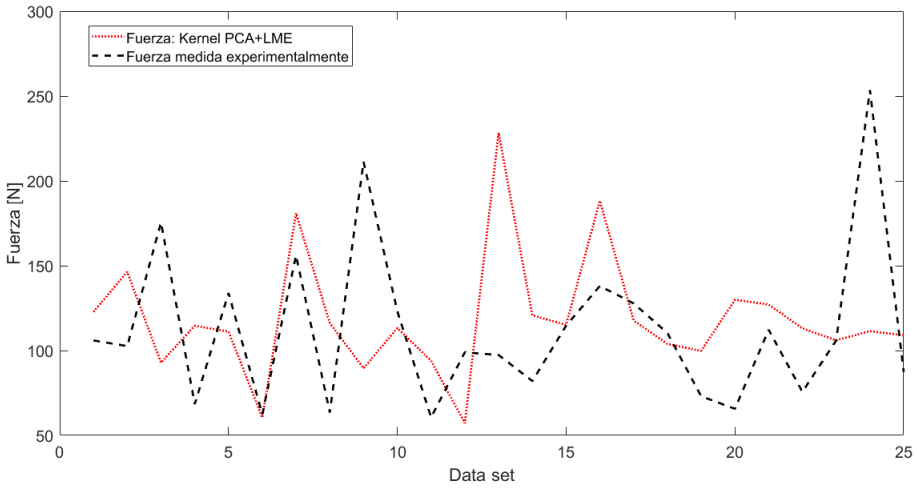

Figura 4.37: Error de cuantificación de la fuerza, en función de los datos de prueba

4.4. Algoritmo Isomap+LME

4.4.1. Evaluación de Isomap+LME, en función del número de vecinos del algoritmo LME

Los gráficos mostrados a continuación, muestran los errores promedio en la localización de impactos, coordenada X e Y, E_X e E_Y , el error porcentual de área E_A, el error porcentual de cuantificación de la fuerza E_F y el error de identificación de impacto normalizado E_I, para distintos números de vecinos de la técnica de reducción de dimensioanlidad Isomap. Estos errores están graficados en función del número de vecinos ocupados para el algoritmo LME.

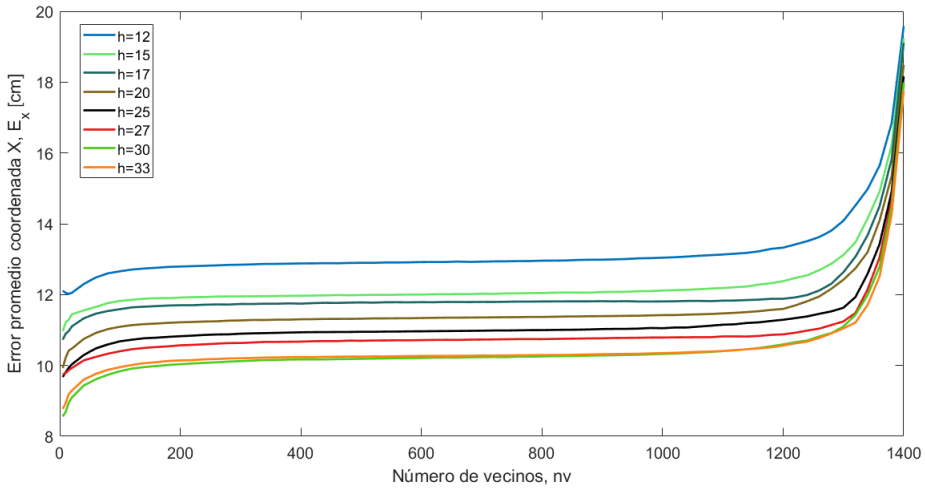

Figura 4.38: Errores promedio en la coordenada que X en función del número de vecinos seleccionados por el algoritmo LME, para cada número de vecinos del algoritmo de Isomap, h=12, h=15, h=17, h=20, h=25, h=27, h=30, h=33.

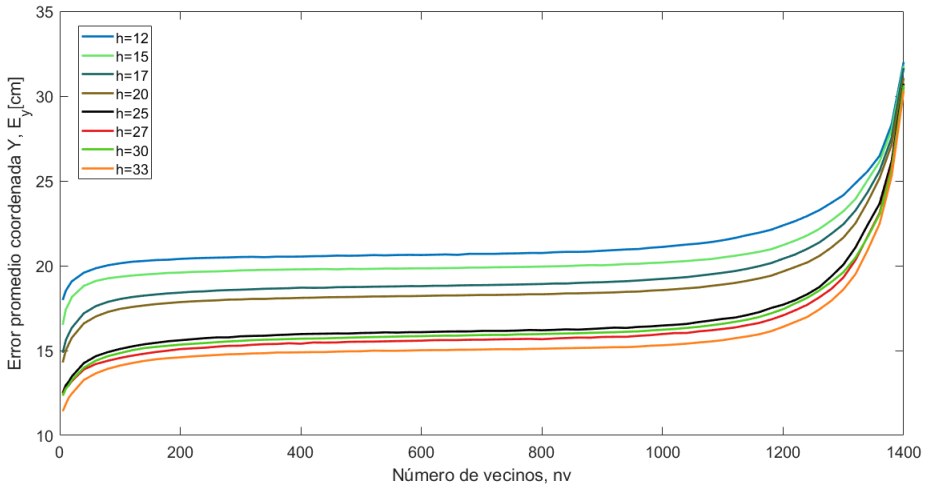

Figura 4.39: Errores promedio en la coordenada que Y en función del número de vecinos seleccionados por el algoritmo LME, para cada número de vecinos del algoritmo de Isomap, h=12, h=15, h=17, h=20, h=25, h=27, h=30, h=33.

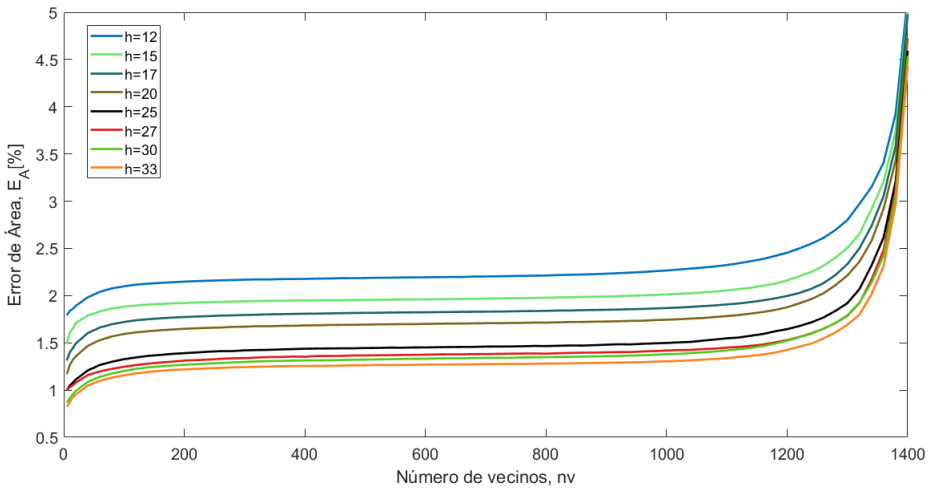

Figura 4.40: Errores porcentuales de área en función del número de vecinos seleccionados por el algoritmo LME, para cada número de vecinos del algoritmo de Isomap, h=12, h=15, h=17, h=20, h=25, h=27, h=30, h=33.

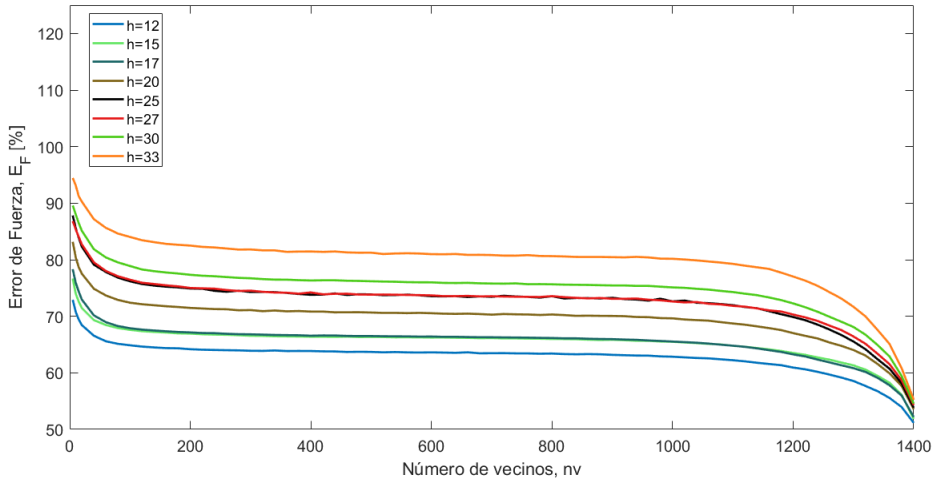

Figura 4.41: Errores porcentuales de cuantificación de la fuerza en función del número de vecinos seleccionados por el algoritmo LME, para cada número de vecinos del algoritmo de Isomap, h=12, h=15, h=17, h=20, h=25, h=27, h=30, h=33.

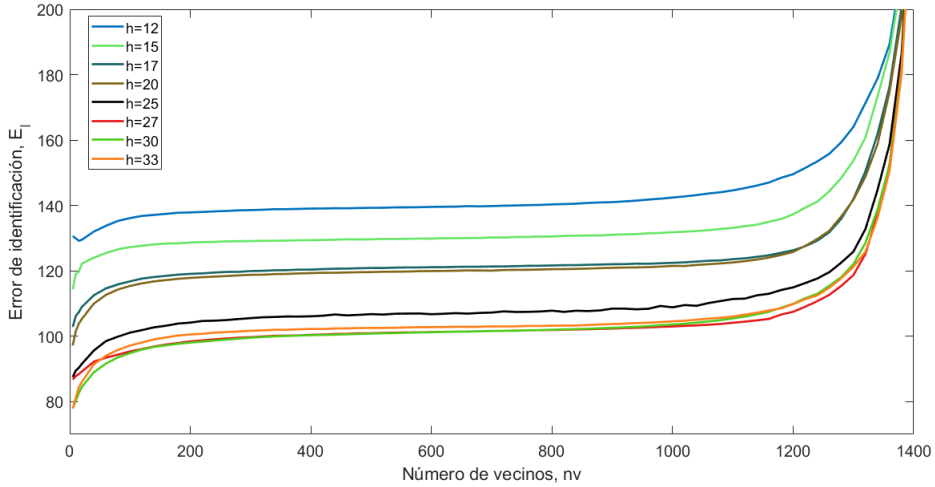

Figura 4.42: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo LME, para cada número de vecinos del algoritmo de Isomap, h=12, h=15, h=17, h=20, h=25, h=27, h=30, h=33.

De las figuras, 4.38,4.39,4.40,4.41 y 4.42 se puede apreciar que si bien la reducción de dimensionalidad isomap mantiene muy estable el error en función del número de vecinos de LME, esta técnica empeora los resultados mostrados por algoritmos precedentes. Se observa que el error porcentual de área aumenta a medida que se aumenta el número de vecinos del algoritmo LME, ocurre lo opuesto con error porcentual de fuerza, el cual disminuye en función del aumento de número de vecinos del algoritmo LME, esto dificulta aún encontrar un error de identificación más bajo del encontrado.

4.4.2. Selección de parámetros y evaluación en el conjunto de prueba

El número de vecinos del algoritmo Isomap, se busco en un rango entre 12 y 80, pasando por los valores de h=12, 15, 17, 20, 25, 27 , 30, 33, 35, 40, 50, 60, 70 y 80, eligiendo un número de vecinos óptimo de 33 vecinos. (Es por esto que las figuras solo muestran entre 12 y 33 como número de vecinos en el algoritmo Isomap, ya que es donde mejores resultados se obtuvieron).

Tabla 4.12: Parámetros y resultados del algoritmo Isomap+LME con mejor rendimiento

Parámetros y resultados del algoritmo Isomap+LME	**Parámetros**
Número de vecinos Isomap	h=33
Número de vecinos del algoritmo LME	5
Error promedio de localización en la coordenada X	8.77[cm]
Error promedio de localización en la coordenada Y	11.41[cm]
Error porcentual de área	0.82[%]
Error de cuantificación de fuerza	94.44 [%]
Error de identificación de impacto normalizado	77.85 [%]

La tabla 4.12 muestra los parámetros en el cual se obtiene un menor error de identificación de impacto normalizado.

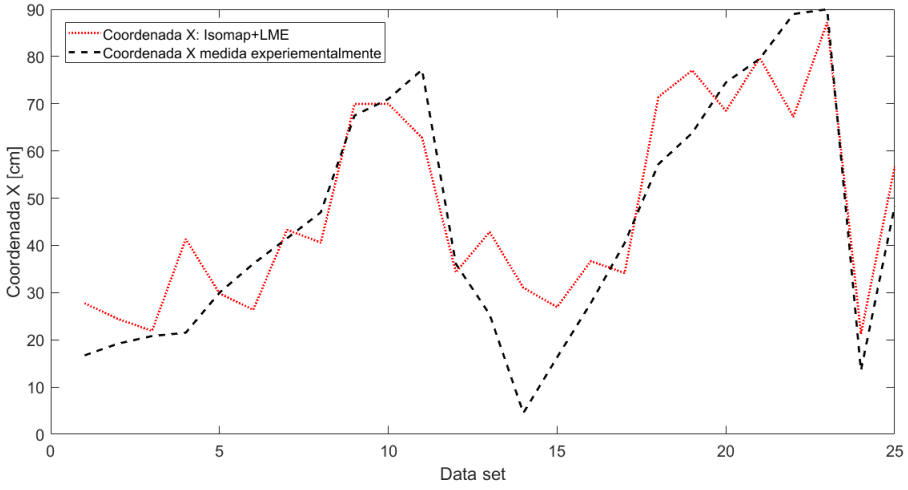

Figura 4.43: Error de localización de la coordenada X, en función de los datos de prueba

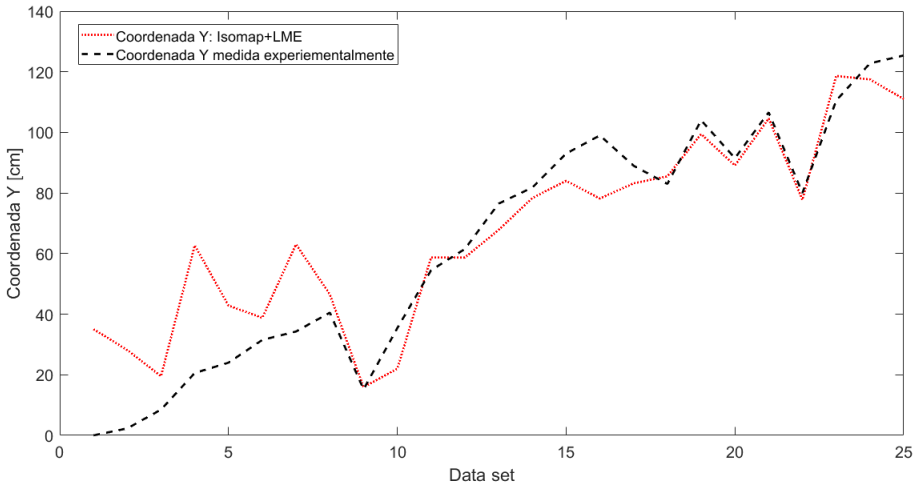

Figura 4.44: Error de localización de la coordenada Y, en función de los datos de prueba

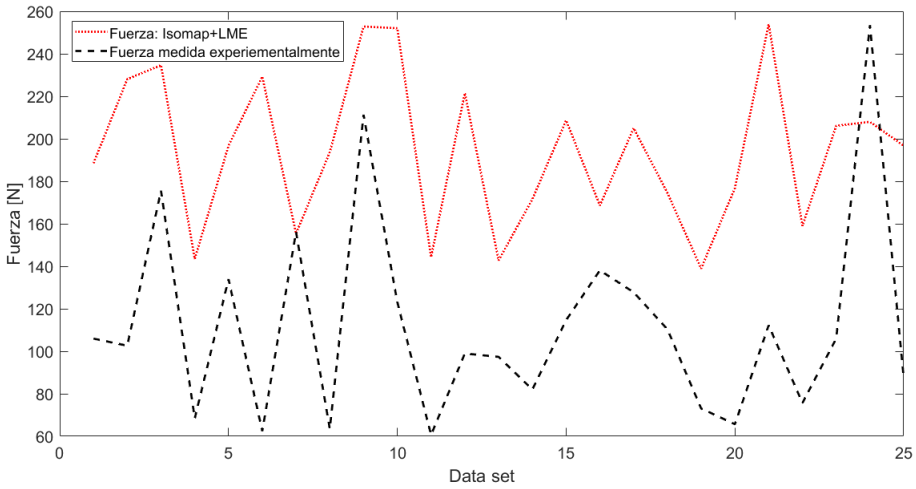

Figura 4.45: Error de en la cuantificación de la fuerza, en función de los datos de prueba

Al evaluar Isomap+LME en el conjunto de pruebas (ver en 4.43, 4.44, 4.45) se puede observar un retroceso con respecto a los algoritmo precedentes, se descarta el uso de Isomap como técnica de reducción de dimensionalidad previa al uso del algoritmo LME.

4.5. Algoritmo MDS+LME

4.5.1. Evaluación de MDS+LME, en función del número de vecinos del algoritmo LME

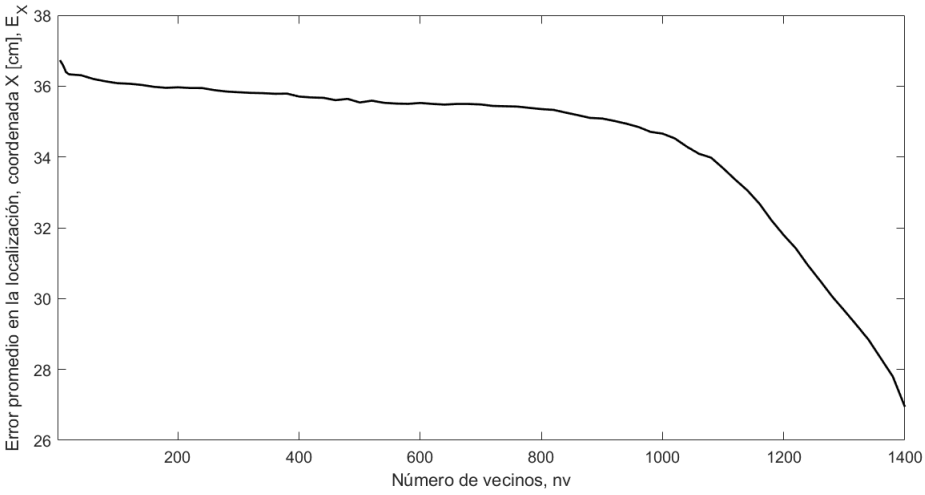

Figura 4.46: Errores promedio en la coordenada que X en función del número de vecinos seleccionados por el algoritmo MDS+LME.

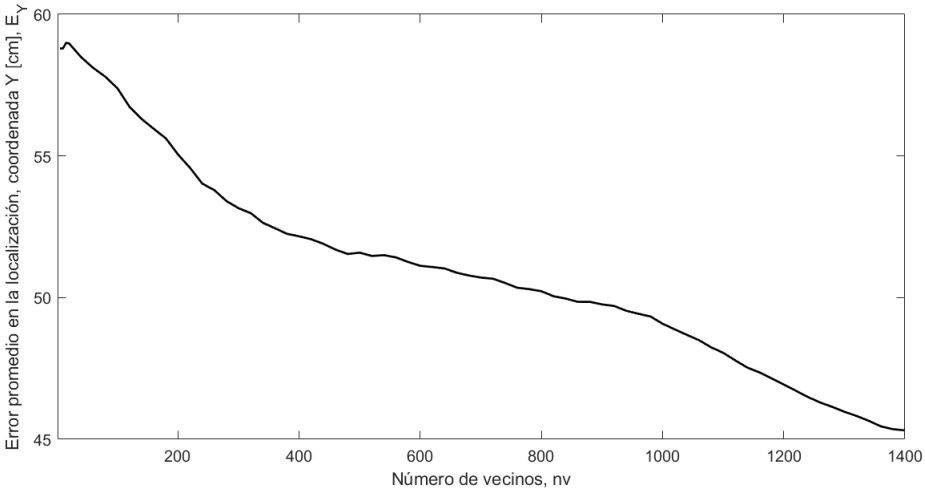

Figura 4.47: Errores promedio en la coordenada que Y en función del número de vecinos seleccionados por el algoritmo MDS+LME.

