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

IDENTIFICACIÓN DE DAÑO EN PANELES COMPUESTOS UTILIZANDO REDES NEURONALES CONVOLUCIONALES Y LOS MODOS DE VIBRACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

DANIEL MATÍAS MARDINI GONZÁLEZ

PROFESORA GUÍA:
VIVIANA MERUANE NARANJO

MIEMBROS DE LA COMISIÓN:
ENRIQUE LÓPEZ DROGUETT
RAFAEL RUIZ GARCÍA

Este trabajo ha sido parcialmente financiado por proyecto Fondecyt 1170535

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: DANIEL MATÍAS MARDINI GONZÁLEZ
FECHA: 2020
PROF. GUÍA: VIVIANA MERUANE NARANJO

IDENTIFICACIÓN DE DAÑO EN PANELES COMPUESTOS UTILIZANDO REDES NEURONALES CONVOLUCIONALES Y LOS MODOS DE VIBRACIÓN

Los paneles de aluminio compuestos son materiales estructurales con una amplia gama de aplicaciones, desde la construcción de turbinas eólicas hasta equipos de ingeniería aeroespacial. Esto se debe a sus propiedades mecánicas pues permite tener altas resistencias con un muy bajo peso en comparación a otros materiales. Sin embargo, existe un inconveniente a la hora de utilizar estos paneles: Pueden sufrir silenciosamente de delaminación, la cual no se puede detectar a simple vista y afecta severamente en las propiedades mecánicas además de alterar los modos de vibración del panel. Es por esto que es necesario recurrir a ensayos no destructivos para diagnosticar y/o monitorear paneles y evitar que fallen de manera catastrófica.

Las redes neuronales arificiales fueron primeramente desarrolladas en los años 40' y sólo recientemente han podido ser utilizadas con facilidad debido al desarrollo de la aceleración por hardware. Estas redes están diseñadas para tratar con grandes cantidades de datos y aprender de los mismos. En este contexto se encuentran las redes neuronales convolucionales, las cuales son especiales para tratar datos que se encuentren ordenados en una configuración matricial.

El objetivo de este trabajo consiste en utilizar estas redes neuronales convolucionales para realizar una identificación de daño en paneles compuestos. Para ello se cuenta con un código en Matlab que permite simular la respuesta vibratoria de los paneles estando sanos o dañados, éste programa se utilizará para generar una gran cantidad de datos de paneles dañados y sanos que contengan tanto la ubicación del daño como los modos de vibración. Con ello se entrenarán redes neuronales que aprendan a identificar daño y, finalmente, se hará uso de datos experimentales obtenidos anteriormente en laboratorio para validar las redes entrenadas.

El proceso de entrenamiento se realiza mediante una metodología de generación aleatoria de redes neuronales convolucionales que permite una basta exploración de hiperparámetros con el fin de seleccionar la arquitectura con mejores resultados.

Finalmente se logra evaluar el desempeño de las redes neuronales en la tarea de identificación de daño en los paneles, obteniendo una mejoría en los resultados en relación a los trabajos anteriormente realizados en la misma linea de investigación. Los resultados finales indican un alto potencial en el método para aplicaciones más complejas.

A mi familia y amigos,
por siempre apoyarme en todos mis proyectos.

Agradecimientos

Me resulta tremendamente difícil escribir un discurso de agradecimiento sin tener el pensamiento constante de que me faltará por mencionar a muchas personas que han hecho posible que hoy esté finalizando este gran proyecto.

En primer lugar, quiero agradecer tremendamente a mis padres, por haberse dedicado incondicionalmente a cuidarme, criarme, ayudarme y enseñarme durante todos los años de mi vida. A mi madre, Verónica, por ser ese pilar en el que siempre me pude apoyar, quien más confió en mi y en mis decisiones y me ha ayudado a levantarme de cada caída que he tenido. A mi padre, Rafael, por sacrificarse día a día para que nunca falte nada y por entregarme una cantidad increíble de conocimientos que me han ayudado a convertirme en una persona mucho más integral. A mis hermanos, Rafael, Rodrigo y Ariel, por estar siempre presentes, por compartir tantos buenos momentos en familia y por crear ese lazo tan fuerte entre nosotros. A mi tata, Baltazar, por su tremenda e infinita voluntad para ayudar en lo que se necesite.

En segundo lugar, agradezco de corazón a todos mis amigos por todos los momentos que hemos pasado, por su apoyo y por su confianza a lo largo de todo este tiempo, gracias por hacer la mi paso por la universidad mucho más ameno. Quiero dedicar una mención especial a Sebastian Astudillo, por acompañarme cercanamente a lo largo de todo el desarrollo de este trabajo, por todas esas horas de debate, intercambio de ideas y distensión que tuvimos.

Por último, quiero agradecer a mis profesores, por confiar plenamente en mi en las etapas más tempranas de mi formación y luego, en la universidad, por entregarme herramientas para enfrentar el mundo profesional. Quiero agradecer profundamente a mi profesora guía, Viviana Meruane, por su tremendo apoyo y disposición para resolver dudas y orientarme durante este trabajo.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

1. Introducción

1.1. Antecedentes generales

Los paneles de aluminio compuestos son materiales estructurales fabricados adhiriendo dos planchas de aluminio a un núcleo intermedio mediante el uso de un pegamento epóxico. Entre los núcleos que se pueden utilizar está el tipo panal de abeja[1]. Este material es utilizado en una amplia gama de aplicaciones, desde la arquitectura hasta la ingeniería aeroespacial, debido a que su estructura permite tener una alta resistencia al esfuerzo y a la corrosión combinados con un bajo peso [1]. Sin embargo, estos paneles poseen un inconveniente inherente a su configuración y es el hecho de que los daños no superficiales, como la delaminación, son visualmente imperceptibles y pueden llegar a ser catastróficos ya que, como se ha demostrado en [2], la resistencia y rigidez de estos paneles se ven negativamente afectadas en caso de presentarse delaminación al interior del panel.

La forma de vibrar de una estructura entrega información valiosa acerca de la misma sin afectarla mayormente, es por esto que el uso de los modos normales de vibración permite conocer fenómenos internos, como distintos tipos de daño, sin la necesidad de realizar ensayos que puedan deteriorar los elementos. Prueba de ello son los resultados obtenidos por Burlayenko y Sadowski [2], los que exponen un cambio de comportamiento en la rigidez y, por tanto, en los modos normales de un panel compuesto de aluminio ante la presencia de un a zona delaminada. A partir de este enfoque, se han estudiado distintos métodos para clasificar e identificar el daño en estos paneles basándose en la información otorgada por los modos normales [3–5]. Sin embargo las técnicas utilizadas en esta labor se basan principalmente en la aplicación de modelos o bien suposiciones con respecto al comportamiento del material, por lo que existe una dependencia de los resultados con la técnica o los supuestos realizados.

Por otro lado, el uso de redes neuronales artificiales se ha vuelto una herramienta importante en la solución de problemas de ingeniería, puesto que éstas entregan resultados de calidad en diversas tareas sin requerir directamente de un modelo del fenómeno de estudio, sino que su funcionamiento se basa únicamente en los datos. Además, con el desarrollo de la aceleración por hardware, el uso de redes neuronales artificiales se ha vuelto viable para su uso en diversas áreas pues los tiempos de entrenamiento se han reducido al punto de que los ordenadores comunes son capaces de entrenarlas. En este contexto, se encuentra una categoría de redes cuyas características permiten el tratamiento de datos en configuración matricial como imágenes, las cuales son serían útiles para detectar daños en los paneles, estas son llamadas redes neuronales convolucionales. Adicional a esto, existe un trabajo previo

[1] Aluminium honeycomb sandwich panel

realizado por Meruane et al. [6] en el que se ha validado un modelo que permite simular la respuesta de paneles con delaminación, lo que posibilita la creación de la información necesaria para entrenar redes neuronales con el fin de detectar daño en estos paneles sin la necesidad de generar un modelo predictivo, sino con el uso de redes capaces de trabajar únicamente con las respuestas modales.

1.2. Motivación

El desarrollo de este trabajo está motivado por la exploración de las redes neuronales convolucionales como un medio efectivo y confiable para la identificación de daño en los paneles compuestos con núcleo tipo panal de abeja. Buscando una mejora en los resultados obtenidos en trabajos anteriores, además de desprenderse de la dependencia los pre-procesamientos de las imágenes. La elección de las redes convolucionales en particular, y no otras arquitecturas, radica en el hecho de que los modos normales se representan como imágenes (matrices), en las que cada pixel indica la amplitud del desplazamiento en un punto particular de la placa.

1.3. Objetivos

Objetivo general

- Evaluar el desempeño de las redes neuronales convolucionales (CNN) en la tarea de identificar el daño por delaminación en paneles compuestos de aluminio con núcleo tipo panal de abeja.

Objetivos específicos

- Diseñar, entrenar y probar un set de redes neuronales convolucionales para identificar daño en los paneles.
- Validar experimentalmente la predicción del modelo con datos obtenidos en laboratorio.
- Comparar los resultados experimentales con trabajos anteriores.

1.4. Alcances

Este trabajo busca evaluar el desempeño de algunas arquitecturas de redes neuronales convolucionales en la tarea de identificación de daño por delaminación en panales compuestos de aluminio con núcleo tipo panal de abeja, basándose en la respuesta oscilatoria de los mismos. Considerando esto, éste trabajo no contempla ser una implementación práctica del método, sino una evaluación del mismo con el fin de avanzar en la linea de investigación.

Todos los datos de entrenamiento de las redes son generados digitalmente mediante un modelo numérico y posteriormente validados con datos medidos experimentalmente. Los paneles, tanto en la simulación como en la validación, poseen dimensiones de 0.25×0.35 [m].

2. Antecedentes

2.1. Paneles compuestos

Los paneles compuestos, también conocidos como paneles sandwich, son elementos estructurales del tipo placa o tablero, consistentes en un núcleo de un material o geometría particular aislado mediante dos planchas exteriores. La finalidad de esta combinación es conseguir características específicas del conjunto.

En este trabajo se estudian los paneles compuestos de aluminio con núcleo tipo panal de abeja (en inglés, *Aluminium honeycomb panel*) los cuales, como se muestra en la Figura 2.1, constan de tres partes bien definidas: Un par de placas de aluminio rígidas, que corresponden a las caras exteriores de la estructura; un núcleo del mismo material dispuesto en forma de matriz hexagonal, similar a un panal de abejas; y por último, una capa adhesivo que mantiene la unión entre el núcleo y los paneles. Esto elementos estructurales son ampliamente utilizadas en distintos campos de la ingeniería como la aeronáutica, la construcción civil o la generación de energía, debido a sus distinguidas propiedades mecánicas y físicas, entre ellas destacan [1]:

- Bajo peso en comparación a otros materiales que soporten las mismas solicitaciones.
- Bajo nivel de corrosión si las caras son tratadas de forma adecuada.
- Bajo riesgo de incendio.
- Facilidad de doblado y corte, lo que reduce el tiempo de trabajo con las mismas.

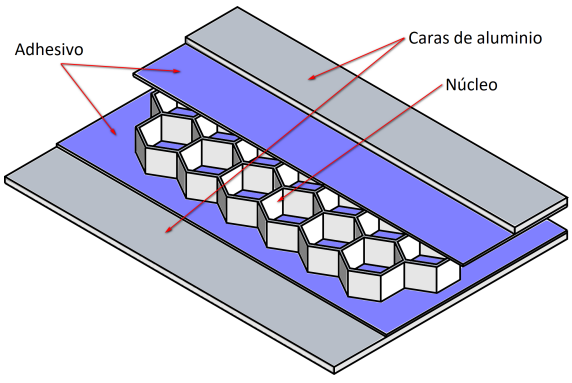


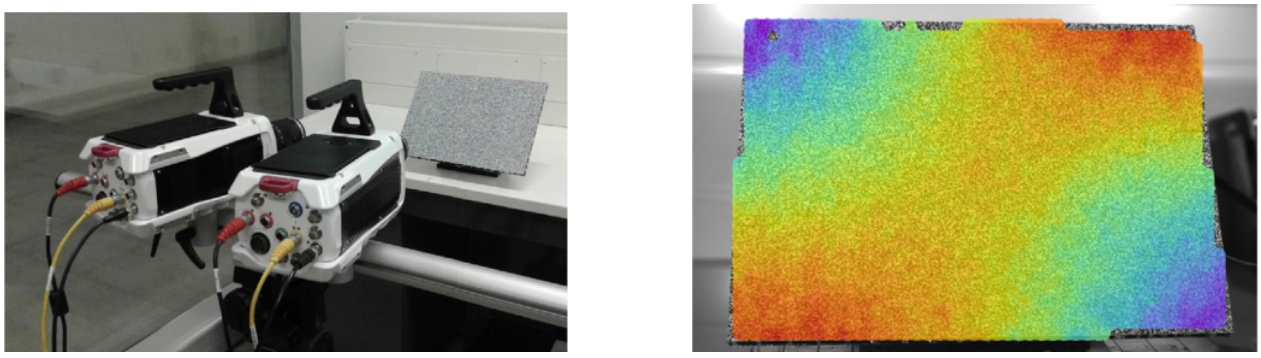
Figura 2.1: Esquema de panel compuesto

A pesar de sus características, éste material sigue siendo utilizado principalmente para la fabricación de componentes secundarios, pues es propenso a sufrir una amplia cantidad de

defectos y daños debido tanto a los procesos de manufactura como mantenimiento. Algunos defectos en la manufactura como la cobertura incompleta del adhesivo o las burbujas de aire encerradas en el mismo pueden provocar una unión no uniforme del núcleo con las placa [2].

La delaminación es uno de los defectos comunes en este tipo de paneles y consiste en la pérdida de adherencia entre el núcleo y las caras del panel. Esta situación provoca un deterioro severo en las propiedades mecánicas como lo son la resistencia y la rigidez del panel, además, trae consigo cambios en las propiedades vibracionales del mismo como el corrimiento de la frecuencia natural del panel y la modificación de los modos de vibración del mismo.

El hecho de que la delaminación provoque cambios en las propiedades modales, permite que se puedan realizar ensayos no destructivos al panel para conocer el estado de la estructura basándose en un análisis de vibraciones como se ha realizado en los trabajos *"Gapped Gaussian smoothing technique for debonding assessment with automatic thresholding"* [5] y *"Damage assessment in a sandwich panel based on full-field vibration measurements"* [4], en donde se han utilizados distintos para predecir el daño en paneles cuyos modos de vibración fueron medidos con un método llamado Image Correlation (DIC), el cual registra el desplazamiento del panel ante la aplicación de una vibración a una frecuencia específica. La Figura 2.2 muestra el montaje experimental utilizado en las investigaciones mencionadas para obtener los modos de vibración de los paneles estudiados.



(a) Cámaras de alta velocidad (b) Respuesta del panel

Figura 2.2: Montaje experimental para la medición de los modos de vibración [4].

2.2. Modos de vibración

Los modos de vibración o modos normales de un sistema oscilatorio se definen como un patrón de movimiento en el cual cada parte del sistema oscila a una misma frecuencia manteniendo una relación de fase. El movimiento descrito por cada modo normal es desarrollado a una frecuencia fija llamada frecuencia natural. Además, al oscilar, cada modo describe un lugar geométrico en el que su movimiento es nulo, este espacio es llamado nodo. Todo cuerpo o sistema posee un conjunto de modos normales que depende de la rigidez del mismo y, por lo tanto, del material, la configuración, las condiciones de borde, etc.

En general, la vibración de un sistema se puede describir como la superposición de sus modos de vibración. Se dice que los modos normales de un sistema son independientes pues la excitación de uno de ellos no causaría el movimiento de otro. Esto quiere decir, en términos matemáticos, que los modos de vibración son ortogonales entre sí.

2.2.1. Modos de una placa rectangular

Para tener noción de los modos normales en un medio continuo se presenta a continuación el ejemplo de los modos de vibración en una placa rectangular de lados $a \times b$ representada en la figura 2.3. En este caso se considera que sus extremos (aristas) se encuentran fijas y la placa está sometida a una tensión determinada en su contorno.

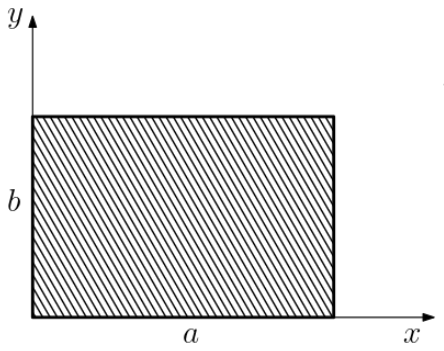


Figura 2.3: Esquema de la placa

La ecuación de movimiento que describe el desplazamiento (Ψ) de la placa cuando no existe término forzarte es:

$$\frac{\partial^2 \Psi}{\partial t^2} = c^2 \left(\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} \right) \tag{2.1}$$

Donde c es un parámetro que depende de la Tensión y la densidad.

Para resolver la ecuación 2.1 se utilizan el método de separación de variables, proponiendo como solución [7]:

$$\Psi = \Phi(x, y) \cdot T(t) \tag{2.2}$$

De lo que se obtiene:

$$\Phi(x, y) = \sin\left(\frac{m\pi}{a}x\right) \sin\left(\frac{n\pi}{b}y\right) \qquad m, n \in \mathbb{N} \tag{2.3}$$

La ecuación 2.3 indica que, en este caso, los primeros modos de la placa son los que se muestran en la figura 2.4, mientras que sus nodos son graficados en la figura 2.5.

Nótese de las figuras 2.4 y 2.5 que los modos normales se nombran por el par (m, n) según su cantidad de crestas/valles que posea el modo según la dirección x o y en la que se encuentre, respectivamente.

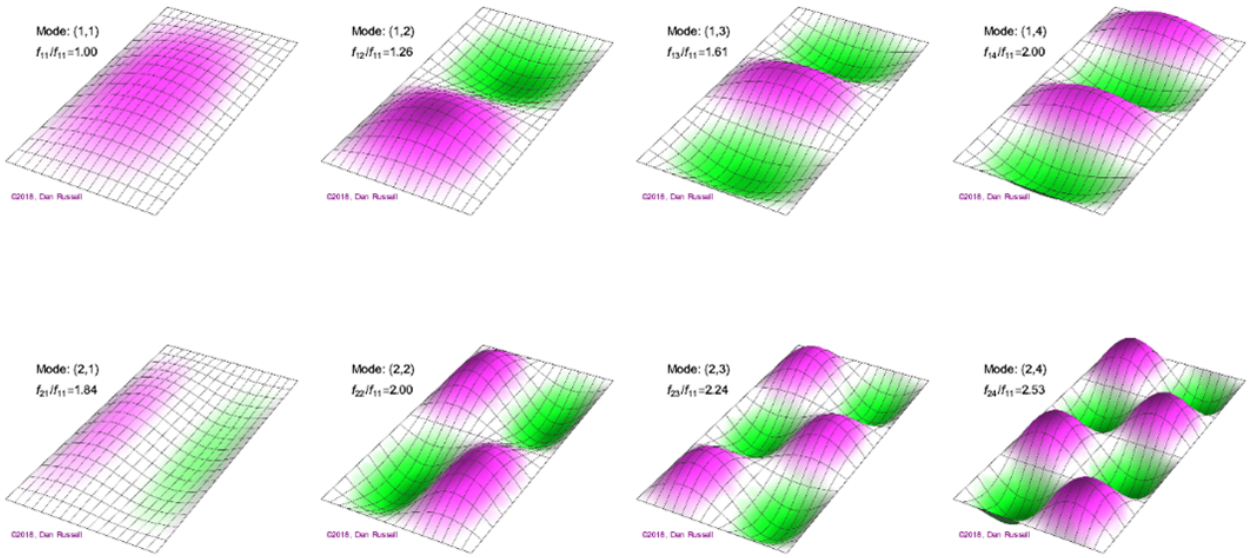


Figura 2.4: Primeros modos normales de una placa en tensión [8].

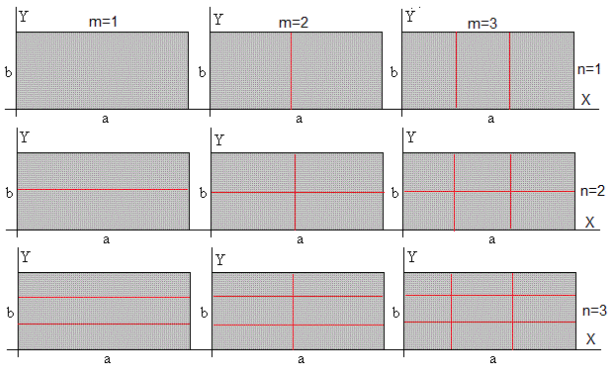


Figura 2.5: Primeros nodos de una placa en tensión [7].

2.3. Indicador MAC

El Modal Assurance Criterion (MAC) [18] es un indicador estadístico utilizado para cuantificar el grado de consistencia entre modos normales. Este indicador compara dos vectores $\vec{v_1}$ y $\vec{v_2}$ y retorna un valor entre 0 y 1 según su similitud, en este sentido, un valor del MAC igual a cero indica total discordancia mientras que un MAC igual a uno indicaría que los modos representados por los vectores son el mismo.

La fórmula para calcular matemáticamente el MAC se encuentra en la Ecuación 2.4. En caso de presentarse los modos en forma de matrices de $m \times n$ (imágenes) se utiliza la Ecuación 2.5. Ambas ecuaciones son equivalentes y, dada su formulación, el indicador presenta una alta sensibilidad ante grandes diferencias y baja sensibilidad ante las pequeñas.

$$MAC(\vec{v_1}, \vec{v_2}) = \frac{\left|\vec{v_1}^T \cdot \vec{v_2}\right|^2}{\left(\vec{v_1}^T \cdot \vec{v_1}\right)\left(\vec{v_2}^T \cdot \vec{v_2}\right)} \tag{2.4}$$

$$MAC(A,B) = \frac{\left|\sum_{i=0}^{n}\sum_{j=0}^{m} A_{i,j}\cdot B_{i,j}\right|^2}{\left(\sum_{i=0}^{n}\sum_{j=0}^{m} A_{i,j}^2\right)\cdot\left(\sum_{j=0}^{m}\sum_{i=0}^{n} B_{i,j}^2\right)} \tag{2.5}$$

En general se considera que un valor MAC superior a 0.9 indica una correspondencia consistente en los modos, mientras que valores inferiores indican una concordancia pobre.

El MAC puede tomar valores cercanos a cero por las siguientes razones:

- El sistema es no estacionario, resultando en un cambio de masa, rigidez o amortiguamiento durante el periodo de pruebas.
- El sistema es no lineal.
- Hay ruido en el modo de referencia.
- El método de extracción de parámetros es invalido para el set de datos medidos
- Los modos son linealmente independientes. Si bien el MAC no es una revisión de la ortogonalidad de los modos, este valor puede ser utilizado como una aproximación de la misma.

Las razones por las que el MAC puede tomar valores cercanos a uno son:

- El número de grados de libertad de la respuesta es insuficiente para distinguir entre modos de vibración independientes.
- Los modos de vibración son el resultados de una fuerza no medida del sistema.
- Los modos de vibración son principalmente ruido coherente.
- Los modos de vibración representan el mismo movimiento, difiriendo únicamente por su escala.

De este modo, si las primeras tres razones pueden ser descartadas, el MAC puede ser usado para indicar la consistencia entre los modos.

