



VNiVERSiDAD
DE SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

FACULTAD DE CIENCIAS

CARACTERIZACIÓN DE HACES LÁSER MEDIANTE INTELIGENCIA ARTIFICIAL

Trabajo de Fin de Grado

Autor:

Avril González González

Tutores:

Enrique Conejero Jarque

Francisco Javier Serrano Rodríguez

Curso 2024-2025

Certificado de los tutores TFG Grado en Física

Enrique Conejero Jarque

Francisco Javier Serrano Rodríguez

HACEN CONSTAR:

Que el trabajo titulado “*Caracterización de haces láser mediante inteligencia artificial*” que se presenta, ha sido realizado por la alumna Avril González González, con DNI 21085391A y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Físicas en esta Universidad.

Salamanca, a fecha de firma electrónica.

Agradecimientos:

A mis padres, por innumerables motivos, pero sobre todo por actuar como refugio cuando las cosas no salen como una quiere.

A mi familia, por celebrar mis éxitos como si fueran tuyos.

A mis amigos, por ser ese respiro en los momentos alegres y en los que no lo fueron tanto.

A Patricia, por enseñarme a relativizar y a otras muchas cosas.

A mis tutores, Enrique y Javier, que además de estar siempre dispuestos a ayudarme, han hecho que trabajar y aprender con ellos sea un verdadero honor. Quisiera expresar un reconocimiento especial a Enrique, quien en sus clases de Óptica I logró hacer de este campo de la física mi favorito.

Abstract

In the theoretical framework of paraxial optics, Hermite-Gaussian (HG) transverse modes are defined as one of the solutions to the paraxial Helmholtz equation, which describes the propagation of Gaussian beams in homogeneous media. These modes form a complete set of orthogonal functions characterized by their modal indices (l, m) , which determine the spatial profile of the beam in the transverse directions. In the case of a composite beam, that is, a superposition of several modes, it is necessary to specify not only the modal indices but also the relative weights and phases of each of the modes present in the combination.

The ability to manipulate these superpositions is of particular interest in the context of free-space optical communications. By encoding information in the spatial patterns formed by composite beams, it is possible to significantly increase the capacity of the transmission channel. This technique of modal multiplexing¹ could represent an efficient alternative to traditional methods, especially in applications that require high transmission rates or robust systems against interference and attenuation.

This work focuses on the characterization of laser beams using artificial intelligence techniques, with the aim of decomposing an unknown intensity profile in terms of the Hermite-Gaussian mode basis. To this end, an extensive campaign of numerical simulations in *Python* has been carried out, generating a synthetic database consisting of various arbitrary combinations of HG modes with indices $l, m \leq 5$. Specifically, profiles corresponding to superpositions of two simple modes, two modes with relative phase shift, two modes under different levels of Gaussian noise, and three simple modes have been simulated. These profiles have been properly normalized to ensure the consistency of the database.

Based on this dataset, neural network models with convolutional architectures, widely used in image processing, have been trained. These networks have been designed to automatically extract the relevant spatial features of the intensity profiles and associate them with the corresponding labels. The implemented architecture combines convolutional and pooling layers, followed by dense layers that allow the model to learn higher-level representations. The design of the loss function and the network outputs has been conceived to adapt to the particularities of the problem, facilitating the identification of the modes present and the estimation of the parameters that characterize them.

The results obtained show that the proposed deep learning models are capable of identifying the modal composition of the profiles with high accuracy, achieving success rates above 96 % in all the cases studied. In the simplest scenario —the superposition of two simple modes without phase shift or noise— the network achieves an accuracy of 98,76 %. Furthermore, the robustness of the system has been verified under adverse conditions such as relative phase shift or the presence of noise, supporting the viability of this methodology for its application in experimental or industrial environments where ideal conditions are not always guaranteed.

This work lays the foundations for the development of automatic laser beam characterization tools based on neural networks, which could have a significant impact in areas such as optical system alignment, real-time beam monitoring, or the implementation of advanced optical communication systems. Future prospects include extending the model

¹It is a technique that allows several signals or information flows to be sent through the same physical or logical communication channel, either simultaneously or alternately, making better use of the channel and increasing its capacity

to higher-order modes, the inclusion of Laguerre-Gaussian beams, and the experimental validation of the results obtained in simulation.

Resumen

En el marco teórico de la óptica paraxial, los modos transversales Hermite-Gauss (HG) se definen como una de las soluciones de la ecuación de Helmholtz paraxial, la cual describe la propagación de haces gaussianos en medios homogéneos. Estos modos constituyen un conjunto completo de funciones ortogonales caracterizadas por sus índices modales (l, m), que determinan el perfil espacial del haz en las direcciones transversales. En el caso de un haz compuesto, es decir, una superposición de varios modos, además de los índices modales, es necesario especificar los pesos relativos y las fases de cada uno de los modos presentes en la combinación.

La capacidad de manipular estas superposiciones resulta de especial interés en el contexto de las comunicaciones ópticas en espacio libre. Mediante la codificación de la información en los patrones espaciales que forman los haces compuestos, es posible incrementar significativamente la capacidad del canal de transmisión. Esta técnica de multiplexación² modal podría constituir una alternativa eficiente frente a los métodos tradicionales, especialmente en aplicaciones que requieran altas tasas de transmisión o sistemas robustos frente a interferencias y atenuaciones.

El trabajo presentado se centra en la caracterización de haces láser mediante técnicas de inteligencia artificial, con el objetivo de descomponer un perfil de intensidad desconocido en términos de la base de modos Hermite-Gauss. Para ello, se ha llevado a cabo un extenso conjunto de simulaciones numéricas en *Python*, generando una base de datos sintética compuesta por diversas combinaciones arbitrarias de modos HG con índices $l, m \leq 5$. En concreto, se han simulado perfiles correspondientes a superposiciones de dos modos simples, dos modos con desfase relativo, dos modos bajo diferentes niveles de ruido gaussiano y tres modos simples. Estos perfiles han sido normalizados adecuadamente para garantizar la consistencia de la base de datos.

A partir de este conjunto de datos, se han entrenado modelos de redes neuronales basados en arquitecturas convolucionales, ampliamente utilizadas en el procesamiento de imágenes. Estas redes han sido diseñadas para extraer de forma automática las características espaciales relevantes de los perfiles de intensidad, y para asociarlas con las etiquetas correspondientes. La arquitectura implementada combina capas de convolución y de agrupamiento, seguidas de capas densas que permiten al modelo aprender representaciones de mayor nivel de abstracción. El diseño de la función de pérdida y de las salidas de la red se ha planteado para adaptarse a las particularidades del problema, facilitando la identificación de los modos presentes y la estimación de los parámetros que los caracterizan.