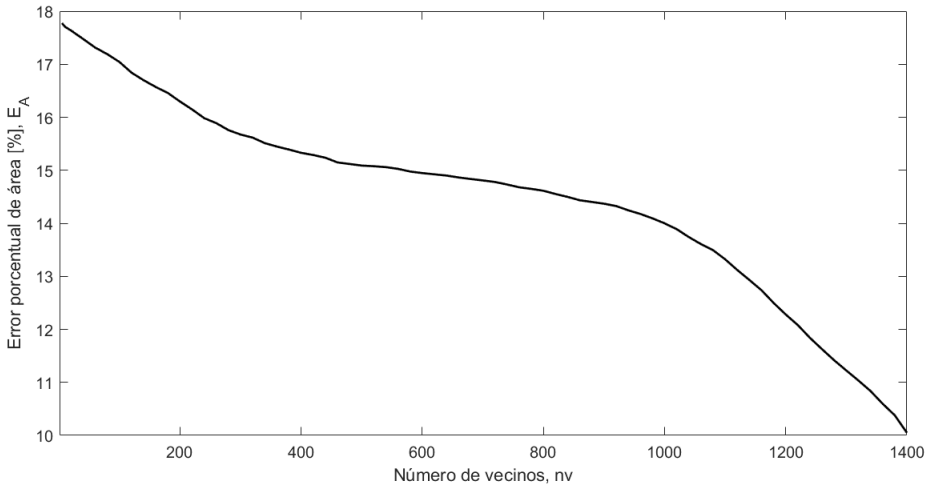

Figura 4.48: Errores porcentuales de área en función del número de vecinos seleccionados por el algoritmo MDS+LME.

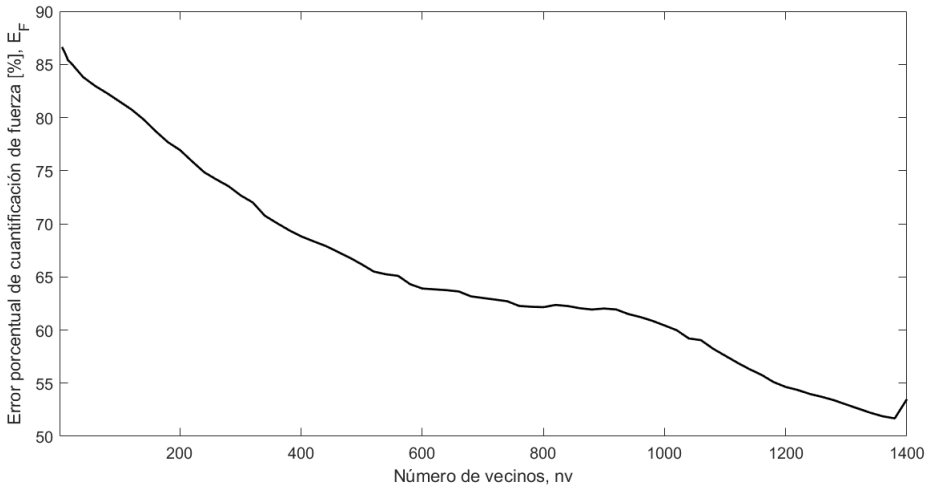

Figura 4.49: Errores porcentuales de cuantificación de la fuerza en función del número de vecinos seleccionados por el algoritmo MDS+LME.

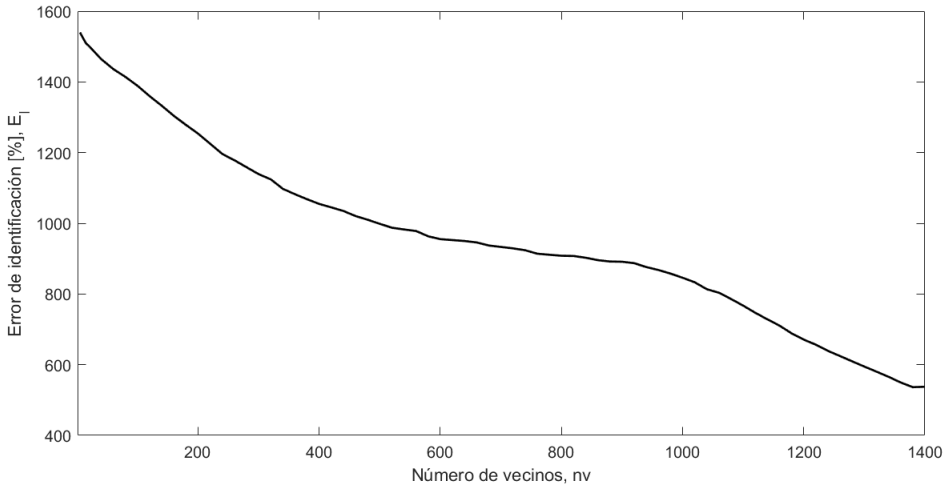

Figura 4.50: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo MDS+LME.

De los resultados vistos en los gráficos 4.46,4.47, 4.48, 4.49 y 4.50 se observan elevados errores tanto en la localización como cuantificación del impacto, se descarta MDS como una técnicas de reducción de dimensionalidad adecuada para el problema.

Tabla 4.13: Resultados algoritmo MDS+LME con el número de vecinos con mejor rendimiento

Resultados del algoritmo del algortimo MDS+LME	**Valor**
Número de vecinos del algortimo LME	1380
Error promedio de localización en la coordenada X	27.80 [cm]
Error promedio de localización en la coordenada Y	45.35 [cm]
Error porcentual de área	10.37[%]
Error de cuantificación de fuerza	51.68 [%]
Error de identificación de impacto normalizado	536.37 [%]

4.6. Algoritmo Prob PCA+LME

4.6.1. Evaluación de Prob PCA+LME, en función del número de vecinos del algoritmo LME

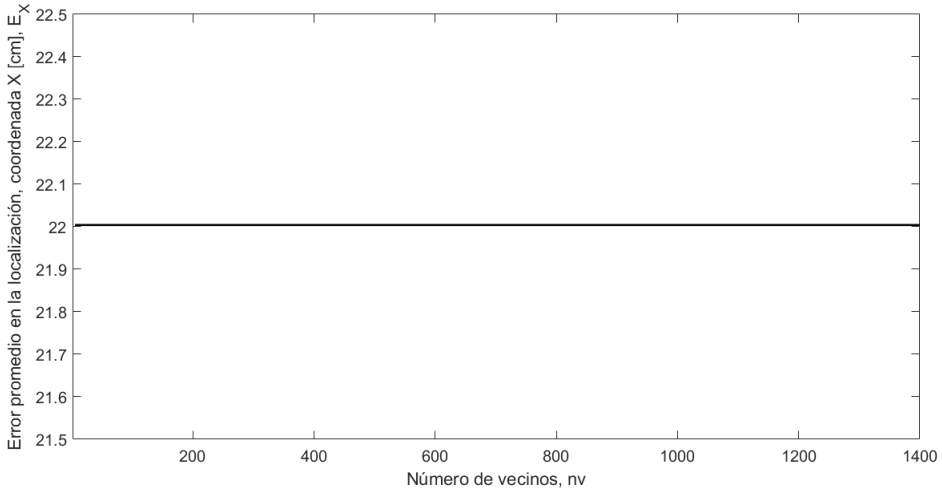

Figura 4.51: Errores promedio en la coordenada que X en función del número de vecinos seleccionados por el algoritmo Prob PCA+LME.

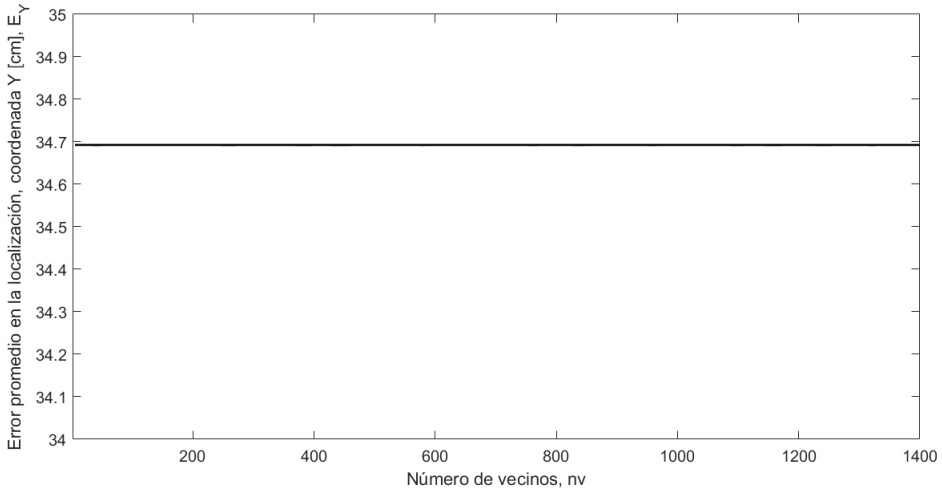

Figura 4.52: Errores promedio en la coordenada que Y en función del número de vecinos seleccionados por el algoritmo Prob PCA+LME.

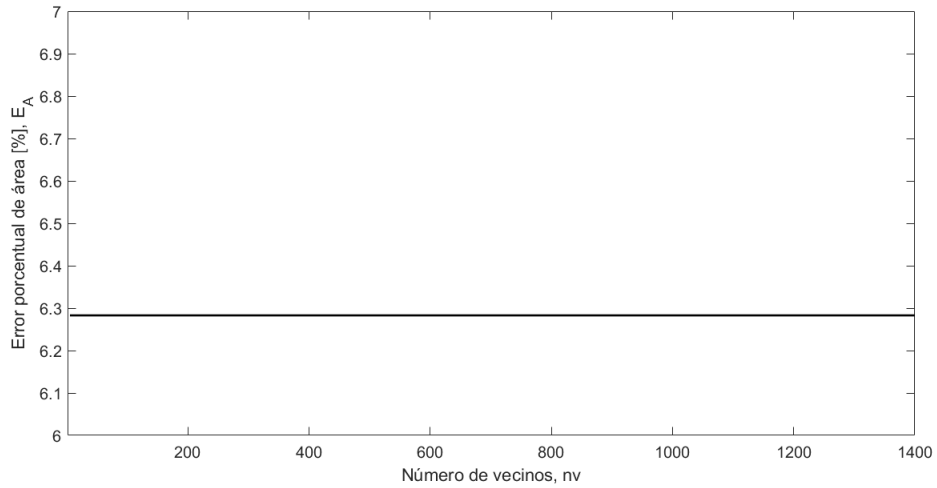

Figura 4.53: Errores porcentuales de área en función del número de vecinos seleccionados por el algoritmo Prob PCA+LME.

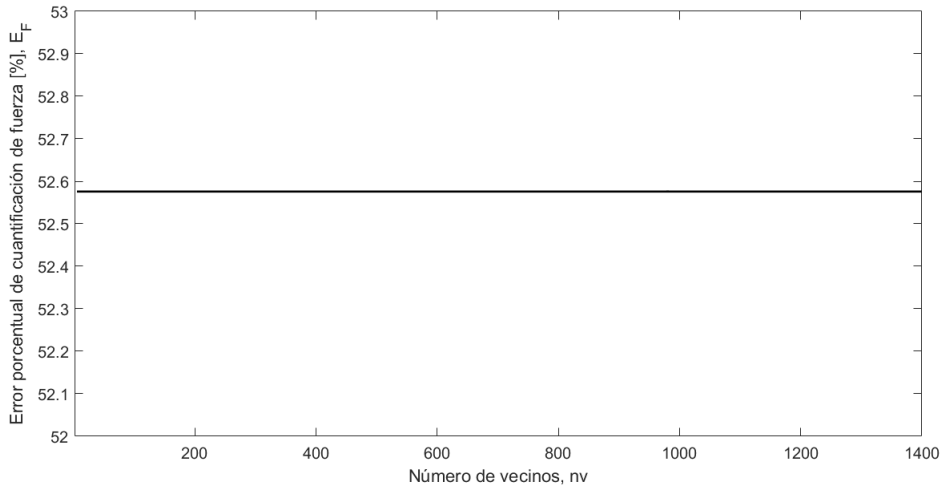

Figura 4.54: Errores porcentuales de cuantificación de la fuerza en función del número de vecinos seleccionados por el algoritmo Prob PCA+LME.

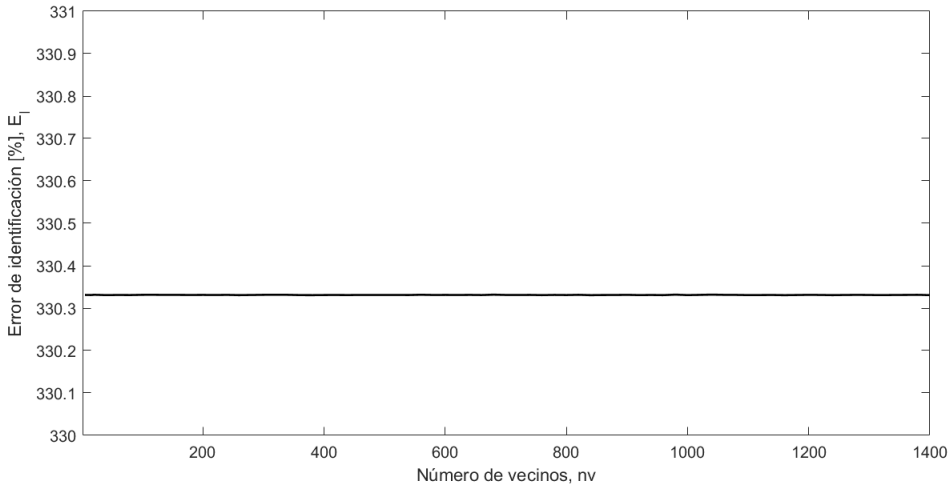

Figura 4.55: Errores porcentuales de identificación de impacto normalizado en función del número de vecinos seleccionados por el algoritmo Prob PCA+LME.

De los resultados vistos en los gráficos 4.51,4.52, 4.53, 4.54 y 4.55 se descarta el uso de la técnica de reducción de dimensionalidad Prob PCA, dado que se obtuvo errores porcentuales demasiado grandes en comparación a otros algoritmos.

La tabla 4.14 muestra los resultados para algoritmo con menor error, usando la técnica de reducción de dimensionalidad PCA probabilístico.

Tabla 4.14: Resultados algoritmo ProbPCA+LME para el número de vecinos del algoritmo LME con mejor rendimiento

Resultados del algoritmo de detección de impactos del algortimo ProbPCA+LME	**Valor**
Número de vecinos del algoritmo LME	840
Error promedio de localización en la coordenada X	22.00 [cm]
Error promedio de localización en la coordenada Y	34.69 [cm]
Error porcentual de área	6.28 [%]
Error de cuantificación de fuerza	52.57 [%]
Error de identificación de impacto normalizado	330.33 [%]

4.7. Resultados modelo red neuronal convolucional

4.7.1. Arquitectura Base

Dado que P=100, fue la cantidad de pasos temporales que mejores resultados obtuvo en anteriores algoritmos de identificación de impactos, como PCA+LME y AE+LME, se elige trabajar con la misma cantidad, en la construcción del modelo de la red neuronal convolucional.

La arquitectura base con la que se comienza a trabajar este problema es una red convolucional de 5 bloques de convolución, con kernels de tamaño (2,2), las cuales contienen dropout (d=0.1) y maxpooling de tamaño (2,1). Esta arquitectura es aquella con la que se definen los parámetros iniciales para el entrenamiento de otros modelos.

Dado que el problema el problema que se trata de la identificación de impactos, la función de costos utilizada es error de identificación de impactos (ver 2.81), que es definida como una función en python y que puede ser utilizada para entrenar modelos con la librería Keras.

Tres hiperparámetros controlan el tamaño del volumen de salida: profundidad, stride (pasos) y zeropadding (margen cero).

- La profundidad del volumen de salida corresponde al número de filtros que queremos ocupar, cada uno aprendiendo a buscar algo diferente en la entrada, para este trabajo se eligió ocupar un número creciente de filtros y no un número en particular, tomando valores de 8 a 128 en función de las capas convolucionales utilizadas.
- El stride, que son los pasos que damos al deslizar el filtro. Cuando el stride es 1 movemos los filtros un "pixel" a la vez. Cuando el stride es 2 los filtros brincan 2 "pixeles" a la ves cuando los deslizamos, se seleccionó ir de strides de 1 "pixel".
- Algunas veces es conveniente cambiar el margen del volumen de entrada por ceros. La característica del zero-padding es que nos permite controlar el tamaño espacial de nuestro volumen de salida, característica que se uso en todas las capas de convolución.

Para la elección del número de capas de la red fully connected de la CNN se prueban distintas configuraciones de capas de salida, con distintos números de neuronas, de las cuales se elige aquella que disminuye el error de identificación de impactos de validación. Los resultados se pueden ver en la tabla 4.15.

Tabla 4.15: Número de neuronas en las capas de la red fully connected considerando 3 capas, con el error de identificación normalizado que entregan

Neuronas de la red fully connected	**Error de identificación de impactos E_I [%]**
64-32-3	44.19
128-64-3	10.93
256-128-3	10.60
512-256-3	8.70
1024-512-3	**8.30**

De la tabla 4.15 se observa que la mejor configuración de capas de salida es aquella con 3 capas fully connected, 2 internas y una conectada a al output.Dado el volumen de datos y los recursos computacionales limitados se decide no seguir probando con más de 1024 neuronas.

Las funciones de activación usadas fueron la función ReLU para todas las capas de la red, excepto para la capa final de la red de salida, en donde se ocupo una función de activación lineal, ya que es aquella que funciona mejor para regresiones [10].

La figura 4.56 y tabla 4.16 muestran el resultado del entrenamiento de la arquitectura base por 5000 épocas.

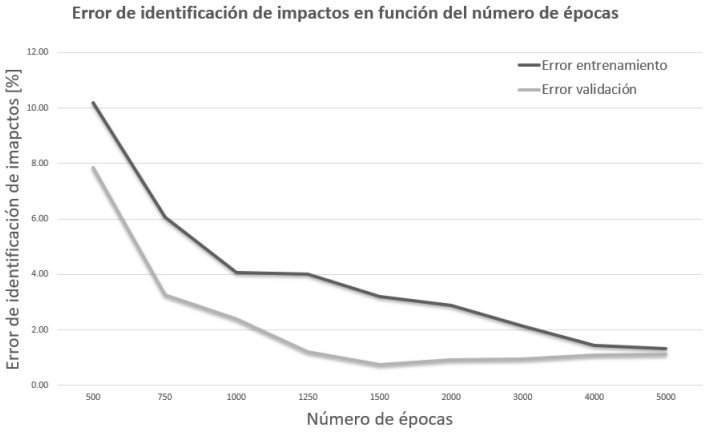

Figura 4.56: Error de identificación de impactos de entrenamiento y validación en función del número de épocas de entrenamiento.

Tabla 4.16: Error de identificación de impactos de entrenamiento y validación en función del número de épocas de entrenamiento.

Épocas	E_I [%] **entrenamiento**	E_I [%] **validación**
1	2169.07	1171.61
50	145.62	189.22
100	54.26	87.57
250	19.13	31.12
500	10.18	7.86
750	6.05	3.26
1000	4.08	2.40
1250	4.00	1.21
1500	**3.21**	**0.73**
2000	2.88	0.93
3000	2.13	0.94
4000	1.43	1.10
5000	1.33	1.13

La selección del número de épocas de entrenamiento, se define, al igual como se hizo con lo autoencoders, a partir del callback "EarlyStopping" de keras.

Del gráfico 4.56 y de la tabla 4.16 se puede observar que a partir de 1500 épocas, comienza a producirse "overffiting" en el entrenamiento del modelo, con lo cual se espera que en la mayor parte de los modelos entrenados, una cantidad de épocas de entrenamiento cercana a las 1500. De la tabla se puede ver que entre las épocas 1250 y aproximadamente 1500 el modelo no fue capaz de reducir en 1 % el error de identificación de impactos, posterior a ello, se encuentra en una situación de "overffiting". Debido a esta observación se considera la siguientes condiciones de detención del EarlyStopping:

- monitor= Variable a monitorear, en este caso el error de identificación de impactos en los datos de validación.
- Patience= 400 : Número de épocas a esperar sin que se produzcan mejoras de mínimo δ en la variable de monitoreo.
- δ= 0.1: Es el valor que debe producirse en la función de costos para considerarse como progreso.