Las formas convencionales de uso de este valor son:

AutoMAC criterion

AutoMAC compara cuantitativamente todas las posibles combinaciones de los pares de modos de vibración de un mismo set de datos. Los resultados son presentados a través de una matriz cuadrada y simétrica en la que se indica el MAC para cada combinación.

CrossMAC criterion

CrossMAC es similar a AutoMAC, sin embargo, en éste se utilizan dos set de modos como filas y columnas de la matriz. De esta manera, cada elemento $C_{i,j}$ de la matriz de CrossMAC representa el MAC entre el modo i de un set de datos y el modo j de otro. Este método puede ser utilizado para:

- Calcular el MAC entre dos modelos de prueba distintos
- Calcular el MAC entre un modelo de prueba y un Modelo de Elemento Finito (FEM)
- Calcular el MAC entre dos modelos calculados mediante dos algoritmos distintos de análisis modal.

2.4. Redes Neuronales Artificiales

Las redes neuronales artificiales (en inglés, *Artificial Neural Networks* o ANN) son modelos computacionales bioinspirados destinados al procesamiento de datos mediante la acción conjunta de estructuras menores llamadas neuronas [9]. Estas redes son el núcleo del Deep Learning (DL) pues son unidades versátiles y escalables que permiten llevar a cabo grandes y complejas tareas en el área del Machine Learning. Estas redes fueron primeramente desarrolladas en el año 1943 por Warren McCulloch y Walter Pitts, en ese entonces no existía la potencia computacional para sacar provecho del potencial de su invento. Luego de diversos desarrollos, tanto en la teoría como en el hardware necesario para la implementación de estos algoritmos, se ha hecho posible el uso práctico de las ANN.

La unidad más básica de las redes es la neurona, que se considera una unidad de procesamiento no lineal de información. Ésta unidad se encuentra esquematizada en la figura 2.6, en ella se observa que posee valores de entrada (x_i), una serie de pesos ponderadores (w_i), un *bias* (b), una función llamada *función de activación* (f) y que retorna un valor de salida (y) de modo que la acción individual de una neurona se describe matemáticamente con la ecuación 2.6, pudiendo conectarse la salida a la entrada de una o más neuronas. Tanto los pesos como el *bias* son parámetros modificables, los cuales se actualizan iterativamente para ajustar la respuesta de la neurona o la red completa a un resultado en particular.

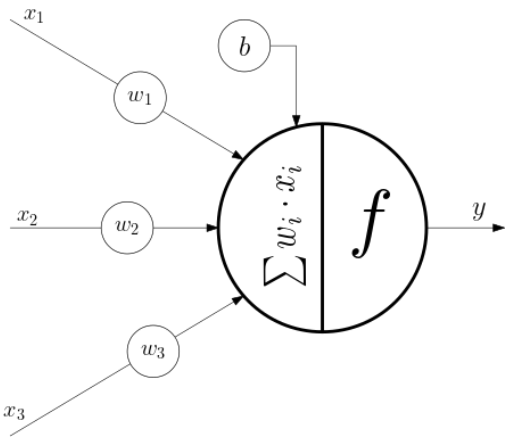


Figura 2.6: Esquema básico de una neurona

$$y = f\left(b + \sum_i w_i \cdot x_i\right) \tag{2.6}$$

Para formar una red neuronal, las neuronas se conectan formando capas como se representa en la figura 2.7. En donde las entradas de cada neurona son salidas de capas anteriores. Si todas las neuronas de una capa están completamente conectadas con las de la siguiente se dice que la capa es densa. Al combinarse todas las capas de la red se obtiene una función multivariable cuyos parámetros (w_i, b_i) pueden ajustarse con el fin de obtener un resultado deseado. Tanto al tipo de capa como sus carácteristicas (cantidad de parámetros, conexiones, funciones de activación, etc) se les llama hiperparámetros de la red.

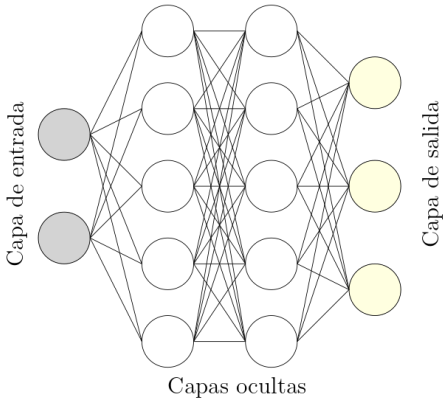


Figura 2.7: Esquema de una red neuronal densa.

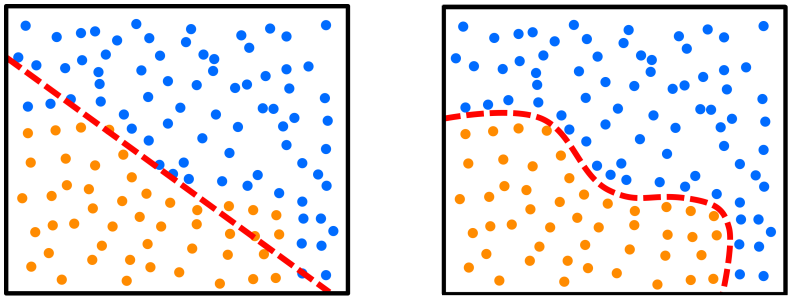
2.4.1. Funciones de activación

Una parte fundamental en las redes neuronales son las funciones de activación (presentadas como f en la ecuación 2.6), ya que éstas otorgan no linealidad a la red, incrementando su capacidad de adaptación y permitiendo, en algunos casos, controlar el rango de las salidas de la red. A modo de ejemplo se presenta la Figura 2.8, donde se esquematiza una situación en que se requiere realizar una clasificación binaria en un conjunto de datos, problema abordado por un clasificador lineal y uno no lineal. Se observa que la frontera definida por el segundo realiza una mejor clasificación al ajustar suavemente los datos, en contraposición con el primero, el cual realiza un ajuste más rígido y menos preciso.

Existe una amplia variedad de funciones de activación y la elección de cada una depende, entre otras cosas, del problema a solucionar, el rango en que se encuentran los datos y el diseño de la red. Algunas de las funciones de activación más utilizadas se encuentran programadas en el paquete TensorFlow [10] y sus gráficas se presentan en la Figura 2.9. En particular es deseable que la derivada de la función de costo sea conocida y fácil de calcular con el fin de que el proceso de actualización de pesos sea más eficiente.

2.4.2. Entrenamiento de la red

La actualización de los pesos o entrenamiento de la red consiste, como se mencionó anteriormente, en la modificación metódica de los pesos con el objetivo mejorar la respuesta del



(a) Lineal (b) No lineal

Figura 2.8: Comparativa entre clasificadores.

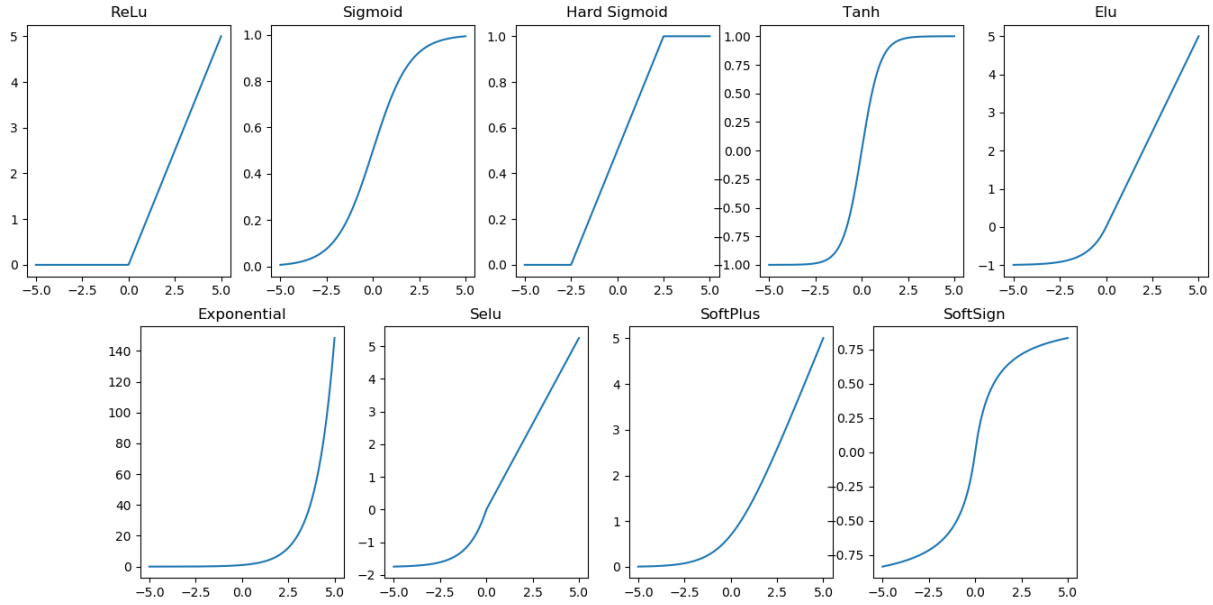


Figura 2.9: Funciones de activación

modelo. Para explicar el proceso de entrenamiento se puede considerar a la ANN como una función $y_i = Y(X, \theta_i)$, donde y_i corresponde a la respuesta de la red en la iteración i-esima, X a los datos de entrada y θ_i el estado de los parámetros en la iteración i-esima. Además, se puede considerar que se cuenta con un set de datos (llamados datos de entrenamiento) $\hat{x}$ cuya respuesta esperada $\hat{y}$ es conocida, con esto último se quiere la red "aprenda" el comportamiento de la respuesta ante entradas similares.

En términos generales, el entrenamiento es un proceso iterativo de minimización de una función f_c, conocida como función de costo (en inglés, *Loss Function*), que compara la respuesta de la red en un estado de parámetros en particular, $y_i = Y(\hat{x}, \theta_i)$, con el valor esperado $\hat{y}$. Para realizar la minimización se utiliza el método de *Backpropagation* junto con el concepto de *Gradient Descent* para actualizar los pesos.

Gradient Descent

El Descenso del Gradiente o *Gradient Descent* es un algoritmo genérico de optimización capaz de encontrar la solución optima a un amplio rango de problemas [9]. La idea general del *Gradient Descent* es modificar los parámetros θ_i iterativamente para minimizar la función de costo f_c. En términos prácticos, este método se basa en calcular el gradiente local (puntual) de la función de costo con respecto al vector de parámetros θ_i, para luego desplazarse (dar un *paso*) en la dirección del gradiente de manera proporcional a la magnitud del mismo. Una vez que el gradiente es cero, se ha encontrado un mínimo. Para iniciar el método se requiere que los parámetros sean inicializados de forma aleatoria y luego aplicar el algoritmo como se ilustra en la Figura 2.10, en ella se muestra la situación de una función con un solo parámetro y como el set de parámetros se mueve en dirección al mínimo.

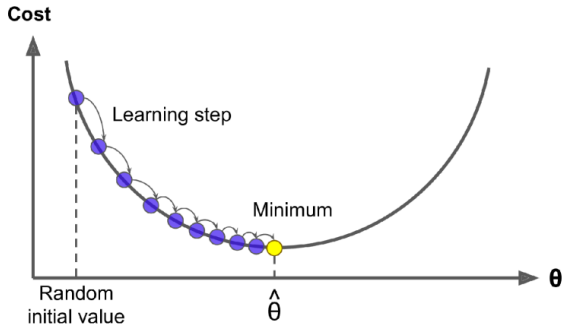
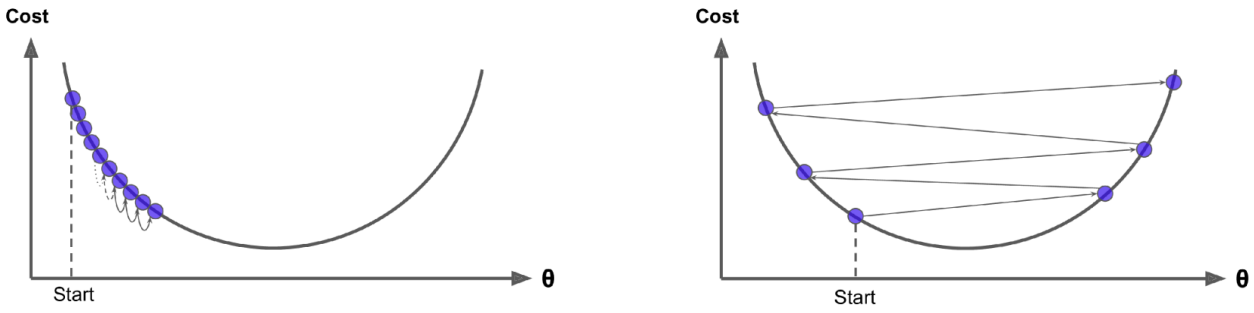


Figura 2.10: Método del Descenso del gradiente con un sólo parámetro [9].

Un parámetro importante en el método es el tamaño de los *pasos*, determinado por la tasa de aprendizaje (en inglés, *Learning Rate*). Si el *learning rate* es demasiado pequeño, al algoritmo tendrá que realizar muchas iteraciones para converger. Por otro lado, si el *learning rate* es demasiado grande, las respuestas oscilaran dentro del valle en la función de costo, posiblemente sin poder encontrar el mínimo e incluso tomando valores cada vez más grandes, provocando que el algoritmo diverja. Estas dos situaciones son representadas en la figura



(a) *Learning Rate* demasiado pequeño

(b) *Learning Rate* demasiado grande

Figura 2.11: Efecto del Learning Rate en la convergencia del Gradient Descent [9].

Backpropagation

Backpropagation es una técnica utilizada para entrenar redes neuronales artificiales introducido en el año 1986 por David Rumelhart, Geoffrey Hinton, y Ronald Williams. Este método es el *Gradient Descent* utilizando una metodología eficiente para calcular los gradientes automáticamente, necesitando recorrer la red únicamente dos veces, una de ida y vuelta. El algoritmo es capaz de calcular el gradiente de la función de costo con respecto a cada parámetro θ de la red, para luego realizar un *paso* utilizando el *Gradient Descent* y repetir el procedimiento hasta que la red converja a la solución [9].

Para realizar el *Backpropagation* el set de datos de entrenamiento se divide en conjuntos menores, conocidos como *mini-batch*, y cada uno de ellos es tratado independientemente hasta operar todo el set de datos. A cada iteración con el dataset completo se le llama época (en inglés, *epoch*). Cada *mini-batch* pasa por toda la red desde la entrada a la salida, exactamente igual que cuando se realiza una predicción pero, en este caso, los resultados de cada neurona en particular son preservados. Luego, el algoritmo calcula el error de salida evaluando la función de costo para comparar el valor esperado $\hat{y}$ con el resultado predicho y_i. Posteriormente se cuantifica la contribución de cada conexión de salida al error mediante el uso de la regla de la cadena, esto mismo se realiza con las capas anteriores hasta que se encuentra con la capa de entrada. Finalmente se aplica un paso del *Gradient Descent* para actualizar los pesos con los errores calculados.

2.5. Redes Neuronales Convolucionales

Existe un tipo de red especializada en el tratamiento de datos en formato matricial, como las imágenes. Éstas son conocidas como redes neuronales convolucionales (en inglés, *Convolutional Neural Networks* o CNN). Las arquitecturas basadas en CNN utilizan como entrada datos ordenados matricialmente con uno o más canales y utilizan capas diseñadas para tratar con este tipo de datos.

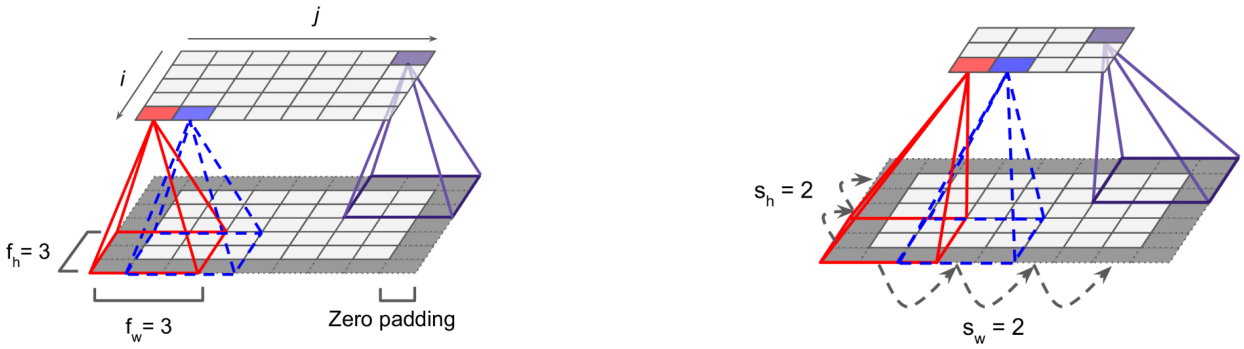
2.5.1. Capas convolucionales

La estructura más importante en las CNN son las capas convolucionales. En este caso, las neuronas no se encuentran conectadas a cada pixel (o dato en la matriz) como se haría en una ANN convencional, sino que se conecta a una serie de pixeles en una región de percepción [9].

Las neuronas de una Capa Convolucional se ordenan de manera matricial. De este modo, una neurona ubicada en la fija i, columna j de una capa en particular está conectada a las salidas de las neuronas de la capa anterior en las filas $[i, i + f_h - 1]$, columnas $[j, j + f_w - 1]$, donde f_h y f_w son la altura y el ancho del campo de percepción. Una práctica común es añadir ceros alrededor de la imagen analizada con el fin de mantener las dimensiones a través de las capas. Esta técnica es llamada *zero padding*.

Además, se puede reducir generar una reducción de la cantidad de neuronas en la capa espaciando los campos de percepción mediante un parámetro llamado *stride*. De este modo, una neurona ubicada en la fila i, columna j se encuentra conectada a las neuronas de la capa anterior ubicadas en la fila $[i \times s_h,\, i \times s_h + f_h - 1]$, columna $[j \times s_w,\, j \times s_w + f_w - 1]$, donde s_h y

s_w corresponden al *stride* horizontal y vertical, respectivamente. La Figura 2.12 muestra una esquemáticamente la implementación del *stride*. El uso este parámetro reduce drásticamente el costo computacional del uso de estas capas.



(a) Sin stride (b) Con stride

Figura 2.12: Esquema comparativo del uso de *stride* [9].

Los *pesos* de las neuronas en las capas convolucionales se encuentran contenidos en pequeña matriz con las dimensiones de la región de recepción (f_h, f_w), esta matriz es llamada filtro (o *kernel*). Los coeficientes del filtros se utilizan multiplicándose con los valores de la capa anterior dentro de la región de recepción y luego sumando sus resultados, asignando éste valor a la neurona de la capa. El resultado de aplicar esta operación a todas las neuronas de una capa es una matriz cuyo tamaño depende tanto de las dimensiones de la capa anterior como del parámetro del *stride* y es conocida como *mapa de carácteristicas* (en inglés, *feature map*), el cual es almacenado apilándolo con otros cuando se aplican diferentes filtros a una imagen, tal como se ilustra en la Figura 2.13.

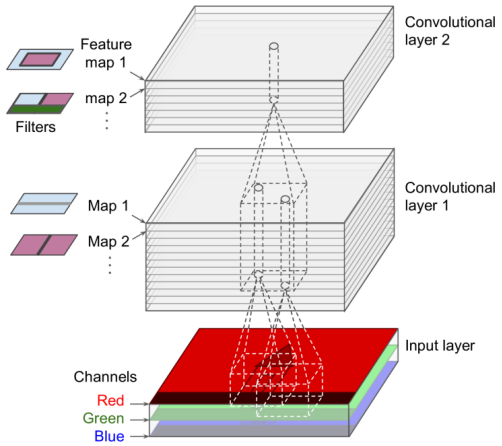


Figura 2.13: Capas convolucionales con *Feature Maps* concatenados [9].

La operación anteriormente descrita es llamada convolución y se expresa matemáticamente de la siguiente manera: Sea A un tensor de dimensiones $n \times m \times r$, el cual podría representar una imagen de tamaño $n \times m$ con r canales, W un tensor de dimensión $f_h \times f_w \times r$, que representa el filtro a utilizar y que contiene los pesos. Considere además, que se está utilizando

un *stride* (s_h, s_w). Con ello, convolución tiene como resultado una matriz Z, cuyos elementos son descritos mediante la Ecuación 2.7, en ella se agrega un termino b conocido como *bias* o sesgo, el cual también es un parámetro entrenable pero es opcional.

$$z_{i,j} = b + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k=0}^{r-1} w_{u,v,k} \cdot a_{i \cdot s_h+u,\, j \cdot s_w+v,\, k} \tag{2.7}$$

De esta manera, al apilar las matrices Z obtenida se obtiene el tensor llamado *feature map* que contiene la información procesada por los filtros (kernels). Adicional a este procesamiento se suele incluir una función de activación para agregar no-linealidad al resultado de la red, de la misma manera que se explicó en la sección 2.4.1.

2.5.2. Capas de Pooling

Las capas de Pooling o *Pooling Layers* son capas que se utilizan conjuntamente con las capas convolucionales y su principal objetivo es generar una compresión de la imagen. Su funcionamiento es similar al de las capas convolucionales pero en este caso no se aplica un filtro en la región de percepción, sino una función a todos los datos de la imagen. Al igual que en el caso anterior, puede ser utilizada con un parámetro de *stride* para reducir el tamaño de la matriz de salida. En general, estas capas no poseen parámetros entrenables, algunos ejemplos de éstas son:

MaxPooling

El MaxPooling consiste en guardar en una neurona el valor máximo de la región de percepción de la capa anterior, esto es:

$$z_{i,j,k} = \max_{\substack{u \in [0,\, f_h-1] \\ v \in [0,\, f_w-1]}} a_{i \cdot s_h+u,\, j \cdot s_w+v,\, k} \tag{2.8}$$

AveragePooling

El AveragePooling consiste en guardar en una neurona el valor promedio de la región de percepción de la capa anterior, esto es:

$$z_{i,j,k} = \frac{1}{f_h + f_w} \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} a_{i \cdot s_h+u,\, j \cdot s_w+v,\, k} \tag{2.9}$$

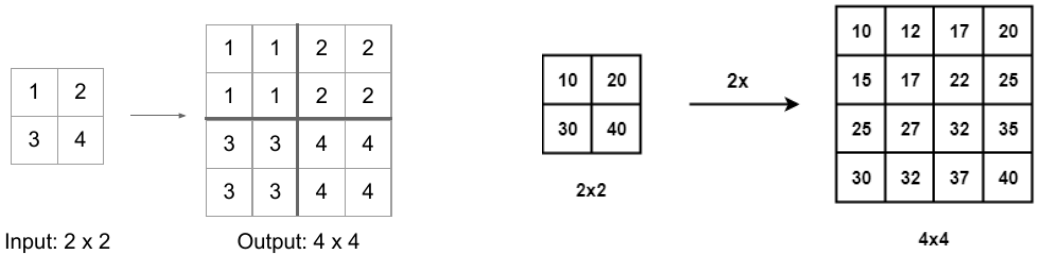
2.5.3. Capas de UpSampling

Al contrario de las capas de Pooling, las capas de UpSampling buscan aumentar la dimensión de la matriz de entrada, para ello existen diversas estrategias según la aplicación que éstas tengan. Algunos ejemplos de estas capas son:

UpSampling2D

Esta capa genera una matriz con el doble del tamaño de la matriz de entrada copiando los valores originales y rellenando los intermedios según algún algoritmo designado, por ejemplo:

- **Nearest Neighbors**: Con este método los pixeles añadidos son rellenados copiando el valor original más cercano, tal como se muestra en la Figura 2.14a.
- **Bi-Linear Interpolation**: Con este método los pixeles añadidos son rellenados tomando un promedio ponderado en función de la distancia, tal como se muestra en la Figura 2.14b.