Los resultados obtenidos muestran que los modelos de aprendizaje profundo propuestos son capaces de identificar con alta precisión la composición modal de los perfiles, alcanzando tasas de acierto superiores al 96 % en todos los casos estudiados. En el escenario más sencillo —superposición de dos modos simples sin desfase ni ruido—, la red logra un acierto del 98,76 %. Además, se ha comprobado la robustez del sistema frente a condiciones adversas como el desfase relativo o la presencia de ruido, lo que respalda la viabilidad de esta metodología para su aplicación en entornos experimentales o industriales donde las condiciones ideales no siempre están garantizadas.

Este trabajo sienta las bases para el desarrollo de herramientas automáticas de ca-

²Es una técnica que permite enviar varias señales o flujos de información a través de un mismo canal físico o lógico de comunicación, de forma simultánea o alternada, aprovechando mejor el canal y aumentando su capacidad

racterización de haces láser mediante redes neuronales, lo que podría tener un impacto notable en áreas como el alineamiento de sistemas ópticos, la monitorización de haces en tiempo real o la implementación de sistemas avanzados de comunicación óptica. Las perspectivas futuras incluyen la extensión del modelo a modos de orden superior, la inclusión de haces Laguerre-Gaussianos, y la validación experimental de los resultados obtenidos en simulación.

Palabras clave

Haz gaussiano, paraxial, modos Hermite-Gauss, intensidad, redes neuronales convolucionales (CNN), dataset, decodificación.

Keywords

Gaussian beam, paraxial, Hermite-Gaussian modes, intensity, convolutional neural networks (CNN), dataset, decoding

Índice

1. Introducción.	10
2. Preludio y contexto histórico.	11
2.1. Láseres	11
2.2. Redes neuronales	11
3. Ondas paraxiales. Ecuación paraxial de Helmholtz.	13
3.1. La Ecuación de Helmholtz Paraxial	14
4. Haces láser	15
4.1. Haz Gaussiano	16
4.1.1. Propiedades	17
4.1.2. Parámetros que caracterizan un haz gaussiano	23
4.2. Haces Hermite-Gaussianos (HG)	23
5. Redes neuronales	27
5.1. Redes neuronales de una capa	27
5.2. Red multi-clase de una capa	29
5.3. Perceptrón multicapa	31
5.4. El paradigma del Deep-Learning	34
5.5. Redes neuronales convolucionales (CNN)	36
6. Red neuronal para decodificar una superposición	41
6.1. Simulación	41
6.1.1. Superposición de dos modos simples	41
6.1.2. Superposición de dos modos desfasados	43
6.1.3. Superposición de dos modos con ruido gaussiano	43
6.1.4. Superposición de más de dos modos	45
6.2. Redes empleadas para cada tipo de superposición	46
6.2.1. Superposiciones de dos modos	46
6.2.2. Superposición de más de dos modos	48
6.3. Resultados	51
6.3.1. Superposición de dos modos simples	51
6.3.2. Superposición de dos modos desfasados	52
6.3.3. Superposición de dos modos con ruido	53
6.3.4. Superposición de más de dos modos	57

7. Discusión de resultados	58
8. Conclusiones	60
Referencias	62

1. Introducción.

El estudio de la estructura espacial de los haces láser es uno de los campos de la óptica que ha atraído más interés en los últimos años debido a que dicha estructura puede abarcar un sinfín de configuraciones de intensidad, fase y polarización que afectan a su propagación y dan lugar a aplicaciones de gran interés.

En los casos más sencillos, los haces son monomodo (o prácticamente monomodo). En el siguiente nivel de complejidad se encuentran conformados por la superposición de unos pocos modos gaussianos [1] que generan un determinado perfil de intensidad. Extraer la composición de cada uno de los modos a partir de ese perfil no es una tarea trivial.

Aprovechando que en los últimos años se ha extendido el uso de técnicas de inteligencia artificial en la caracterización de propiedades de haces láser, particularmente la de haces con índices radiales altos [2] y con momento angular orbital [3], en este trabajo proponemos emplear estas técnicas para caracterizar haces formados por distintas combinaciones de modos de orden bajo [4]. Se trata de un trabajo teórico con una parte inicial bibliográfica y otra de cálculo numérico. La parte de cálculo numérico se realizará con Python y las librerías Keras/Tensorflow, usadas habitualmente en técnicas de inteligencia artificial.

Se parte del marco teórico de la óptica paraxial, en el que se definen los modos transversales como soluciones del modelo de propagación de haces gaussiano. A través de simulaciones numéricas se genera un conjunto de datos sintéticos que representan haces formados por combinaciones arbitrarias de modos HG, incluyendo efectos como el desfase relativo, el ruido gaussiano y la interferencia entre modos.

Una vez construida la base de datos, se implementan y entrena nodelos de redes neuronales profundas —concretamente redes convolucionales (CNN)— con el objetivo de identificar qué modos están presentes en un perfil dado. La red es entrenada como un clasificador multietiqueta capaz de detectar múltiples modos simultáneamente, y se analiza su rendimiento frente a diferentes configuraciones de superposición, niveles de ruido, y fases relativas.

El trabajo concluye exponiendo las principales conclusiones obtenidas de este estudio así como las interesantes perspectivas que estas plantean para aplicaciones futuras.

2. Preludio y contexto histórico.

2.1. Láseres

Cabe comenzar definiendo formalmente el láser -acrónimo del inglés "*Light Amplification by stimulated Emission of Radiation*"[5]- como dispositivo de amplificación óptica que, mediante emisión estimulada y retroalimentación en cavidad, produce un haz electromagnético monocromático, coherente, colimado y sustancialmente direccional [1].

En general, un láser posee tres elementos fundamentales: el medio activo, que es el encargado de generar la inversión de población de los átomos, permitiendo la amplificación de la luz; el elemento de bombeo, que suministra la energía necesaria para excitar los átomos del medio activo; y finalmente, la cavidad resonante, que proporciona un recorrido para la luz, facilitando el proceso de emisión estimulada y la producción de radiación coherente.