Lo que se traduce en que si el modelo, no es capaz de reducir el error de los datos de validación en una puntuación de 0.1 % en el transcurso de 400 épocas, se puede concluir que se ha estancado y esta sobre ajustando los datos. Se elige un número de épocas tan amplio, ya que en la práctica se observó que habían modelos que a causa del ruido se estancaban, pero poco a poco estos conseguían avanzar. EarlyStopping nos permite recuperar las mejores pesos de validación, con lo que, mismo si se entrena más épocas de las que se requerían, la recuperación de las mejores puntuaciones nos asegura estar siempre en la mejor situación.

Tal y como se reflexionó anteriormente con los autoencoders, el optimizador usado es el ADAM

(adaptative momentum), (ver sección de optimizador sección de resultados AE+LME).

Al igual como se hace en el número de épocas, para encontrar la tasa de aprendizaje óptima, también se utiliza un callback de keras, "ReduceLROnPlateau", la cual adapta la tasa de aprendizaje en función del aprendizaje. Las entradas de este callback son; La tasa mínima con la que se puede entrenar, la paciencia y el factor de reducción de la tasa de aprendizaje, se ocupa esta función, ya que nos ahorra buscar por grid search la tasa óptima en diferentes modelos (lo que sería bastante tedioso), además da la posibilidad al programador de ir cambiando los rangos de tasas de aprendizaje en función de los resultados de las pruebas.

- Patience= 50 : Número de épocas a esperar sin mejoras en el entrenamiento.
- Factor= 0.5: $Lr_n = Lr_{n-1} \cdot Factor$, factor de disminución de la tasa de aprendizaje.

Usando esta función se encontró que el modelo se entrena mejor con tasas de aprendizaje en rangos de 10^{-5} y$5 \cdot 10^{-4}$.

4.7.2. Segunda arquitectura

Para la elección de la segunda arquitectura se utilizan dos métodos. El primero, utilizando kernels de tamaño constante en las capas convolucionales. En el segundo método, se varía el tamaño de los kernels en las capas convolucionales, seleccionando el kernel que obtuvo mejor resultado en cada capa, para utilizarlo en la arquitectura siguiente (partiendo de 1 a 6 capas convolucionales). Se ocupan los tamaños de kernels presentados en la sección de metodología.

Tabla 4.17: Modelos construidos con un kernel de tamaño constante

CNN	**Tamaño de los kernels**	E_x **[cm]**	E_y **[cm]**	E_A **[%]**	E_F **[%]**	E_I **[%]**
1 capa convolucional						
2 capas de salida	2,1	7.29	6.85	0.41	41.40	17.04
	2,2	6.46	8.53	0.45	38.20	17.32
	3,1	7.09	7.21	0.42	43.17	18.16
	3,3	5.07	8.34	0.35	37.83	13.15
	5,1	7.36	7.43	0.45	40.00	18.00
3 capas de salida	2,1	5.43	7.67	0.34	38.42	13.16
	2,2	6.81	9.02	0.51	34.60	17.48
	3,1	5.61	5.69	0.26	39.50	10.38
	3,3	5.70	7.04	0.33	40.12	13.23
	5,1	7.26	7.42	0.44	39.90	17.69
2 capa convolucionales						
2 capas de salida	2,1	6.09	8.64	0.43	34.65	15.01
	2,2	8.11	11.40	0.76	29.17	22.19
	3,1	6.99	9.62	0.55	33.08	18.32
	3,3	6.54	7.90	0.43	34.61	14.71
	5,1	7.17	8.01	0.47	35.63	16.85
3 capas de salida	2,1	6.29	9.21	0.48	29.74	14.17
	2,2	9.17	8.27	0.62	32.09	20.03
	3,1	7.60	8.43	0.53	35.50	18.72
	3,3	9.10	10.03	0.75	29.55	22.19
	5,1	6.67	9.33	0.51	28.37	14.54

Tabla 4.17: Modelos construidos con un kernel de tamaño constante

CNN	**Tamaño de los kernels**	E_x **[cm]**	E_y **[cm]**	E_A **[%]**	E_F **[%]**	E_I **[%]**
3 capa convolucionales						
2 capas de salida	2,1	7.05	9.77	0.57	28.37	16.07
	2,2	6.83	8.98	0.51	30.94	15.62
	3,1	7.29	9.16	0.55	33.46	18.39
	3,3	7.02	8.44	0.49	30.11	14.68
	5,1	6.70	8.32	0.46	30.36	13.93
3 capas de salida	2,1	6.00	7.95	0.39	28.57	11.22
	2,2	6.47	8.59	0.46	27.18	12.44
	3,1	5.94	7.20	0.35	33.40	11.76
	3,3	7.83	7.11	0.46	29.57	13.54
	5,1	6.53	6.99	0.38	33.66	12.63
4 capas convolucionales						
2 capas de salida	2,1	8.10	10.83	0.72	26.24	18.93
	2,2	6.50	7.94	0.42	27.52	11.68
	3,1	6.76	9.60	0.53	29.93	15.99
	3,3	5.80	8.17	0.39	28.80	11.22
	5,1	6.37	8.57	0.45	30.64	13.78
3 capas de salida	2,1	6.35	6.92	0.36	27.32	9.88
	2,2	8.53	7.69	0.54	27.88	15.05
	3,1	5.62	7.05	0.33	30.36	9.90
	3,3	5.14	8.03	0.34	30.20	10.27
	5,1	5.85	8.19	0.39	31.45	12.40
5 capas convolucionales						
2 capas de salida	2,1	7.02	9.75	0.56	28.55	16.10
	2,2	5.45	7.23	0.32	25.91	8.40
	3,1	6.24	7.57	0.39	24.13	9.38
	3,3	6.33	6.33	0.33	29.10	9.60
	5,1	6.23	7.71	0.40	32.70	12.93
3 capas de salida	2,1	8.23	8.80	0.60	25.45	15.16
	2,2	7.04	7.39	0.43	28.36	12.15
	3,1	5.58	8.81	0.40	31.61	12.79
	3,3	5.37	4.99	0.22	31.02	**6.83**
	5,1	5.80	7.21	0.34	27.52	9.47

Tabla 4.18: Modelos construidos variando el tamaño de los kernels a medida que aumenta el número de capas

CNN		$E_x[cm]$	$E_y[cm]$	$E_A[\%]$	$E_F[\%]$	$E_I[\%]$
1 capa convolucional						
2 capas de salida	2,1	7.29	6.85	0.41	41.40	17.04
	2,2	6.46	8.53	0.45	38.20	17.32
	3,1	7.09	7.21	0.42	43.17	18.16
	3,3	5.07	8.34	0.35	37.83	**13.15**
	5,1	7.36	7.43	0.45	40.00	18.00
3 capas de salida	2,1	5.43	7.67	0.34	38.42	13.16
	2,2	6.81	9.02	0.51	34.60	17.48
	3,1	5.61	5.69	0.26	39.50	**10.38**
	3,3	5.70	7.04	0.33	40.12	13.23
	5,1	7.26	7.42	0.44	39.90	17.69
2 capas convolucionales						
2 capas de salida: k1=(3,3)	2,1	7.82	12.02	0.77	30.79	23.81
	2,2	7.00	7.24	0.42	29.27	**12.20**
	3,1	7.49	8.95	0.55	29.69	16.38
	3,3	6.11	8.14	0.41	32.94	13.47
	5,1	6.21	8.85	0.45	32.77	14.83
3 capas de salida:k1=(3,1)	2,1	5.64	8.02	0.37	34.03	12.67
	2,2	8.73	11.01	0.79	31.92	25.24
	3,1	6.27	9.30	0.48	30.68	14.72
	3,3	7.18	9.11	0.54	29.06	15.63
	5,1	6.20	8.03	0.41	33.63	**13.77**
3 capas convolucionales						
2 capas de salida: k2=(2,2)	2,1	8.27	7.32	0.50	27.68	13.80
	2,2	6.21	7.45	0.38	26.89	**10.24**
	3,1	6.56	6.51	0.35	31.65	11.12
	3,3	9.69	9.39	0.75	30.05	22.50
	5,1	11.46	13.85	1.31	31.96	41.76
3 capas de salida:k2=(2,1)	2,1	7.42	7.50	0.46	28.54	13.08
	2,2	7.23	9.95	0.59	26.76	15.84
	3,1	7.90	10.83	0.70	32.16	22.63
	3,3	5.90	5.85	0.28	35.51	**10.09**
	5,1	5.88	8.65	0.42	33.51	14.01

Tabla 4.18: Modelos construidos variando el tamaño de los kernels a medida que aumenta el número de capas

CNN		$E_x[cm]$	$E_y[cm]$	$E_A[\%]$	$E_F[\%]$	$E_I[\%]$
4 capas convolucionales						
2 capas de salida:k3=(2,2)	2,1	5.78	6.71	0.32	26.57	8.48
	2,2	6.60	7.19	0.39	26.50	10.35
	3,1	9.06	8.77	0.65	28.95	18.92
	3,3	4.97	6.66	0.27	32.81	**8.34**
	5,1	8.34	8.47	0.58	23.91	13.91
3 capas de salida:k3=(3,3)	2,1	6.98	7.53	0.43	28.42	12.29
	2,2	6.17	3.99	0.20	29.76	6.03
	3,1	5.94	6.56	0.32	24.93	7.99
	3,3	6.85	4.52	0.25	27.66	7.05
	5,1	6.65	4.15	0.23	23.96	**5.44**
5 capas convolucionales						
3 capas de salida:k4=(5,1)	2,1	6.35	5.11	0.27	27.94	7.47
	2,2	6.08	6.09	0.30	29.72	9.05
	3,1	5.88	5.95	0.29	25.75	7.42
	3,3	5.86	4.94	0.24	21.69	**5.17**
	5,1	6.30	5.42	0.28	24.29	6.83
6 capas convolucionales						
3 capas de salida:k5=(3,3)	2,1	6.48	4.48	0.24	30.27	7.24
	2,2	5.23	5.17	0.22	39.94	8.89
	3,1	5.50	4.77	0.22	34.10	7.36
	3,3	5.87	6.22	0.30	29.87	8.99
	5,1	5.72	6.14	0.29	29.82	8.62

Como se observa en las tablas 4.17 y 4.18 el tamaño de los kernels afecta en gran medida el rendimiento de una arquitectura, siendo el método que varía los kernels en función de las capas el que de manera general mejores resultados obtuvo.

Tabla 4.19: Tamaño de los kernels de la arquitectura con mejor rendimiento

Posición de la capa	Tamaño del kernel
Capa#1	k=(3,1)
Capa#2	k=(2,1)
Capa#3	k=(3,3)
Capa#4	k=(5,1)
Capa#5	k=(3,3)

La arquitectura que obtuvo mejores resultados, consta de 5 capas convolucionales, capas de maxpooling y dropout en las 5 capas, con tasa de abandono de d=0.1, 3 capas en la red neuronal de salida (FC) con 1024-512-3 neuronas, con dropout con tasa de abandono de d=0.3 en las dos primeras capas, siendo la última la salida de los targets. Ver A.7.

Las figuras 4.57 y 4.58 muestran el error de identificación de impactos en el conjunto de prueba, para diferentes arquitecturas.

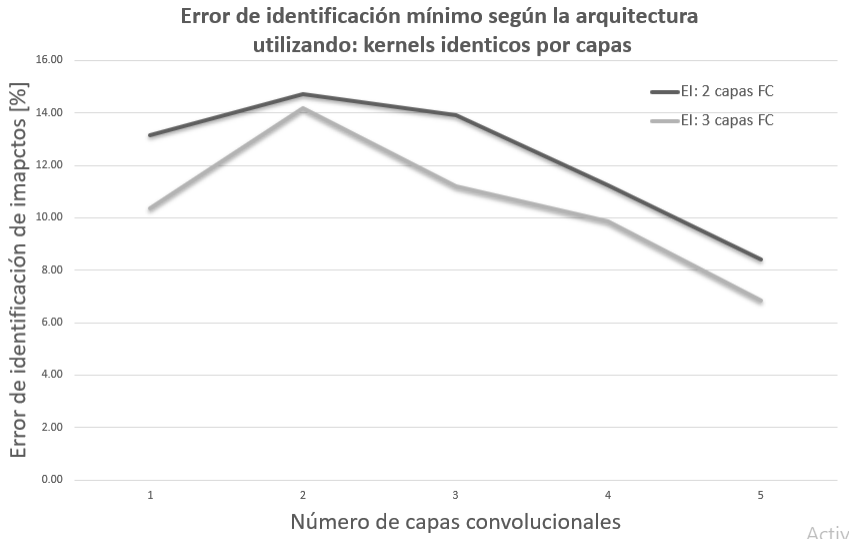

Figura 4.57: Error de identificación mínimo en función de la arquitectura seleccionada

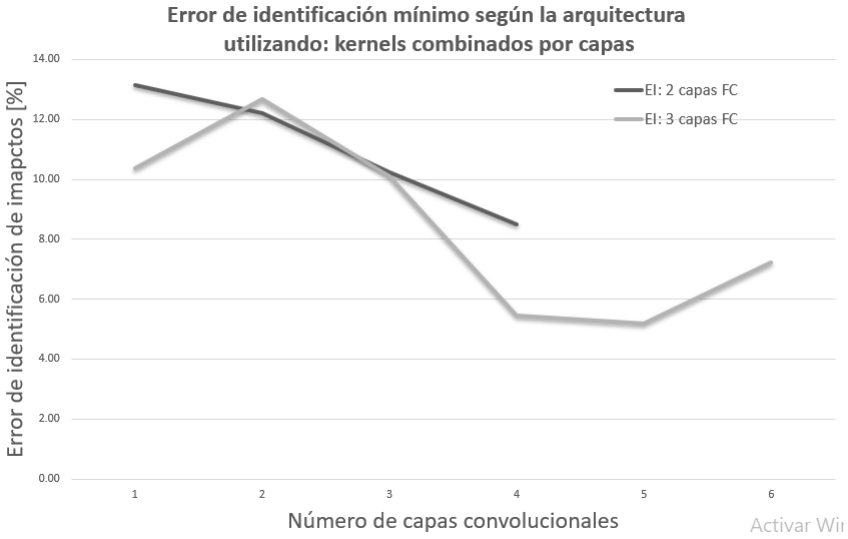

Figura 4.58: Error de identificación mínimo en función de la arquitectura seleccionada

4.7.3. Regularización del modelo

Una vez seleccionada la arquitectura del modelo, se intentan nuevas configuraciones de regularización para este. Estas son cambiar la tasa de dropout y agregar regularización por pesos norma L_2.

4.7.3.1. Regularización L_2

Se prueban dos regularizaciones de tipo L_2, los resultados son mostrados en la tabla 4.20.

Tabla 4.20: Regularizadores L_2 probados para el modelo de la CNN

Regularizadores	**Error de identificación de impactos** $E_I[\%]$
$\lambda = 10^{-4}$	5.42
$\lambda = 10^{-5}$	9.92

4.7.3.2. Regularización por dropout

Los modelos precedentes fueron entrenados con una tasa de dropout de 0.1 en las capas convolucionales y de 0.3 en las capas de la red neuronal FC.

La tabla 4.21 muestra los resultados del modelo escogido en la etapa anterior con distintas tasas de abandono, sin embargo estás pruebas no mejoraron los resultados ya mostrados, con la configuración de dropout anterior, es decir de d=0.1 en capas convolucionales y de 0.3 en las capas de la red FC.

Tabla 4.21: Modelo probado con distintas tasas de abandono

Regularizadores	**Error de identificación de impactos** $E_I[\%]$
d=0.1	6.37
d=0.2	6.02
d=0.3	5.55
d=0.4	6.61

4.7.3.3. Resumen regularización

Los resultados de las pruebas de regularización, no mejoraron los resultados obtenidos previamente, por lo que se decidió dejar la misma configuración utilizada para entrenar los modelos en la segunda arquitectura. Esta es la siguiente: capas de dropout con tasa de abandono de d=0.1 en la salida de las capas convolucionales y tasa de abandono de d=0.3, para las capas de la red de salida; En conjunto con capas de Batch Normalization, aplicadas después de la salida de las dos primeras capas de la red FC.

La tabla 4.22 muestra los hiperparámetros del modelo final.

Tabla 4.22: Hiperparámetros de la CNN con mejor rendimiento

Hiperparámetros	
Número de épocas	1400
Tasa de aprendizaje	$[10^{-5}, 5 \cdot 10^{-4}]$
Funciones de activación	ReLU y lineal (capa de salida)
Número de filtros	8-16-32-64-128
Tamaño kernels	ver tabla 4,19
Padding	Zero padding
Strides	(1,1)
Función de costo	Error de identificación de impacto normalizado
Dropout	d=0.1(capas convolucionales) d=0.3(FC)

4.7.4. Evaluación del modelo en el conjunto de prueba

Tabla 4.23: Resultados del modelo final, red neuronal convolucional

Resultados del modelo	**Valor**
Error promedio en la localización coordenada X	5.86 [cm]
Error promedio en la localización coordenada Y	4.94 [cm]
Error porcentual de área	0.24 [%]
Error de porcentual de cuantificación de fuerza	21.69 [%]
Error de identificación de impacto normalizado	5.17 [%]

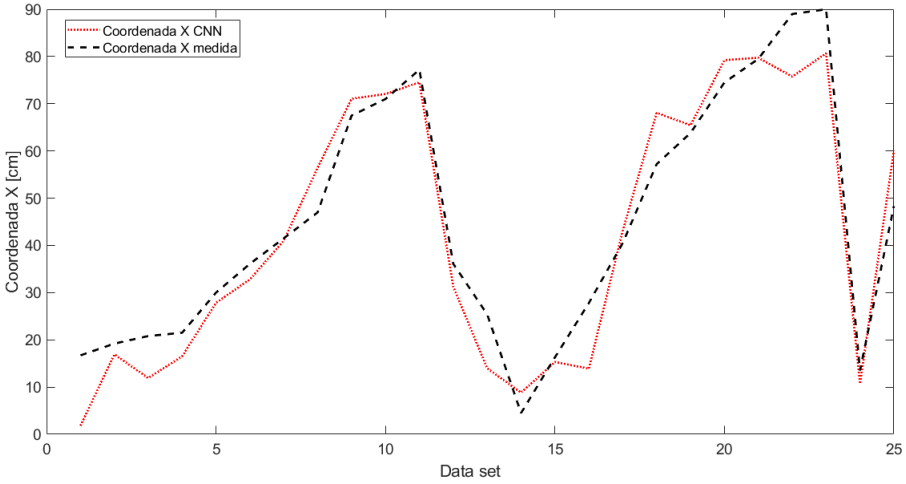

Figura 4.59: Error de localización de la coordenada X, en función de los datos de prueba

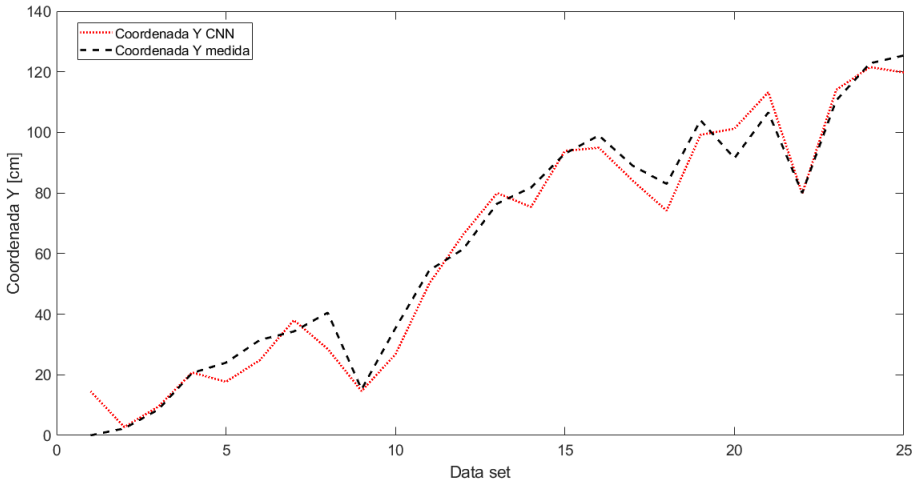

Figura 4.60: Error de localización de la coordenada Y, en función de los datos de prueba

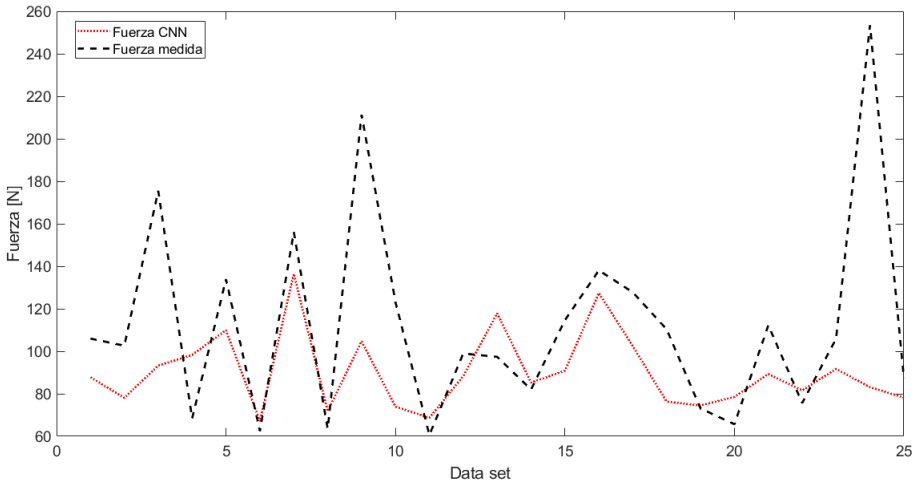

Figura 4.61: Error de en la cuantificación de la fuerza, en función de los datos de prueba

En las 4.59, 4.60 y 4.61 se muestra el rendimiento del modelo creado en el conjunto de datos de prueba. En comparación con los anteriores algoritmos, se puede observar mejor rendimiento en la predicción de la fuerza, sin embargo menos preciso en la localización del impacto, que algoritmos como PCA+LME, AE+LME y Kernel PCA+LME.