(a) Nearest Neighbors (b) Bi-Linear Interpolation

Figura 2.14: Métodos de Upsampling2D [11].

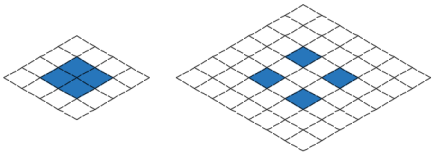
Transposed Convolutions

La capa Convolucinal Transpuesta, conocida como capa Deconvolucional, también se utiliza para aumentar el tamaño de la capa anterior pero, a diferencia de las capas de UpSampling 2D, ésta posee parámetros entrenables pues, al igual que las capas convolucionales, posee un kernel con pesos.

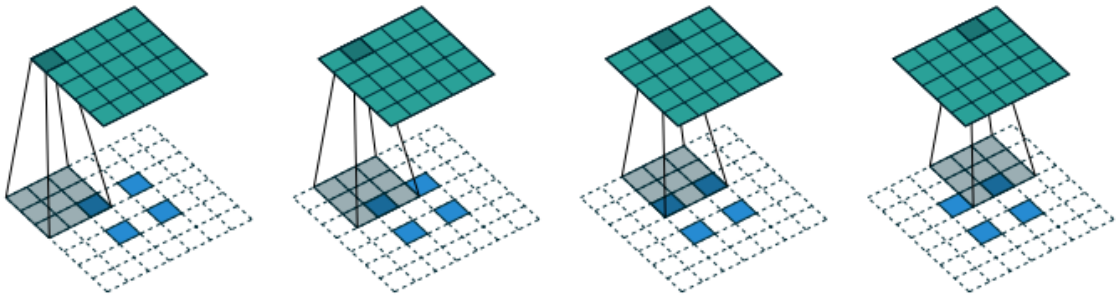
El mecanismo de funcionamiento es el siguiente: A cada pixel de la imagen de entrada se le intercala una fila y columna de ceros como se encuentra ilustrado en la Figura 2.15a. Posterior a esto se le aplica un filtro como si fuera una capa convolucional, tal como se ilustra en la Figura 2.15b. De este modo se cuenta con los pesos del filtro para mejorar la respuesta de la capa.

Unpooling

Esta capa requiere de una matriz y un vector de posiciones para crear una matriz con el doble del tamaño que la de entrada. Esta matriz es generada copiando los valores originales en las posiciones que el vector le indica, sin rellenar los pixeles intermedios. Esta metodología se explica gráficamente en la Figura 2.16.



(a) Preparación de la imagen



(b) Filtro convolucional

Figura 2.15: Funcionamiento de la capa deconvolucional [12].

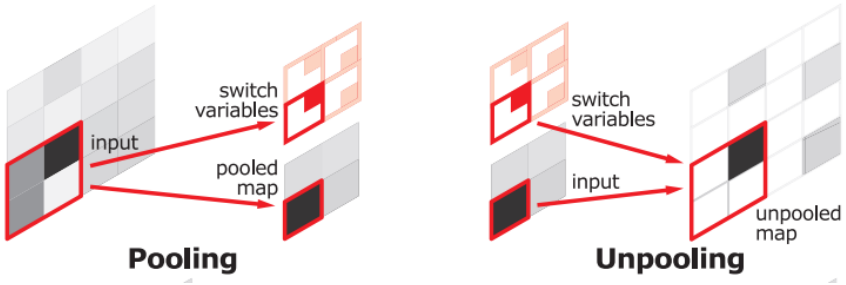


Figura 2.16: Funcionamiento de la capa Unpooling [13].

2.6. Arquitecturas para segmentación de imágenes

En visión computacional, la *segmentación de imágenes* es una forma de segregar una imagen digital en múltiples regiones según las diferentes propiedades de cada pixel. A diferencia de la clasificación y detección de objetos, ésta se define como una tarea a nivel de pixel pues la información espacial de una imagen se torna relevante al momento de segmentar diferentes regiones semánticamente. El objetivo principal de la segmentación es extraer información relevante para facilitar el análisis [14].

El mecanismo de la segmentación consiste en etiquetar cada pixel de una imagen según comparta carácteristicas con otros, por ejemplo: color, intensidad, textura, etc. Principalmente existen dos tipos de segmentación: Segmentación Semántica (en inglés, *Semantic Segmentation*) y Segmentación de Instancias (en inglés, *Instance Segmentation*). También, existe un tercer tipo que unifica las dos anteriores, llamado Segmentación Panóptica (en inglés, *Panoptic Segmentation*). La Figura 2.17 muestra las principales diferencias entre los resultados que se obtiene con cada uno de los segmentadores mencionados.

Para resolver el problema de segmentación existen diversas técnicas en el área de visión computacional, entre ellas se encuentra el uso de redes neuronales convolucionales para el

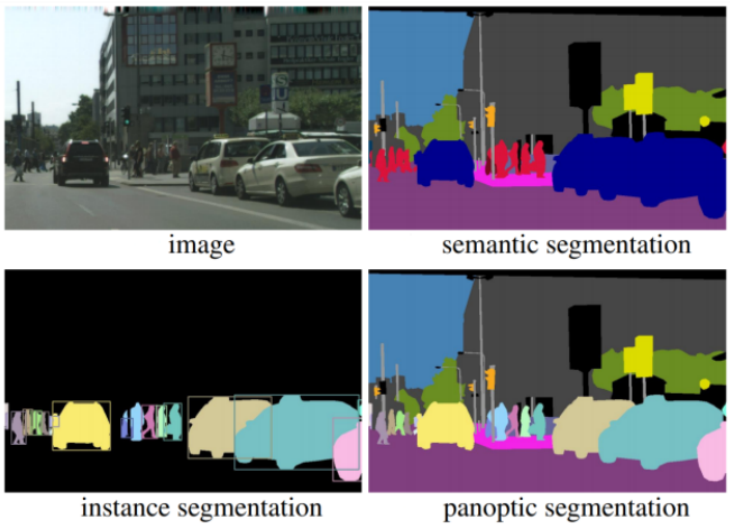


Figura 2.17: Diferentes tipos de segmentación [14].

tratamiento de imágenes. A continuación se presentan algunas de las arquitecturas utilizadas en la resolución de este problema.

2.6.1. Deconvnet

Propuesta por Noh *et al.* en el año 2015 [13], la red Deconvnet es una red de dos etapas: Una convolucional (compresión) y otra deconvolucional (expansión). La primera parte es idéntica a las primeras 13 capas convolucionales de la red VGG16 [15]. Sin embargo, la segunda parte posee una red deconvolucional simétrica con su primera parte, esto es, manteniendo las dimensiones de las capas de salida pero en dirección opuesta como se muestra en la Figura 2.18. Tal como se indica en la figura, en la zona de compresión se utilizan capas MaxPooling para reducir el tamaño de la imagen seguidas de varias capas convolucionales y capas de regularización. Por otro lado, en la zona de descompresión, se utilizan capas Unpooling seguidas de capas convolucionales hasta retornar una imágen del tamaño de la imagen de entrada. La salida de la red tiene tantos canales como clases se requiera segmentar y la función de activación de esta capa es del tipo *Softmax*.

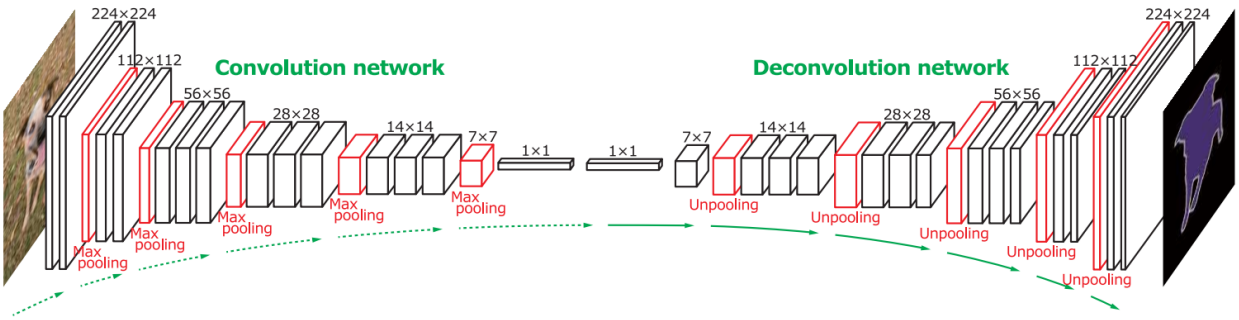


Figura 2.18: Esquema de la Deconvnet [13].

2.6.2. U-Net

Propuesta por Ronneberger *et al.* en el año 2015 [16] y pensada originalmente para segmentación de imágenes en el área de biomedicina, la red U-Net es una red similar a la Deconvnet, pues también consta de una etapa de compresión y una de expansión. Sin embargo, ésta red incluye conexiones adicionales, llamadas *skip-connections*, que comunican las salidas de las capas de compresión con las de expansión.

El funcionamiento de la U-Net es esquematizado en la Figura 2.19, en ella se observa, a la izquierda, la ruta de compresión y, a la derecha, la de expansión. Cada paso de en la etapa de compresión consiste en la aplicación de dos capas convolucionales sucesivas con una cantidad específica de filtros con una función de activación *ReLu* seguida de una capa tipo MaxPooling que reduce el tamaño de la imagen para pasar a un siguiente paso en donde la cantidad de filtros se duplica. La ruta de expansión consiste en la aplicación de un método de UpSampling (UpSumpling2D o Deconvolución) que duplica el tamaño de la imagen, seguido de dos capas convolucionales cuya cantidad de filtros va reduciéndose a la mitad hasta llegar a la última capa en donde se tiene que la salida posee una cantidad de canales igual a la cantidad de clases que se quiera segmentar. Las *skip-connections* que comunican la compresión con la expansión se implementan mediante la concatenación de la salida inmediatamente anterior al Maxpooling en la zona de compresión con la salida de UpSampling en la zona de expansión correspondiente a un mismo tamaño (mismo paso).

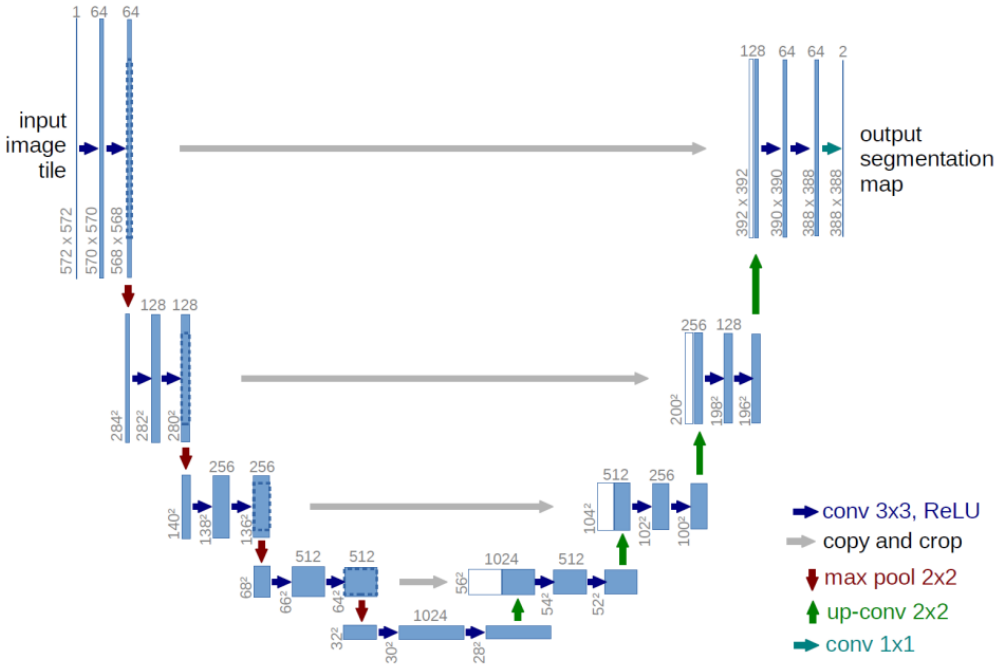


Figura 2.19: Esquema de la U-Net [16].

2.6.3. MultiResUNet

La MultiResUNet corresponde a una modificación de la U-Net propuesta por Ibtehaz y Rahman [17] el año 2020, en donde se pretende mejorar la respuesta de la U-Net ante situa-

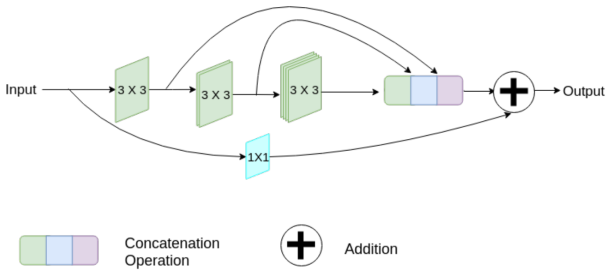
ciones en las que un fenómeno se presenta con múltiples escalas en las imágenes. Para ello, se utilizó como base la arquitectura de la U-Net y se modificaron tanto sus componentes de procesamiento (bloques convolucionales) como el proceso de comunicación entre los procesos de compresión y expansión. Para ello, se definen los conceptos de *MultiRes Block* y *Res Path*.

MultiRes Block

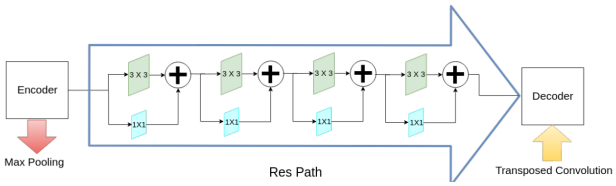
El MultiRes Block consiste en una serie de tres capas convolucionales con *kernels* de tamaño 3×3 aplicadas sucesivamente y aumentando gradualmente el número de filtros, además sus salidas independientes son concatenadas y posteriormente sumadas con la salida de una capa convolucional con un único filtro de tamaño 1×1, tal como se ilustra en la Figura 2.20a. Este bloque reemplaza a las dos capas convolucionales que se utilizan en la U-Net, como se muestra en la Figura 2.20c.

Res Path

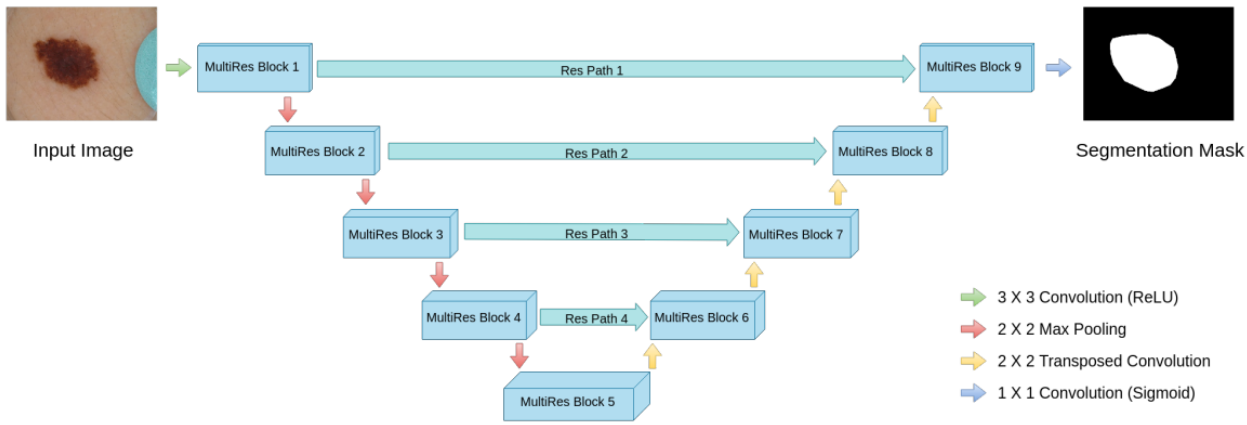
El Res Path consiste en una adición sucesiva de pequeños bloques de procesamiento compuestos por dos capas convolucionales en paralelo, una de 3×3 y otra 1×1, tal como se esquematiza en la Figura 2.20b. Como entrada a esta unidad se utiliza la salida de un MultiRes Block en un nivel de la zona de compresión y la salida se concatena con la entrada del MultiRes Block de la zona de expansión en el mismo nivel, como se ilustra en la figura 2.20. Esta unidad reemplaza a las *skip-conections* de la U-Net, añadiendo un procesamiento adicional antes de la comunicación entre el Encoder y el Decoder.



(a) MultiRes Block



(b) Res Path



(c) MultiResUNet

Figura 2.20: Partes de la MultiResUNet [17].

2.7. Intersection Over Union

El intersection Over Union, en adelante IoU, es una métrica ampliamente utilizada en el área de visión computacional y en las tareas de segmentación de imagen a través de Deep Learning [19]. Esta métrica retorna un valor entre 0 y 1 según cuan cercana sea la predicción de una clase con su valor de referencia. La figura 2.21 muestra un ejemplo de predicción junto a la clase que se desea predecir (*Ground truth*), en ella se identifican las zonas según la coincidencia entre estas dos muestras:

- Verdaderos Positivos (TP): Zonas en las que la identificación de la clase fue acertada.
- Verdaderos Negativos (TN): Zonas en las que se identifica correctamente que los puntos no corresponden a la clase.
- Falsos Positivos (FP): Zonas en las que se predice la presencia de la clase de manera errada.
- Falsos Negativos (FN): Zonas en las que la clase está presente pero se predice su ausencia.

Con ello, se define matemáticamente el IoU con la fórmula de la ecuación 2.10. Sin embargo, en la práctica se le sumar un valor ϵ pequeño tanto al numerador como al denominador, como se expresa en la ecuación 2.11, con el fin de evitar las divisiones entre cero. De este modo, en caso de no existir traslape entre las zonas (TP=0), el IoU se mantiene en un valor pequeño, cercano a cero. Esta situación también puede darse en el caso de que se acierte correctamente la ausencia de la clase, en tal caso, el IoU toma el valor 1.

$$\frac{TP}{TP + FP + FN} \tag{2.10}$$

$$\frac{TP + \epsilon}{TP + FP + FN + \epsilon} \tag{2.11}$$

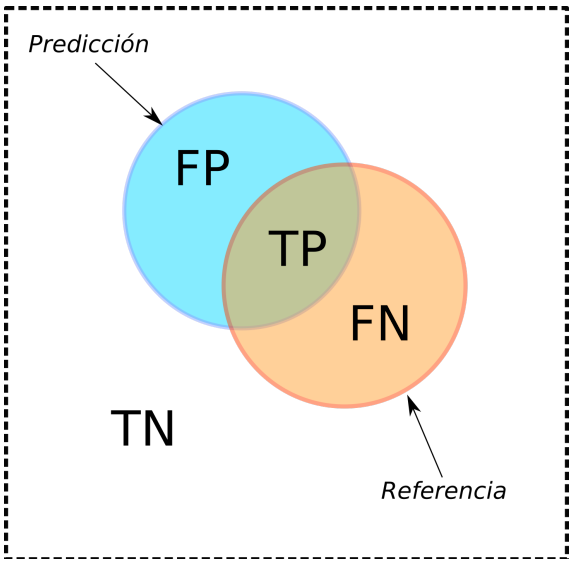


Figura 2.21: Esquema del IoU

3. Metodología

La metodología general de trabajo está orientada a crear y entrenar modelos de redes neuronales convolucionales que permitan identificar daño por delaminación en paneles compuestos de aluminio mediante el uso de su respuesta vibratoria. Para cada panel, esta respuesta se representa como un conjunto de matrices, cada una de las cuales se corresponde con un modo de vibración del panel, y cada elemento de la matriz corresponde a la amplitud del desplazamiento puntual de la placa en ese modo. La idea es, entonces, simular la respuesta vibratoria de paneles tanto sanos como con delaminación con el fin de generar un set de datos que permitan entrenar un conjunto de redes convolucionales, cada una dedicada a predecir la respuesta de un modo en particular. Luego de ello, unificar la predicción del conjunto de redes, con el fin de poder ingresar una cantidad variable de modos normales y se tenga una única respuesta. Por último se realiza una validación experimental utilizando la información obtenida medida en laboratorios realizados anteriormente.

3.1. Generación y exploración de datos

El primer paso para entrenar las redes es generar un set de datos de paneles dañados que represente fielmente el fenómeno de delaminación en ellos. Para esto se ha utilizado un programa de Matlab basado en Elementos Finitos, creado en [6], con el que se han generado un total de 5000 simulaciones de paneles con daños de tamaño circular cuya localización está uniformemente distribuida a lo largo y ancho del panel y su diámetro está distribuido uniformemente entre 0 y 0.25 en términos del diámetro normalizado[2]. Como resultado se obtiene, por cada simulación, siete matrices de 51×71 correspondiendo cada una a un modo de vibración en particular el panel, además de conocer la zona delaminada que genera ésta respuesta, pues es el dato de entrada para generar la simulación. Adicionalmente, se le ha agregado un nivel de ruido uniforme de un 10 % en la respuesta de los modos.

Posterior a la simulación se toman muestras aleatorias con el fin de observar el comportamiento de la simulación, así como la distribución de los tamaños de diámetros de delaminación. Luego se preparan los dataset para el entrenamiento de las redes, para ello se divide el total de datos de manera aleatoria en conjuntos, utilizando un 70 % (3 500) para entrenamiento, un 15 % (750) para validación y un 15 % (750) para testeo.

Por último, se procede a normalizar los datos utilizando la siguiente ecuación:

[2]El *diámetro normalizado* corresponde al diámetro de la zona delaminada dividido entre la distancia diagonal del panel.

$$\overline{x_{i,j,k,m}} = \frac{\max\limits_{\forall i,\forall j} x_{i,j,k,m} - x_{i,j,k,m}}{\max\limits_{\forall i,\forall j} x_{i,j,k,m} - \min\limits_{\forall i,\forall j} x_{i,j,k,m}} \tag{3.1}$$

Donde:

- $x_{i,j,k,m}$ es el elemento k de la respuesta de la simulación en la posición i, j de la matriz en el modo m.
- $\overline{x_{i,j,k,m}}$ es el elemento k de la respuesta normalizada en la posición i, j de la matriz en el modo m.

3.2. Creación de redes

La generación de redes neuronales se realizó utilizando la API de TensorFlow-Keras [10] implementada en el lenguaje *Python*, utilizando el paradigma funcional de estructuración de las capas. Para realizar una exploración basta de redes se ha decidido utilizar una implementación del método *Random Search* en la elección de los hiperparámetros con los que se estructuran las redes. Esta implementación está inspirada en la metodología utilizada en [20], la cual consiste en un *torneo de redes*. El procedimiento se resume en la figura 3.1, el cual consta de las etapas descritas a continuación. La idea principal de este método es generar un set de redes convolucionales con hiperparámetros elegidos aleatoriamente, para luego entrenarlos en *tandas de entrenamientos* consistentes en entrenar cada una cantidad de épocas. Con ello, se selecciona un porcentaje de las mejores redes y se elimina el resto para volver a realizar otra tanda de entrenamientos hasta terminar con una única red.