La historia del láser [6] comienza a principios del siglo XX. Es Albert Einstein quien en 1917 sienta las bases para la tecnología láser cuando predice el fenómeno de "*emisión estimulada*"[7], que es fundamental para el funcionamiento de todos los láseres. Valentin Fabrikant en 1939 teoriza el uso de la emisión estimulada para amplificar la radiación [8]. Más tarde, en 1950, Charles Townes, Nikolay Basov y Alexander Prokhorov desarrollan la teoría cuántica de emisión estimulada y demuestran la emisión estimulada en las microondas. Recibirían más adelante el Premio Nobel en física por este trabajo de vanguardia. En 1960, Gordon Gould, graduado de la Universidad Columbia, propone que la emisión estimulada se puede usar para amplificar la luz. Describe un resonador óptico que puede crear un haz estrecho de luz coherente y lo llama LASER [9]. Es en 1960 cuando Theodore Maiman [10] construye el primer prototipo de trabajo de un láser en Hughes Research Laboratories en Malibu, California [11]. La primera aplicación para el láser de rubí fue en telemetría militar, y aún se usa comercialmente para perforar orificios en diamantes debido a su alta potencia pico. Después del láser de rubí se construyeron muchos otros tipos de láseres, cada uno de los cuales tiene sus características peculiares y se han utilizado para diferentes aplicaciones. Como ejemplo de uno de los primeros estudios sobre la estructura espacial de los haces láser, particularmente de los haces gaussianos, cabe destacar el artículo clásico de Kogelnik y Li [12].

2.2. Redes neuronales

A lo largo de esta subsección y de la sección 5 se ha tomado como referencia y principal fuente bibliográfica el libro '*Introducción a la Inteligencia Artificial. Aprendizaje Automático y Redes Neuronales*' de Daniel López Sánchez, lamentablemente no publicado de forma oficial [13].

Las neuronas artificiales y las redes compuestas por estas se inspiran en el funcionamiento biológico: en una red neuronal artificial hay neuronas o nodos (módulos de software) que trabajan en conjunto para la resolución de problemas. La arquitectura elemental de una red neuronal contempla una interconexión entre tres tipos de capas distintos: de entrada, capas ocultas y de salida. El ingreso de información se realiza en la capa de entrada, donde hay un procesamiento y una clasificación de los contenidos antes de su envío a la siguiente capa: la capa oculta. En realidad, cada red neuronal puede contar con múltiples capas ocultas, que avanzan con el análisis de los datos. Finalmente la información llega a la capa de salida, que ofrece el resultado final (Figura 1).

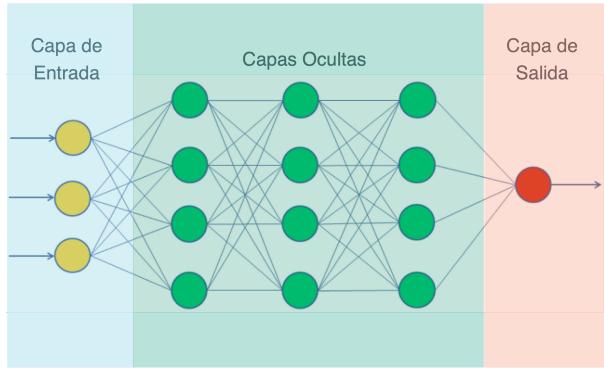


Figura 1: Esquema de una red neuronal artificial [14]

La historia de las redes neuronales artificiales puede dividirse en cinco etapas diferenciadas:

- **Primeros modelos de redes neuronales artificiales (1940-1970)**

Durante este periodo histórico se desarrollaron los primeros intentos de construir modelos computacionales inspirados en el funcionamiento de las redes neuronales biológicas. Entre ellos destaca la propuesta inicial de McCulloch y Pitts [15], quienes presentaron un modelo de neurona artificial con capacidad para identificar patrones simples. Sin embargo, este modelo presentaba limitaciones importantes, como la necesidad de que ciertos parámetros fueran definidos manualmente por el usuario.

Posteriormente, Frank Rosenblatt introdujo el *Perceptrón* [16], considerado el primer modelo de red neuronal capaz de ajustar de forma automática sus parámetros (pesos sinápticos) a partir de un conjunto de ejemplos de entrenamiento, eliminando así la dependencia de una configuración manual. Poco tiempo después surgió el modelo conocido como ADALINE (*adaptive linear element*), que supuso un nuevo avance en este campo.

- **El invierno de las redes neuronales (1970-1986)**

El auge inicial del perceptrón y del modelo ADALINE despertó el interés de numerosos investigadores por estudiar las limitaciones que presentaban estos primeros sistemas de inteligencia artificial (IA). Entre las contribuciones más relevantes de aquella época destaca la obra *Perceptrons* de Marvin Minsky, considerado uno de los principales referentes en el ámbito de la IA, en la que examinaba las capacidades computacionales de este tipo de redes [17]. Sus conclusiones fueron decepcionantes, ya que puso de manifiesto que el perceptrón no podía resolver problemas básicos como el de la función lógica XOR. Este tipo de resultados desfavorables provocaron un estancamiento en el desarrollo y la investigación de las redes neuronales durante ese periodo.

- **Segunda ola de investigación de las redes neuronales (1986-1995)**

El principal hito de esta segunda ola de investigación de las ANN (*artificial neural network*) fue el exitoso uso del algoritmo *back-propagation* para entrenar modelos de red neuronal más complejos con varias capas de neuronas [18].

- **Segundo invierno de las redes neuronales (1995-2006)**

La incapacidad de las redes neuronales durante este periodo para responder a las elevadas expectativas de la segunda ola dio lugar a este segundo invierno.

- **Tercera ola de las redes neuronales: el Deep Learning (2006-Actualidad)**

En 2006 Geoffrey Hinton publica un artículo [19] en el cual explica cómo había logrado entrenar una red neuronal profunda utilizando la estrategia *"layer-wise pre-training"*³. La investigación de las redes neuronales profundas desde entonces ha recibido una atención sin precedentes entre los investigadores del campo de la IA (Figura 2).