Capítulo 5

Discusión

5.1. Sobre la determinación del número de vecinos en el algoritmo LME

Ya que se trabaja con una dimensión intrínseca fija, solo se necesita buscar el número de vecinos que minimice el error de identificación de impactos en cada técnica de reducción de dimensionalidad.

En la tabla 5.1 se presenta un resumen de los valores de n_v elegidos por algoritmo en cada técnica de reducción de dimensionalidad.

Tabla 5.1: Número de vecinos del algoritmo LME, asociados a cada técnica de reducción de dimensionalidad.

Número de vecinos del algortimo LME	**Valor**
PCA	5
Kernel PCA: Polinomial	100
Kernel PCA: Gaussiano	15
PCA Probabilístico	840
Isomap	5
Escalamiento multidimensional	1380
Autoencoders	15

Se observa que, frente a la variación del número de vecinos del algoritmo LME, el error de identificación es constante para la mayoría de las técnicas de reducción de dimensionalidad (ver 4.5, 4.19, 4.33, 4.24, 4.42, 4.55) (excepto MDS), además se ve que hasta como mínimo un número de vecinos de 800, el error de identificación de impactos E_I, presenta variaciones muy leves (del orden de $\pm$ 1.5 %, las técnicas con mejor resultado). Este resultado es bastante útil ya que el rendimiento de un algoritmo de identificación de impactos, pasa a ser únicamente dependiente de la técnica de reducción de dimensionalidad usada previa al LME.

La excepción fue la técnica de escalamiento multidimensional (MDS), que presenta un error decreciente en función del número de vecinos, sin embargo esta es creciente entre 1380 y 1400 vecinos, por lo que no dice nada concluyente sobre si el aumentar el número de vecinos (i.e expandir base de datos de entrenamiento), mejoraría su rendimiento.

De la tabla 5.1 se puede constatar que para los algoritmos que obtuvieron un mejor desempeño el número de vecinos que minimiza el error de identificación, no supera los 15 vecinos. Esto discrepa con resultados obtenidos en estructuras como una paneles de aluminio tipo sandwich y un cilindro de acero ([8], [17]), en los cuales se obtuvieron números de vecinos mucho más grandes.

5.2. Sobre el rendimiento de los algoritmos de identificación de impactos

La tabla 5.2 muestra un resumen de los mejores resultados obtenidos por cada uno de los algoritmos utilizados. Se puede observar que son 4 los algoritmos con mejor rendimiento, AE+LME, CNN, Kernel PCA con kernel gaussiano+LME y PCA+LME. Dentro de estos 4 algoritmos, aquellos que se componen de una técnica de reducción de dimensionalidad y LME, superan en rendimiento la predicción de la posición a la CNN, pero son superados en la predicción de la fuerza de impacto.

Tabla 5.2: Tabla resumen rendimiento de los algoritmos

Método	**Error porcentual de área [%]**	**Error porcentual de fuerza [%]**	**Error de identificación de impactos [%]**
Autoencoders+LME	**0.116**	39.537	**4.578**
Redes neuronales convolucionales	0.239	**21.692**	5.173
Kernel PCA: Gaussiano+LME	0.142	38.607	5.463
PCA+LME	0.187	35.895	6.709
Kernel PCA: Polinomial+LME	0.327	37.726	12.325
Isomap+LME	0.824	94.446	77.861
PCA Probabilístico+LME	6.283	52.576	330.329
MDS+LME	10.378	51.682	536.371

5.2.1. Sobre el rendimiento de las técnicas de reducción de dimensionalidad+LME

El rendimiento de cada una de las técnicas varían según la aplicación de estas, como lo explican Maaten et Al [1]. Lo que conlleva que a pesar de ser la misma aplicación (i.e, identificación de impactos), los conjuntos de datos dependen de la estructura o forma en la que fueron creados.

Las técnicas que no emplean gráficos de vecindarios, como PCA, kernel PCA, y autoencoders, funcionaron mejor, y el técnica de reducción de dimensionalidad que proporcionó los mejores resultados fue el autoencoder. Estos resultados concuerdan con lo que se mostró por Meruane [16], para aplicaciones de un cilindro de acero y para paneles de aluminio tipo sándwich.

La construcción de gráficos de vecindario (como lo hacen las técnicas MDS e ISOMAP), son susceptibles a problemas de sobreajuste, es decir la técnica de reducción de dimensionalidad reproduce muy bien las geometrías locales de los datos de entrenamiento de entrenamiento, pero no ajustándose a los datos de prueba. Además la interferencia de valores atípicos en la construcción de nuevas geometrías locales, puede también ser un problema para la posterior utilización en el set de datos de prueba.

Probablemente es que el en el buen rendimiento de los autoencoders en esta aplicación, es la gran capacidad que estos tienen para aprender las no linealidades de variedades muy complejas. Además de ser capaz de pasar por alto los problemas de las técnicas que usan gráficos, manteniendo un control en el "sobre ajuste" de los datos y a través del mismo funcionamiento del mismo, diferenciar valores atípicos.

PCA probabilístico, a pesar de no utilizar gráficos de vecindarios, no obtuvo buenos resultados, esto es atribuible a la gran parte de información que se pierde en la proyección de los datos a una baja dimensión, o a un sobre ajuste de las densidades de probabilidad atribuidas a los datos de entrenamiento.

5.2.2. Sobre el rendimiento de la red neuronal convolucional (CNN)

El estudio del problema de identificación de impactos con una red neuronal convolucional, entrego resultados bastante favorables en cuanto a la predicción de la fuerza de impacto, prediciendo 1.83 veces mejor la fuerza en comparación al algoritmo AE+LME, sin embargo prediciendo el área de impacto su error es 2 veces mayor.

En términos de mantenimiento industrial lo que indica si el daño es o no grave y debe ser revisado, es la magnitud de la fuerza de impacto, en esto, la red neuronal convolucional, tuvo mejor desempeño que el resto de los algoritmos, haciéndola muy útil si lo que se privilegia por el algoritmo es la precisión en el rango de fuerzas de impacto, para la estimación del posible daño.

5.3. Sobre el error porcentual de área y fuerza

Según la literatura [17][3][18],[19] el error de identificación de la fuerza en estructuras como un cilindro de acero, o un panel de aluminio tipo sandwich, esta dentro del orden del 10-18 %, en este trabajo el error porcentual de la fuerza en los algoritmos que utilizan técnica de reducción de dimensionalidad +LME, bordea el 39 % (ver figura 5.1), lo que es el doble de lo visto en trabajos anteriores. Esto anterior se puede explicar, debido a que los trabajos anteriores eran estructuras de laboratorio y bastantes regulares en su forma y composición. El fuselaje posee características como la curvatura del cilindro de acero y la laminación de capas que posee la placa de aluminio, dos de las características estudiadas precedentemente por separado, lo que lo hace una estructura mucho más compleja a estudiar. Otro posible factor que puede haber influido son errores humanos asociados a los golpes con el martillo (dinamómetro o posición normal del martillo frente al golpe), o en la medición de las coordenadas (precisión del punto de impacto). Esto se ratifica con los con los resultados mostrados en la tabla 5.3 en la cual se puede apreciar que a medida que la estructura se torna más realista, los error porcentuales de tanto de área como de fuerza, aumentan. Se destaca que en el caso de la CNN ($\approx$ 21 %), se obtuvo un resultado similar a trabajos anteriores, sin embargo trabajando con una estructura mucho más compleja.

Por otro lado el error porcentual de área en los algoritmos, se mostró bastante similar a trabajos anteriores, no mostrando grandes diferencias entre las estructuras anteriormente estudiadas y un fuselaje real.

Tabla 5.3: Tabla comparativa de resultados de distintos algoritmos y estructuras en las cuales se realizaron algoritmos de identificación de impactos.
LS-SVM: Least squares support vector machines, **Kernel-ELM: Kernel extreme leaning machine**, **LME: Linear approximation with maximum entropy**, **PCA: Principal components analisys**, **AE: Autoencoders**, **CNN: Convolutional neural networks**

Referencia	**Algoritmo**	**Estructura**	**Error de área [%]**	**Error de fuerza [%]**	**Error de identificación [%]**
Xu [19]	LS-SVM	Placa simple	1.06	51.2	54.27
Xu-Fu [20]	Kernel-ELM	Placa simple	0.74	-	-
Sanchez-Meruane [3]	LME	**Placa simple**	**0.12**	**7.18**	**0.86**
Meruane-Veliz [4]	PCA+LME	**Placa compuesta**	**0.05**	**10.79**	**0.54**
Meruane-Espinoza [18]	AE+LME	**Cilindro de acero**	**0.10**	**17.09**	**1.71**
Actual trabajo	AE+LME	**Fuselaje**	**0.11**	**39.53**	**4.57**
Actual trabajo	CNN	**Fuselaje**	**0.23**	**21.69**	**5.17**

El gráfico 5.1 muestra el error porcentual de área y de fuerza para los 4 mejores algoritmos de identificación de imapctos, construidos en este trabajo.

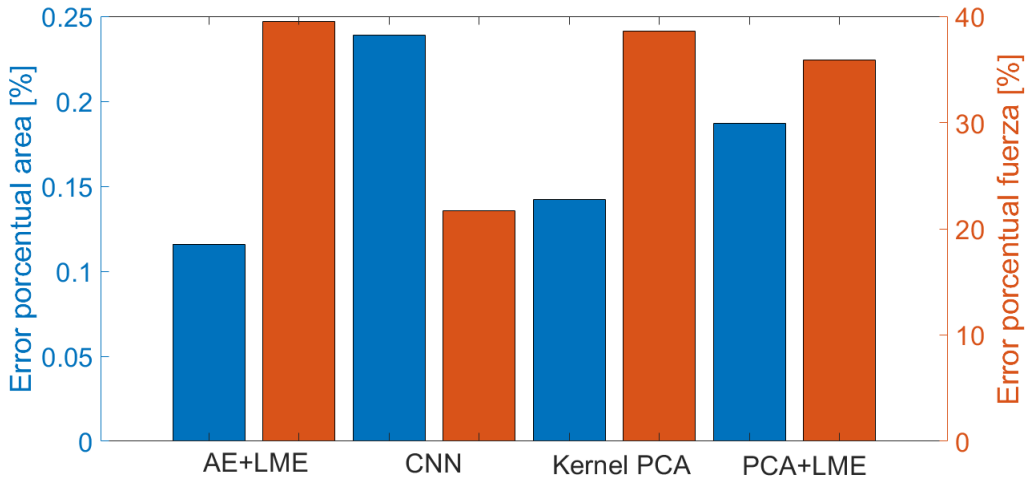

Figura 5.1: Error porcentual de área y fuerza para los 4 mejores algoritmos de identificación de impactos del trabajo actual

5.4. Sobre el sentido físico de la identificación de impactos en el fuselaje de un avión

Con el fin de dar una interpretación física, las figuras 5.2,5.3,5.4 y 5.5, representan gráficamente los datos de prueba vs los impactos predichos de los 4 mejores algoritmos de identificación de impactos, se representa el fuselaje en los ejes x e y en [cm] y en la gama de colores de los círculos, la magnitud de la fuerza en [N].

Para las gráficas se utilizaron una superficie de impacto de diámetro de 8 [cm] (que puede corresponder al impacto de un pájaro en fuselaje) y una superficie de inspección del mismo diámetro (que es el área donde se buscarían daños una vez ocurrido el impacto).

La tabla 5.4 magnitudes físicas que se obtuvieron de la comparación de datos de prueba y los impactos predichos por los algoritmos.

Tabla 5.4: Cantidades físicas de la identificación de impactos

Método	**Distancia promedio entre puntos [cm]**	**Distancia máxima entre puntos [cm]**	**Error de fuerza promedio [N]**
Autoencoders+LME	5.34	15.32	41.50
Redes neuronales convolucionales	7.67	20.87	29.62
Kernel PCA: Gaussiano+LME	5.92	19.57	41.69
PCA+LME	6.74	23.19	39.19

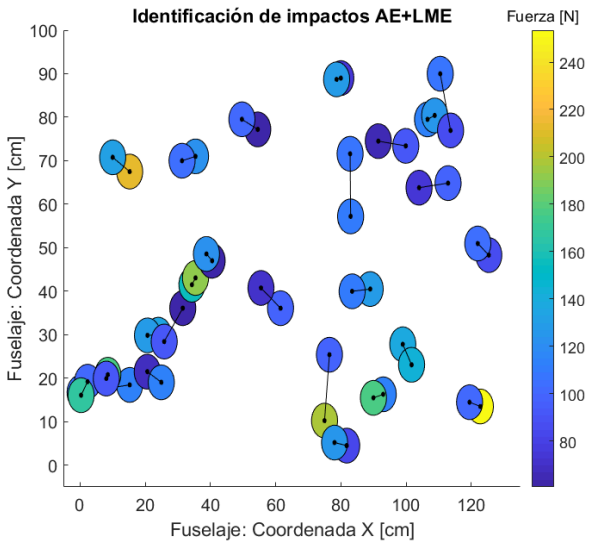

Figura 5.2: Identificación de impactos: Algoritmo AE+LME. Error de identificación de impacto= 4,57 %.

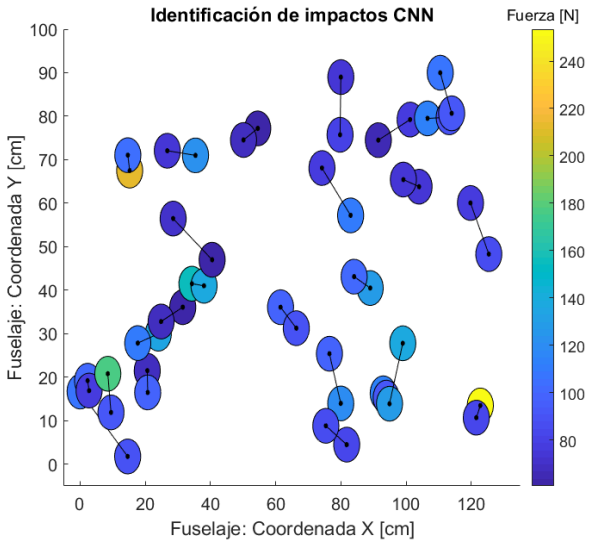

Figura 5.3: Identificación de impactos: CNN. Error de identificación de impacto= 5,17 %.

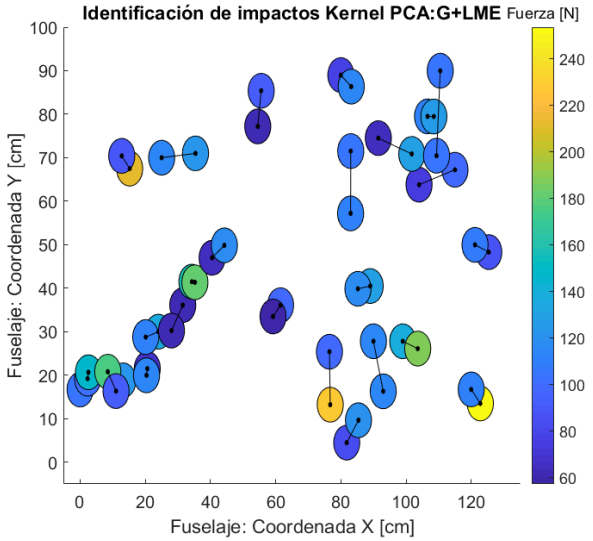

Figura 5.4: Identificación de impactos: Algoritmo Kernel PCA+LME. Error de identificación de impacto= 5,46 %.

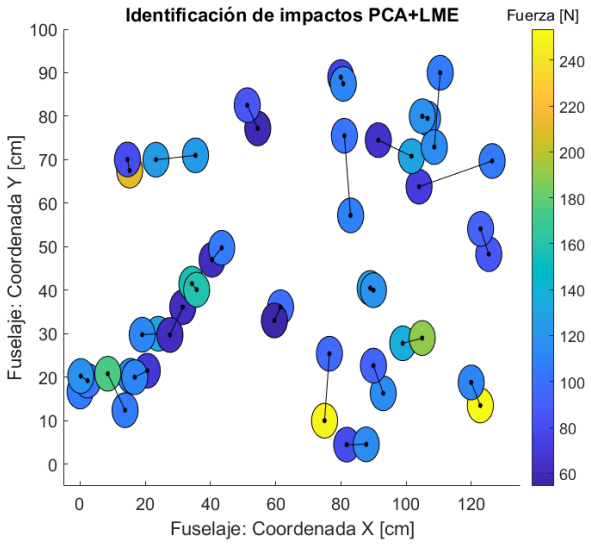

Figura 5.5: Identificación de impactos: Algoritmo PCA+LME. Error de identificación de impacto= 6,70 %.

La tabla 5.4 muestra la distancia máxima entre puntos detectada por cada algoritmo, lo que físicamente correspondería al radio del área de donde se debiesen buscar daños una vez detectado el impacto. Esta distancia no supera los 24 [cm] de radio en los 4 algoritmos mostrados, haciendo en términos de mantenimiento industrial, algoritmos bastante precisos en la detección de la posición del impacto. Además todos los algoritmos, detectan la magnitud de la fuerza en un rango del mismo orden de magnitud. Como ya se discutió previamente el algoritmo que tuvo mejor desempeño en la predicción de la fuerza, fue la CNN y de las figuras mostradas, se puede ver su impacto físico (en el colorbar con una gama de colores).

En el caso del algoritmo AE+LME y la CNN, el error porcentual de fuerza es de 0.11 % en el primer caso y en el segundo es de un 0.23 %, sin embargo esto se traduce físicamente en que en promedio AE+LME, predice con ≈ 2 [cm] más de exactitud que la CNN, por otro lado en el caso de la fuerza en la CNN ese ≈ 18 % más de exactitud, se traduce en 12 [N] de mejoría en promedio en la predicción de la fuerza.

De la tabla 5.4 y de las figuras precedentes se deduce que los algoritmos de identificación de impactos AE, Kernel PCA, PCA, en conjunto con el algoritmo LME, y la utilización de redes neuronales convolucionales, son funcionales con la metodología experimental usada y obteniendo buenos resultados, tanto en la predicción de la posición, como de la fuerza de impacto, lo que permitiría una evaluación en el uso en el marco de mantenimiento industrial.