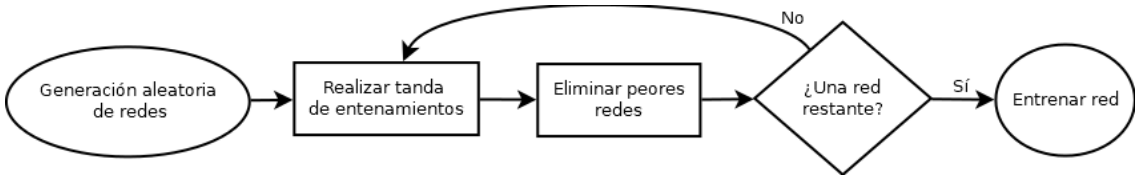


Figura 3.1: Diagrama de flujo del torneo de redes.

3.2.1. Generación aleatoria de redes

Para la creación de redes se ha tomado como referencia la publicación *Evolution of image segmentation using deep convolutional neural network: A survey* [14], en donde se hace una reseña de las arquitecturas basadas en redes neuronales convolucionales más populares utilizadas en la tarea de segmentación. De las mencionadas se han seleccionado tres debido, principalmente, a la simplicidad en su implementación, estas son: DeconvNet, UNet y MultiResUNet.

DecovNet y U-Net

Estas redes se describen en la sección 2.6.1 y 2.6.2, respectivamente. Para la construcción de estas arquitecturas se utilizan los parámetros descritos a continuación, la única diferencia entre la construcción los modelos inspirados en estas dos redes radica en las *skip-conections*

presentes en la U-Net. Un aspecto importante es que en el Decoder se utiliza una capa de Cropping para hacer coincidir el tamaño de la salida de la capa de descompresión con el tamaño de la salida de la capa de compresión del mismo nivel. Por ultimo, en la última capa se utiliza una función de activación tipo *sigmoide*.

- **Tamaño de la red** (*Entero*): Define la cantidad de compresiones/descompresiones se harán en el Encoder y el Decoder. Los valores utilizados van entre 2 y 6.
- **Número inicial de filtros** (*Entero*): Define el número inicial de kernels que se utilizaran en la primera capa de convolución, la cantidad de filtros posteriores son el doble de la capa anterior. Se utiliza un valor aleatorio entre 2 y 256.
- **Doble convolución en el Encoder** (*Booleano*): Define si se utilizara una segunda capa de convolución en el Encoder.
- **Convolución en el Decoder** (*Booleano*): Define si se utilizara una capa de convolución luego de la descompresión en el Decoder.
- **Capas de Compresión** (*Lista*): Contiene las capas que se utilizarán para realizar la compresión. Las posibles son MaxPooling2D y AveragePooling2D. Estas capas se utilizan con un stride (2,2), de modo que el tamaño de salida de estas capas es la mitad del tamaño de entrada.
- **Tamaño de los filtros en compresión** (*Lista*): Contiene tuplas de dos enteros iguales que definen el tamaño de los kernel de las capas de Pooling del Encoder. Los tamaños utilizados son entre 2 y 6.
- **Tamaño del kernel de convolución** (*Lista*): Contiene tuplas de dos enteros iguales que definen el tamaño de los filtros de las capas convolucionales tanto del Encoder como el Decoder. Los tamaños utilizados son entre 2 y 6.
- **Dropout en el Encoder** (*Lista*): Lista de valores Booleanos que indican la presencia de una capa de Dropout en las capas del Encoder.
- **Batchnorm en el Encoder** (*Lista*): Lista de valores Booleanos que indican la presencia de una capa de Batchnorm en las capas del Encoder.
- **Activaciones en el Encoder** (*Lista*): Lista de strings que indica la activación a la salida de las capas convolucionales del Encoder. Las funciones utilizadas son: *relu*, *sigmoid*, *tanh*.
- **Dropout en el Decoder** (*Lista*): Lista de valores Booleanos que indican la presencia de una capa de Dropout en las capas del Decoder.
- **Batchnorm en el Decoder** (*Lista*): Lista de valores Booleanos que indican la presencia de una capa de Batchnorm en las capas del Decoder.
- **Activaciones en el Decoder** (*Lista*): Lista de strings que indica la activación a la salida de las capas convolucionales del Decoder. Las funciones utilizadas son: *relu*, *sigmoid*, *tanh*.

MultiResUNet

Esta red se describe en la sección 2.6.3. Los modelos aleatorios creados a partir de esta arquitectura comparten con las anteriores los parámetros **Tamaño de la red**, **Capas de Compresión**, **Capas de Compresión**, **Activaciones en el Encoder** y **Activaciones en el Decoder**. Sin embargo, se definen los siguientes parámetros para manipularlas al MultiResBlock y el ResPath, estos son:

- w (*entero*): Indica la cantidad total de filtros aplicados en el Multiresblock. Esta estructura se compone de tres capas convolucionales secuenciales con w_1, w_2 y w_3 cantidad de kernels, respectivamente. Estos valores se calculan mediante la siguiente fórmula:

$$w_1 = \lceil w/6 \rceil \qquad w_2 = \lceil w/3 \rceil \qquad w_3 = \lceil w/2 \rceil \tag{3.2}$$

- **Número de convoluciones** (*Lista*): Contiene la cantidad de convoluciones que se realizan en los ResPath. Puede ser un valor entre 1 y 11.

Esta red también posee una capa de activación tipo *sigmoide* en la capa de salida.

Para el entrenamiento de todas las redes se ha utilizado el optimizador Adam, cuyo *Learning Rate* es elegido también de manera aleatoria pudiendo tomar un valor de la forma $\alpha = n \times 10^m$, pudiendo n tomar valores entre 1 y 9 y m, entre -7 y -1. Por último se ha restringido la cantidad de parámetros de la red entre 10 000 y 6 000 000, con el fin de evitar los tiempos excesivos de entrenamiento y las redes demasiado pequeñas.

3.3. Selección de redes

Para llevar a cabo el *torneo* se generan 700 modelos (100 por cada modo de vibración) aleatorios según los parámetros de creación descritos anteriormente. Luego de esto realizas *tandas de entrenamiento* en las que cada modelo es entrenado un total de 10 épocas utilizando como función de costo el IoU y registrando su error en la predicción de los datos de validación. Posterior a esto se eliminan el 10 % de las redes con peor error de eliminación (la cantidad de redes es redondeada hacia arriba) y se vuelve a repetir el proceso de las tandas de entrenamiento hasta terminar con una única red. Ésta es entrenada durante 5000 épocas restantes con un criterio de EarlyStopping de parámetros $\Delta_{min} = 1 \times 10^{-6}$ y *patience*=50 épocas, monitoreando el error de validación.

Concluido este proceso, se tiene siete modelos entrenados con un modo en particular de vibración y cuya respuesta es una matriz de 51×71 cuyos valores son números tipo float entre 0 y 1 según identifiquen delaminación (1) o ausencia de ella (0). Para elegir el umbral que define la intersección en la respuesta se evalúa la predicción de cada una de las redes en el dataset de entrenamiento y se selecciona un valor umbral de retorne el mayor IoU promedio.

3.4. Unificación de redes

En la práctica, es posible que la respuesta de una placa no presente los siete modos con los que fueron entrenadas las redes. Además, no se sabe *a priori* qué imagen corresponde con cada modo en particular, por lo que es necesario realizar un proceso de emparejamiento de

los modos obtenidos experimentalmente para asignarle correctamente a cada uno la red que le corresponda con la finalidad de lograr una predicción adecuada. Para llevar a cabo esta labor se utiliza el indicador MAC, descrito en la sección 2.3. Llamamos X_r la respuesta de un panel sano utilizado como referencia, correspondiendo a un conjunto de siete matrices y X_m el conjunto de los modos normales medidos, luego se siguen las siguientes instrucciones:

1. Calcular la matriz de CrossMAC ($C = \text{CMAC}(X_r, X_m)$).
2. Encontrar el máximo de la matriz C.
3. Emparejar los modos (X_r^i, X_m^j), siendo i,j los indices del valor máximo encontrado.
4. Quitar la fila i y columna j de la matriz C
5. Quitar a X_r el elemento i y a X_m el elemento j.
6. Repetir desde 2 hasta eliminar la matriz C completamente.

Luego de tener los modos emparejados, se genera la predicción de la delaminación para cada una de los modos utilizando la red correspondiente, obteniendo un conjunto de matrices con la zona de delaminación que cada red predijo individualmente. Para unificar la respuesta se utiliza un proceso de votación, el cual corresponde a realizar una suma pixel a pixel de la cantidad n de matrices que se tengan, luego realizar una revisión pixel a pixel de la matriz resultante, si el valor es superior a $\lceil n/2 \rceil$, el pixel toma el valor 1, expresando una presencia de delaminación, de lo contrario toma el valor 0.

3.5. Validación experimental y comparación de resultados

La validación experimental de este trabajo consiste en la aplicación de las 7 redes entrenadas anteriormente a un set de datos de respuestas modales de placas obtenidas en laboratorio [5]. Los paneles han sido construidos con una zona delaminada conocida la cual se describe en la Tabla 3.1 y su ubicación se encuentra en la Figura 3.2. Adicionalmente, las propiedades mecánicas de los paneles son descritas en la tablas 3.2 y 3.3.

Tabla 3.1: Tamaño de la delaminación en casos experimentales

ID	Diámetro de daño normalizado		Forma
	Daño 1	Daño 2	
0	-	-	-
1	0.12	-	Circular
2	0.14	0.07	Cuadrado
3	0.11	0.17	Circular
4	0.09	-	Circular

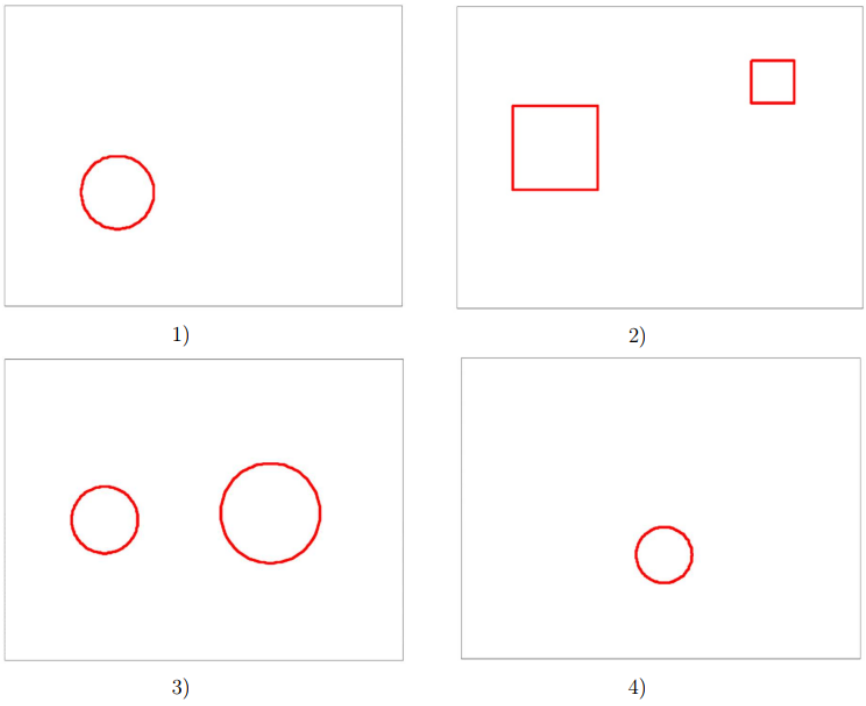


Figura 3.2: Delaminación en escenarios experimentales [5]

Tabla 3.2: Propiedades mecánicas del núcleo de aluminio

Tamaño de la celda	19.1 mm
Espesor de la pared de la celda	0.05 mm
Espesor del núcleo	10 mm
Densidad	20.8 kg/m^3
Resistencia a la compresión	0.448 MPa
Resistencia al corte longitudinal (σ_{xy})	0.345 MPa
Módulo de corte longitudinal (G_{xy}	89.63 MPa
Resistencia al corte transversal (σ_{yz})	0.241 MPa
Módulo de corte transversal (G_{yz})	41.37 MPa

Tabla 3.3: Propiedades mecánicas del núcleo de aluminio

Espesor	0.8 mm
Módulo de Elasticidad	69 GPa
Coeficiente de Poisson	0.33
Densidad	2700 kg/m^3

Las respuestas modales de las placas se pueden observar en el Anexo A.1. Finalmente, los resultados obtenidos son comparados con aquellos obtenidos en [5].

4. Resultados

4.1. Generación y exploración de datos

De los datos generados se han tomado 4 muestras para tener noción de su comportamiento, las cuales son presentadas en el Anexo A.2. Además, se muestra en la figura 3.2 el histograma de los diámetros de delaminación utilizados en la simulación.

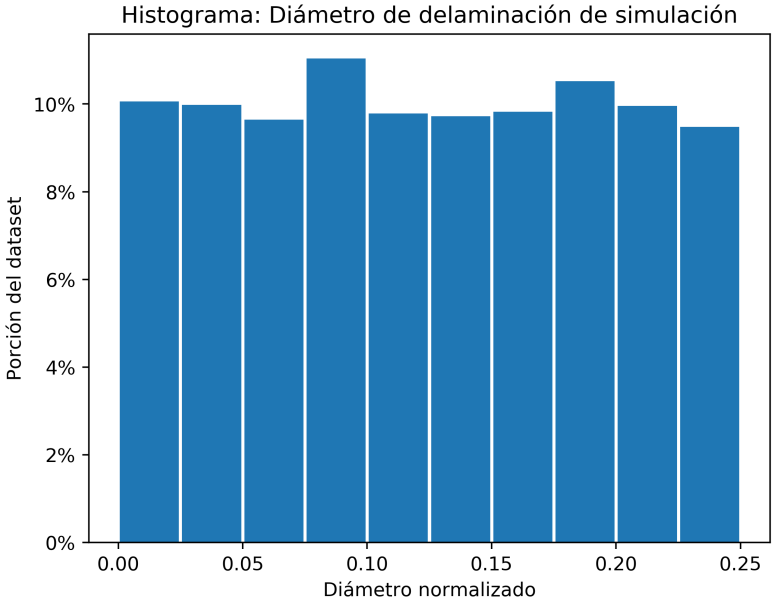


Figura 4.1: Histograma del tamaño de la zona delaminada en simulaciones.

4.2. Selección de redes

En la figura 4.2 se muestran los gráficos de cada uno de los siete torneos realizado. Las lineas indican el IoU promedio (IoUm) calculado en base al dataset de validación. La linea con menor transparencia (azul) corresponde con la red que fue, finalmente, ganadora del torneo al llegar hasta la última tanda de entrenamiento con el IoUm de validación más elevado. Las arquitecturas que obtuvieron estos resultados se encuentran esquematizadas en el Anexo A.3.

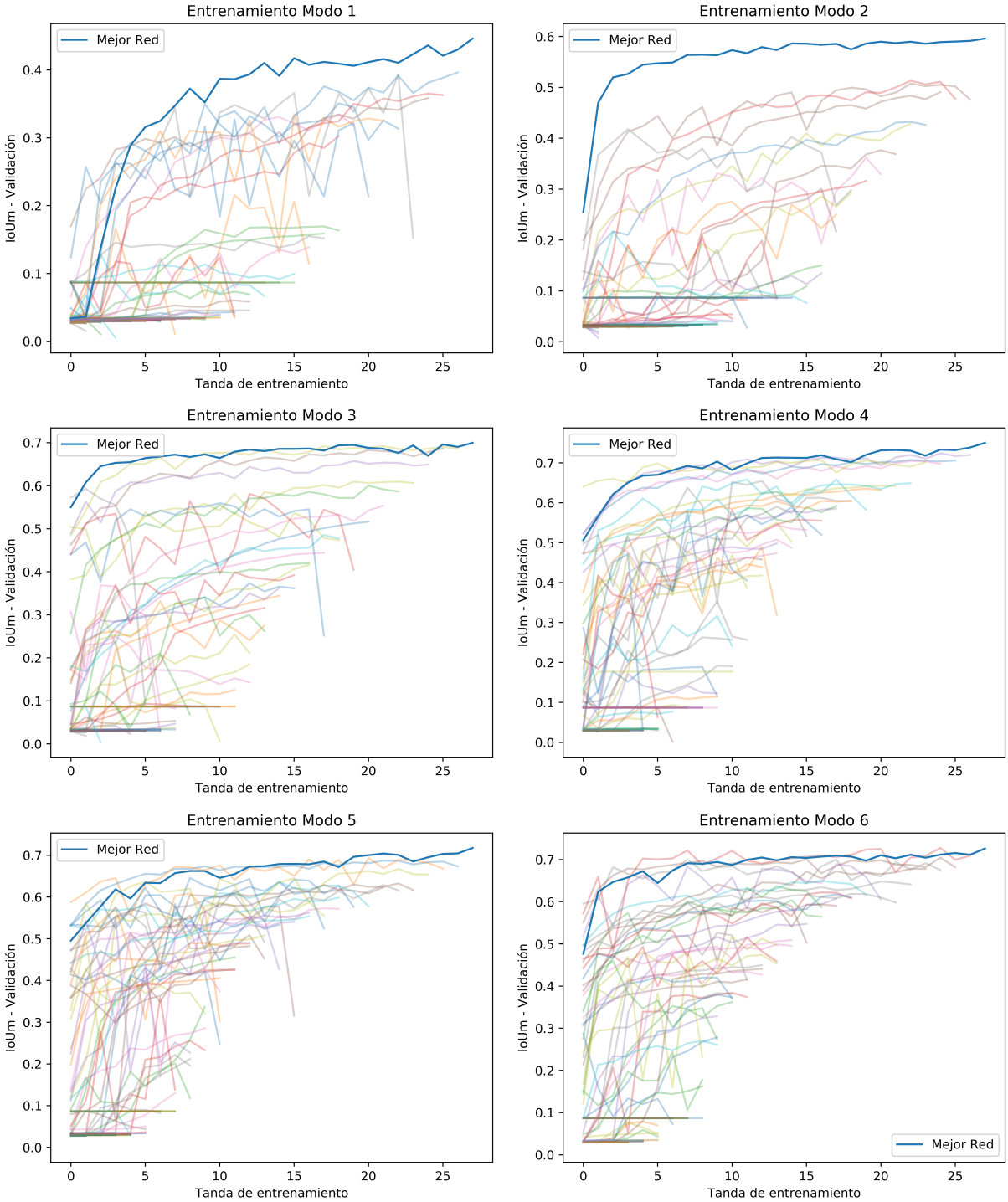


Figura 4.2: Gráfico del torneo de redes.

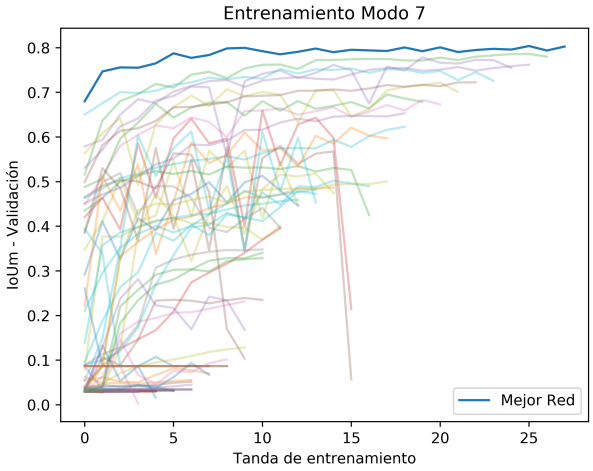


Figura 4.2: Gráfico del torneo de redes (cont.).

Dado que la salida de las redes es una función de activación *sigmoide* y se quiere binarizar para realizar la segmentación es necesario elegir un valor umbral para cada red que determine el nivel en que un pixel de una predicción se considera correspondiente a delaminación. Para esta elección se ha realizado la Figura 4.3 en la que se muestra el comportamiento del IoU para diferentes tamaños de delaminación en función del valor umbral elegido para cada red. En esta figura se aprecia que las respuestas poseen un punto de inflexión en valores umbral bajos, de modo que, en este dataset se puede elegir un umbral entre 0.1 y 0.9 sin notar cambio en el resultado final de la predicción. Es por ello que se ha elegido un umbral $th_m = 0.5$ para cada una de las siete redes.

Con el umbral definido se ha construido la Figura 4.4, en la que se muestra un diagrama de caja y bigotes con el IoU obtenido por cada una de las redes ganadoras del torneo en distintos radios de delaminación. En estos diagramas la linea continua representa la mediana de la distribución, mientras que la línea segmentada muestra el promedio.

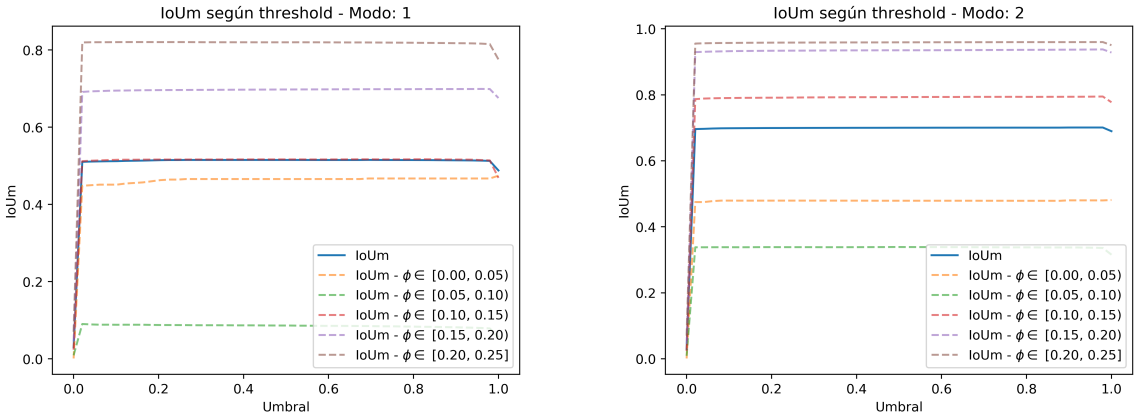


Figura 4.3: IoUm en función del valor umbral.