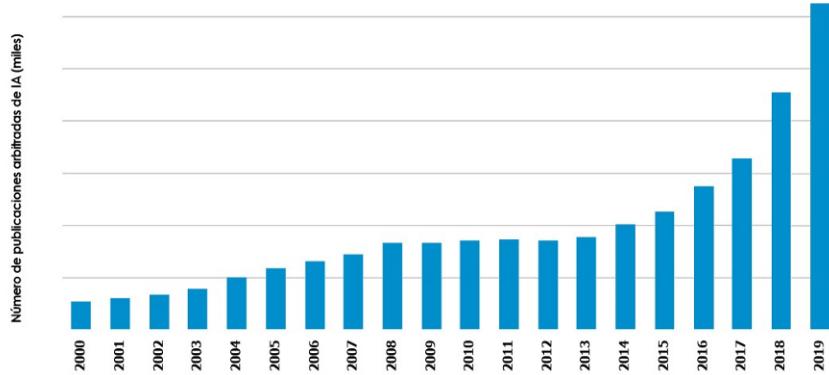


Figura 2: Número de publicaciones arbitradas de IA 2000-2019 [20]

3. Ondas paraxiales. Ecuación paraxial de Helmholtz.

A lo largo de esta sección y de la siguiente se ha tomado como referencia y principal fuente bibliográfica el libro *"Fundamentals of Photonics"* de Saleh y Teich [21].

Una onda se considera paraxial si sus direcciones normales a los frentes de onda son rayos paraxiales (i.e. que forman pequeños ángulos con el eje de propagación, habitualmente denotado en óptica como eje z). Una forma de construir una onda paraxial es empezar con una onda plana (i.e. una onda cuya dependencia espacial es $A(z) \exp(-jkz)$, donde j es la unidad compleja y k el módulo del vector de ondas). Esta onda puede verse como el producto de dos ondas: $\exp(-jkz)$ a la que se hará referencia en lo siguiente como *onda portadora*, esta onda va a verse modificada por $A(z)$, que será llamada *envolvente* y será en general una función compleja. A tendrá que ser una función lentamente variable (a lo largo del eje z) dependiente de la posición ($A(\mathbf{r})$) para que la amplitud compleja de la onda modulada sea:

$$U(\mathbf{r}) = A(\mathbf{r}) \exp(-jkz) \quad (1)$$

La variación de $A(\mathbf{r})$ con la posición debe ser lenta dentro de la distancia de una longitud de onda $\lambda = 2\pi/k$, de modo que la onda mantenga aproximadamente su naturaleza subyacente de onda plana.

La función de onda en notación real: $u(\mathbf{r}, t) = |A(\mathbf{r})| \cos[2\pi\nu t - kz + \arg\{A(\mathbf{r})\}]$ de una onda paraxial se representa en la Figura 3 (a) como una función de z en $t = 0$ y $x = y = 0$. Esta es una función sinusoidal de z con amplitud $|A(0, 0, z)|$ y fase $\arg\{A(0, 0, z)\}$ que

³El *layer-wise pretraining* es una técnica de entrenamiento de redes neuronales profundas que consiste en entrenar cada capa de la red de manera secuencial y gradual, antes de ajustar todos los pesos conjuntamente, lo que facilita la optimización y mejora la convergencia.

varían lentamente con z . Dado que el cambio de la fase $\arg\{A(x, y, z)\}$ es pequeño dentro de la distancia de una longitud de onda, los frentes de onda planos, $kz = 2\pi q$, de la onda plana portadora se doblan solo ligeramente, de modo que sus normales son rayos paraxiales (ver figura 3 (b)).

3.1. La Ecuación de Helmholtz Paraxial

Para que la onda paraxial (1) satisfaga la ecuación de Helmholtz (2),

$$(\nabla^2 + k^2)U(\mathbf{r}) = 0 \quad (2)$$

la envolvente compleja $A(\mathbf{r})$ debe satisfacer otra ecuación diferencial parcial obtenida al sustituir (1) en (2). La suposición de que $A(\mathbf{r})$ varía lentamente con respecto a z significa que dentro de una distancia $\Delta z = \lambda$, el cambio ΔA es mucho menor que A mismo; es decir, $\Delta A \ll A$. Esta desigualdad de variables complejas se aplica a las magnitudes de las partes real e imaginaria por separado. Dado que $\Delta A = (\partial A / \partial z) \Delta z = (\partial A / \partial z) \lambda$, se sigue que $\partial A / \partial z \ll A / \lambda = Ak/2\pi$, y por lo tanto

$$\frac{\partial A}{\partial z} \ll kA \quad (3)$$

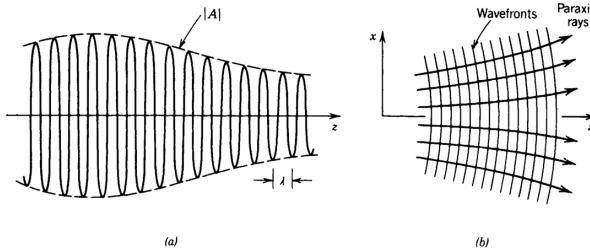


Figura 3: (a) La magnitud de una onda paraxial como función de la distancia axial z . (b) Los frentes de onda y las normales a los frentes de onda de una onda paraxial. [21]

De manera similar, la derivada $\partial A / \partial z$ varía lentamente dentro de la distancia λ , de modo que $\partial^2 A / \partial^2 z \ll k \partial A / \partial z$, y por lo tanto

$$\frac{\partial^2 A}{\partial z^2} \ll k^2 A \quad (4)$$

Sustituyendo (1) en (2) y despreciando $\partial^2 A / \partial^2 z$ en comparación con $k \partial A / \partial z$ o $k^2 A$, obtenemos

$$\nabla_T^2 A - j2k \frac{\partial A}{\partial z} = 0 \quad (5)$$

donde $\nabla_T^2 = \partial^2 / \partial x^2 + \partial^2 / \partial y^2$ es el operador laplaciano transversal. La ecuación (5) es la aproximación de envolvente lentamente variable de la ecuación de Helmholtz. Simplemente la llamaremos ecuación paraxial de Helmholtz. Es una ecuación diferencial en derivadas parciales que se asemeja a la ecuación de Schrödinger de la física cuántica.

La solución más simple de la ecuación paraxial de Helmholtz es la onda paraboloidal, que es la aproximación paraxial de la onda esférica. Sin embargo, la solución más interesante y útil es el haz gaussiano, al cual se dedica la sección 4.1.

4. Haces láser

Es lógico preguntarse si puede la luz estar espacialmente confinada y transportarse en el espacio libre sin dispersión angular. Aunque la naturaleza ondulatoria de la luz impide la existencia de tal idealización, la luz puede adoptar la forma de haces que se acercan lo más posible a ondas espacialmente localizadas y no divergentes.

Una onda plana y una onda esférica representan los dos extremos opuestos de confinamiento angular y espacial:

Las normales a los frentes de onda (rayos) de una onda plana son paralelos a la dirección de la onda, de modo que no hay dispersión angular, pero la energía se extiende espacialmente por todo el espacio. Por otro lado, la onda esférica, que se origina en un solo punto, tiene sus normales que divergen en todas las direcciones.