Desde el punto de vista de las aplicaciones en ingeniería, vemos que estos cuatro algoritmos presentan resultados muy similares, sin embargo la metodología empleada en la construcción de estos algoritmos no es la misma, provocando diferencias de tiempos la construcción de los algoritmos sustanciales, siendo el algoritmo AE+LME el que más tiempo conlleva al ingeniero para construir un modelo funcional, en contraste con PCA+LME, que es de la técnica de más rápido empleo. Por lo tanto, en función de una aplicación, es necesario hacer un "trade-off" entre la precisión que tendrá el algoritmo y el tiempo que se tiene disponible para la implementación.

5.5. Trabajos futuros

5.5.1. CNN

En este trabajo se mostró que el uso de redes neuronales convolucionales, puede obtener buenos resultados en el problema de identificación de impactos, sin embargo su metodología podría ser mejorada, una posible forma de mejorar los resultados sería utilizar arquitecturas que contengan "skip connections", saltos entre las capas convolucionales y la entrada de la red neuronal FC, entregando la posibilidad a la red, de tomar la información que le sea más conveniente para su aprendizaje.

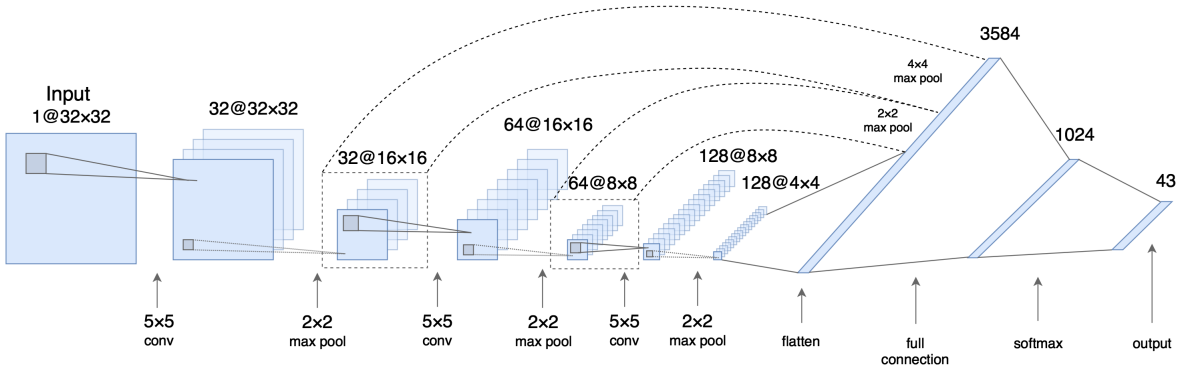

Figura 5.6: Funcionamiento de las "skip connections" en una red neuronal convolucional

Otra forma de mejorar el algoritmo sería probar con más combinaciones de kernels; por ejemplo, trabajando con los 5 tamaños de kernels ocupados y 5 capas, existen 5^5, formas de combinar los kernels en las capas y en este trabajo, solo se probaron 10 formas. Por ejemplo, la teoría dice que es mejor comenzar con kernels más grandes en las capas iniciales e ir achicándolos para encontrar características más abstractas.

5.5.2. Montaje experimental

Dada que los algoritmos, PCA+LME, Kernel PCA+LME, AE+LME y CNN, obtuvieron buenos resultados, construir un montaje experimental que cubra más superficie del fuselaje podría ser una buena idea para probar los algoritmos.

Capítulo 6

Conclusiones

Este trabajo presenta la metodología, aplicación y evaluación en una estructura real, de dos estrategias de identificación de impactos. La primera estrategia constituida de una técnica de reducción de dimensionalidad de datos, junto a la utilización de una aproximación lineal del principio de máxima entropía, que permite obtener la ubicación y magnitud de los impactos de prueba. La segunda estrategia ocupa redes neuronales convolucionales, aplicadas al problema de identificación de impactos.

Los datos de vibraciones fueron obtenidos con acelerómetros piezoeléctricos pegados a la superficie del fuselaje estudiado.

Los resultados indican que ambas estrategias de identificación de impactos son capaces de obtener resultados aplicables en estructuras reales. Los mejores resultados son los obtenidos por los algoritmos, AE+LME y CNN. Comparando sus resultados, se muestra que AE+LME predice con más exactitud la posición del impacto que la fuerza de este, ocurriendo todo lo contrario con la CNN, la que presenta mejor desempeño en la detección de la magnitud del impacto, que su posición. Este es un resultado relevante dado que en la caracterización de un posible daño en la estructura, la predicción de la fuerza tiene mucho más interés.

Las técnicas de reducción de dimensionalidad que no emplean gráficos, como PCA, kernel PCA y autoencoders, funcionan mejor, que las que si lo hacen como Isomap y MDS. Además se ve que la versión probabilística de PCA no obtiene buenos resultados, muy probablemente debido a problemas de sobre ajuste de las densidades de probabilidades con los datos de entrenamiento. Los autoencoders proporcionan los mejores resultados, muy probablemente porque pueden aprender las estructuras estructuras de datos altamente variables y no lineales, haciendo posible proyectar nuevos datos de alta dimensión en la representación de baja dimensión existente, con errores muy bajos.

Finalmente, es posible afirmar que las metodologías implementadas cumplen con la función de identificar la ubicación y magnitud de los impactos efectuados en la estructura, permitiendo afirmar que que tanto el objetivo general como los objetivos específicos fueron cumplidos, entregando entregando resultados satisfactorios en cuatro de los algoritmos de identificación de impactos, que sin embargo pueden ser aún mejorados.

Bibliografía

[1] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res*, vol. 10, no. 66-71, p. 13, 2009.

[2] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[3] N. Sanchez, V. Meruane, and A. Ortiz-Bernardin, "A novel impact identification algorithm based on a linear approximation with maximum entropy," *Smart Materials and Structures*, vol. 25, no. 9, p. 095050, 2016.

[4] V. Meruane, P. Véliz, E. López Droguett, and A. Ortiz-Bernardin, "Impact location and quantification on an aluminum sandwich panel using principal component analysis and linear approximation with maximum entropy," *Entropy*, vol. 19, no. 4, p. 137, 2017.

[5] C. Modarres, N. Astorga, E. L. Droguett, and V. Meruane, "Convolutional neural networks for automated damage recognition and damage type identification," *Structural Control and Health Monitoring*, vol. 25, no. 10, p. e2230, 2018.

[6] V. A. d. Fierro Aguirre, "Detección de daños en una placa de material compuesto tipo panal de abeja mediante métodos de aprendizaje supervisado," 2014.

[7] V. Meruane, "Dinámica estructural. apuntes para el curso me706," *Santiago: Departamento de Ingeniería Mecánica, Universidad de Chile*, 2013.

[8] P. E. Véliz Alonso, "Identificación de impactos en una placa compuesta utilizando el principio de máxima entropía y análisis de componentes principales," 2017.

[9] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. .ºReilly Media, Inc.", 2017.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[12] L. G. Sánchez, G. A. Osorio, and J. F. Suárez, "Introduction to kernel pca and other spectral methods applied to unsupervised learning," *Revista Colombiana de Estadística*, vol. 31, no. 1,

pp. 19–40, 2008.

[13] J. Valencia-Aguirre, G. Daza-Santacoloma, C. D. Acosta, and G. Castellanos-Domínguez, "Comparación de métodos de reducción de dimensión basados en análisis por localidades," *TecnoLógicas*, no. 25, pp. 131–150, 2010.

[14] L. Liberti and C. d'Ambrosio, "The isomap algorithm in distance geometry," 2017.

[15] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[16] V. Meruane, C. Espinoza, E. Lopez, and A. Ortiz-Bernardin, "Impact identification using nonlinear dimensionality reduction and supervised learning," *Smart Materials and Structures, En verificación*.

[17] C. d. l. Á. Espinoza Quitral, "Identificación de impactos en una estructura tridimensional utilizando autoencoders y una aproximación lineal del principio de máxima entropía," 2018.

[18] V. Meruane, C. Espinoza, E. L. Droguett, and A. Ortiz-Bernardin, "Impact identification using nonlinear dimensionality reduction and supervised learning," *Smart Materials and Structures*, 2019.

[19] Q. Xu, "Impact detection and location for a plate structure using least squares support vector machines," *Structural Health Monitoring*, vol. 13, no. 1, pp. 5–18, 2014.

[20] H. Fu, C.-M. Vong, P.-K. Wong, and Z. Yang, "Fast detection of impact location using kernel extreme learning machine," *Neural Computing and Applications*, vol. 27, no. 1, pp. 121–130, 2016.

Apéndice A

Algoritmo LME

Las figuras A.1, A.3, A.2, A.4 y A.5 muestran el rendimiento del algoritmo LME sin usar técnicas de reducción de dimensionalidad en función del número de vecinos.

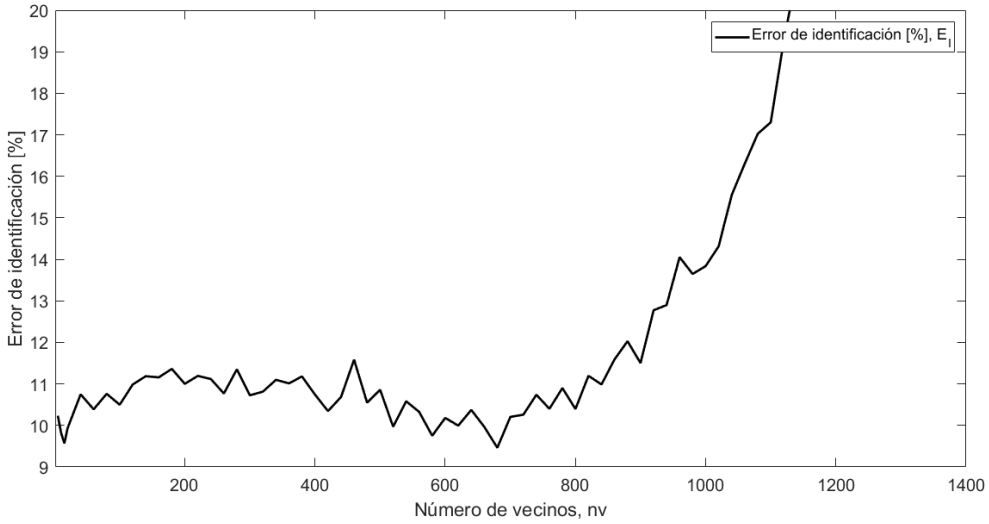

Figura A.1: Error de identificación de impactos algoritmo LME

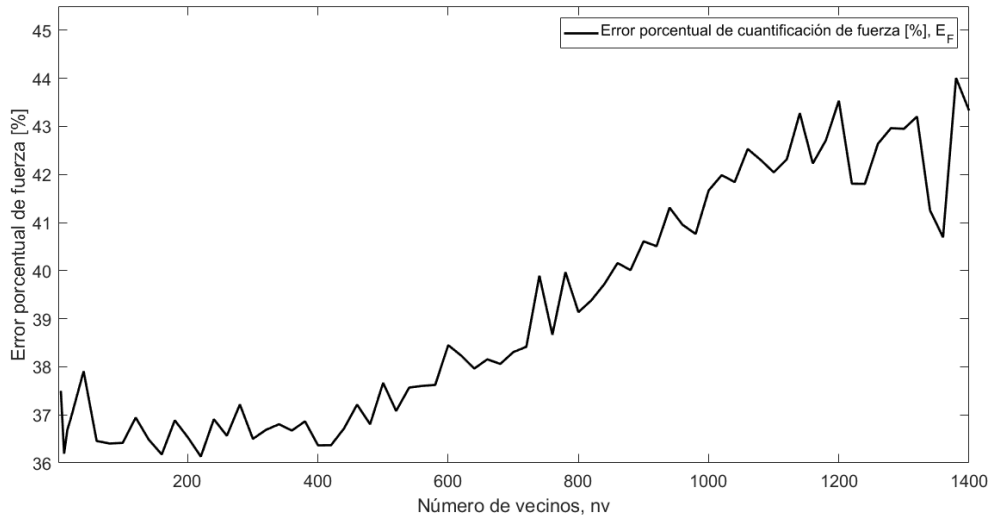

Figura A.2: Error porcentual de fuerza algoritmo LME

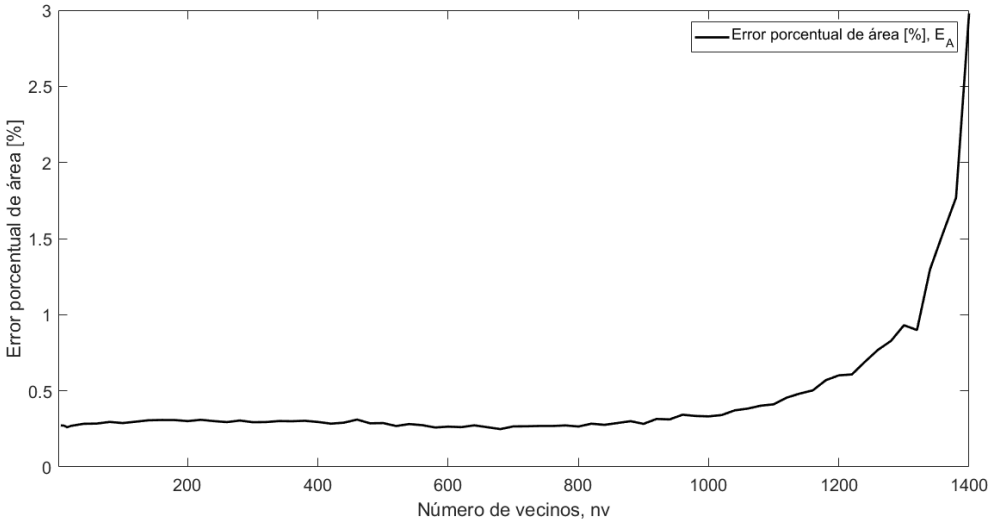

Figura A.3: Error porcentual de área algoritmo LME

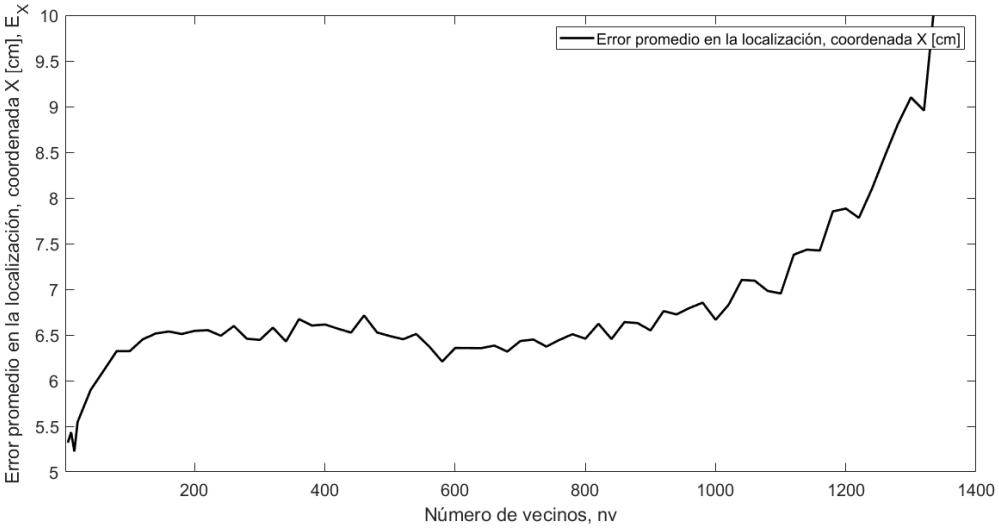

Figura A.4: Error promedio coordenada X algoritmo LME

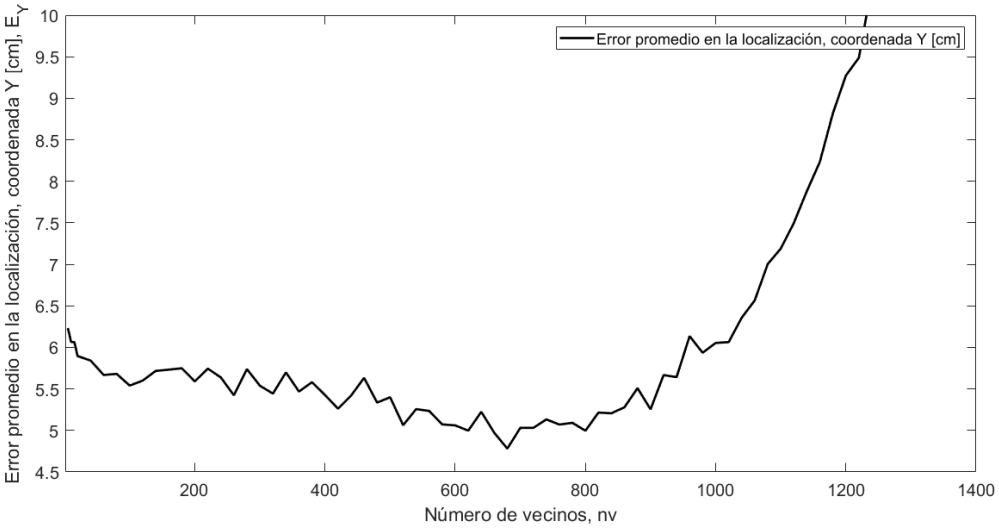

Figura A.5: Error promedio coordenada Y algoritmo LME

Tabla A.1: Resultados del algoritmo LME sin usar técnicas de reducción de dimensionalidad

Resultados del algoritmo LME	**Valor**
Número de vecinos del algortimo LME	680
Error promedio en la localización, coordenada X	6.31 [cm]
Error promedio en la localización, coordenada Y	4.78 [cm]
Error porcentual de área	0.24 [%]
Error de cuantificación de fuerza	38.06 [%]
Error de identificación de impacto normalizado	9.46 [%]

A.1. Matriz

La figura A.6 muestra la matriz de un impacto transformada en imagen y replicada 35 veces en la dirección horizontal, para aumentar la cantidad de pixeles de esta (un impacto esta representado por el cuadro rojo).