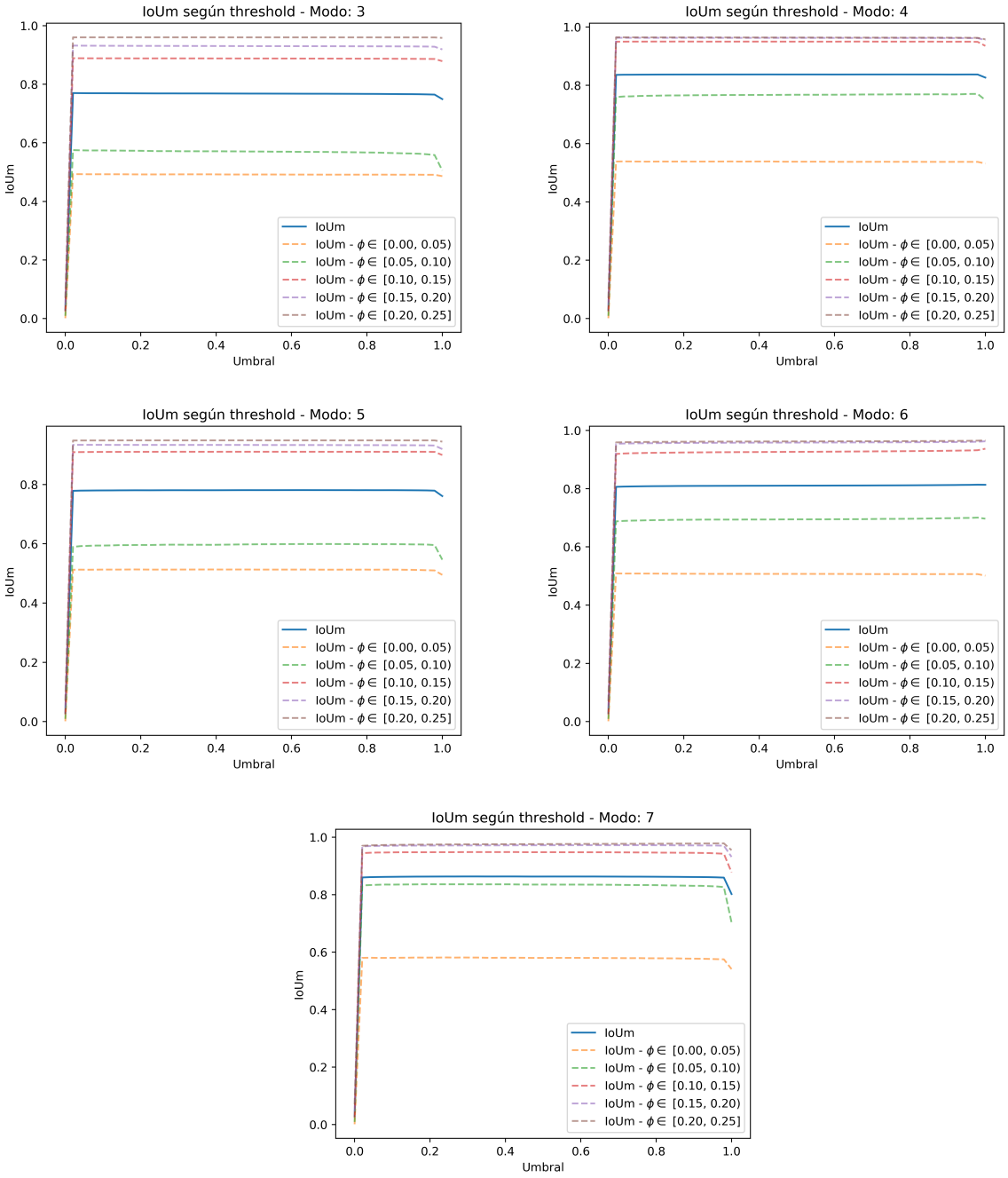


Figura 4.3: IoUm en función del valor umbral (cont.).

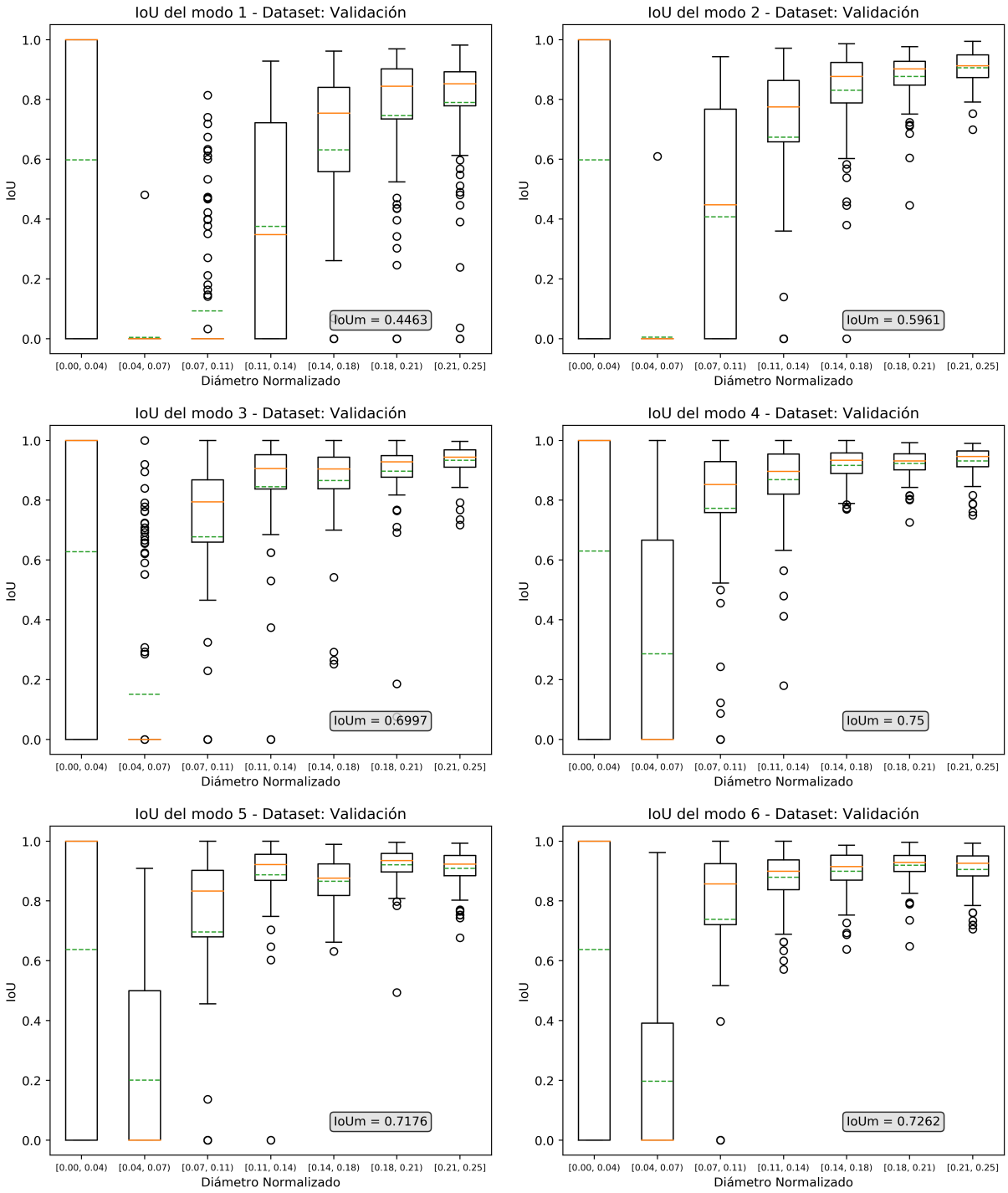


Figura 4.4: Intersection over Union de las predicciones.

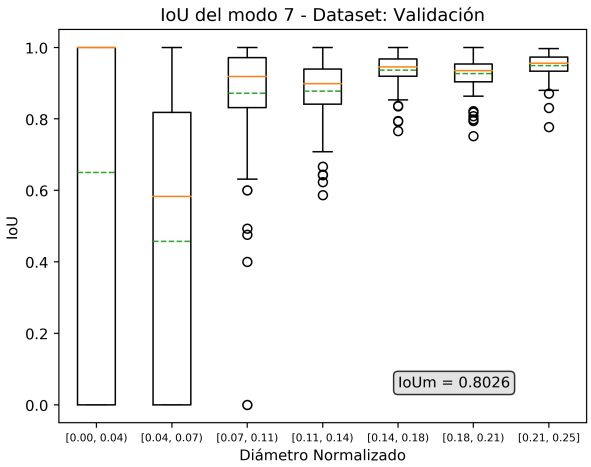


Figura 4.4: Intersection over Union de las predicciones (cont.).

Con esto ya se tiene un set de redes especializadas, cada una, en predecir un modo de vibración en particular.

4.3. Unificación de Redes

Para retornar una única respuesta para una entrada de múltiples modos de vibración se ha utilizado el método de votación descrito en la sección 3.4. Utilizando este algoritmo se ha realizado la predicción del dataset de testeo, obteniendo un IoUm = 0.763, distribuido como se aprecia en la Figura 4.5, utilizando el mismo diagrama de caja y bigotes descrito anteriormente.

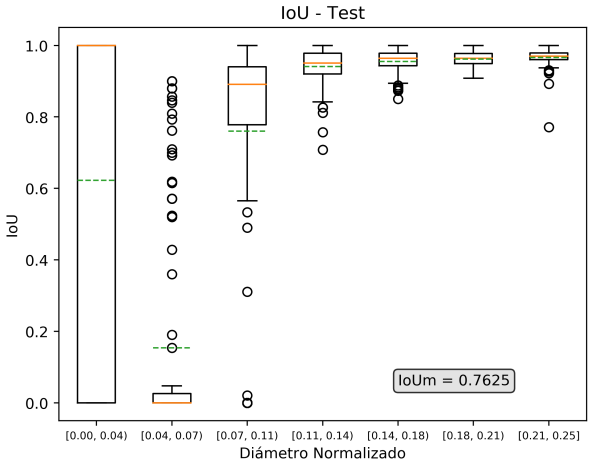
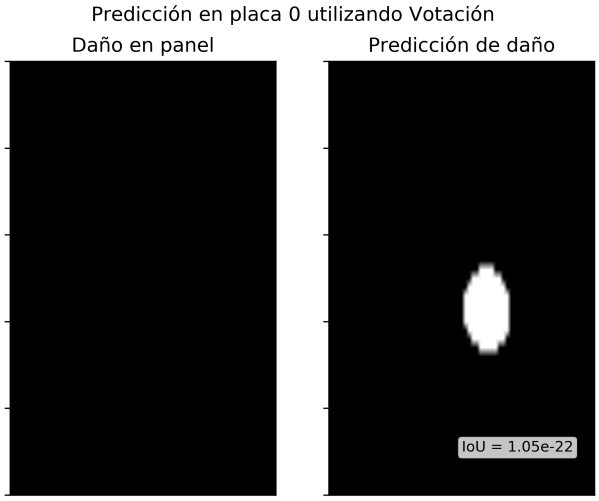


Figura 4.5: IoU del dataset de Testeo utilizando Votación.

4.4. Validación Experimental

Para la validación experimental se ha realizado el proceso de *pairing* que se ha descrito en la metodología. Las matrices de *CrossMAC* son mostradas en el Anexo A.4 y el emparejamiento resultante se muestra en el Anexo A.5.

Con el emparejamiento realizado se procede a predecir los modos de las placas medidas experimentalmente con los daños descritos en la Tabla 3.1. Estas predicciones son presentadas junto a su respuesta esperada (*ground truth*) en la Figura 4.5, también se encuentran el IoU de cada una de ellas en la Tabla 4.1, en donde se muestra, paralelamente, el IoU obtenido mediante el método de Gapped Smoothing en [5] y una comparativa entre los resultados de los dos métodos.



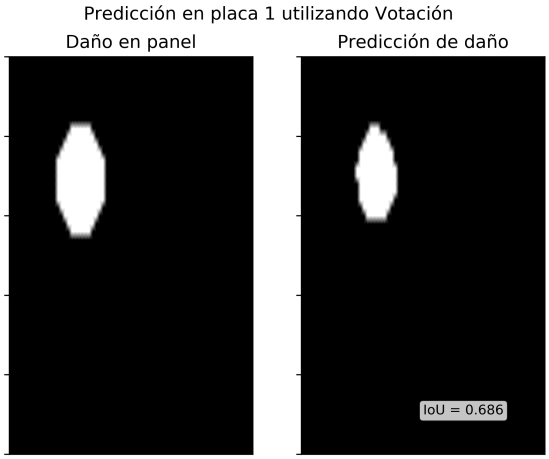


Figura 4.5: Predicciones de placas experimentales.

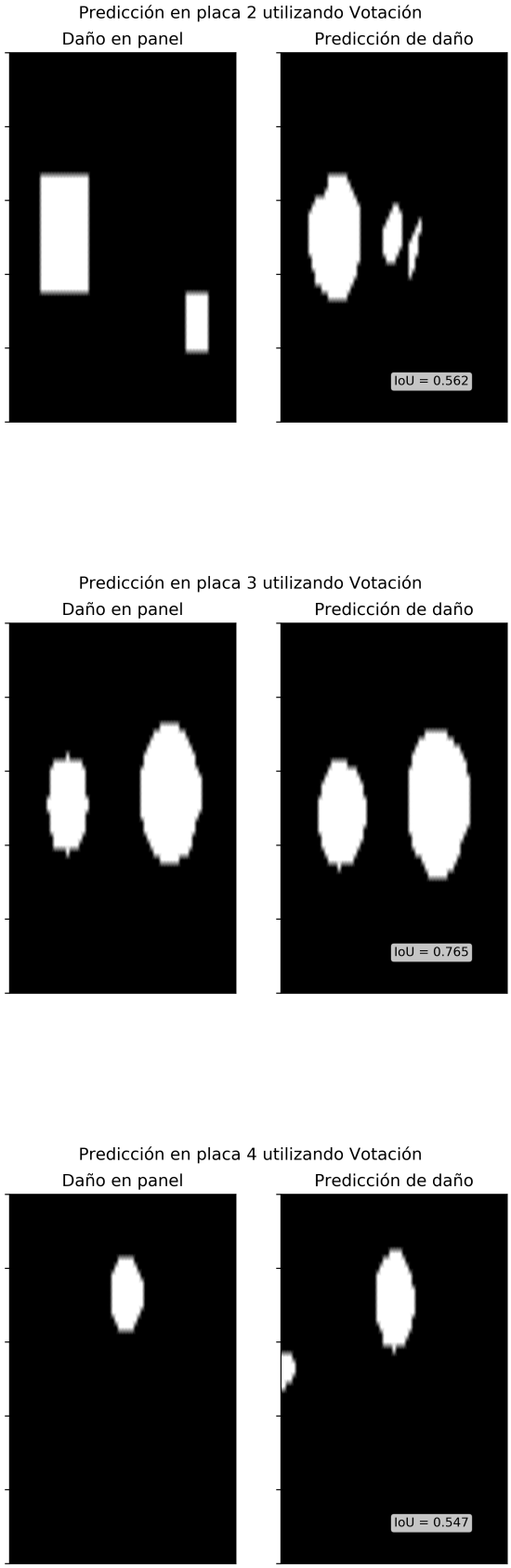


Figura 4.3: Predicciones de placas experimentales (cont.).

Tabla 4.1: Comparativa de resultados entre los métodos.

Placa	Gapped Smoothing	Redes Neuronales Convolucionales	Diferencia Porcentual
0	-	0.00 %	-
1	46.0 %	68.6 %	49.1 %
2	46.0 %	56.2 %	22.2 %
3	65.0 %	76.5 %	17.7 %
4	47.0 %	54.7 %	16.4 %

5. Análisis

5.1. Resultados numéricos

En la figura 4.1 se observa que los datos proporcionados por el modelo experimental se encuentran, efectivamente, distribuidos los diámetros normalizados 0 y 0.25, esto es necesario para que el modelo logre generalizar su predicción ante distintos tamaños y no se sesgue con alguno en particular.

Al observar las figuras del Anexo A.2 se aprecian a simple vista dos comportamientos en la relación entre la zona delaminada del panel y sus modos normales. El primero es el hecho de que el efecto de la delaminación es más notorio en los modos con una mayor cantidad de nodos, pues se observa claramente la diferencia que existe entre los paneles con delaminación y el panel sano en los modos 5, 6 y 7. El segundo comportamiento observado corresponde a que el tamaño del defecto provoca que éste sea notable en modos con cada vez menor cantidad de nodos a medida que éste se hace más grande. Si bien estos comportamiento son claramente expresados en las muestras de la simulación, ellos no se presentan claramente en los datos experimentales mostrados en el Anexo A.1. Esta diferencia podría provocar que exista una desigualdad en el desempeño de las redes al predecir datos experimentales.

Con respecto a la generación de redes, en las figuras del Anexo A.3 se observa que únicamente sobrevivieron arquitecturas tipo U-Net y MultiResUNet. Esto podría deberse a la capacidad que entregan las *skip-conections* en ambos casos, ya que, al tener esta vía de comunicación entre el Encoder y el Decoder, se logra que se pierda menos información al momento de comprimir las imágenes. Por otro lado, esto no significa que estas arquitecturas encuentren siempre mejores resultados que las basadas en DeconvNet. Estas últimas poseen la cualidad de ser más sencillas en su implementación y más rápidas al momento de entrenar, entregando de igual manera buenos resultados.

En la figura 4.2 se aprecia, en todos los modos, que una gran cantidad de redes no aprende a lo largo del entrenamiento, siendo finalmente eliminadas al no aumentar si IoUm. Paralelamente, se observa que, en general, las redes que llegan al final del torneo son aquellas que tienen un IoUm al momento de inicializar sus parámetros. Estos dos fenómenos sugieren una modificación al proceso del torneo pues el entrenar las redes que sostenidamente no evolucionan es una pérdida de tiempo, por lo que una mejora en el mecanismo de eliminación podría ser realizar un seguimiento activo del comportamiento del *score* de la red, de modo que si éste se mantiene estático una cantidad de tandas de entrenamiento, la red sea eliminada. Por otro lado, el hecho de que las redes finalistas comiencen con un alto IoUm indica que

es posible realizar una eliminación más rápida, es decir, aumentar la tasa de eliminación de las redes. Estos cambios permitirían, por ejemplo, realizar un entrenamiento con una mayor cantidad de redes por explorar en un mismo tiempo.

Con respecto a la selección del umbral para la binarización, es interesante notar que en todos los modos las redes tengan un comportamiento extremista, es decir, que su respuesta sea siempre cercana a los valores extremos. Esta cualidad facilita la elección de un valor umbral para su binarización. Éste comportamiento puede deberse a la forma en que se implementó la función de costo pues, para calcular el IoU, en particular la cantidad de verdaderos positivos, se realiza una multiplicación pixel a pixel entre la salida de la red (que corresponde a un continuo entre 0 y 1) y la matriz de referencia (que es solamente posee ceros o unos). Estas multiplicaciones serán tan cercana a uno como el pixel de salida sea cercano a la unidad. De este modo, para maximizar su *score* la red aprende a extremar el valor de su salida. Sin embargo, este aspecto no puede ser tomado, *a priori*, como una generalidad pues no hay garantías de que esto suceda en todos los casos de segmentación.

Los resultados del *score* de validación de la Figura 4.4 muestran que las redes presentan un Intersection over Union promedio entre 44.6 % y un 80 %, siguiendo el comportamiento esperado al observar las muestras de las simulaciones, esto es, el hecho de que se prediga con mayor precisión los modos con mayor cantidad de nodos. Por otro lado, se observa que existe una dificultad para predecir delaminaciones pequeñas, esto se hace patente en las medidas de tendencia central en los diámetros de delaminación [0, 0.04] y [0.04, 0.07], en donde se observa que la media y el promedio se encuentran cercanos a cero en todos los casos, exceptuando el modo 7. Se observa que en todas las redes, la mediana en el primer rango se encuentra en el valor 1, mientras que el promedio se acerca al 60 % de IoU. Esto puede deberse a que los defectos pequeños tienen tan poco impacto en la respuesta dinámica que la red es incapaz de detectar una diferencia con el panel sano, de este modo, los paneles que en efecto están sanos, son correctamente detectados y la identificación de daño en placas que poseen fallas incipientes tiene un IoU = 0, pues es catalogado como completamente sano. Este comportamiento, si bien no es deseable, tampoco es preocupante en la práctica pues los paneles con daños pequeños pequeños no presentan un gran peligro. En este sentido es más importante que las redes no subestimen fallas de tamaño intermedio o mayores. Con respecto a ésto último, se observa un buen comportamiento de la respuesta de las redes, pues, en general, los defectos entre [0.11 y 0.25] posee un IoU promedio superior al 80 % en todos los casos, exceptuando el modo 1, el cual tiene dificultad para predecir daño en los rangos intermedio y bajo, siendo concordante con poseer *score* promedio menor.

Al unificar las redes se aprecia que el Intersection over Union promedio se ve beneficiado, pues es superior al que poseen las redes de manera individual, exceptuando la del modo 7. Ésta red podría ser la causante de este efecto debido a su alto *score* promedio. Esto puede ser una desventaja debido a que el rendimiento podría verse mermado por la ausencia de este modo en la práctica.

5.2. Resultados experimentales

En primer lugar se observan las imágenes del Anexo A.5, en ellas se aprecia que el emparejamiento es visualmente concordante cuando el MAC es superior a 0.2, existiendo varios

casos en los que se emparejan modos teniendo un MAC cercano a 0. Sin embargo, no se puede descartar que a pesar de poseer un MAC bajo, los modos experimentales correspondan a los de referencia, pues la zona delaminada produce una distorsión en el modo que no es fácilmente identificable a simple vista. Ejemplo de ello es el modo 4 de la placa 3, en donde se observa de fondo la silueta del modo aparentemente correcto, sin embargo, el tamaño del defecto provoca una distorsión tal que eclipsa la forma "original" del modo, resultando un indicador MAC = 0.0941.

En las predicciones con las placas medidas experimentalmente se observan distintos fenómenos, los cuales conviene analizar por separado:

Placa 0

En esta placa se observa que la predicción es completamente erronea pues se trata de una placa sana en la que se predice un daño aproximadamente al centro de la placa. En este caso, el error se explica por la metodología de medición de los modos, ya que en las cercanías de la zona presuntamente delaminada es donde se instala el oscilador (*shaker*) para generar la respuesta dinámica en la red. La unión de este aparato con la placa genera una rigidez que no fue considerada en el modelo numérico, de modo que las redes detectan esta zona como una anomalía con respecto al modo de vibrar de un panel sano. Este es un problema práctico que debe ser considerado en la investigación pues genera un falso positivo importante que podría, detectando un daño de aproximadamente 0.1 diámetro normalizado donde no existe. Este efecto de la rigidez adicional no se hace presente en el resto de placas y esto es debido a que la distorsión en la respuesta que provoca una delaminación es mayor que el efecto de esta rigidez. Dos posibles soluciones a este problema son: Considerar la rigidez adicional en la simulación para el entrenamiento de las redes o buscar una forma alternativa de inducir vibraciones en el panel al momento de medir.

Placa 1

En este panel se observa que, si bien la predicción subestima el área de delaminación, es capaz de predecir la ubicación y el rango del diámetro de delaminación con una precisión aceptable.

Placa 2

En esta placa se observa que el daño principal es detectado correctamente a diferencia del daño secundario. Esto puede deberse a que el daño mayor eclipsa el efecto del la delaminación pequeña. Otro fenómeno interesante es que la predicción del daño tiene forma aproximadamente circular, lo cual puede deberse tanto al hecho de que las redes fueron entrenadas únicamente con defectos circulares o, más probablemente, es que la respuesta de un defecto cuadrado genere, en efecto, una distorsión del modo de vibración similar a la que provoca un defecto circular. Adicionalmente se detecta un falso positivo en la zona central de la placa, el cual puede estar relacionado con el mismo fenómeno presentado en la placa 0.

Placa 3

Esta placa presenta los mejores resultados en la predicción. En este caso se observa que es posible detectar dos daños simultáneamente. En este caso el daño principal no opacó al secundario como en el caso de la placa anterior, esto debido a que los tamaños de delaminación son comparables entre sí.

Placa 4

La predicción de este último panel sobrestima el tamaño del daño pero es capaz de predecir precisamente la ubicación del mismo. Si se observa la Figura 4.4, la respuesta promedio en el rango de tamaños de este defecto oscila entre un 35 % y un 85 % de IoU, por lo que, debido a la metodología de la votación, es esperable un valor que ronde el 60 % de IoU. La predicción se ve negativamente afectada por una zona de falsos positivos en un borde de la placa.