Las ondas con normales a los frentes de onda que forman pequeños ángulos con el eje z se llaman ondas paraxiales. Deben satisfacer la ecuación de Helmholtz paraxial (5) derivada en la sección 3.

Una solución importante de esta ecuación que exhibe las características de un haz óptico es una onda llamada **haz gaussiano**. Aunque se discutirán más en profundidad en la sección 4.1, las principales características de un haz gaussiano son:

- La potencia de este haz está principalmente concentrada dentro de un pequeño cilindro que rodea el eje del haz.
- La distribución de intensidad en cualquier plano transversal es una función gaussiana simétrica circularmente centrada sobre el eje del haz.
- El ancho de esta función es mínimo en la cintura del haz y crece gradualmente en ambas direcciones.
- Los frentes de onda son aproximadamente planos cerca de la cintura del haz, pero gradualmente se curvan y se convierten aproximadamente en esféricos lejos de la cintura.
- La divergencia angular de las normales a los frentes de onda es la mínima permitida por la ecuación de ondas para un ancho de haz dado. Por lo tanto, las normales a los frentes de onda son muy parecidas a un delgado lápiz de rayos. En condiciones ideales, la luz de un láser adopta la forma de un haz gaussiano.
- Una expresión para la amplitud compleja del haz gaussiano se deriva en la sección 4.1 y se proporciona una discusión detallada de sus propiedades físicas (intensidad, potencia, radio del haz, divergencia angular, profundidad de foco y fase).

En la sección 4.2 se introduce una familia de haces ópticos llamada haces Hermite-Gauss, de los cuales el haz gaussiano es un miembro. Los haces Hermite-Gauss no son la única solución a la ecuación de Helmholtz paraxial, existen otras soluciones como los haces Laguerre-Gauss e incluso soluciones a la ecuación de Helmholtz completa sin la aproximación paraxial como es el caso de los haces de Bessel. Sin embargo, en estos dos últimos tipos de haces mencionados no ahondaremos mucho puesto que no afectan al contenido del trabajo.

4.1. Haz Gaussiano

Amplitud Compleja

El concepto de ondas paraxiales fue introducido en la sección 3. La amplitud compleja venía dada por la ecuación (1). Para que esta satisficiera la ecuación de Helmholtz (2), la envolvente compleja $A(\mathbf{r})$ debía satisfacer la ecuación de Helmholtz paraxial (5).

Una solución simple a la ecuación de Helmholtz paraxial proporciona la onda paraboloidal

$$A(\mathbf{r}) = \frac{A_1}{z} \exp\left(-jk\frac{\rho^2}{2z}\right), \quad \rho^2 = x^2 + y^2. \quad (6)$$

para la cual A_1 es una constante. La onda paraboloidal es la aproximación paraxial de la onda esférica $U(\mathbf{r}) = \frac{A_1}{r} \exp(-jkr)$ cuando x e y son mucho menores que z .

Otra solución de la ecuación de Helmholtz paraxial proporciona el haz gaussiano. Se obtiene a partir de la onda paraboloidal mediante una transformación simple. Dado que la envolvente compleja de la onda paraboloidal (6) es una solución de la ecuación de Helmholtz paraxial (5), una versión desplazada de esta, con $z - \xi$ reemplazando z donde ξ es una constante,

$$A(\mathbf{r}) = \frac{A_1}{q(z)} \exp\left[-jk\frac{\rho^2}{2q(z)}\right], \quad q(z) = z - \xi, \quad (7)$$

también es una solución. Esto proporciona una onda paraboloidal centrada en el punto $z = \xi$ en lugar de $z = 0$. Cuando ξ es complejo, (7) sigue siendo una solución de (5), pero adquiere propiedades dramáticamente diferentes. En particular, cuando ξ es puramente imaginario, digamos $\xi = -jz_0$ donde z_0 es real, (7) da lugar a la envolvente compleja del haz gaussiano

$$A(\mathbf{r}) = \frac{A_1}{q(z)} \exp\left[-jk\frac{\rho^2}{2q(z)}\right], \quad q(z) = z + jz_0. \quad (8)$$

El parámetro z_0 se conoce como longitud de Rayleigh.

Para separar la amplitud y fase de esta envolvente compleja, escribimos la función compleja $1/q(z) = 1/(z + jz_0)$ en términos de sus partes real e imaginaria definiendo dos nuevas funciones reales $R(z)$ y $W(z)$, de modo que

$$\frac{1}{q(z)} = \frac{1}{R(z)} - j\frac{\lambda}{\pi W^2(z)}. \quad (9)$$

Se mostrará posteriormente que $W(z)$ y $R(z)$ son medidas del ancho del haz y el radio de curvatura de los frentes de onda, respectivamente. Las expresiones para $W(z)$ y $R(z)$ como funciones de z y z_0 se proporcionan en (11) y (12), respectivamente. Sustituyendo (9) en (8) y usando (1), se obtiene una expresión para la amplitud compleja $U(\mathbf{r})$ del haz gaussiano:

$$U(\mathbf{r}) = A_0 \frac{W_0}{W(z)} \exp\left[-\frac{\rho^2}{W^2(z)}\right] \exp\left[-jkz - jk\frac{\rho^2}{2R(z)} + j\zeta(z)\right] \quad (10)$$

donde:

$$W(z) = W_0 \left[1 + \left(\frac{z}{z_0} \right)^2 \right]^{1/2}, \quad (11)$$

$$R(z) = z \left[1 + \left(\frac{z_0}{z} \right)^2 \right], \quad (12)$$

$$\zeta(z) = \tan^{-1} \frac{z}{z_0}, \quad (13)$$

$$W_0 = \left(\frac{\lambda z_0}{\pi} \right)^{1/2}. \quad (14)$$

Se ha definido una nueva constante $A_0 = A_1/jz_0$ para mayor comodidad. La expresión de la amplitud compleja del haz gaussiano es central en este capítulo. Contiene dos parámetros, A_0 y z_0 , que se determinan a partir de las condiciones de frontera. Todos los demás parámetros están relacionados con el rango de Rayleigh z_0 y la longitud de onda λ mediante las ecuaciones (11) a (14).

4.1.1. Propiedades

Las ecuaciones (10) a (14) se usarán ahora para determinar las propiedades del haz gaussiano.