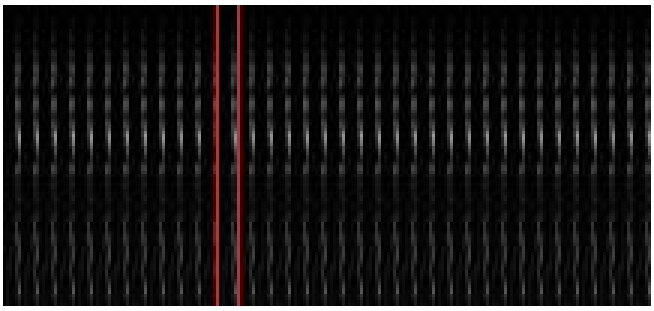

Figura A.6: Matriz de impacto transformada a imagen

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 6, 8)         32
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 6, 8)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 6, 8)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 6, 16)         272
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 25, 6, 16)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 25, 6, 16)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 6, 32)         4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 6, 32)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 12, 6, 32)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 6, 64)         10304
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 128)         24704
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 6, 128)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 3, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2360320
_________________________________________________________________
activation_1 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 1539
_________________________________________________________________
activation_3 (Activation)    (None, 3)                 0
=================================================================
Total params: 2,926,611
Trainable params: 2,926,611
Non-trainable params: 0
_________________________________________________________________
```

Figura A.7: Arquitectura de la CNN con mejor rendimiento

Apéndice A

Códigos

A.1. LME

```
clear all;
close all;
clc

addpath('C:\LME');
%%%%BUILD DATABASES%%%%%%
load Test;
load Train;

%%%%ESTIMATE IMPACTS %%%%%%%%%

%read observation vector
load Yt; %testing data
load Yi; %training data

nv=[5:5:15 20:20:1400];
tic
for kk=1:length(nv)
    kk
for k=1:25
    k
    gamma=gammanb(Test(:,k),Train,nv(kk)); %determinate parameter gamma in eq. (10)
    [phi,contribute] = lme(Test(:,k),Train,gamma); %solve LME
    Ye(:,k)=Yi(:,contribute)*phi;
end

Impact{1,kk}=Ye(1,:);
Impact{2,kk}=Ye(2,:);
Impact{3,kk}=Ye(3,:);
```

```
%%COMPUTE ERRORS AND PLOT RESULTS%%
n=25;
E_x=0;
E_y=0;
E_f=0;

for k=1:25
    E_x = E_x + abs(Ye(1,k)-Yt(1,k));
    E_y = E_y + abs(Ye(2,k)-Yt(2,k));
    E_f = E_f + abs((Ye(3,k)-Yt(3,k))/(Yt(3,k)));
end
E_X(kk)=E_x/n;
E_Y(kk)=E_y/n;
E_A(kk) = (100*(E_x*E_y))/((n^2)*(135*90));
E_F(kk) = (E_f*100)/(n);
E_I(kk) = E_A(kk)*E_F(kk)
end
save E_X E_X
save E_Y E_Y
save E_A E_A;
save E_F E_F;
save E_I E_I;
save Impact Impact

toc
disp('Termino de correr')
```

A.2. PCA

A.2.1. Construcción de la base de datos dimensión reducida

```
clear all;
close all;
clc

addpath('C:\LME');

%%BUILD DATABASES%%
ni=100;

load XP_1;
load XP_2;
load XP_3;
load XP_4;
load XP_5;
```

```
load XP_6;

load Xi1_E;
load Xi2_E;
load Xi3_E;
load Xi4_E;
load Xi5_E;
load Xi6_E;

Z1=Xi1_E(1:100,:);
Z2=Xi2_E(1:100,:);
Z3=Xi3_E(1:100,:);
Z4=Xi4_E(1:100,:);
Z5=Xi5_E(1:100,:);
Z6=Xi6_E(1:100,:);

desv(1)=std(reshape(Z1,100*1400,1));
desv(2)=std(reshape(Z2,100*1400,1));
desv(3)=std(reshape(Z3,100*1400,1));
desv(4)=std(reshape(Z4,100*1400,1));
desv(5)=std(reshape(Z5,100*1400,1));
desv(6)=std(reshape(Z6,100*1400,1));

Z1=Z1/desv(1);
Z2=Z2/desv(2);
Z3=Z3/desv(3);
Z4=Z4/desv(4);
Z5=Z5/desv(5);
Z6=Z6/desv(6);

XP_1=XP_1(1:100,:)./desv(1);
XP_2=XP_2(1:100,:)./desv(2);
XP_3=XP_3(1:100,:)./desv(3);
XP_4=XP_4(1:100,:)./desv(4);
XP_5=XP_5(1:100,:)./desv(5);
XP_6=XP_6(1:100,:)./desv(6);

% Build matrices Pi
covariance1 = (Z1*Z1')./(size(Z1,1)-1);
[Vi1 Di1] = eigs(covariance1,ni);
covariance2 = (Z2*Z2')./(size(Z2,1)-1);
[Vi2 Di2] = eigs(covariance2,ni);
covariance3 = (Z3*Z3')./(size(Z3,1)-1);
[Vi3 Di3] = eigs(covariance3,ni);
covariance4 = (Z4*Z4')./(size(Z4,1)-1);
[Vi4 Di4] = eigs(covariance4,ni);
covariance5 = (Z5*Z5')./(size(Z5,1)-1);
[Vi5 Di5] = eigs(covariance5,ni);
covariance6 = (Z6*Z6')./(size(Z6,1)-1);
[Vi6 Di6] = eigs(covariance6,ni);
```

```
a1=cumsum(diag(Di1))/sum(diag(Di1));
a2=cumsum(diag(Di2))/sum(diag(Di2));
a3=cumsum(diag(Di3))/sum(diag(Di3));
a4=cumsum(diag(Di4))/sum(diag(Di4));
a5=cumsum(diag(Di5))/sum(diag(Di5));
a6=cumsum(diag(Di6))/sum(diag(Di6));

for i=1:ni
    if
        a1(i)>=0.9999&&a2(i)>=0.9999&&a3(i)>=0.9999&&a4(i)>=0.9999&&a5(i)>=0.9999&&a6(i)>=0.9999
        nt=i;
        break
    end
end

P1=Vi1(:,1:nt)';
P2=Vi2(:,1:nt)';
P3=Vi3(:,1:nt)';
P4=Vi4(:,1:nt)';
P5=Vi5(:,1:nt)';
P6=Vi6(:,1:nt)';

% Build matrices Si
S1 = P1*Z1;
S2 = P2*Z2;
S3 = P3*Z3;
S4 = P4*Z4;
S5 = P5*Z5;
S6 = P6*Z6;
Train=[S1;S2;S3;S4;S5;S6];

%PCs for the testing data
St1 = P1*XP_1;
St2 = P2*XP_2;
St3 = P3*XP_3;
St4 = P4*XP_4;
St5 = P5*XP_5;
St6 = P6*XP_6;

Test=[St1;St2;St3;St4;St5;St6];

save Test Test;
save Train Train;
```

A.3. Autoencoders

A.3.1. Ejemplo de uno de los autoencoders utilizados

```
import keras
import scipy.io
import scipy.io as sio
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy.matlib
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential

from keras.layers import Input, Dense, Dropout, Activation
from keras.models import Model
import numpy as np
import pandas as pd
get_ipython().run_line_magic('matplotlib', 'inline')
from sklearn.externals import joblib
from keras import regularizers

# Data set
x_train = scipy.io.loadmat('Xi1_100.mat')['Xi1_100']
x_train = x_train.transpose()

t = scipy.io.loadmat('XP_1_100.mat')['XP_1_100']
x_test = t.transpose()
x_val = x_test

# Normalizacin de los datos
sc = MinMaxScaler(feature_range=(0,1))
x_train = sc.fit_transform(x_train)
x_val = sc.transform(x_val)
x_test = sc.transform(x_test)
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))


# Dimensin de entrada
dim_input= Input(shape=(100,))
forma_input=(x_train.shape[0],x_train.shape[1],1)

# Autoencoder
b1=keras.layers.BatchNormalization()(dim_input)
z = Dense(units=49, activation='sigmoid')(b1)
b2=keras.layers.BatchNormalization()(z)
decoded = Dense(units=100, activation='sigmoid')(b2)
```

```
# Definicin del modelo
autoencoder=Model(dim_input, decoded)
encoder = Model(dim_input, z)

# Modelos
autoencoder.summary()
encoder.summary()

# Parmetros de entrenamiento
batch_size = 100
epochs = 20000

# Hiperparmetros
autoencoder.compile(optimizer=keras.optimizers.Adam(lr=4E-5), loss='mse')

# Fit del autoencoder, valdiacin 10%
history=autoencoder.fit(x_train, x_train,
               validation_split=0.1,
               epochs=epochs,
               batch_size=batch_size,
               shuffle=True)

import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager

# Historial de la funcion de costos en las epocas
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('RMSE', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Prediccin
encoded_test = encoder.predict(x_test)
encoded_train = encoder.predict(x_train)

# Guardar los datos.
numpy.savetxt('AE1_train.txt' , encoded_train.transpose() , delimiter=',')
numpy.savetxt('AE1_test.txt' , encoded_test.transpose() , delimiter=',')

filename='Autoencoder_1_100'
joblib.dump(autoencoder,filename)
```

```
# Data set
x_train = scipy.io.loadmat('Xi2_100.mat')['Xi2_100']
x_train = x_train.transpose()

t = scipy.io.loadmat('XP_2_100.mat')['XP_2_100']
x_test = t.transpose()
x_val = x_test

# Normalizacin de los datos
sc = MinMaxScaler(feature_range=(0,1))
x_train = sc.fit_transform(x_train)
x_val = sc.transform(x_val)
x_test = sc.transform(x_test)
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))

# Dimensin de entrada
dim_input= Input(shape=(100,))
forma_input=(x_train.shape[0],x_train.shape[1],1)

# Autoencoder
b1=keras.layers.BatchNormalization()(dim_input)
z = Dense(units=49, activation='sigmoid')(b1)
b2=keras.layers.BatchNormalization()(z)
decoded = Dense(units=100, activation='sigmoid')(b2)

# Definicin del modelo
autoencoder=Model(dim_input, decoded)
encoder = Model(dim_input, z)

# Modelos
autoencoder.summary()
encoder.summary()

# Parmetros de entrenamiento
batch_size = 100
epochs = 20000

# Hiperparmetros
autoencoder.compile(optimizer=keras.optimizers.Adam(lr=4E-5), loss='mse')

# Fit del autoencoder, valdiacin 10%
history=autoencoder.fit(x_train, x_train,
                validation_split=0.1,
                epochs=epochs,
                batch_size=batch_size,
                shuffle=True)

import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager
```

```
# Historial de la funcion de costos en las epocas
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('RMSE', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Prediccin
encoded_test = encoder.predict(x_test)
encoded_train = encoder.predict(x_train)

# Guardar los datos.
numpy.savetxt('AE2_train.txt' , encoded_train.transpose() , delimiter=',')
numpy.savetxt('AE2_test.txt' , encoded_test.transpose() , delimiter=',')

filename='Autoencoder_2_100'
joblib.dump(autoencoder,filename)

# Data set
x_train = scipy.io.loadmat('Xi3_100.mat')['Xi3_100']
x_train = x_train.transpose()

t = scipy.io.loadmat('XP_3_100.mat')['XP_3_100']
x_test = t.transpose()
x_val = x_test

# Normalizacin de los datos
sc = MinMaxScaler(feature_range=(0,1))
x_train = sc.fit_transform(x_train)
x_val = sc.transform(x_val)
x_test = sc.transform(x_test)
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))

# Dimensin de entrada
dim_input= Input(shape=(100,))
forma_input=(x_train.shape[0],x_train.shape[1],1)

# Autoencoder
b1=keras.layers.BatchNormalization()(dim_input)
z = Dense(units=49, activation='sigmoid')(b1)
b2=keras.layers.BatchNormalization()(z)
decoded = Dense(units=100, activation='sigmoid')(b2)
```

```
# Definicin del modelo
autoencoder=Model(dim_input, decoded)
encoder = Model(dim_input, z)

# Modelos
autoencoder.summary()
encoder.summary()

# Parmetros de entrenamiento
batch_size = 100
epochs = 20000

# Hiperparmetros
autoencoder.compile(optimizer=keras.optimizers.Adam(lr=4E-5), loss='mse')

# Fit del autoencoder, valdiacin 10%
history=autoencoder.fit(x_train, x_train,
               validation_split=0.1,
               epochs=epochs,
               batch_size=batch_size,
               shuffle=True)

import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager

# Historial de la funcion de costos en las epocas
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('RMSE', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Prediccin
encoded_test = encoder.predict(x_test)
encoded_train = encoder.predict(x_train)

# Guardar los datos.
numpy.savetxt('AE3_train.txt' , encoded_train.transpose() , delimiter=',')
numpy.savetxt('AE3_test.txt' , encoded_test.transpose() , delimiter=',')

filename='Autoencoder_3_100'
joblib.dump(autoencoder,filename)
```

```
# Data set
x_train = scipy.io.loadmat('Xi4_100.mat')['Xi4_100']
x_train = x_train.transpose()

t = scipy.io.loadmat('XP_4_100.mat')['XP_4_100']
x_test = t.transpose()
x_val = x_test

# Normalizacin de los datos
sc = MinMaxScaler(feature_range=(0,1))
x_train = sc.fit_transform(x_train)
x_val = sc.transform(x_val)
x_test = sc.transform(x_test)
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))

# Dimensin de entrada
dim_input= Input(shape=(100,))
forma_input=(x_train.shape[0],x_train.shape[1],1)

# Autoencoder

b1=keras.layers.BatchNormalization()(dim_input)
z = Dense(units=49, activation='sigmoid')(b1)
b2=keras.layers.BatchNormalization()(z)
decoded = Dense(units=100, activation='sigmoid')(b2)

# Definicin del modelo
autoencoder=Model(dim_input, decoded)
encoder = Model(dim_input, z)

# Modelos
autoencoder.summary()
encoder.summary()

# Parmetros de entrenamiento
batch_size = 100
epochs = 20000

# Hiperparmetros
autoencoder.compile(optimizer=keras.optimizers.Adam(lr=4E-5), loss='mse')

# Fit del autoencoder, valdiacin 10%
history=autoencoder.fit(x_train, x_train,
                validation_split=0.1,
                epochs=epochs,
                batch_size=batch_size,
                shuffle=True)

import matplotlib.pyplot as plt
```

```
import matplotlib.font_manager as font_manager

# Historial de la funcion de costos en las epocas
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('RMSE', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Prediccin
encoded_test = encoder.predict(x_test)
encoded_train = encoder.predict(x_train)

# Guardar los datos.
numpy.savetxt('AE4_train.txt' , encoded_train.transpose() , delimiter=',')
numpy.savetxt('AE4_test.txt' , encoded_test.transpose() , delimiter=',')

filename='Autoencoder_4_100'
joblib.dump(autoencoder,filename)

# Data set
x_train = scipy.io.loadmat('Xi5_100.mat')['Xi5_100']
x_train = x_train.transpose()

t = scipy.io.loadmat('XP_5_100.mat')['XP_5_100']
x_test = t.transpose()
x_val = x_test

# Normalizacin de los datos
sc = MinMaxScaler(feature_range=(0,1))
x_train = sc.fit_transform(x_train)
x_val = sc.transform(x_val)
x_test = sc.transform(x_test)
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))

# Dimensin de entrada
dim_input= Input(shape=(100,))
forma_input=(x_train.shape[0],x_train.shape[1],1)

# Autoencoder
b1=keras.layers.BatchNormalization()(dim_input)
z = Dense(units=49, activation='sigmoid')(b1)
b2=keras.layers.BatchNormalization()(z)
```

```
decoded = Dense(units=100, activation='sigmoid')(b2)

# Definicin del modelo
autoencoder=Model(dim_input, decoded)
encoder = Model(dim_input, z)

# Modelos
autoencoder.summary()
encoder.summary()

# Parmetros de entrenamiento
batch_size = 100
epochs = 20000

# Hiperparmetros
autoencoder.compile(optimizer=keras.optimizers.Adam(lr=4E-5), loss='mse')

# Fit del autoencoder, valdiacin 10%
history=autoencoder.fit(x_train, x_train,
               validation_split=0.1,
               epochs=epochs,
               batch_size=batch_size,
               shuffle=True)

import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager

# Historial de la funcion de costos en las epocas
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('RMSE', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()


# Prediccin
encoded_test = encoder.predict(x_test)
encoded_train = encoder.predict(x_train)

# Guardar los datos.
numpy.savetxt('AE5_train.txt' , encoded_train.transpose() , delimiter=',')
numpy.savetxt('AE5_test.txt' , encoded_test.transpose() , delimiter=',')

filename='Autoencoder_5_100'
```

```
joblib.dump(autoencoder,filename)

# Data set
x_train = scipy.io.loadmat('Xi6_100.mat')['Xi6_100']
x_train = x_train.transpose()

t = scipy.io.loadmat('XP_6_100.mat')['XP_6_100']
x_test = t.transpose()
x_val = x_test

# Normalizacin de los datos
sc = MinMaxScaler(feature_range=(0,1))
x_train = sc.fit_transform(x_train)
x_val = sc.transform(x_val)
x_test = sc.transform(x_test)
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))

# Dimensin de entrada
dim_input= Input(shape=(100,))
forma_input=(x_train.shape[0],x_train.shape[1],1)

# Autoencoder
b1=keras.layers.BatchNormalization()(dim_input)
z = Dense(units=49, activation='sigmoid')(b1)
b2=keras.layers.BatchNormalization()(z)
decoded = Dense(units=100, activation='sigmoid')(b2)

# Definicin del modelo
autoencoder=Model(dim_input, decoded)
encoder = Model(dim_input, z)

# Modelos
autoencoder.summary()
encoder.summary()

# Parmetros de entrenamiento
batch_size = 100
epochs = 20000


# Hiperparmetros
autoencoder.compile(optimizer=keras.optimizers.Adam(lr=4E-5), loss='mse')

# Fit del autoencoder, valdiacin 10%
history=autoencoder.fit(x_train, x_train,
                validation_split=0.1,
                epochs=epochs,
                batch_size=batch_size,
                shuffle=True)
```

```
import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager

# Historial de la funcion de costos en las epocas
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('RMSE', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Prediccin
encoded_test = encoder.predict(x_test)
encoded_train = encoder.predict(x_train)

# Guardar los datos.
numpy.savetxt('AE6_train.txt' , encoded_train.transpose() , delimiter=',')
numpy.savetxt('AE6_test.txt' , encoded_test.transpose() , delimiter=',')

filename='Autoencoder_6_100'
joblib.dump(autoencoder,filename)
```

A.4. Kernel PCA

```
% clear all;
% close all;
% clc
function [Test Train]=Data_Bases(nn)
addpath('C:\LME');
addpath('C:\drtoolbox');
addpath('C:\drtoolbox\techniques');
addpath('C:\drtoolbox\gui');

%%BUILD DATABASES%%
ni=100;

%Read training data

load Xi1_E;
load Xi2_E;
load Xi3_E;
load Xi4_E;
load Xi5_E;
load Xi6_E;

load XP_1;
load XP_2;
load XP_3;
load XP_4;
load XP_5;
load XP_6;

% Data selection
Z1=Xi1_E(1:100,:);
Z2=Xi2_E(1:100,:);
Z3=Xi3_E(1:100,:);
Z4=Xi4_E(1:100,:);
Z5=Xi5_E(1:100,:);
Z6=Xi6_E(1:100,:);

% Data Normalization
desv(1)=std(reshape(Z1,100*1400,1));
desv(2)=std(reshape(Z2,100*1400,1));
desv(3)=std(reshape(Z3,100*1400,1));
desv(4)=std(reshape(Z4,100*1400,1));
desv(5)=std(reshape(Z5,100*1400,1));
desv(6)=std(reshape(Z6,100*1400,1));
save desv desv

% Normalization
load desv desv
```

```
Z1=Z1./desv(1);
Z2=Z2./desv(2);
Z3=Z3./desv(3);
Z4=Z4./desv(4);
Z5=Z5./desv(5);
Z6=Z6./desv(6);

XP_1=XP_1(1:100,:)./desv(1);
XP_2=XP_2(1:100,:)./desv(2);
XP_3=XP_3(1:100,:)./desv(3);
XP_4=XP_4(1:100,:)./desv(4);
XP_5=XP_5(1:100,:)./desv(5);
XP_6=XP_6(1:100,:)./desv(6);


% Usar esta cuando es kernel gaussiano
[S1, M1] = compute_mapping(Z1', 'KernelPCA', 49,'gauss',nn);
[S2, M2] = compute_mapping(Z2', 'KernelPCA', 49,'gauss',nn);
[S3, M3] = compute_mapping(Z3', 'KernelPCA', 49,'gauss',nn);
[S4, M4] = compute_mapping(Z4', 'KernelPCA', 49,'gauss',nn);
[S5, M5] = compute_mapping(Z5', 'KernelPCA', 49,'gauss',nn);
[S6, M6] = compute_mapping(Z6', 'KernelPCA', 49,'gauss',nn);

% Usar esta cuando es kernel polinomial
[S1, M1] = compute_mapping(Z1', 'KernelPCA', 49,'poly',0,nn);
[S2, M2] = compute_mapping(Z2', 'KernelPCA', 49,'poly',0,nn);
[S3, M3] = compute_mapping(Z3', 'KernelPCA', 49,'poly',0,nn);
[S4, M4] = compute_mapping(Z4', 'KernelPCA', 49,'poly',0,nn);
[S5, M5] = compute_mapping(Z5', 'KernelPCA', 49,'poly',0,nn);
[S6, M6] = compute_mapping(Z6', 'KernelPCA', 49,'poly',0,nn);

St1 = out_of_sample(XP_1', M1);
St2 = out_of_sample(XP_2', M2);
St3 = out_of_sample(XP_3', M3);
St4 = out_of_sample(XP_4', M4);
St5 = out_of_sample(XP_5', M5);
St6 = out_of_sample(XP_6', M6);

Train=[S1';S2';S3';S4';S5';S6'];
Test=[St1';St2';St3';St4';St5';St6'];
end

% save Test Test;
% save Train Train;

% Llamar Data_Bases de otro script y aplicar la reduccin de dimensionalidad
```

A.5. Isomap

```
% clear all;
% close all;
% clc

function [Test Train]=Data_Bases(nn)
%%%%%%%%%%%%%%%%%%%%%%%%%%%BUILD
    DATABASES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ni=100;
% Path
addpath('C:\LME');
addpath('C:\drtoolbox');
addpath('C:\drtoolbox\techniques');
addpath('C:\drtoolbox\gui');