Finalmente, en la comparativa de los resultados de este trabajo con los obtenidos anteriormente en [5] se observa una mejoría en la predicción de todos los casos, llegando hasta un 49 % en el caso de la placa 1. Además de presentar esta significante mejoría en la predicción, las redes neuronales convolucionales poseen la ventaja de no necesitar de un preprocesamiento mayor para ser implementadas. Además el método utilizado para la unificación de las redes es independiente de cualquier parámetro, lo que independiza los resultados de la toma de decisiones. Actualmente los únicos parámetros cuya selección no se ha automatizado son los valores umbrales para binarizar la salida de las redes neuronales. Sin embargo, dado su comportamiento, este parámetro se puede considerar como $th_s = 0.5$ por defecto o bien implementar algún método de selección automática de *threshold*.

6. Conclusión

En el presente trabajo se ha implementado un algoritmo para la identificación de daño por delaminación en paneles compuestos de aluminio procesando la respuesta dinámica de éstos por medio de redes neuronales convolucionales. Para ello se ha hecho uso del IoU tanto como métrica evaluadora como función de costo en el entrenamiento de las redes.

El proceso de selección de arquitecturas se ha desarrollado mediante un algoritmo de creación aleatoria de redes, el cual permitió la exploración de diversas configuraciones basadas en arquitecturas de redes neuronales convolucionales especializadas en la segmentación de imágenes. Además, se utilizó un método para unificar la respuesta de la red que no depende de ningún parámetro, otorgando una gran independencia con respecto a las decisiones a tomar para implementar la metodología.

Una de las ventajas que entrega el uso de las redes neuronales es el hecho de poder realizar predicciones con buena precisión aún cuando no se ha realizado un pre-procesamiento de los datos. Permitiendo una mayor rapidez en la aplicación de este método en la implementación práctica.

El método implementado muestra un buen desempeño en su evaluación de testeo, llegando a un 76 % de IoU promedio. Por otro lado, presenta una mejora significativa oscilando entre un 16 % y un 49 % con respecto a los resultados obtenidos en trabajos anteriores. Con ello y las carácteristicas previamente mencionadas, las redes neuronales se ven como una alternativa con un potencial interesante para ser empleado en la identificación de daño en estos paneles. Sin embargo, presenta serias dificultades en la predicción de daños con tamaños inferiores a 0.07, lo cual es un aspecto a tener en cuenta al momento de su uso.

Bibliografía

[1] D. N. Cross, "The use of honeycomb sandwich panels in the engineering applications," *Ciba-Geigy*, [circa 1984].

[2] V. N. Burlayenko and T. Sadowski, "Influence of skin/core debonding on free vibration behavior of foam and honeycomb cored sandwich plates," *International Journal of Non-Linear Mechanics*, vol. 45, no. 10, pp. 959–968, 2010.

[3] S. Grupta, S. Satpal, S. Banerjee, and A. Guha, "Vibration based health monitoring of honeycomb core sandwich panels using support vector machine," *International Journal on Smart Sensing and Intelligent Systems*, 2016.

[4] F. Seguel and V. Meruane, "Damage assessment in a sandwich panel based on full-field vibration measurements," *Journal of Sound and Vibration*, 2017.

[5] V. Meruane, I. Fernandez, R. O. Ruiz, G. Petrone, and E. Lopez-Drouguett, "Gapped gaussian smoothing technique for debonding assessment with automatic thresholding," *Wiley*, 2019.

[6] V. Meruane, V. D. Fierro, and A. Ortiz-Bernardin, "A maximum entropy approach to assess debonding in honeycomb aluminum plates," *Entropy*, vol. 16, no. 5, pp. 2869–2889, 2014.

[7] A. F. García, "Modos de vibración de una membrana rectangular." `http://www.sc.ehu.es/sbweb/fisica3/ondas/membrana_1/membrana_1.html`, 2016. Accedido: 01/12/2019.

[8] D. A. Russell, "Vibrational modeshapes of a rectangular membrane (fixed at the edges)." `https://www.acs.psu.edu/drussell/Demos/rect-membrane/rect-mem.html`, 2018. Accedido: 01/12/2019.

[9] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc, 2da ed., 2019.

[10] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[11] D. Mishra, "Transposed convolution demystified." `https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba`, 2020. Accedido: 20/06/2020.

[12] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[13] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, pp. 1520–1528, 2015.

[14] F. Sultana, A. Sufian, and P. Dutta, "Evolution of image segmentation using deep convolutional neural network: A survey," *Knowledge-Based Systems*, p. 106062, 2020.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[17] N. Ibtehaz and M. S. Rahman, "Multiresunet: Rethinking the u-net architecture for multimodal biomedical image segmentation," *Neural Networks*, vol. 121, pp. 74–87, 2020.

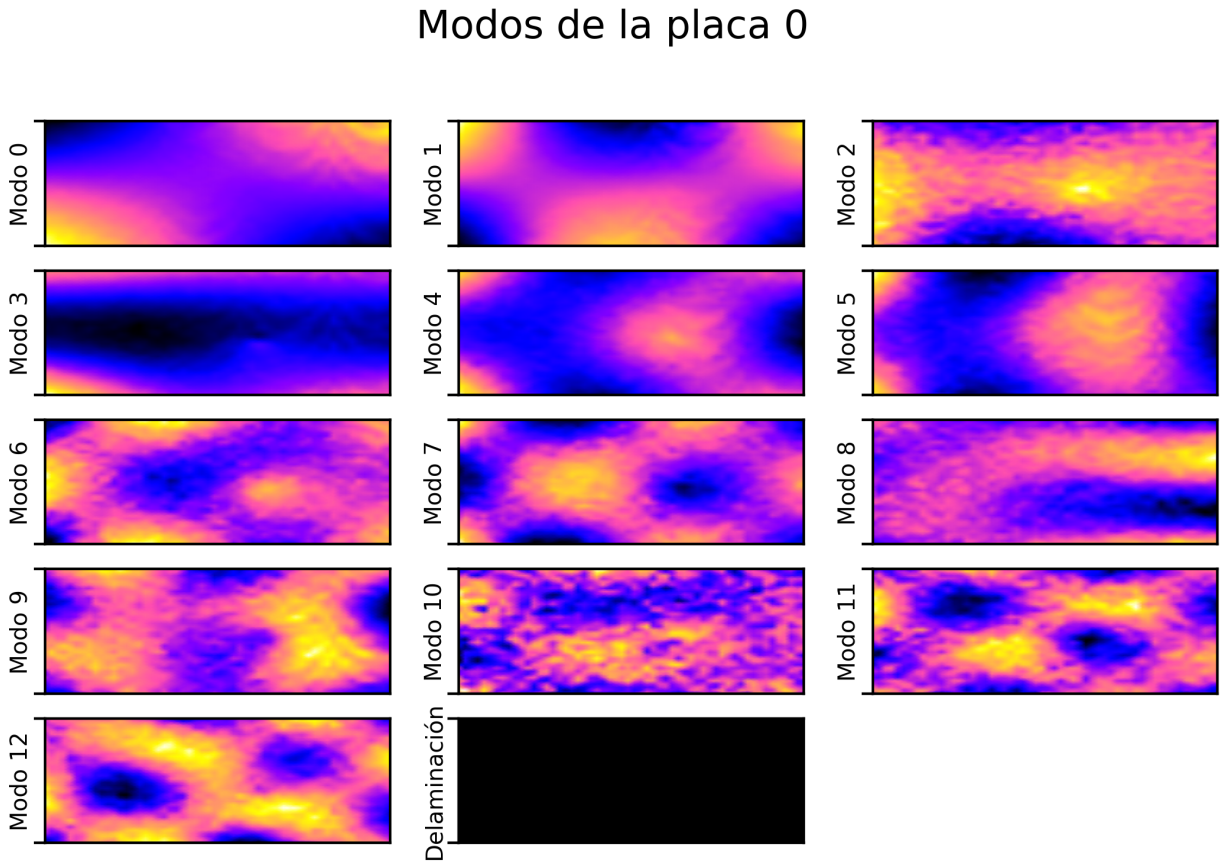
[18] M. Pastor, M. Binda, and T. Harčarik, "Modal assurance criterion," *Procedia Engineering*, vol. 48, pp. 543–548, 2012.

[19] M. A. Rahman and Y. Wang, "Optimizing intersection-over-union in deep neural networks for image segmentation," in *International symposium on visual computing*, pp. 234–244, Springer, 2016.

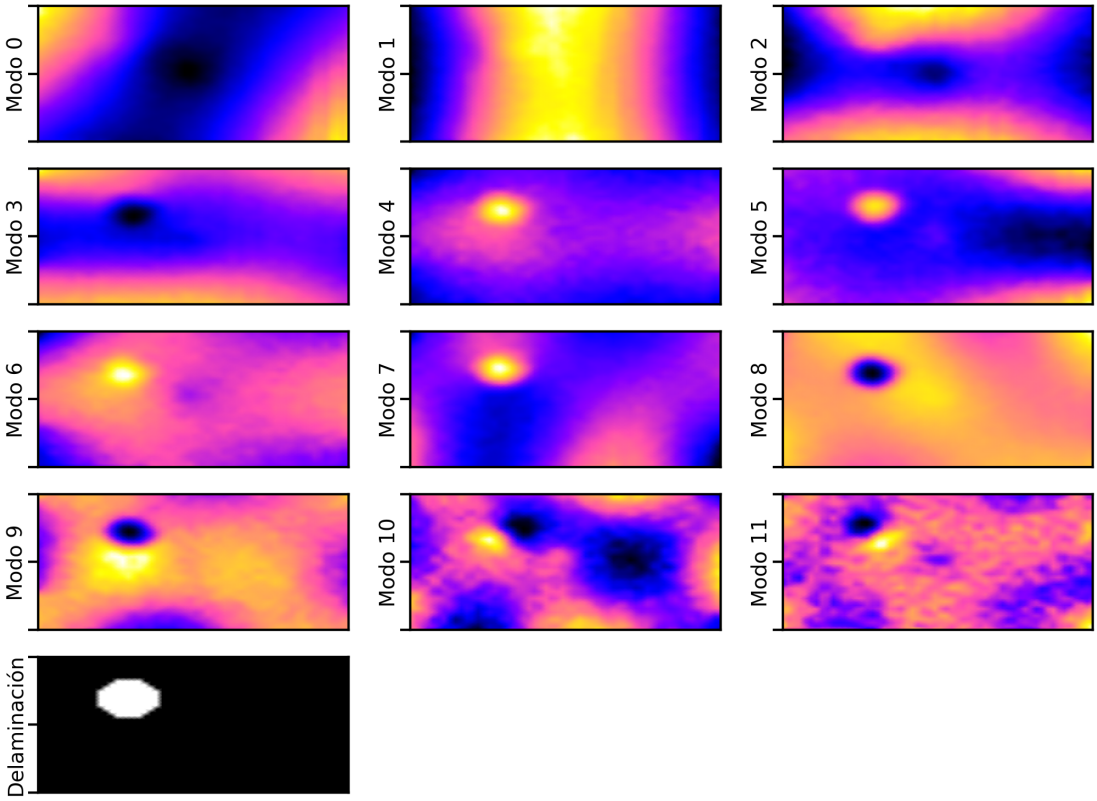
[20] N. S. Gulgec, M. Takáč, and S. N. Pakzad, "Convolutional neural network approach for robust structural damage detection and localization," *Journal of Computing in Civil Engineering*, vol. 33, no. 3, 2019.

Anexo

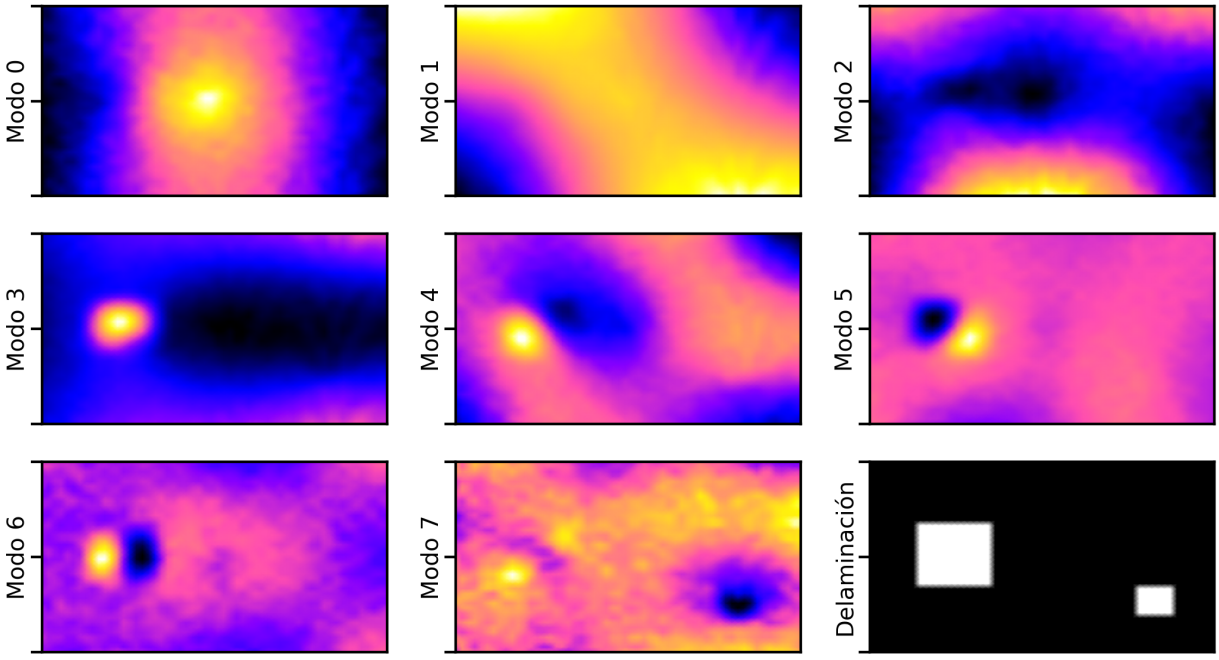
A.1 Respuesta modal de placas experimentales