- La **intensidad óptica** $I(\mathbf{r}) \propto |U(\mathbf{r})|^2$ es una función de las distancias axial y radial: z y $\rho = (x^2 + y^2)^{1/2}$,

$$I(\rho, z) = I_0 \left[\frac{W_0}{W(z)} \right]^2 \exp \left[-\frac{2\rho^2}{W^2(z)} \right], \quad (15)$$

donde $I_0 = |A_0|^2$. Para cada valor de z , la intensidad es una función gaussiana de la distancia radial ρ . Por esta razón, esta onda se llama haz gaussiano. La función gaussiana tiene su pico en $\rho = 0$ (en el eje) y disminuye monótonamente con el aumento de ρ . El ancho $W(z)$ de la distribución gaussiana aumenta con la distancia axial z como se ilustra en la Figura 4

En el eje del haz ($\rho = 0$), la intensidad es:

$$I(0, z) = I_0 \left[\frac{W_0}{W(z)} \right]^2 = \frac{I_0}{1 + (z/z_0)^2} \quad (16)$$

tiene su valor máximo I_0 en $z = 0$ y disminuye gradualmente con el aumento de z , alcanzando la mitad de su valor máximo en $z = \pm z_0$ (Figura 5). Cuando $|z| \gg z_0$, $I(0, z) \approx I_0 z_0^2 / z^2$, por lo que la intensidad disminuye con la distancia de acuerdo con una ley de inverso cuadrado, como ocurre con las ondas esféricas y paraboloidales. La intensidad máxima global $I(0, 0) = I_0$ ocurre en el centro del haz ($z = 0, \rho = 0$).

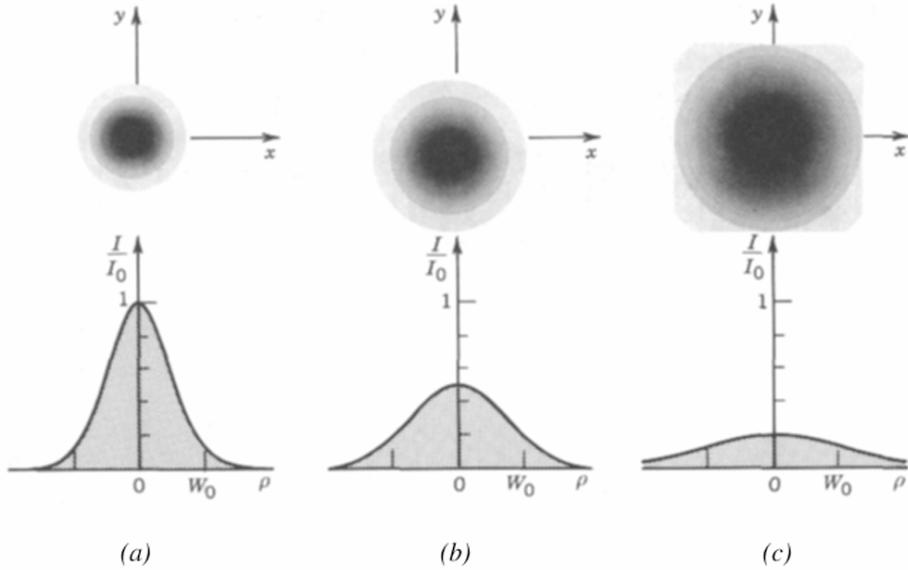


Figura 4: La intensidad del haz normalizado I/I_0 como función de la distancia radial ρ a diferentes distancias axiales: (a) $z = 0$, (b) $z = z_0$, (c) $z = 2z_0$. [21]

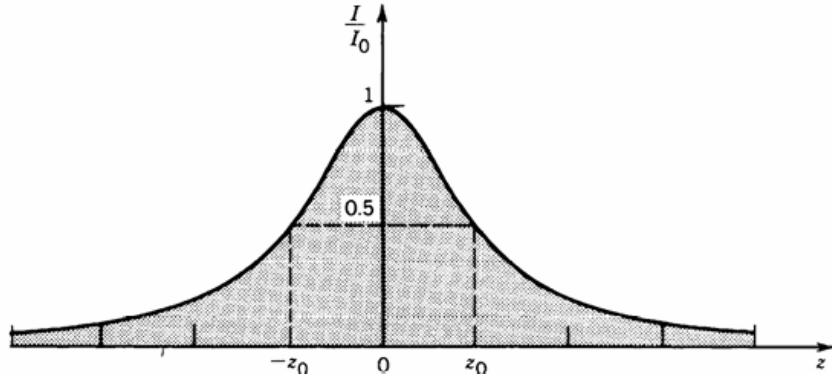


Figura 5: La intensidad del haz normalizada I/I_0 en los puntos del eje del haz ($\rho = 0$) en función de z [21]

- La **potencia óptica** total transportada por el haz es la integral de la intensidad óptica sobre un plano transversal (digamos a una distancia z),

$$P = \int_0^\infty I(\rho, z) 2\pi\rho d\rho,$$

lo que da

$$P = \frac{1}{2} I_0 (\pi W_0^2). \quad (17)$$

El resultado es independiente de z , como se esperaba. Por lo tanto, la potencia del haz es la mitad de la intensidad máxima multiplicada por el área del haz. Dado que los haces a menudo se describen por su potencia P , es útil expresar I_0 en términos de P usando (17) y reescribir (15) en la forma

$$I(\rho, z) = \frac{2P}{\pi W^2(z)} \exp \left[-\frac{2\rho^2}{W^2(z)} \right]. \quad (18)$$

La fracción de potencia transportada dentro de un círculo de radio ρ_0 en el plano transversal en la posición z con respecto a la potencia total es

$$\frac{1}{P} \int_0^{\rho_0} I(\rho, z) 2\pi\rho d\rho = 1 - \exp \left[-\frac{2\rho_0^2}{W^2(z)} \right]. \quad (19)$$

La potencia contenida dentro de un círculo de radio $\rho_0 = W(z)$ es aproximadamente el 86 % de la potencia total. Aproximadamente el 99 % de la potencia está contenida dentro de un círculo de radio $1,5W(z)$.

- Dentro de cualquier plano transversal, la intensidad del haz alcanza su valor máximo en el eje del haz y disminuye en un factor de $1/e^2 \approx 0,135$ a la distancia radial $\rho = W(z)$. Dado que el 86 % de la potencia se transporta dentro de un círculo de radio $W(z)$, consideramos $W(z)$ como el **radio del haz** (también conocido como anchura del haz). La desviación estándar de la distribución de intensidad es $\sigma = \frac{1}{2}W(z)$.