%Read training data

load Xi1_E;
load Xi2_E;
load Xi3_E;
load Xi4_E;
load Xi5_E;
load Xi6_E;

load XP_1;
load XP_2;
load XP_3;
load XP_4;
load XP_5;
load XP_6;

% Data selection
Z1=Xi1_E(1:100,:);
Z2=Xi2_E(1:100,:);
Z3=Xi3_E(1:100,:);
Z4=Xi4_E(1:100,:);
Z5=Xi5_E(1:100,:);
Z6=Xi6_E(1:100,:);

% Data Normalization
desv(1)=std(reshape(Z1,100*1400,1));
desv(2)=std(reshape(Z2,100*1400,1));
desv(3)=std(reshape(Z3,100*1400,1));
desv(4)=std(reshape(Z4,100*1400,1));
desv(5)=std(reshape(Z5,100*1400,1));
desv(6)=std(reshape(Z6,100*1400,1));
save desv desv
```

```
Z1=Z1/desv(1);
Z2=Z2/desv(2);
Z3=Z3/desv(3);
Z4=Z4/desv(4);
Z5=Z5/desv(5);
Z6=Z6/desv(6);

% Normalization
load desv desv
Z1=Z1./desv(1);
Z2=Z2./desv(2);
Z3=Z3./desv(3);
Z4=Z4./desv(4);
Z5=Z5./desv(5);
Z6=Z6./desv(6);

XP_1=XP_1(1:100,:)./desv(1);
XP_2=XP_2(1:100,:)./desv(2);
XP_3=XP_3(1:100,:)./desv(3);
XP_4=XP_4(1:100,:)./desv(4);
XP_5=XP_5(1:100,:)./desv(5);
XP_6=XP_6(1:100,:)./desv(6);

[S1, M1] = compute_mapping(Z1','Isomap', 49,nn,'Matlab');
[S2, M2] = compute_mapping(Z2','Isomap', 49,nn,'Matlab');
[S3, M3] = compute_mapping(Z3','Isomap', 49,nn,'Matlab');
[S4, M4] = compute_mapping(Z4','Isomap', 49,nn,'Matlab');
[S5, M5] = compute_mapping(Z5','Isomap', 49,nn,'Matlab');
[S6, M6] = compute_mapping(Z6','Isomap', 49,nn,'Matlab');

St1 = out_of_sample(XP_1', M1);
St2 = out_of_sample(XP_2', M2);
St3 = out_of_sample(XP_3', M3);
St4 = out_of_sample(XP_4', M4);
St5 = out_of_sample(XP_5', M5);
St6 = out_of_sample(XP_6', M6);

Train=[S1';S2';S3';S4';S5';S6'];
Test=[St1';St2';St3';St4';St5';St6'];

save Test Test;
save Train Train;

end

% Llamar Data_Bases de otro script y aplicar la reduccin de dimensionalidad
```

A.6. Escalamiento multidimensional

```
% clear all;
% close all;
% clc

function [Test Train]=Data_Bases(nn)

nn=1;
addpath('C:\LME');
addpath('C:\drtoolbox');
addpath('C:\drtoolbox\techniques');
addpath('C:\drtoolbox\gui');

%%BUILD DATABASES%
ni=100;
%Read training data

load Xi1_E;
load Xi2_E;
load Xi3_E;
load Xi4_E;
load Xi5_E;
load Xi6_E;

load XP_1;
load XP_2;
load XP_3;
load XP_4;
load XP_5;
load XP_6;

% Data selection
Z1=Xi1_E(1:100,:);
Z2=Xi2_E(1:100,:);
Z3=Xi3_E(1:100,:);
Z4=Xi4_E(1:100,:);
Z5=Xi5_E(1:100,:);
Z6=Xi6_E(1:100,:);

% Data Normalization
desv(1)=std(reshape(Z1,100*1400,1));
desv(2)=std(reshape(Z2,100*1400,1));
desv(3)=std(reshape(Z3,100*1400,1));
desv(4)=std(reshape(Z4,100*1400,1));
desv(5)=std(reshape(Z5,100*1400,1));
desv(6)=std(reshape(Z6,100*1400,1));
save desv desv
```

```
% Normalization
load desv desv
Z1=Z1./desv(1);
Z2=Z2./desv(2);
Z3=Z3./desv(3);
Z4=Z4./desv(4);
Z5=Z5./desv(5);
Z6=Z6./desv(6);

XP_1=XP_1(1:100,:)./desv(1);
XP_2=XP_2(1:100,:)./desv(2);
XP_3=XP_3(1:100,:)./desv(3);
XP_4=XP_4(1:100,:)./desv(4);
XP_5=XP_5(1:100,:)./desv(5);
XP_6=XP_6(1:100,:)./desv(6);

[S1, M1] = compute_mapping(Z1', 'MDS');
[S2, M2] = compute_mapping(Z2', 'MDS');
[S3, M3] = compute_mapping(Z3', 'MDS');
[S4, M4] = compute_mapping(Z4', 'MDS');
[S5, M5] = compute_mapping(Z5', 'MDS');
[S6, M6] = compute_mapping(Z6', 'MDS');

[St1, Mt1] = compute_mapping(XP_1', 'MDS');
[St2, Mt2] = compute_mapping(XP_2', 'MDS');
[St3, Mt3] = compute_mapping(XP_3', 'MDS');
[St4, Mt4] = compute_mapping(XP_4', 'MDS');
[St5, Mt5] = compute_mapping(XP_5', 'MDS');
[St6, Mt6] = compute_mapping(XP_6', 'MDS');

Train=[S1';S2';S3';S4';S5';S6'];
Test=[St1';St2';St3';St4';St5';St6'];
end

% save Test Test;
% save Train Train;

% Llamar Data_Bases de otro script y aplicar la reduccin de dimensionalidad
```

A.7. PCA probabilístico

```
% clear all;
% close all;
% clc

% nn= nmero de iteraciones del prob PCA
function [Test Train]=Data_Bases(nn)

addpath('C:\LME');
addpath('C:\drtoolbox');
addpath('C:\drtoolbox\techniques');
addpath('C:\drtoolbox\gui');

%%%%%%%%%%%%%%%%%%%%%%%%%%BUILD
    DATABASES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ni=100;

%Read training data
load Xi1_E;
load Xi2_E;
load Xi3_E;
load Xi4_E;
load Xi5_E;
load Xi6_E;

load XP_1;
load XP_2;
load XP_3;
load XP_4;
load XP_5;
load XP_6;

% Data selection
Z1=Xi1_E(1:100,:);
Z2=Xi2_E(1:100,:);
Z3=Xi3_E(1:100,:);
Z4=Xi4_E(1:100,:);
Z5=Xi5_E(1:100,:);
Z6=Xi6_E(1:100,:);

% Data Normalization
desv(1)=std(reshape(Z1,100*1400,1));
desv(2)=std(reshape(Z2,100*1400,1));
desv(3)=std(reshape(Z3,100*1400,1));
desv(4)=std(reshape(Z4,100*1400,1));
desv(5)=std(reshape(Z5,100*1400,1));
desv(6)=std(reshape(Z6,100*1400,1));
save desv desv
```

```
% Normalization
load desv desv
Z1=Z1./desv(1);
Z2=Z2./desv(2);
Z3=Z3./desv(3);
Z4=Z4./desv(4);
Z5=Z5./desv(5);
Z6=Z6./desv(6);

XP_1=XP_1(1:100,:)./desv(1);
XP_2=XP_2(1:100,:)./desv(2);
XP_3=XP_3(1:100,:)./desv(3);
XP_4=XP_4(1:100,:)./desv(4);
XP_5=XP_5(1:100,:)./desv(5);
XP_6=XP_6(1:100,:)./desv(6);

[S1, M1] = compute_mapping(Z1', 'ProbPCA',49,nn);
[S2, M2] = compute_mapping(Z2', 'ProbPCA',49,nn);
[S3, M3] = compute_mapping(Z3', 'ProbPCA',49,nn);
[S4, M4] = compute_mapping(Z4', 'ProbPCA',49,nn);
[S5, M5] = compute_mapping(Z5', 'ProbPCA',49,nn);
[S6, M6] = compute_mapping(Z6', 'ProbPCA',49,nn);

[St1, Mt1] = compute_mapping(XP_1', 'ProbPCA',49,nn);
[St2, Mt2] = compute_mapping(XP_2', 'ProbPCA',49,nn);
[St3, Mt3] = compute_mapping(XP_3', 'ProbPCA',49,nn);
[St4, Mt4] = compute_mapping(XP_4', 'ProbPCA',49,nn);
[St5, Mt5] = compute_mapping(XP_5', 'ProbPCA',49,nn);
[St6, Mt6] = compute_mapping(XP_6', 'ProbPCA',49,nn);

Train=[S1';S2';S3';S4';S5';S6'];
Test=[St1';St2';St3';St4';St5';St6'];
end

% save Test Test;
% save Train Train;

% Llamar Data_Bases de otro script y aplicar la reduccin de dimensionalidad
```

A.8. CNN

A.8.1. Expansión del conjunto de datos

Usar solo esta expansión cuando fuese necesario, si no se puede trabajar con los ejemplos ya mostrados

```
clear all;
close all;
clc
%cargar datos de entrenamiento
load Xi1_E;
load Xi2_E;
load Xi3_E;
load Xi4_E;
load Xi5_E;
load Xi6_E;
load Yi;

Xi1=Xi1_E(:,1:100);
Xi2=Xi2_E(:,1:100);
Xi3=Xi3_E(:,1:100);
Xi4=Xi4_E(:,1:100);
Xi5=Xi5_E(:,1:100);
Xi6=Xi6_E(:,1:100);
Yi=Yi(:,1:100);

[nn,mm] = size(Xi1); %nn: respuesta, mm: numero del impacto

%expandir base para el siguiente vector de fuerzas
mf = [50:2:260];

Xi1_E = [];
Xi2_E = [];
Xi3_E = [];
Xi4_E = [];
Xi5_E = [];
Xi6_E = [];
Yi_E = [];

for k = 1:length(mf)
    for i = 1:mm
        af = Yi(3,i);
        Xa1(:,i) = Xi1(:,i)*mf(k)/af;
        Xa2(:,i) = Xi2(:,i)*mf(k)/af;
        Xa3(:,i) = Xi3(:,i)*mf(k)/af;
        Xa4(:,i) = Xi4(:,i)*mf(k)/af;
        Xa5(:,i) = Xi5(:,i)*mf(k)/af;
```

```
        Xa6(:,i) = Xi6(:,i)*mf(k)/af;
        Ya(1:2,i) = Yi(1:2,i);
        Ya(3,i) = mf(k);
    end

    Xi1_E = [Xi1_E Xa1];
    Xi2_E = [Xi2_E Xa2];
    Xi3_E = [Xi3_E Xa3];
    Xi4_E = [Xi4_E Xa4];
    Xi5_E = [Xi5_E Xa5];
    Xi6_E = [Xi6_E Xa6];
    Yi_E = [Yi_E Ya];
end

Xi1_cnn=Xi1_E;
Xi2_cnn=Xi2_E;
Xi3_cnn=Xi3_E;
Xi4_cnn=Xi4_E;
Xi5_cnn=Xi5_E;
Xi6_cnn=Xi6_E;
Yi_cnn=Yi_E;

save Xi1_cnn Xi1_cnn;
save Xi2_cnn Xi2_cnn;
save Xi3_cnn Xi3_cnn;
save Xi4_cnn Xi4_cnn;
save Xi5_cnn Xi5_cnn;
save Xi6_cnn Xi6_cnn;
save Yi_cnn Yi_cnn;
```

A.8.2. Pre-Normalización

```
close all
clear all
clc
load('Xi1_cnn.mat')
load('Xi2_cnn.mat')
load('Xi3_cnn.mat')
load('Xi4_cnn.mat')
load('Xi5_cnn.mat')
load('Xi6_cnn.mat')

load('XP_1.mat')
load('XP_2.mat')
load('XP_3.mat')
load('XP_4.mat')
load('XP_5.mat')
load('XP_6.mat')

%% Crear matrices de entrenamiento
Xi1_cnn=Xi1_cnn(1:100,:);
Xi2_cnn=Xi2_cnn(1:100,:);
Xi3_cnn=Xi3_cnn(1:100,:);
Xi4_cnn=Xi4_cnn(1:100,:);
Xi5_cnn=Xi5_cnn(1:100,:);
Xi6_cnn=Xi6_cnn(1:100,:);

XP_1=XP_1(1:100,:);
XP_2=XP_2(1:100,:);
XP_3=XP_3(1:100,:);
XP_4=XP_4(1:100,:);
XP_5=XP_5(1:100,:);
XP_6=XP_6(1:100,:);

Xtrain=[Xi1_cnn;Xi2_cnn;Xi3_cnn;Xi4_cnn;Xi5_cnn;Xi6_cnn];
Xtest=[XP_1;XP_2;XP_3;XP_4;XP_5;XP_6];

save Xtrain
save Xtest
```

A.8.3. Normalización

```
import scipy
import scipy.io as sio
import keras
import scipy.io
import scipy.io as sio
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy.matlib
from sklearn.preprocessing import MinMaxScaler
from keras.layers import Input, Dense
from keras.models import Model
import numpy as np
import pandas as pd
get_ipython().run_line_magic('matplotlib', 'inline')
from sklearn.externals import joblib

# Cargar datos
x_train = scipy.io.loadmat('Xtrain.mat')['Xtrain']
x_train = x_train.transpose()
t = scipy.io.loadmat('Xtest.mat')['Xtest']
x_test = t.transpose()

# Normalizar dataset entrenamiento
sc = MinMaxScaler(feature_range=(0,1))
x_train = sc.fit_transform(x_train)

# Normalizar dataset prueba c/r entrenamiento
x_test = sc.transform(x_test)

# Ajustar dimensiones
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))

# Guardar los datos.
numpy.savetxt('Xtrain_N.txt' , x_train.transpose() , delimiter=',')
numpy.savetxt('Xtest_N.txt' , x_test.transpose() , delimiter=',')
```

A.8.4. Ejemplo de una de las CNN

```
   # -*- coding: utf-8 -*-
import scipy
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import numpy.matlib
import pandas as pd

# Datos de entrenamiento
Datos = scipy.io.loadmat('X_train.mat')['X_train']
X_train= Datos
# devuelve las dimensiones de la matriz
a=X_train.shape[0:]
# Le suma 1
a = a + (1,)
X_train=np.reshape(X_train, a)

# Datos de testeo
Datos = scipy.io.loadmat('X_test.mat')['X_test']
X_test= Datos
# devuelve las dimensiones de la matriz
a=X_test.shape[0:]
# Le suma 1
a = a + (1,)
X_test=np.reshape(X_test, a)

# Targets de entrenamiento
Datos = scipy.io.loadmat('Y_train.mat')['Y_train']
Y_train= Datos

# devuelve las dimensiones de la matriz
a=Y_train.shape[0:]
# Le suma 1
a = a + (1,)
Y_train=np.squeeze(np.reshape(Y_train, a))

# Targets de prueba
Datos = scipy.io.loadmat('Y_test.mat')['Y_test']
Y_test= Datos
# devuelve las dimensiones de la matriz
a=Y_test.shape[0:]
# Le suma 1
a = a + (1,)
Y_test=np.squeeze(np.reshape(Y_test, a))

print('X_train:', X_train.shape)
print('Y_train:', Y_train.shape)
```

```
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

y_true_x=Y_train[:,0]
print(y_true_x)


#Red convolucional
from __future__ import print_function
import keras
import tensorflow as tf
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error,mean_absolute_error
import keras.backend as K
from sklearn.externals import joblib

# Parmetros del entranamiento CNN
batch_size = 100
epochs =1500
model_name = 'CNN_Impact_Identification'


# Modelo
model = Sequential()

# Primera capa de convolucion
model.add(Conv2D(filters = 8,
                 kernel_size=(3,1),
                 strides=(1,1),
                 padding='same',
                 activation='relu',
                 input_shape=(X_train.shape[1],X_train.shape[2],1)))
model.add(MaxPooling2D(pool_size=(2,1)))
model.add(Dropout(0.1))


# Segunda capa de convolucion
model.add(Conv2D(filters = 16,
                 kernel_size=(2,1),
                 strides=(1,1),
                 padding='same',
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2,1)))
model.add(Dropout(0.1))
```

```
# Tercera capa de convolucion
model.add(Conv2D(filters = 32,
                 kernel_size=(3,3),
                 strides=(1,1),
                 padding='same',
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2,1)))
model.add(Dropout(0.1))


# Cuarta capa de convolucion
model.add(Conv2D(filters = 64,
                 kernel_size=(5,1),
                 strides=(1,1),
                 padding='same',
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2,1)))
model.add(Dropout(0.1))

# Quinta capa de convolucion
model.add(Conv2D(filters = 128,
                 kernel_size=(3,3),
                 strides=(1,1),
                 padding='same',
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2,1)))
model.add(Dropout(0.1))


# Salida
model.add(Flatten())
model.add(Dense(1024))
keras.layers.BatchNormalization()
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(512))
keras.layers.BatchNormalization()
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(Y_train.shape[1]))
model.add(Activation('linear'))

#linear
model.summary()
```

```
def impact_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)
    y3= abs(y_true_F-y_pred_F)*100/y_true_F

    E_F=y3
    E_A= y1*y2*100/(135*90)
    I=E_F*E_A
    return I


model.compile(loss=impact_error,
              optimizer=keras.optimizers.Adam(lr=5E-4))

from keras.callbacks import ReduceLROnPlateau, EarlyStopping
keras.callbacks.Callback()

detencion=keras.callbacks.EarlyStopping(monitor='val_loss',mode='min',
                                        patience=400,
                                        verbose=1,
                                        min_delta=0.1,
                                        restore_best_weights=True)

tasa_aprendizaje=ReduceLROnPlateau(monitor = "val_loss", factor = 0.5,
                                   patience = 50,
                                   verbose = 1,
                                   mode = "auto",
                                   cooldown = 0,
                                   min_lr = 5E-5)

callbacks=[detencion, tasa_aprendizaje]

history=model.fit(X_train, Y_train,validation_split=0.1,
                  batch_size=batch_size,
                  epochs=epochs,
                  shuffle=False,
                  verbose=2,
                  callbacks=callbacks)


# Imprimir historial de entrenamiento
```

```
print(history.history.keys())

import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager


# Historial para la funcin de costos
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('Error de identificacin de impactos', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Score del modelo entrenado en el conjunto de prueba
scores = model.evaluate(X_test, Y_test, batch_size=200)
print('Error de identificacin de impactos, Test:', scores)
```

A.8.5. Abrir el modelo guardado

```
from keras.models import load_model
import keras.backend as K
import keras
from sklearn.externals import joblib
import scipy
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import numpy.matlib
import pandas as pd

# Datos de testeo
Datos = scipy.io.loadmat('X_test.mat')['X_test']
X_test= Datos
a=X_test.shape[0:]
a = a + (1,)
X_test=np.reshape(X_test, a)

Datos = scipy.io.loadmat('Y_test.mat')['Y_test']
Y_test= Datos
a=Y_test.shape[0:]
a = a + (1,)
```

```
Y_test=np.squeeze(np.reshape(Y_test, a))

# Redefinir la funcion de costos
def impact_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)
    y3= abs(y_true_F-y_pred_F)*100/y_true_F

    m1=K.mean(y1)
    m2=K.mean(y2)
    m3=K.mean(y3)

    E_F=y3
    E_A= y1*y2*100/(135*90)
    I=E_F*E_A
    return I

# Agregar la funcion de costos a las funciones de keras
keras.losses.impact_error = impact_error
filename='2capas_kernel_33_early'

# Cargar el modelo
loaded_model = joblib.load(filename)

# Reevaluarlo si se quiere
scores = loaded_model.evaluate(X_test, Y_test, batch_size=200)
print('Error de identificacin de impactos, Test:', scores)
```

A.9. Visualización de los resultados

A.9.1. Hacer plot tridimensional de los puntos (x,y,F)

A.9.2. Función colorear puntos (x,y,F)

```
function h = filledCircle(center,r,N,color)
THETA=linspace(0,2*pi,N);
RHO=ones(1,N)*r;
```

```
[X,Y] = pol2cart(THETA,RHO);
X=X+center(1);
Y=Y+center(2);

h=fill(X,Y,color);
axis square;

end
```

A.9.2.1. Hacer gráfico

```
clear all
close all
clc

Ye=load('2Y_pred_3_3_early.txt');
Ye=Ye';

load Yt
%% Estimaciones de los impactos LME
Ye_x=[];
Ye_y=[];
Ye_F=[];
Ye_x=Ye(1,:);
Ye_y=Ye(2,:);
Ye_F=Ye(3,:);

%% Impactos reales medidos por los sensores
Yi_x=[];
Yi_y=[];
Yi_F=[];
Yi_x=Yt(1,:);
Yi_y=Yt(2,:);
Yi_F=Yt(3,:);

max_F_e=max(Ye_F);
max_F_i=max(Yi_F);
max_f=[max_F_e, max_F_i];
Max_F=max(max_f);

for i=1:length(Yi_x)
    hold on
    filledCircle([Yi_y(i),Yi_x(i)],4,100,Yi_F(i));
    filledCircle([Ye_y(i),Ye_x(i)],4,100,Ye_F(i));

    filledCircle([Yi_y(i),Yi_x(i)],0.5,100,'k');
    filledCircle([Ye_y(i),Ye_x(i)],0.5,100,'k');
```

```
    X1=Yi_y(i);
    X2=Ye_y(i);
    Y1=Yi_x(i);
    Y2=Ye_x(i);

    plot([X1,X2],[Y1,Y2],'k')
    xlim([-5 135])
    ylim([-5 100])
end

set(gca,'FontSize',15)
xlabel('Fuselaje: Coordenada X [cm]')
ylabel('Fuselaje: Coordenada Y [cm]')
hcb=colorbar;
title('Identificacin de impactos CNN')
title(hcb,'Fuerza [N]')
set(gca,'FontSize',15)

%% Mxima distancia de separacin entre puntos
dif_x=abs(Ye_x-Yi_x);
dif_y=abs(Ye_y-Yi_y);
dif_F=abs(Ye_F-Yi_F);

dist= sqrt(dif_x.^2+dif_y.^2);

max_dif_dist=max(dist)
max_dif_F=max(dif_F)

e_x=mean(dif_x);
e_y=mean(dif_y);
dist_promedio= sqrt(e_x.^2+e_y.^2)
fuerza_promedio=mean(dif_F)
resumen= [dist_promedio max_dif_dist fuerza_promedio max_dif_F]
```

A.9.3. Visualización de las matrices de impacto de la CNN como imágenes

Primero se convierte la matriz de impacto inicialmente normalizada, al rango de colores entre 0 y 255. Esto se realiza con un código en matlab.