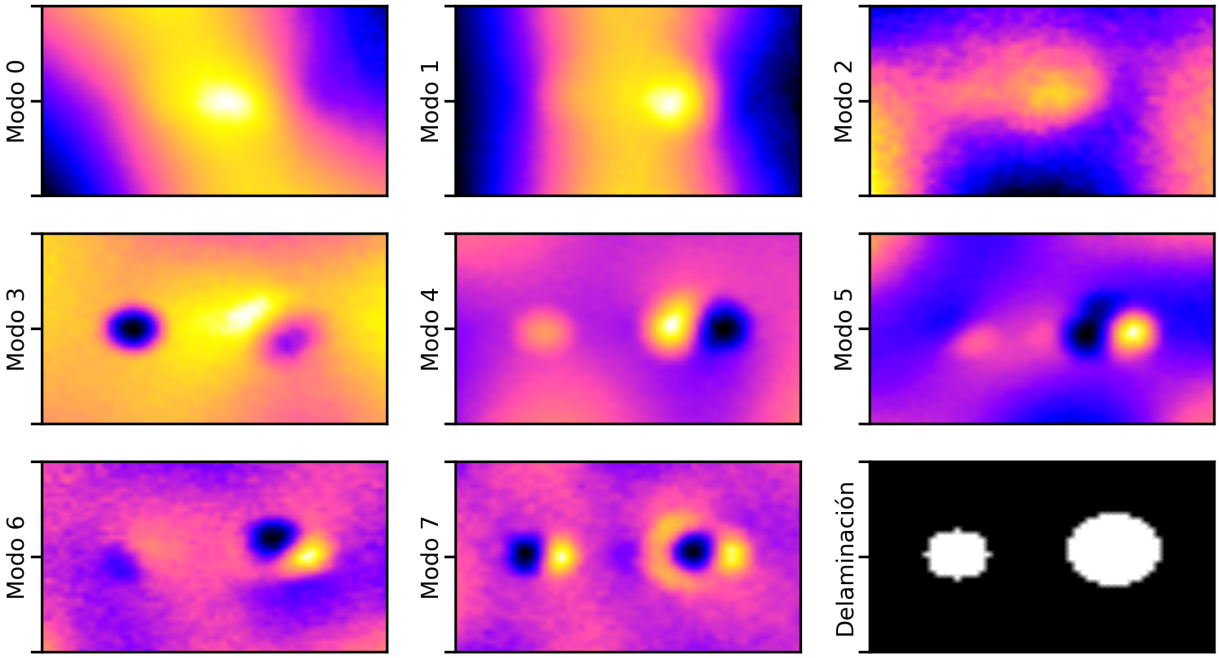
Modos de la placa 1



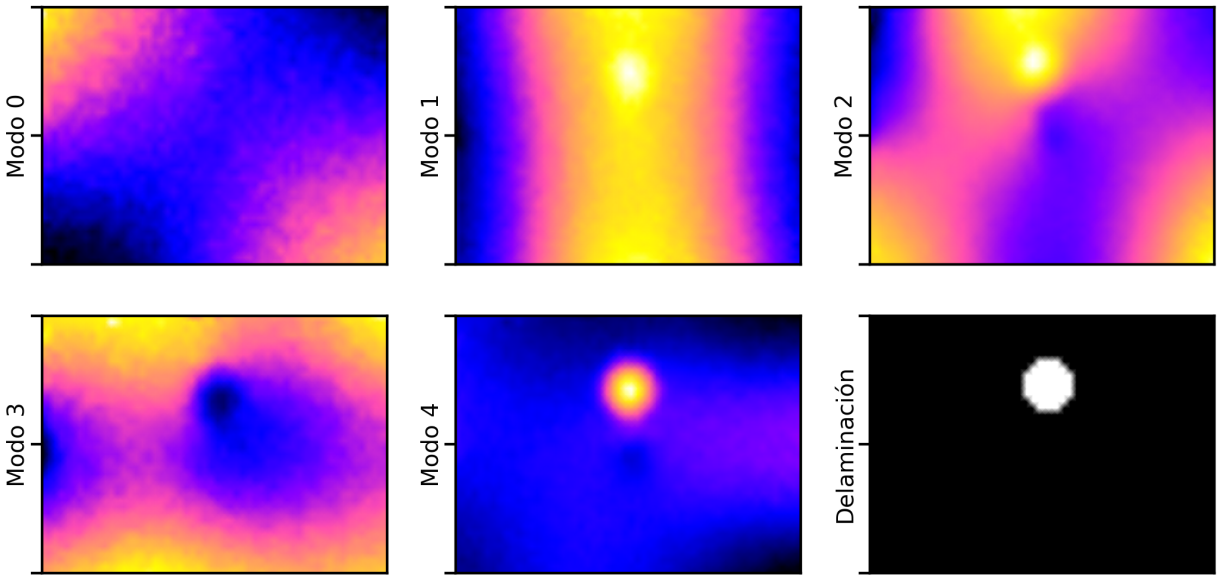
Modos de la placa 2



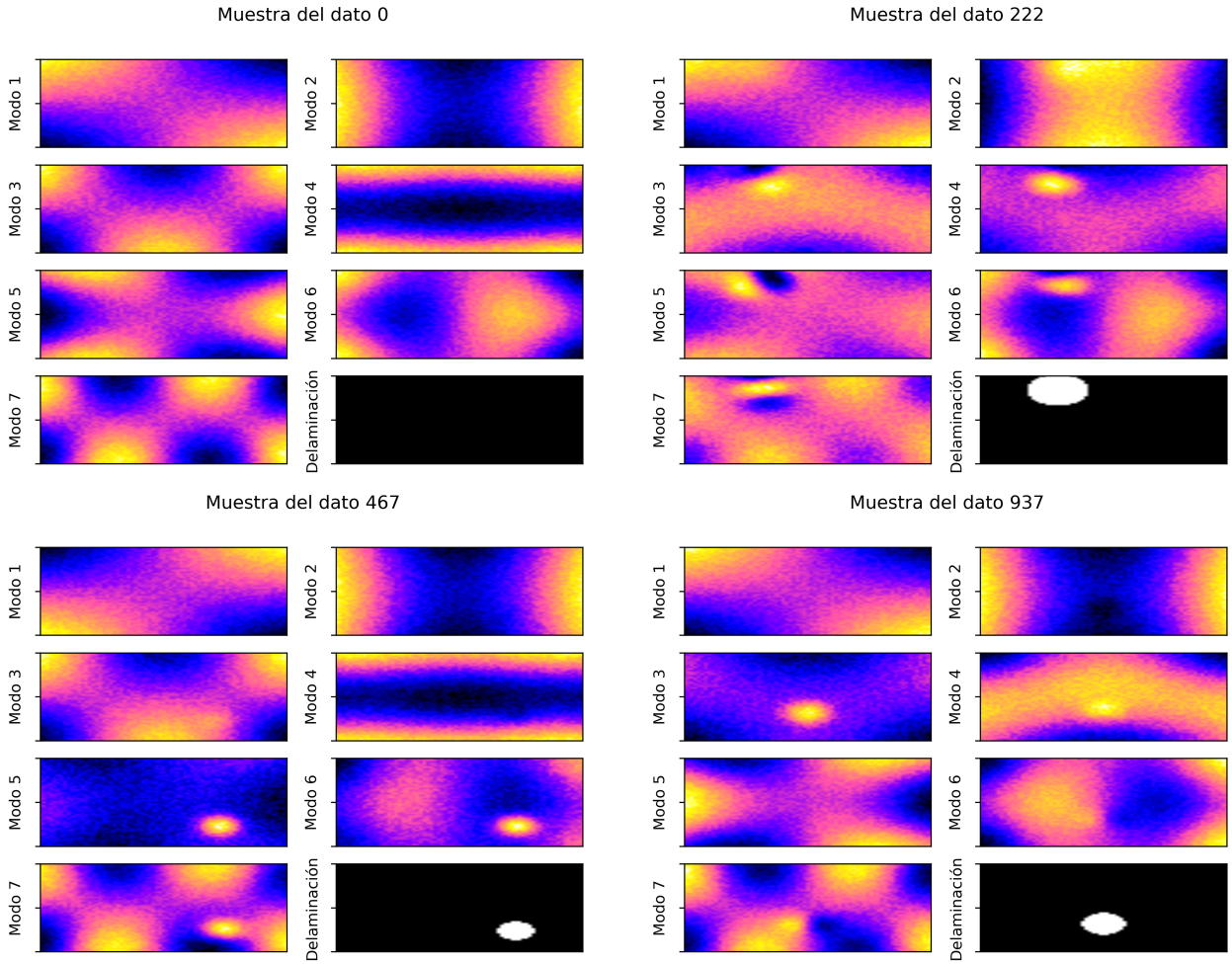
Modos de la placa 3



Modos de la placa 4

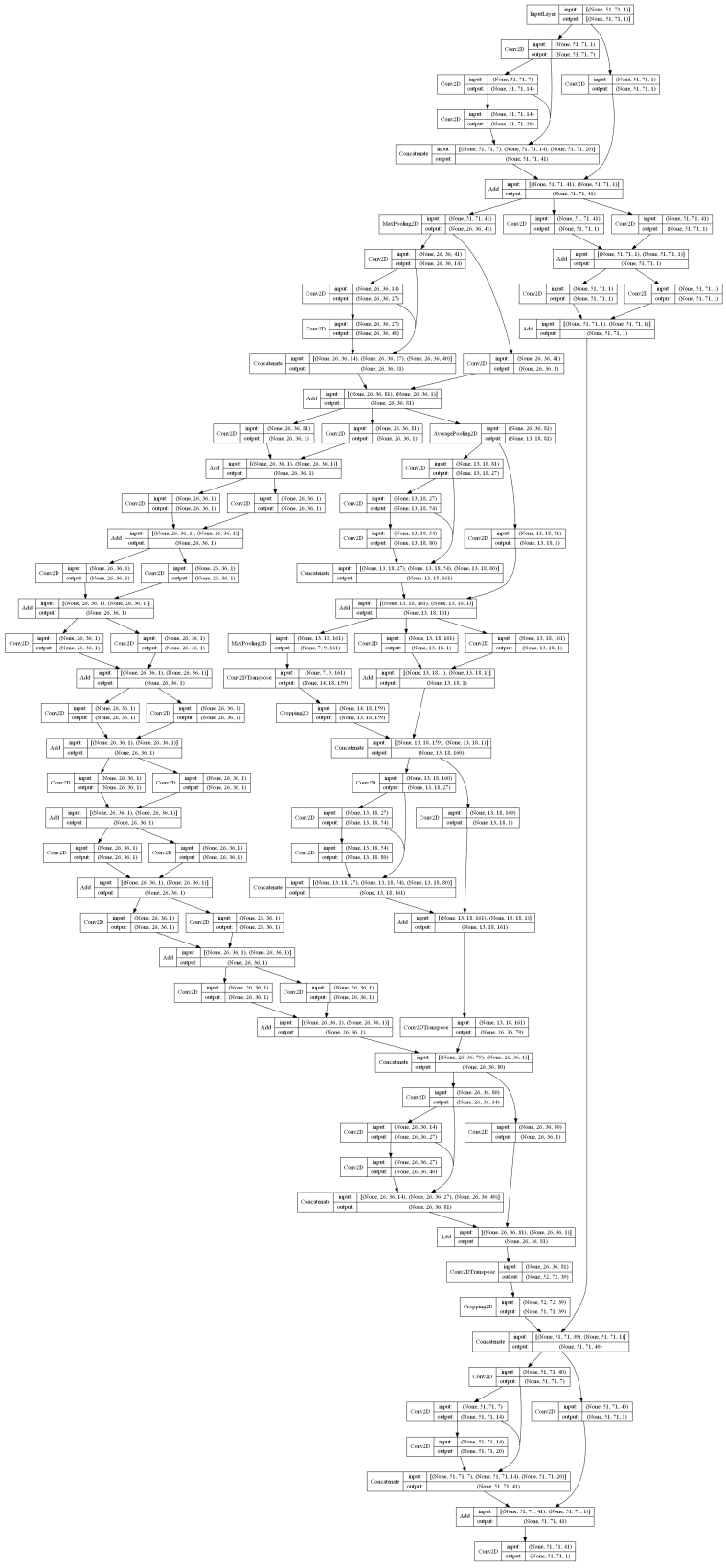


A.2 Muestras de la simulación

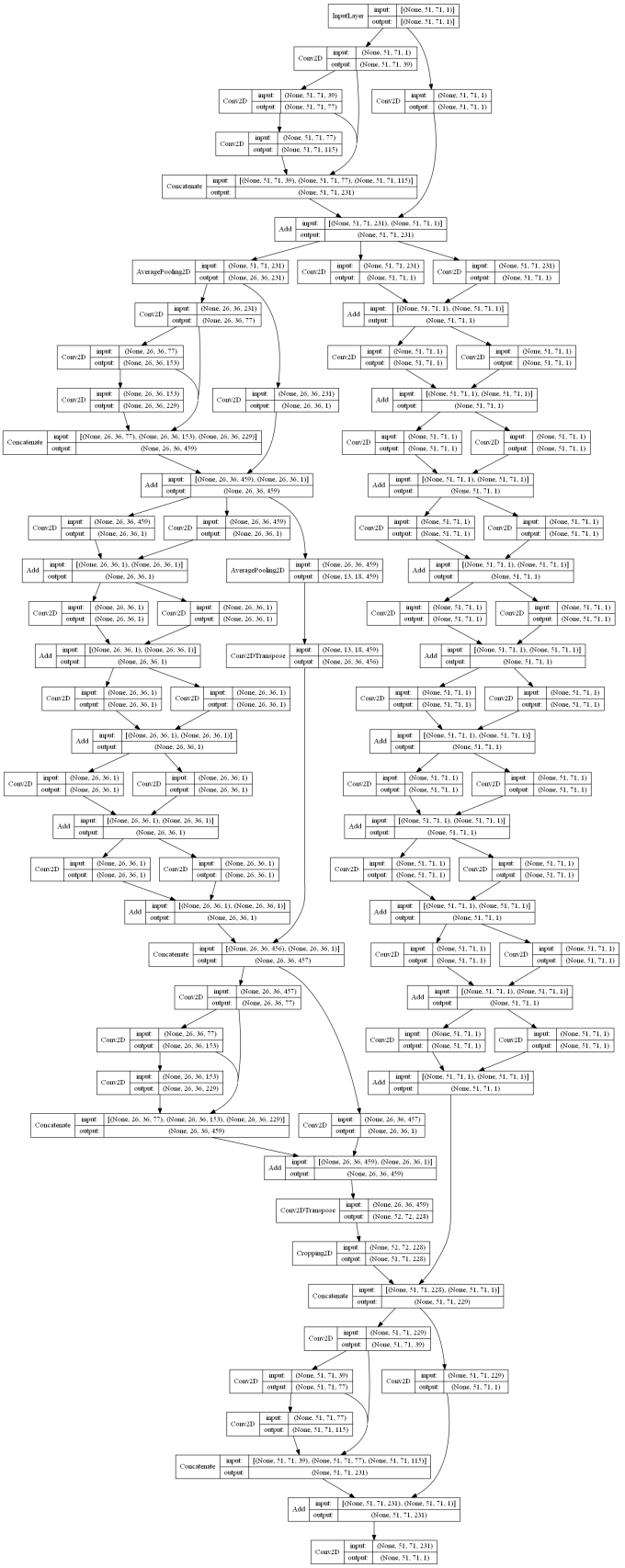


A.3 Arquitecturas creadas

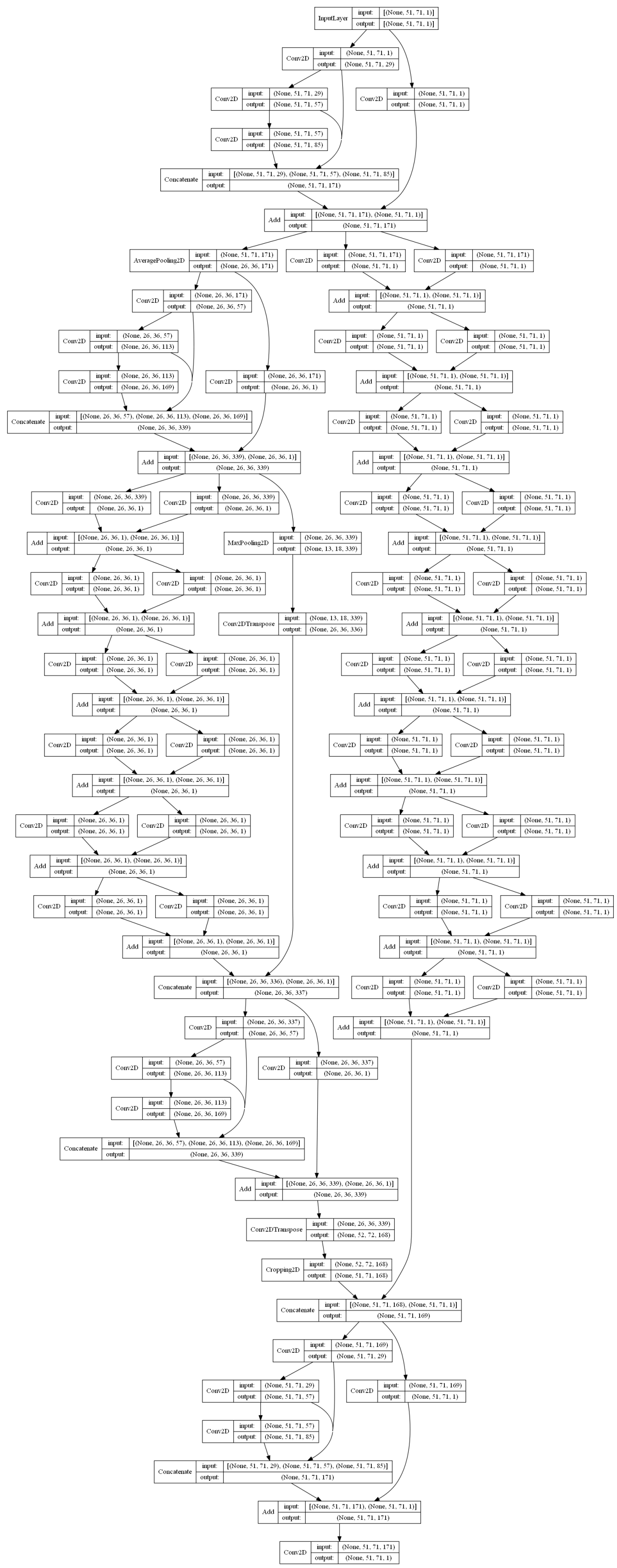
Modo 1



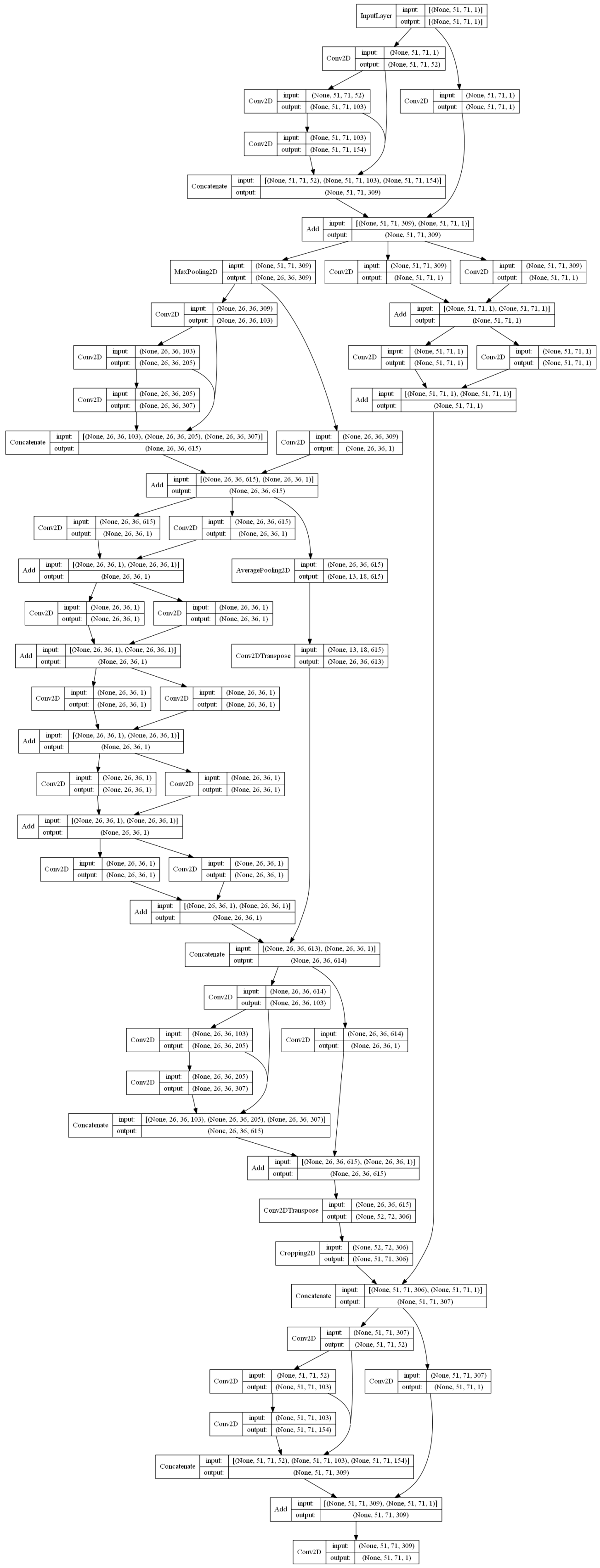
Modo 2


InputLayer
Conv2D
Concatenate
Add
AveragePooling2D
Conv2DTranspose
Cropping2D

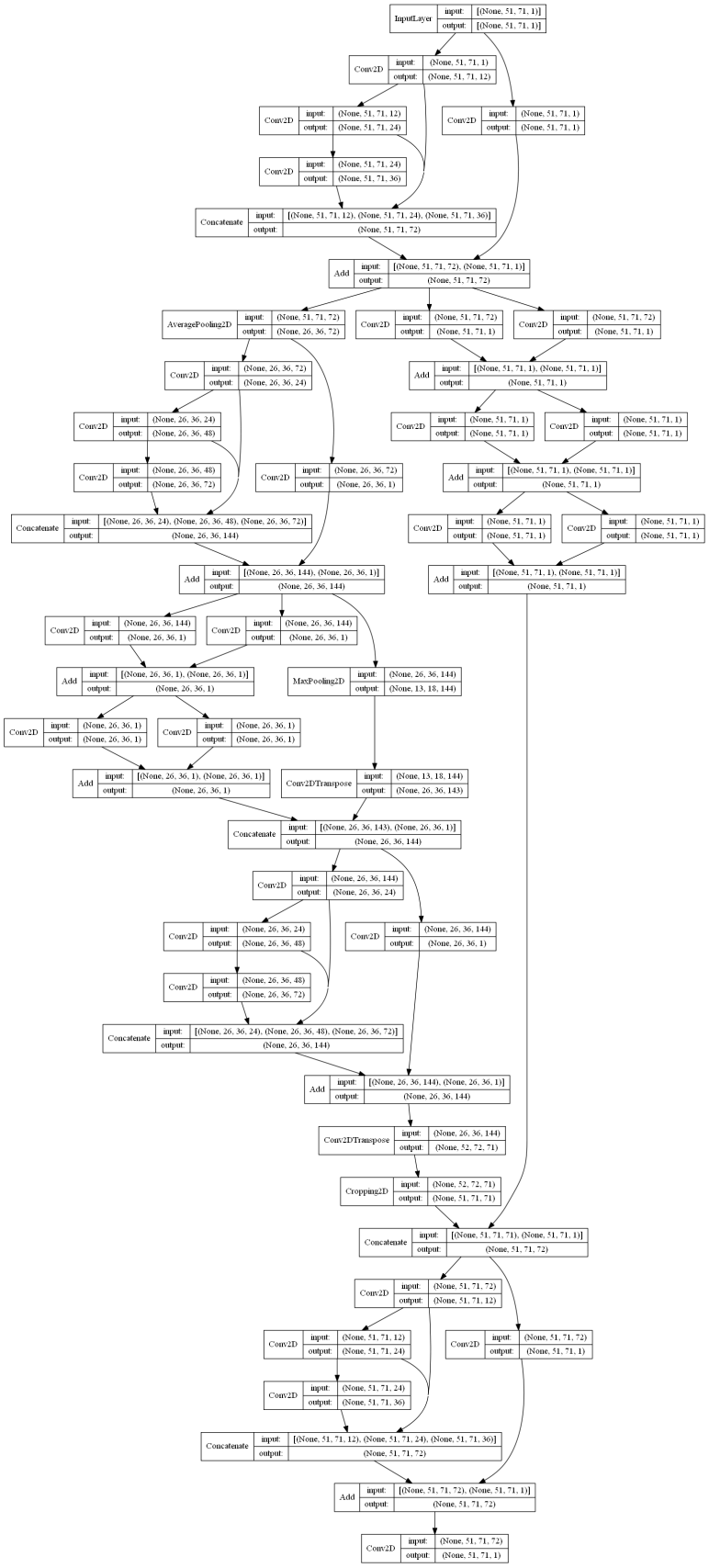
Modo 3



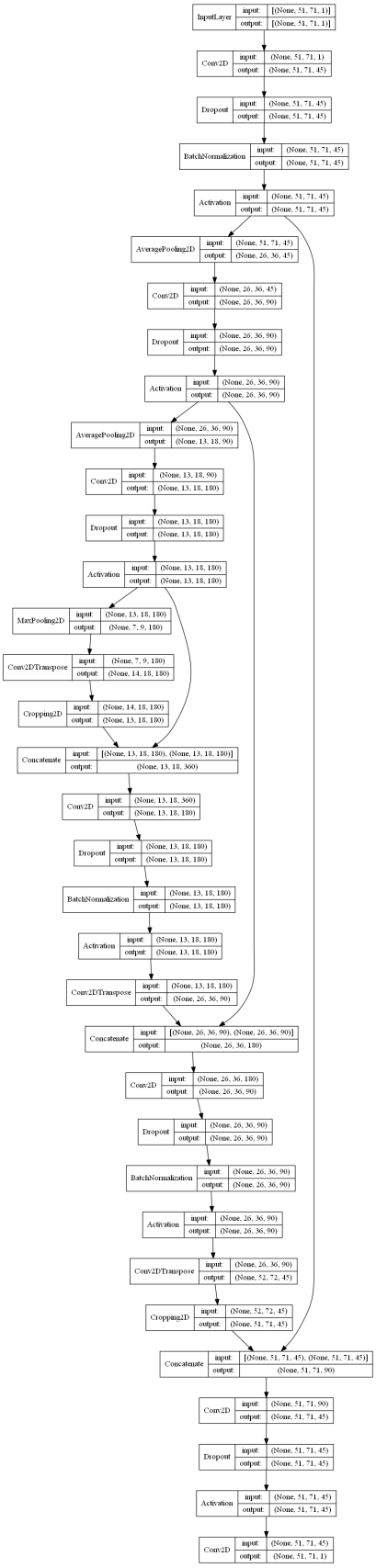
Modo 4



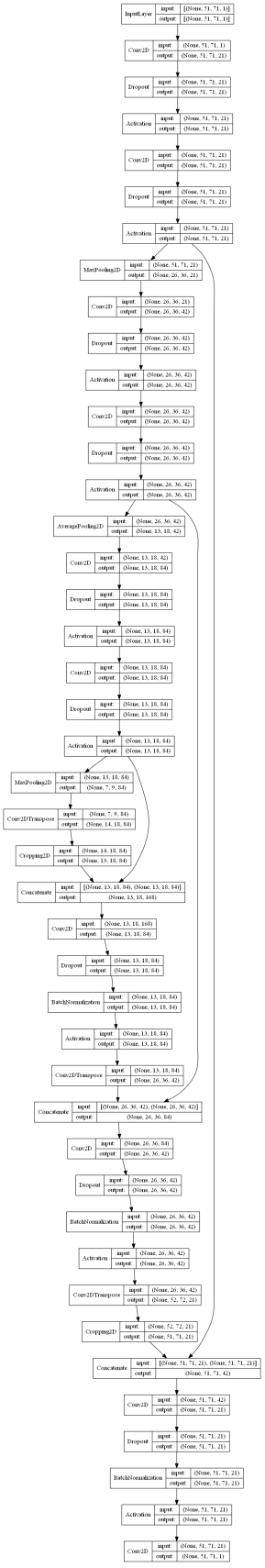
Modo 5


InputLayer | input: [(None, 51, 71, 1)] | output: [(None, 51, 71, 1)]
Conv2D | input: (None, 51, 71, 1) | output: (None, 51, 71, 12)
Conv2D | input: (None, 51, 71, 12) | output: (None, 51, 71, 24)
Conv2D | input: (None, 51, 71, 1) | output: (None, 51, 71, 1)
Conv2D | input: (None, 51, 71, 24) | output: (None, 51, 71, 36)
Concatenate | input: [(None, 51, 71, 12), (None, 51, 71, 24), (None, 51, 71, 36)] | output: (None, 51, 71, 72)
Add | input: [(None, 51, 71, 72), (None, 51, 71, 1)] | output: (None, 51, 71, 72)
AveragePooling2D | input: (None, 51, 71, 72) | output: (None, 26, 36, 72)
Conv2D | input: (None, 51, 71, 72) | output: (None, 51, 71, 1)
Conv2D | input: (None, 51, 71, 72) | output: (None, 51, 71, 1)
Conv2D | input: (None, 26, 36, 72) | output: (None, 26, 36, 24)
Add | input: [(None, 51, 71, 1), (None, 51, 71, 1)] | output: (None, 51, 71, 1)
Conv2D | input: (None, 26, 36, 24) | output: (None, 26, 36, 48)
Conv2D | input: (None, 51, 71, 1) | output: (None, 51, 71, 1)
Conv2D | input: (None, 51, 71, 1) | output: (None, 51, 71, 1)
Conv2D | input: (None, 26, 36, 48) | output: (None, 26, 36, 72)
Conv2D | input: (None, 26, 36, 72) | output: (None, 26, 36, 1)
Add | input: [(None, 51, 71, 1), (None, 51, 71, 1)] | output: (None, 51, 71, 1)
Concatenate | input: [(None, 26, 36, 24), (None, 26, 36, 48), (None, 26, 36, 72)] | output: (None, 26, 36, 144)
Conv2D | input: (None, 51, 71, 1) | output: (None, 51, 71, 1)
Conv2D | input: (None, 51, 71, 1) | output: (None, 51, 71, 1)
Add | input: [(None, 26, 36, 144), (None, 26, 36, 1)] | output: (None, 26, 36, 144)
Add | input: [(None, 51, 71, 1), (None, 51, 71, 1)] | output: (None, 51, 71, 1)
Conv2D | input: (None, 26, 36, 144) | output: (None, 26, 36, 1)
Conv2D | input: (None, 26, 36, 144) | output: (None, 26, 36, 1)
Add | input: [(None, 26, 36, 1), (None, 26, 36, 1)] | output: (None, 26, 36, 1)
MaxPooling2D | input: (None, 26, 36, 144) | output: (None, 13, 18, 144)
Conv2D | input: (None, 26, 36, 1) | output: (None, 26, 36, 1)
Conv2D | input: (None, 26, 36, 1) | output: (None, 26, 36, 1)
Add | input: [(None, 26, 36, 1), (None, 26, 36, 1)] | output: (None, 26, 36, 1)
Conv2DTranspose | input: (None, 13, 18, 144) | output: (None, 26, 36, 143)
Concatenate | input: [(None, 26, 36, 143), (None, 26, 36, 1)] | output: (None, 26, 36, 144)
Conv2D | input: (None, 26, 36, 144) | output: (None, 26, 36, 24)
Conv2D | input: (None, 26, 36, 24) | output: (None, 26, 36, 48)
Conv2D | input: (None, 26, 36, 144) | output: (None, 26, 36, 1)
Conv2D | input: (None, 26, 36, 48) | output: (None, 26, 36, 72)
Concatenate | input: [(None, 26, 36, 24), (None, 26, 36, 48), (None, 26, 36, 72)] | output: (None, 26, 36, 144)
Add | input: [(None, 26, 36, 144), (None, 26, 36, 1)] | output: (None, 26, 36, 144)
Conv2DTranspose | input: (None, 26, 36, 144) | output: (None, 52, 72, 71)
Cropping2D | input: (None, 52, 72, 71) | output: (None, 51, 71, 71)
Concatenate | input: [(None, 51, 71, 71), (None, 51, 71, 1)] | output: (None, 51, 71, 72)
Conv2D | input: (None, 51, 71, 72) | output: (None, 51, 71, 12)
Conv2D | input: (None, 51, 71, 12) | output: (None, 51, 71, 24)
Conv2D | input: (None, 51, 71, 72) | output: (None, 51, 71, 1)
Conv2D | input: (None, 51, 71, 24) | output: (None, 51, 71, 36)
Concatenate | input: [(None, 51, 71, 12), (None, 51, 71, 24), (None, 51, 71, 36)] | output: (None, 51, 71, 72)
Add | input: [(None, 51, 71, 72), (None, 51, 71, 1)] | output: (None, 51, 71, 72)
Conv2D | input: (None, 51, 71, 72) | output: (None, 51, 71, 1)