La dependencia del radio del haz respecto a z está gobernada por (11),

$$W(z) = W_0 \left[1 + \left(\frac{z}{z_0} \right)^2 \right]^{1/2}.$$

Alcanza su valor mínimo W_0 en el plano $z = 0$, llamado cintura del haz. Por lo tanto, W_0 es el radio de la cintura. El diámetro de la cintura $2W_0$ se llama el diámetro del punto. El radio del haz aumenta gradualmente con z , alcanzando $\sqrt{2}W_0$ en $z = z_0$, y continúa aumentando monótonamente con z (Figura 6). Para $z \gg z_0$ el primer término de (11) puede ser despreciado, resultando en la relación lineal

$$W(z) \approx \frac{W_0}{z_0} z = \theta_0 z. \quad (20)$$

donde $\theta_0 = W_0/z_0$. Usando (14), también podemos escribir

$$\theta_0 = \frac{\lambda}{\pi W_0}. \quad (21)$$

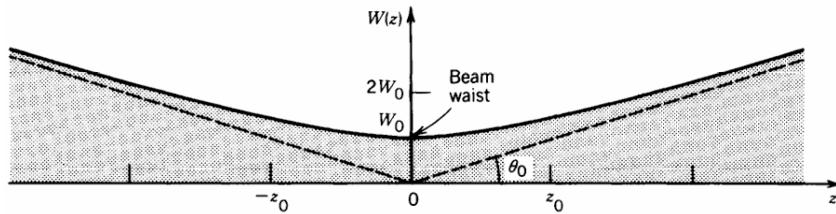


Figura 6: El radio del haz $W(z)$ tiene su valor mínimo W_0 en la cintura del haz ($z=0$). Alcanza $\sqrt{2}W_0$ en $z = \pm z_0$. Aumenta linealmente con z para valores de z superiores a los mencionados [21]

- Lejos del centro del haz, cuando $z \gg z_0$, el radio del haz aumenta aproximadamente de manera lineal con z , definiendo un cono con medio ángulo θ_0 . Aproximadamente el 86 % de la potencia del haz está confinada dentro de este cono. La **divergencia angular del haz** está, por lo tanto, definida por el ángulo

$$\theta_0 = \frac{2}{\pi} \frac{\lambda}{2W_0}. \quad (22)$$

La divergencia del haz es directamente proporcional a la relación entre la longitud de onda λ y el diámetro de la cintura del haz $2W_0$. Si la cintura se reduce, el haz se dispersa. Para obtener un haz altamente direccional, por lo tanto, se debe usar una longitud de onda corta y una cintura de haz ancha.

- Dado que el haz tiene su ancho mínimo en $z = 0$, como se muestra en la Figura 6, alcanza su mejor enfoque en el plano $z = 0$. En cualquier dirección, el haz gradualmente "sale del foco". La distancia axial dentro de la cual el radio del haz se mantiene dentro de un factor $\sqrt{2}$ de su valor mínimo (es decir, su área se mantiene dentro de un factor de 2 de su mínimo) se conoce como la **profundidad de foco** (Figura 7). Se puede observar en (11) que la profundidad de foco es el doble del rango de Rayleigh,

$$2z_0 = \frac{2\pi W_0^2}{\lambda}. \quad (23)$$

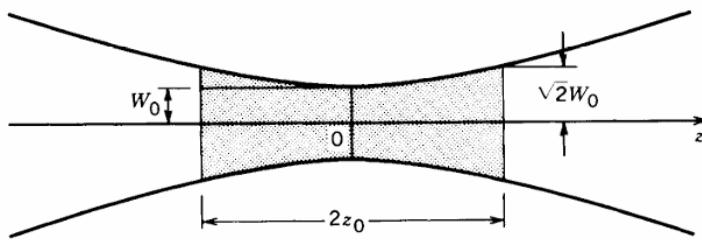


Figura 7: Profundidad de foco de un haz gaussiano [21]

La profundidad de foco es directamente proporcional al área del haz en su cintura e inversamente proporcional a la longitud de onda. Por lo tanto, cuando un haz es enfocado a un tamaño de punto pequeño, la profundidad de foco es corta y el plano de enfoque debe ubicarse con mayor precisión. Un tamaño de punto pequeño y una gran profundidad de foco no pueden obtenerse simultáneamente a menos que la longitud de onda de la luz sea corta. Para $\lambda = 633$ nm (la longitud de onda de una línea láser de He-Ne), por ejemplo, un tamaño de punto $2W_0 = 2$ cm corresponde a una profundidad de foco $2z_0 \approx 1$ km. Un tamaño de punto mucho más pequeño de $20 \mu\text{m}$ corresponde a una profundidad de foco mucho más corta de 1 mm.

- La **fase del haz** es, según (10),

$$\varphi(\rho, z) = kz - \zeta(z) + \frac{k\rho^2}{2R(z)}. \quad (24)$$

En el eje del haz ($\rho = 0$) la fase

$$\varphi(0, z) = kz - \zeta(z) \quad (25)$$

comprende dos componentes. El primero, kz , es la fase de una onda plana. El segundo representa un retardo de fase $\zeta(z)$ dado por (13), que varía de $-\pi/2$ en $z = -\infty$ hasta $+\pi/2$ en $z = \infty$, como se ilustra en la Figura 8. Este retardo de fase corresponde a un retraso excesivo del frente de onda en comparación con una onda plana o una onda esférica (véase también la Figura 9). El retardo total acumulado en el viaje de onda desde $z = -\infty$ hasta $z = \infty$ es π . Este fenómeno se conoce como el *efecto de Gouy*.