```
clear all
close all
clc
clear
%%
Xtest = load('Xtest_N.txt');
```

```
XP_1= Xtest(1:100,:);
XP_2= Xtest(101:200,:);
XP_3= Xtest(201:300,:);
XP_4= Xtest(301:400,:);
XP_5= Xtest(401:500,:);
XP_6= Xtest(501:600,:);

% Imagen para el impacto de prueba nmero 20
c_1=XP_1(:,20);
c_2=XP_2(:,20);
c_3=XP_3(:,20);
c_4=XP_4(:,20);
c_5=XP_5(:,20);
c_6=XP_6(:,20);

% Apilar los pixeles del impacto
A=[c_1 c_2 c_3 c_4 c_5 c_6];

A=A*255;
size(A)
Imagen=A;

save Imagen
```

Posterior a esta etapa se usan las librerias pillow y skimage de python, para convertir la matriz de impacto en una imagen en blanco y negro. Se pasa la imagen por un filtro de color, para notar mejor sus características.

```
# -*- coding: utf-8 -*-
import scipy
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import numpy.matlib
import pandas as pd
#  Crear una imagen del impacto
Impacto = scipy.io.loadmat('Imagen.mat')['Imagen']

print('Matriz de impacto:', Impacto.shape)
print(Impacto)

from PIL import Image
im = Image.fromarray(np.uint8(Impacto))
im.save('Impacto_original.jpg')
#im.show()

import skimage.io as io
from skimage.color import rgb2gray
```

```
img = io.imread('Impacto_original.jpg')
img_grayscale = rgb2gray(img)
io.imsave('Impacto_original-gs.jpg',img_grayscale)

from PIL import Image, ImageOps
img = Image.open("Impacto_original-gs.jpg") # Escala de grises

#dimg = ImageOps.colorize(img,black="Cyan",white="black")
#dimg = ImageOps.colorize(img,black="silver",white="black")
dimg = ImageOps.colorize(img,black="aliceblue",white="black")
dimg.save("Impacto_original_01.jpg")
```

A.10. Transfert Learning en la CNN

Se muestran los códigos de tres modelos en los cuales se probó transfert learning. Estos modelos son pre-entrenados en el data set *ImageNet*.

A.10.1. Modelo pre-entrenado Inception V3

```
import scipy
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import numpy.matlib
import pandas as pd

# Datos de entrenamiento ampliados a los tres canales
Datos = scipy.io.loadmat('X_train_Tr.mat')['X_train_Tr']
X_train= Datos
print('X_train:', X_train.shape)

# Datos de testeo ampliados a los tres canales
Datos = scipy.io.loadmat('X_test_Tr.mat')['X_test_Tr']
X_test= Datos
print('X_test:', X_test.shape)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# Targets de entrenamiento
Datos = scipy.io.loadmat('Y_train_Tr.mat')['Y_train_Tr']
Y_train= Datos
# devuelve las dimensiones de la matriz
a=Y_train.shape[0:]
# Le suma 1
a = a + (1,)
Y_train=np.squeeze(np.reshape(Y_train, a))
print('Y_train:', Y_train.shape)

# Targets de prueba
Datos = scipy.io.loadmat('Y_test_Tr.mat')['Y_test_Tr']
Y_test= Datos
# devuelve las dimensiones de la matriz
a=Y_test.shape[0:]
# Le suma 1
a = a + (1,)
Y_test=np.squeeze(np.reshape(Y_test, a))
print('Y_test:', Y_test.shape)
```

```
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

#Red convolucional
import keras
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Input, Conv2D,
    MaxPooling2D, multiply
from keras.layers import AveragePooling2D, GlobalAveragePooling2D, LocallyConnected2D,
    Lambda
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error,mean_absolute_error
import keras.backend as K

from sklearn.externals import joblib
#from keras.applications.vgg16 import VGG16 as PTModel
#from keras.applications.inception_resnet_v2 import InceptionResNetV2 as PTModel
from keras.applications.inception_v3 import InceptionV3 as PTModel
from keras.models import Model

#  Funcin  de  activacin
def impact_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)
    y3= abs(y_true_F-y_pred_F)*100/y_true_F

    E_F=y3
    E_A= y1*y2*100/(135*90)
    I=E_F*E_A
    return I

keras.losses.impact_error = impact_error

#  Parmetros  del entranamiento CNN
batch_size = 100
epochs =1400

# Agregar un modelo de base
input_tensor = Input(shape=(100, 120, 3))
```

```
base_model = PTModel(input_tensor=input_tensor, weights='imagenet',pooling='none',
    include_top=False)
base_model.trainable = False

# Construir el modelo
x = base_model.output
x= Flatten()(x)

# fully-connected layer
x = Dense(2048, activation='relu')(x)
x = Dense(1024, activation='relu')(x)

#  Regresin
predictions = Dense(Y_train.shape[1], activation='linear')(x)

# Modelo que se va a entrenar
model = Model(inputs=base_model.input, outputs=predictions)

model.summary()

model.compile(loss='impact_error',
              optimizer=keras.optimizers.Adam(lr=5E-4))
#model.compile(optimizer = 'Adam', loss = 'mse')

from keras.callbacks import ReduceLROnPlateau, EarlyStopping
keras.callbacks.Callback()

detencion=keras.callbacks.EarlyStopping(monitor='val_loss',mode='min',
                                        patience=200,
                                        verbose=1,
                                        min_delta=0.1,
                                        restore_best_weights=True)

tasa_aprendizaje=ReduceLROnPlateau(monitor = "val_loss", factor = 0.5,
                                   patience = 40,
                                   verbose = 1,
                                   mode = "auto",
                                   cooldown = 0,
                                   min_lr = 5E-7)
callbacks=[detencion, tasa_aprendizaje]

history=model.fit(X_train, Y_train,validation_split=0.1,
                  batch_size=batch_size,
                  epochs=epochs,
                  shuffle=False,
                  verbose=2,
                  callbacks=callbacks)

# list all data in history
print(history.history.keys())
```

```
import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager


# summarize history for loss
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('Error de identificacin de impactos', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Score trained model.
scores = model.evaluate(X_test, Y_test, batch_size=200)
print('Error de identificacin de impactos, Test:', scores)

model.evaluate(X_test, Y_test)
Y_pred=model.predict(X_test)

numpy.savetxt('inception_v3.txt' , Y_pred , delimiter=',')

error_I=impact_error(Y_test,Y_pred)
#print(error)
def promedio(datos):
    sumatoria = sum(datos)

    longitud = float(len(datos))

    resultado = sumatoria / longitud
    return resultado

def force_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y3= abs(y_true_F-y_pred_F)*100/y_true_F
    E_F=y3
    return E_F
```

```
error_F=force_error(Y_test,Y_pred)
prom_F=promedio(error_F)
print('Error porcentual de fuerza: ', prom_F)


def area_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)

    E_A= y1*y2*100/(135*90)

    return E_A
error_A=area_error(Y_test,Y_pred)
prom_A=promedio(error_A)
print('Error porcentual de area: ', prom_A)

prom_I=prom_A*prom_F
print('Error de identificacin de impactos: ', prom_I)

# Guardar el modelo
filename='inception_v3'
joblib.dump(model, filename)
```

A.10.2. Modelo pre-entrenado Inception Resnet v2

```
import scipy
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import numpy.matlib
import pandas as pd

# Datos de entrenamiento ampliados a los tres canales
Datos = scipy.io.loadmat('X_train_Tr.mat')['X_train_Tr']
X_train= Datos
print('X_train:', X_train.shape)

# Datos de testeo ampliados a los tres canales
```

```
Datos = scipy.io.loadmat('X_test_Tr.mat')['X_test_Tr']
X_test= Datos
print('X_test:', X_test.shape)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# Targets de entrenamiento
Datos = scipy.io.loadmat('Y_train_Tr.mat')['Y_train_Tr']
Y_train= Datos
# devuelve las dimensiones de la matriz
a=Y_train.shape[0:]
# Le suma 1
a = a + (1,)
Y_train=np.squeeze(np.reshape(Y_train, a))
print('Y_train:', Y_train.shape)

# Targets de prueba
Datos = scipy.io.loadmat('Y_test_Tr.mat')['Y_test_Tr']
Y_test= Datos
# devuelve las dimensiones de la matriz
a=Y_test.shape[0:]
# Le suma 1
a = a + (1,)
Y_test=np.squeeze(np.reshape(Y_test, a))
print('Y_test:', Y_test.shape)

print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

#Red convolucional
import keras
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Input, Conv2D,
    MaxPooling2D, multiply
from keras.layers import AveragePooling2D, GlobalAveragePooling2D, LocallyConnected2D,
    Lambda
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error,mean_absolute_error
import keras.backend as K

from sklearn.externals import joblib
#from keras.applications.vgg16 import VGG16 as PTModel
from keras.applications.inception_resnet_v2 import InceptionResNetV2 as PTModel
#from keras.applications.inception_v3 import InceptionV3 as PTModel
from keras.models import Model

#  Funcin  de activacin
```

```
def impact_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)
    y3= abs(y_true_F-y_pred_F)*100/y_true_F

    E_F=y3
    E_A= y1*y2*100/(135*90)
    I=E_F*E_A
    return I

keras.losses.impact_error = impact_error

# Parmetros del entranamiento CNN
batch_size = 100
epochs =1400

# Agregar un modelo de base
input_tensor = Input(shape=(100, 120, 3))
base_model = PTModel(input_tensor=input_tensor, weights='imagenet',pooling='none',
    include_top=False)
base_model.trainable = False

# Construir el modelo
x = base_model.output
x= Flatten()(x)

# fully-connected layer
x = Dense(2048, activation='relu')(x)
x = Dense(1024, activation='relu')(x)

#  Regresin
predictions = Dense(Y_train.shape[1], activation='linear')(x)

# Modelo que se va a entrenar
model = Model(inputs=base_model.input, outputs=predictions)

model.summary()

model.compile(loss='impact_error',
              optimizer=keras.optimizers.Adam(lr=5E-4))
#model.compile(optimizer = 'Adam', loss = 'mse')
```

```
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
keras.callbacks.Callback()

detencion=keras.callbacks.EarlyStopping(monitor='val_loss',mode='min',
                                        patience=200,
                                        verbose=1,
                                        min_delta=0.1,
                                        restore_best_weights=True)

tasa_aprendizaje=ReduceLROnPlateau(monitor = "val_loss", factor = 0.5,
                                   patience = 40,
                                   verbose = 1,
                                   mode = "auto",
                                   cooldown = 0,
                                   min_lr = 5E-7)
callbacks=[detencion, tasa_aprendizaje]


history=model.fit(X_train, Y_train,validation_split=0.1,
                  batch_size=batch_size,
                  epochs=epochs,
                  shuffle=False,
                  verbose=2,
                  callbacks=callbacks)

# list all data in history
print(history.history.keys())

import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager


# summarize history for loss
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('Error de identificacin de impactos', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Score trained model.
scores = model.evaluate(X_test, Y_test, batch_size=200)
print('Error de identificacin de impactos, Test:', scores)

model.evaluate(X_test, Y_test)
```

```
Y_pred=model.predict(X_test)

numpy.savetxt('inception_resnet_v2.txt' , Y_pred , delimiter=',')

error_I=impact_error(Y_test,Y_pred)
#print(error)
def promedio(datos):
    sumatoria = sum(datos)

    longitud = float(len(datos))

    resultado = sumatoria / longitud
    return resultado

def force_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y3= abs(y_true_F-y_pred_F)*100/y_true_F
    E_F=y3
    return E_F

error_F=force_error(Y_test,Y_pred)
prom_F=promedio(error_F)
print('Error porcentual de fuerza: ', prom_F)


def area_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)

    E_A= y1*y2*100/(135*90)

    return E_A
error_A=area_error(Y_test,Y_pred)
prom_A=promedio(error_A)
print('Error porcentual de area: ', prom_A)
```

```
prom_I=prom_A*prom_F
print('Error de identificacin de impactos: ', prom_I)

# Guardar el modelo
filename='inception_resnet_v2'
joblib.dump(model, filename)
```

A.10.3. Modelo pre-entrenado VGG16

```
import scipy
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import numpy.matlib
import pandas as pd

# Datos de entrenamiento ampliados a los tres canales
Datos = scipy.io.loadmat('X_train_Tr.mat')['X_train_Tr']
X_train= Datos
print('X_train:', X_train.shape)

# Datos de testeo ampliados a los tres canales
Datos = scipy.io.loadmat('X_test_Tr.mat')['X_test_Tr']
X_test= Datos
print('X_test:', X_test.shape)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# Targets de entrenamiento
Datos = scipy.io.loadmat('Y_train_Tr.mat')['Y_train_Tr']
Y_train= Datos
# devuelve las dimensiones de la matriz
a=Y_train.shape[0:]
# Le suma 1
a = a + (1,)
Y_train=np.squeeze(np.reshape(Y_train, a))
print('Y_train:', Y_train.shape)

# Targets de prueba
Datos = scipy.io.loadmat('Y_test_Tr.mat')['Y_test_Tr']
Y_test= Datos
# devuelve las dimensiones de la matriz
a=Y_test.shape[0:]
# Le suma 1
```

```
a = a + (1,)
Y_test=np.squeeze(np.reshape(Y_test, a))
print('Y_test:', Y_test.shape)

print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

#Red convolucional
import keras
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Input, Conv2D,
    MaxPooling2D, multiply
from keras.layers import AveragePooling2D, GlobalAveragePooling2D, LocallyConnected2D,
    Lambda
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error,mean_absolute_error
import keras.backend as K

from sklearn.externals import joblib
from keras.applications.vgg16 import VGG16 as PTModel
#from keras.applications.inception_resnet_v2 import InceptionResNetV2 as PTModel
#from keras.applications.inception_v3 import InceptionV3 as PTModel
from keras.models import Model

# Funcin de activacin
def impact_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)
    y3= abs(y_true_F-y_pred_F)*100/y_true_F

    E_F=y3
    E_A= y1*y2*100/(135*90)
    I=E_F*E_A
    return I

keras.losses.impact_error = impact_error

# Parmetros del entranamiento CNN
batch_size = 100
epochs =1400
```

```
# Agregar un modelo de base
input_tensor = Input(shape=(32, 36, 3))
base_model = PTModel(input_tensor=input_tensor, weights='imagenet',pooling='none',
    include_top=False)
#conv_base.trainable = True

set_trainable = False
for layer in base_model.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if layer.name == 'block4_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False


# Construir el modelo
x = base_model.output
x= Flatten()(x)

# fully-connected layer
x = Dense(512, activation='relu')(x)
x = Dense(256, activation='relu')(x)

#  Regresin
predictions = Dense(Y_train.shape[1], activation='linear')(x)

# Modelo que se va a entrenar
model = Model(inputs=base_model.input, outputs=predictions)

model.summary()

model.compile(loss='mse',
              optimizer=keras.optimizers.Adam(lr=5E-4))
#model.compile(optimizer = 'Adam', loss = 'mse')

from keras.callbacks import ReduceLROnPlateau, EarlyStopping
keras.callbacks.Callback()

detencion=keras.callbacks.EarlyStopping(monitor='val_loss',mode='min',
                                        patience=200,
                                        verbose=1,
                                        min_delta=0.1,
                                        restore_best_weights=True)

tasa_aprendizaje=ReduceLROnPlateau(monitor = "val_loss", factor = 0.5,
                                   patience = 40,
```

```
                                verbose = 1,
                                mode = "auto",
                                cooldown = 0,
                                min_lr = 5E-7)
callbacks=[detencion, tasa_aprendizaje]

history=model.fit(X_train, Y_train,validation_split=0.1,
                  batch_size=batch_size,
                  epochs=epochs,
                  shuffle=False,
                  verbose=2,
                  callbacks=callbacks)

# list all data in history
print(history.history.keys())

import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager


# summarize history for loss
fig,ax = plt.subplots(1,figsize=(16, 8))
ax.plot(history.history['loss'],'k', linewidth=2)
ax.plot(history.history['val_loss'],'r', linewidth=2)
ax.set_xlabel('Epocas', fontname="Arial", fontsize=14)
ax.set_ylabel('Error de identificacin de impactos', fontname="Arial", fontsize=14)
ax.legend(['Entrenamiento', 'Validacin'], loc='upper left',prop={'size': 14})
for tick in ax.get_xticklabels():
    tick.set_fontsize(14)
for tick in ax.get_yticklabels():
    tick.set_fontsize(14)
plt.show()

# Score trained model.
scores = model.evaluate(X_test, Y_test, batch_size=200)
print('Error de identificacin de impactos, Test:', scores)
model.evaluate(X_test, Y_test)
Y_pred=model.predict(X_test)
numpy.savetxt('VGG16_unfreeze_2.txt' , Y_pred , delimiter=',')
error_I=impact_error(Y_test,Y_pred)
#print(error)
def promedio(datos):
    sumatoria = sum(datos)

    longitud = float(len(datos))

    resultado = sumatoria / longitud
    return resultado

def force_error(y_true, y_pred):
```

```
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y3= abs(y_true_F-y_pred_F)*100/y_true_F
    E_F=y3
    return E_F

error_F=force_error(Y_test,Y_pred)
prom_F=promedio(error_F)
print('Error porcentual de fuerza: ', prom_F)


def area_error(y_true, y_pred):
    y_true_x=y_true[:,0]
    y_true_y=y_true[:,1]
    y_true_F=y_true[:,2]

    y_pred_x=y_pred[:,0]
    y_pred_y=y_pred[:,1]
    y_pred_F=y_pred[:,2]

    y1= abs(y_true_x-y_pred_x)
    y2= abs(y_true_y-y_pred_y)

    E_A= y1*y2*100/(135*90)

    return E_A
error_A=area_error(Y_test,Y_pred)
prom_A=promedio(error_A)
print('Error porcentual de area: ', prom_A)

prom_I=prom_A*prom_F
print('Error de identificacin de impactos: ', prom_I)

# Guardar el modelo
filename='VGG16_unfreeze_2'
joblib.dump(model, filename)
```