Modo 6


InputLayer | input: [(None, 51, 71, 1)] | output: [(None, 51, 71, 1)]
Conv2D | input: (None, 51, 71, 1) | output: (None, 51, 71, 45)
Dropout | input: (None, 51, 71, 45) | output: (None, 51, 71, 45)
BatchNormalization | input: (None, 51, 71, 45) | output: (None, 51, 71, 45)
Activation | input: (None, 51, 71, 45) | output: (None, 51, 71, 45)
AveragePooling2D | input: (None, 51, 71, 45) | output: (None, 26, 36, 45)
Conv2D | input: (None, 26, 36, 45) | output: (None, 26, 36, 90)
Dropout | input: (None, 26, 36, 90) | output: (None, 26, 36, 90)
Activation | input: (None, 26, 36, 90) | output: (None, 26, 36, 90)
AveragePooling2D | input: (None, 26, 36, 90) | output: (None, 13, 18, 90)
Conv2D | input: (None, 13, 18, 90) | output: (None, 13, 18, 180)
Dropout | input: (None, 13, 18, 180) | output: (None, 13, 18, 180)
Activation | input: (None, 13, 18, 180) | output: (None, 13, 18, 180)
MaxPooling2D | input: (None, 13, 18, 180) | output: (None, 7, 9, 180)
Conv2DTranspose | input: (None, 7, 9, 180) | output: (None, 14, 18, 180)
Cropping2D | input: (None, 14, 18, 180) | output: (None, 13, 18, 180)
Concatenate | input: [(None, 13, 18, 180), (None, 13, 18, 180)] | output: (None, 13, 18, 360)
Conv2D | input: (None, 13, 18, 360) | output: (None, 13, 18, 180)
Dropout | input: (None, 13, 18, 180) | output: (None, 13, 18, 180)
BatchNormalization | input: (None, 13, 18, 180) | output: (None, 13, 18, 180)
Activation | input: (None, 13, 18, 180) | output: (None, 13, 18, 180)
Conv2DTranspose | input: (None, 13, 18, 180) | output: (None, 26, 36, 90)
Concatenate | input: [(None, 26, 36, 90), (None, 26, 36, 90)] | output: (None, 26, 36, 180)
Conv2D | input: (None, 26, 36, 180) | output: (None, 26, 36, 90)
Dropout | input: (None, 26, 36, 90) | output: (None, 26, 36, 90)
BatchNormalization | input: (None, 26, 36, 90) | output: (None, 26, 36, 90)
Activation | input: (None, 26, 36, 90) | output: (None, 26, 36, 90)
Conv2DTranspose | input: (None, 26, 36, 90) | output: (None, 52, 72, 45)
Cropping2D | input: (None, 52, 72, 45) | output: (None, 51, 71, 45)
Concatenate | input: [(None, 51, 71, 45), (None, 51, 71, 45)] | output: (None, 51, 71, 90)
Conv2D | input: (None, 51, 71, 90) | output: (None, 51, 71, 45)
Dropout | input: (None, 51, 71, 45) | output: (None, 51, 71, 45)
Activation | input: (None, 51, 71, 45) | output: (None, 51, 71, 45)
Conv2D | input: (None, 51, 71, 45) | output: (None, 51, 71, 1)

Modo 7


InputLayer
input: [(None, 51, 71, 1)]
output: [(None, 51, 71, 1)]
Conv2D
input: (None, 51, 71, 1)
output: (None, 51, 71, 21)
Dropout
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
Activation
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
Conv2D
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
Dropout
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
Activation
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
MaxPooling2D
input: (None, 51, 71, 21)
output: (None, 26, 36, 21)
Conv2D
input: (None, 26, 36, 21)
output: (None, 26, 36, 42)
Dropout
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
Activation
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
Conv2D
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
Dropout
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
Activation
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
AveragePooling2D
input: (None, 26, 36, 42)
output: (None, 13, 18, 42)
Conv2D
input: (None, 13, 18, 42)
output: (None, 13, 18, 84)
Dropout
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
Activation
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
Conv2D
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
Dropout
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
Activation
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
MaxPooling2D
input: (None, 13, 18, 84)
output: (None, 7, 9, 84)
Conv2DTranspose
input: (None, 7, 9, 84)
output: (None, 14, 18, 84)
Cropping2D
input: (None, 14, 18, 84)
output: (None, 13, 18, 84)
Concatenate
input: [(None, 13, 18, 84), (None, 13, 18, 84)]
output: (None, 13, 18, 168)
Conv2D
input: (None, 13, 18, 168)
output: (None, 13, 18, 84)
Dropout
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
BatchNormalization
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
Activation
input: (None, 13, 18, 84)
output: (None, 13, 18, 84)
Conv2DTranspose
input: (None, 13, 18, 84)
output: (None, 26, 36, 42)
Concatenate
input: [(None, 26, 36, 42), (None, 26, 36, 42)]
output: (None, 26, 36, 84)
Conv2D
input: (None, 26, 36, 84)
output: (None, 26, 36, 42)
Dropout
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
BatchNormalization
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
Activation
input: (None, 26, 36, 42)
output: (None, 26, 36, 42)
Conv2DTranspose
input: (None, 26, 36, 42)
output: (None, 52, 72, 21)
Cropping2D
input: (None, 52, 72, 21)
output: (None, 51, 71, 21)
Concatenate
input: [(None, 51, 71, 21), (None, 51, 71, 21)]
output: (None, 51, 71, 42)
Conv2D
input: (None, 51, 71, 42)
output: (None, 51, 71, 21)
Dropout
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
BatchNormalization
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
Activation
input: (None, 51, 71, 21)
output: (None, 51, 71, 21)
Conv2D
input: (None, 51, 71, 21)
output: (None, 51, 71, 1)

A.4 Matrices de CrossMAC

Placa 0

		Modos Experimentales												
		1	2	3	4	5	6	7	8	9	10	11	12	13
Modos Referencia	1	0.8892	0.0008	0.0017	0.0013	0.0004	0.0010	0.0065	0.0029	0.0603	0.0001	0.0408	0.0001	0.0057
	2	0.0217	0.0033	0.0951	0.0474	0.0069	0.0021	0.0303	0.0183	0.0003	0.0021	0.0001	0.0027	0.0406
	3	0.0051	0.9410	0.0298	0.0111	0.0028	0.0001	0.0000	0.0001	0.0007	0.0129	0.0302	0.0003	0.0004
	4	0.0105	0.0029	0.5189	0.5834	0.0533	0.0013	0.0329	0.0124	0.0022	0.0510	0.0049	0.0157	0.0001
	5	0.0001	0.0006	0.0747	0.0061	0.0652	0.1419	0.0062	0.0047	0.0002	0.0010	0.0013	0.0037	0.0090
	6	0.0000	0.0008	0.0516	0.0280	0.2402	0.4958	0.0119	0.0176	0.0074	0.0260	0.0034	0.0027	0.0086
	7	0.0047	0.0063	0.0115	0.0082	0.0069	0.0060	0.0001	0.0002	0.0009	0.0003	0.0058	0.0712	0.0208

Placa 1

		Modos Experimentales											
		1	2	3	4	5	6	7	8	9	10	11	12
Modos Referencia	1	0.2234	0.0000	0.0017	0.0054	0.0021	0.0014	0.0073	0.0207	0.0012	0.0000	0.0006	0.0011
	2	0.5431	0.9672	0.3064	0.0001	0.0117	0.0337	0.0262	0.0156	0.0434	0.0404	0.0253	0.0466
	3	0.0019	0.0008	0.0456	0.0546	0.0008	0.0000	0.0044	0.0105	0.0006	0.0013	0.0242	0.0132
	4	0.0967	0.0227	0.4636	0.7708	0.1420	0.4714	0.4208	0.0331	0.0573	0.1649	0.0517	0.0630
	5	0.0000	0.0012	0.0116	0.0273	0.0172	0.0765	0.0089	0.0014	0.0007	0.0122	0.0056	0.0021
	6	0.0009	0.0016	0.0005	0.0464	0.1015	0.0665	0.1756	0.0566	0.0002	0.0002	0.0692	0.0006
	7	0.0004	0.0001	0.0086	0.0012	0.0078	0.0152	0.0012	0.3833	0.0091	0.0065	0.0008	0.0058

Placa 2

		Modos Experimentales							
		1	2	3	4	5	6	7	8
Modos Referencia	1	0.0004	0.4801	0.0012	0.0000	0.0144	0.0000	0.0002	0.0110
	2	0.6569	0.2456	0.0212	0.0346	0.0009	0.0484	0.0009	0.0505
	3	0.0006	0.0008	0.2501	0.0000	0.0007	0.0003	0.0019	0.0009
	4	0.0264	0.0509	0.4540	0.2669	0.0407	0.0591	0.0335	0.0629
	5	0.0033	0.0003	0.0000	0.0127	0.0164	0.0004	0.0063	0.0033
	6	0.0000	0.0002	0.0022	0.0644	0.1185	0.0039	0.0036	0.0046
	7	0.0000	0.0019	0.0001	0.0017	0.1464	0.0213	0.0000	0.0001

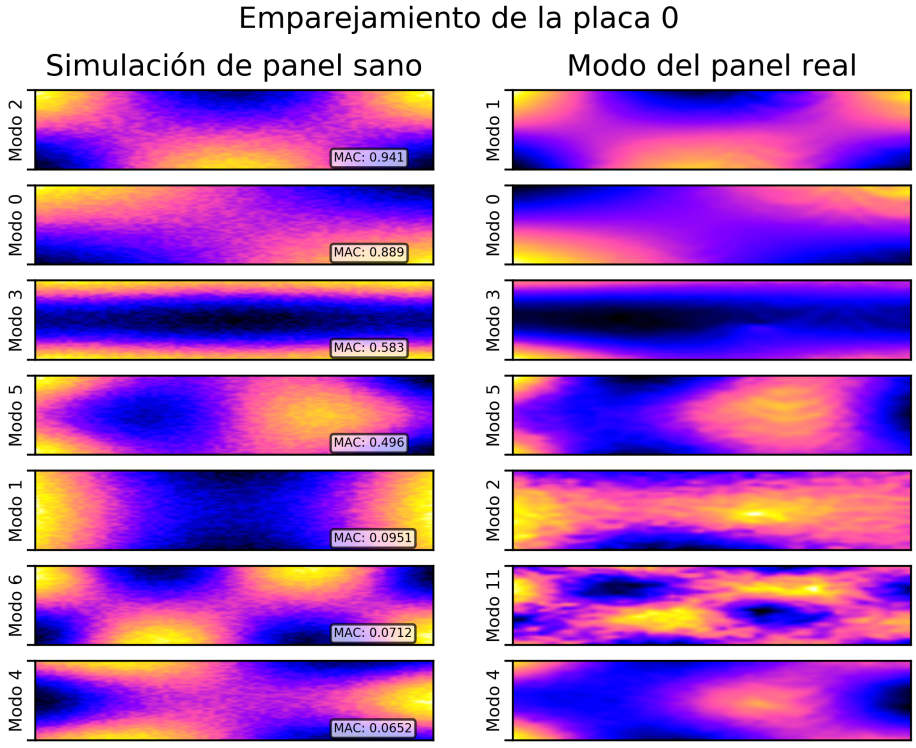
Placa 3

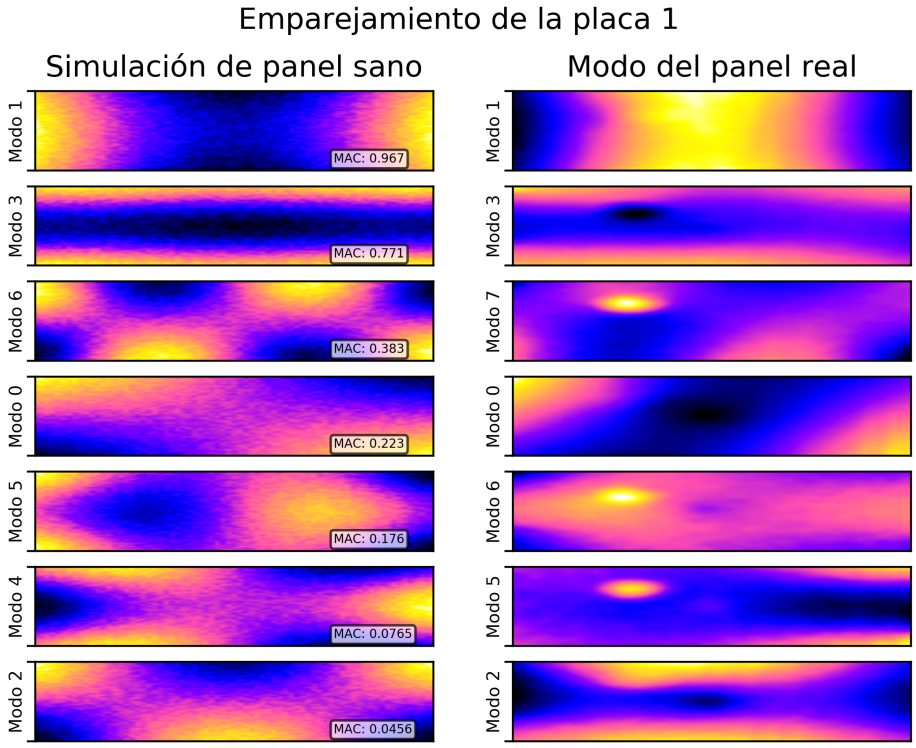
		Modos Experimentales							
		1	2	3	4	5	6	7	8
Modos Referencia	1	0.2328	0.0012	0.0114	0.0003	0.0007	0.0001	0.0001	0.0004
	2	0.5851	0.8775	0.2000	0.0289	0.0187	0.0383	0.0050	0.0205
	3	0.0001	0.0001	0.3462	0.0031	0.0017	0.0005	0.0037	0.0000
	4	0.0741	0.0182	0.2004	0.1038	0.0006	0.0788	0.0061	0.0508
	5	0.0002	0.0008	0.0005	0.0062	0.0237	0.0002	0.0068	0.0026
	6	0.0032	0.0002	0.0100	0.0010	0.0273	0.0002	0.0036	0.0042
	7	0.0002	0.0000	0.0119	0.0097	0.0941	0.0645	0.0000	0.0016

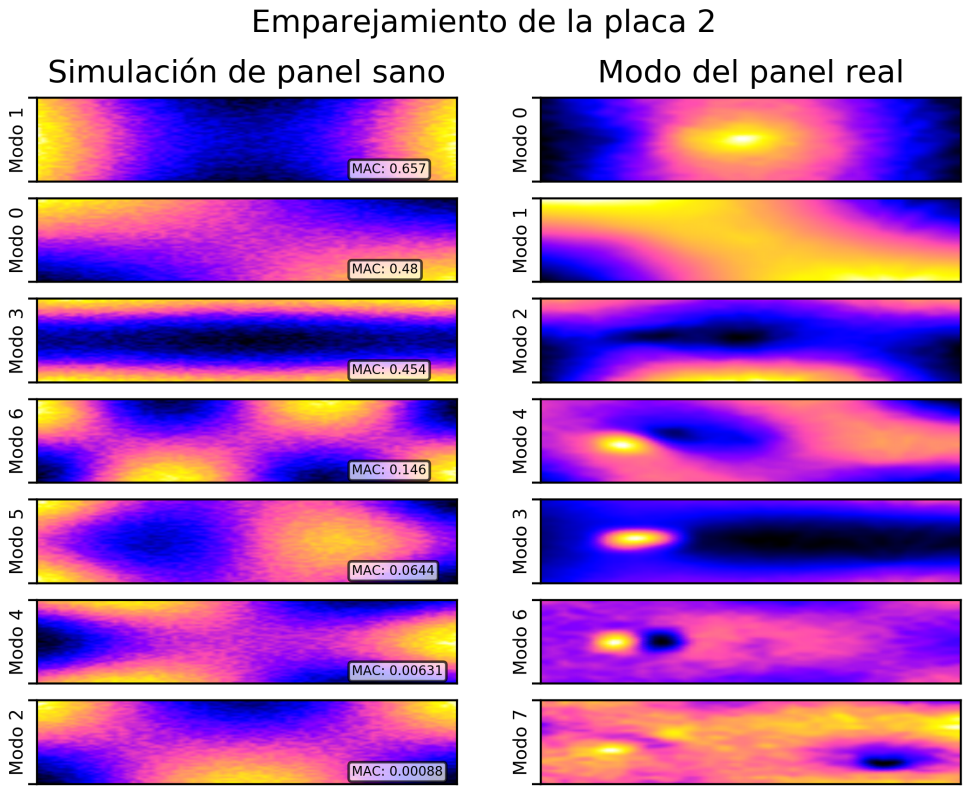
Placa 4

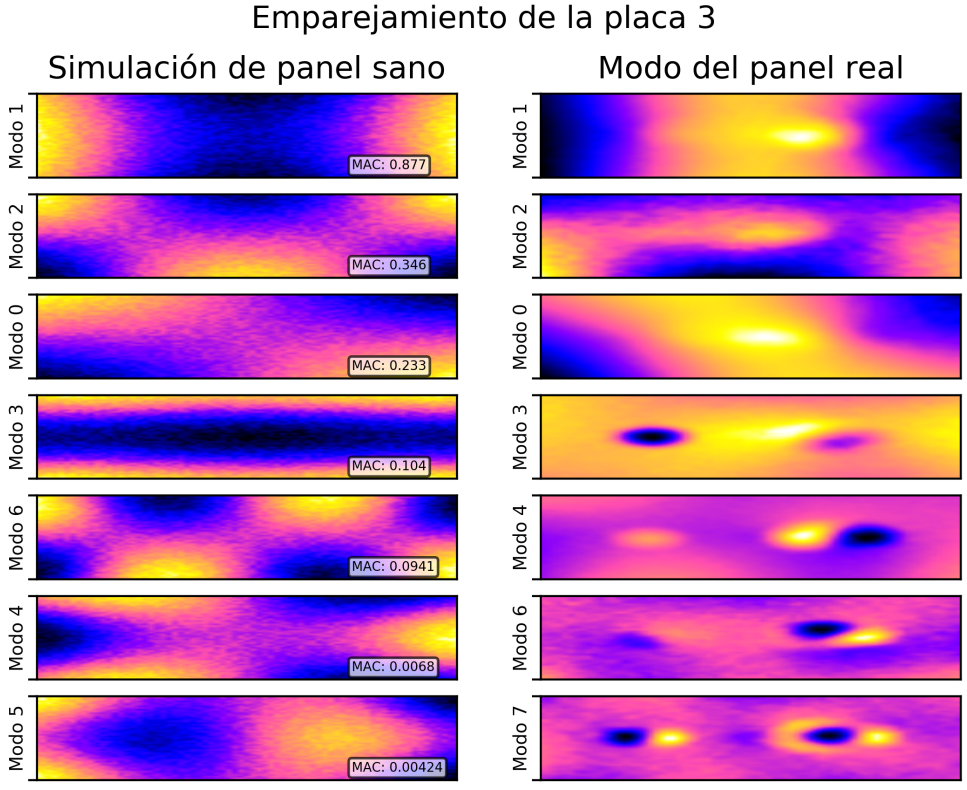
		Modos Experimentales				
		1	2	3	4	5
Modos Referencia	1	0.6574	0.0027	0.0000	0.0041	0.0000
	2	0.0730	0.9488	0.0283	0.0103	0.0263
	3	0.0008	0.0000	0.6784	0.0155	0.0001
	4	0.0366	0.0227	0.0078	0.4241	0.0000
	5	0.0006	0.0002	0.0202	0.0550	0.0113
	6	0.0020	0.0000	0.0680	0.0637	0.0103
	7	0.0000	0.0000	0.0097	0.0006	0.0000

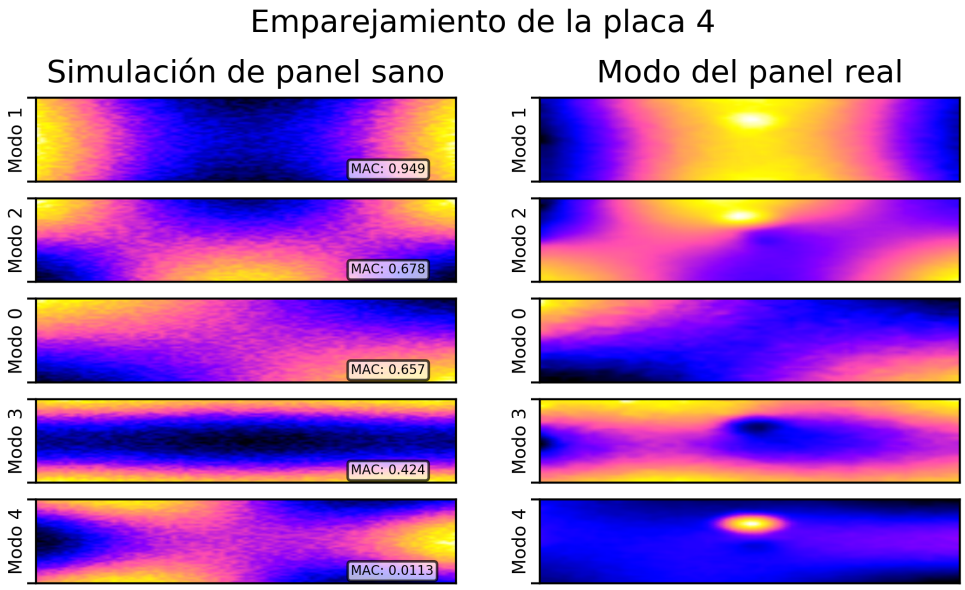
A.5 Pairing de datos experimentales






Emparejamiento de la placa 2
Simulación de panel sano
Modo del panel real
Modo 1
MAC: 0.657
Modo 0
MAC: 0.48
Modo 3
MAC: 0.454
Modo 6
MAC: 0.146
Modo 5
MAC: 0.0644
Modo 4
MAC: 0.00631
Modo 2
MAC: 0.00088
Modo 0
Modo 1
Modo 2
Modo 4
Modo 3
Modo 6
Modo 7


Emparejamiento de la placa 3
Simulación de panel sano
Modo del panel real
Modo 1
MAC: 0.877
Modo 2
MAC: 0.346
Modo 0
MAC: 0.233
Modo 3
MAC: 0.104
Modo 6
MAC: 0.0941
Modo 4
MAC: 0.0068
Modo 5
MAC: 0.00424
Modo 1
Modo 2
Modo 0
Modo 3
Modo 4
Modo 6
Modo 7


Emparejamiento de la placa 4
Simulación de panel sano
Modo del panel real
Modo 1
Modo 2
Modo 0
Modo 3
Modo 4
MAC: 0.949
MAC: 0.678
MAC: 0.657
MAC: 0.424
MAC: 0.0113