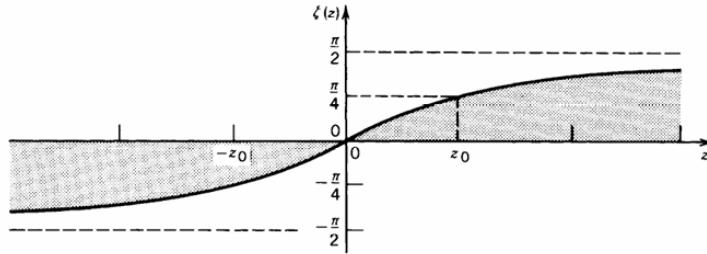


Figura 8: $\zeta(z)$ es el retardo de fase del haz gaussiano respecto a ondas planas uniformes en puntos sobre el eje del haz [21]

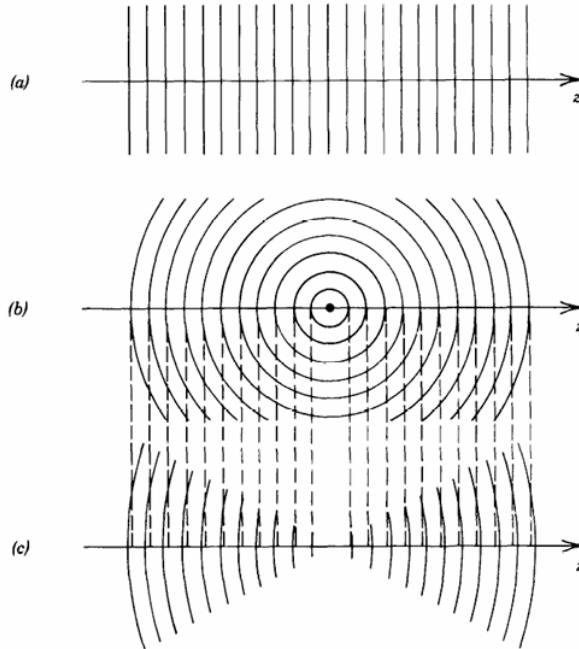


Figura 9: Frentes de onda de (a) una onda plana uniforme; (b) una onda esférica; (c) un haz gaussiano. En los puntos cercanos al centro del haz, el haz gaussiano se asemeja a una onda plana. Para valores grandes de z , el haz se comporta como una onda esférica, excepto que la fase está retrasada en 90° (representado en este diagrama por un cuarto de la distancia entre dos frentes de onda adyacentes). [21]

- El tercer componente en (24) es responsable de la curvatura del **frente de onda**. Representa la desviación de la fase en puntos fuera del eje en un plano transversal dado con respecto a la fase en el punto axial. Las superficies de fase constante satisfacen $k[z + \rho^2/2R(z)] - \zeta(z) = 2\pi q$. Dado que $\zeta(z)$ y $R(z)$ varían relativamente lento, son aproximadamente constantes en los puntos dentro del radio del haz en cada frente de onda. Por lo tanto, podemos escribir $z + \rho^2/2R = q\lambda + \zeta\lambda/2\pi$, donde $R = R(z)$ y $\zeta = \zeta(z)$. Esto es precisamente la ecuación de una superficie paraboloidal de radio de curvatura R . Por lo tanto, $R(z)$, representado en la Figura 10, es el radio de curvatura del frente de onda en la posición z sobre el eje del haz.

Como se ilustra en la Figura 10, el radio de curvatura $R(z)$ es infinito en $z = 0$, lo que corresponde a frentes de onda planos. Disminuye hasta un valor mínimo de $2z_0$ en $z = z_0$. Este es el punto en el que el frente de onda tiene la mayor curvatura (Figura 11). Posteriormente, el radio de curvatura aumenta con el incremento de z hasta que $R(z) \approx z$ para $z \gg z_0$. El frente de onda es entonces aproximadamente el mismo que el de una onda esférica.

Para z negativo, los frentes de onda siguen un patrón idéntico, excepto por un cambio

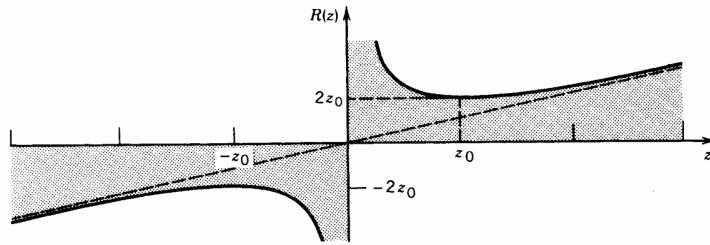


Figura 10: El radio de curvatura $R(z)$ de los frentes de onda de un haz gaussiano. La línea discontinua representa el radio de curvatura de una onda esférica. [21]

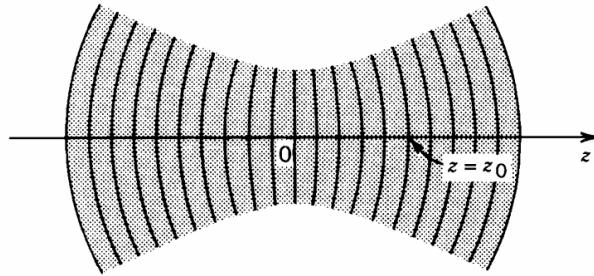


Figura 11: Frentes de onda de un haz gaussiano [21]

de signo. Hemos adoptado la convención de que un frente de onda divergente tiene un radio de curvatura positivo, mientras que un frente de onda convergente tiene un radio de curvatura negativo.

Para resumir las propiedades de los haces gaussianos en algunos puntos especiales:

- En el plano $z = z_0$, a una distancia axial z_0 desde la cintura del haz, la onda tiene las siguientes propiedades:
 1. El radio del haz es $\sqrt{2}$ veces mayor que el radio en la cintura del haz, y el área es mayor por un factor de 2.
 2. La intensidad en el eje del haz es $\frac{1}{2}$ de la intensidad máxima.
 3. La fase en el eje del haz se retrasa en un ángulo de $\pi/4$ en relación con la fase de una onda plana.

■ Cerca del centro del haz

En puntos donde $|z| \ll z_0$ y $\rho \ll W_0$, se cumple que

$$\exp[-\rho^2/W^2(z)] \approx \exp(-\rho^2/W_0^2) \approx 1,$$

por lo que la intensidad del haz es aproximadamente constante.

Además, $R(z) \approx z_0^2/z$ y $\zeta(z) \approx 0$, por lo que la fase

$$kz + \frac{\rho^2}{2R(z)} = kz \left(1 + \frac{\rho^2}{2z_0^2}\right) \approx kz.$$

Como resultado, los frentes de onda son aproximadamente planos. Por lo tanto, el haz gaussiano puede aproximarse cerca de su centro como una onda plana.

- Lejos de la cintura del haz

En puntos dentro del radio de la cintura del haz ($\rho < W_0$), pero lejos de la cintura del haz ($z \gg z_0$), la onda es aproximadamente una onda esférica.

Dado que $W(z) = W_0 z / z_0 \gg W_0$ y $\rho < W_0$, se cumple que

$$\exp[-\rho^2/W^2(z)] \approx 1,$$

por lo que la intensidad del haz es aproximadamente uniforme.

Puesto que $R(z) \approx z$, los frentes de onda son aproximadamente esféricos. Así, excepto por un desfase adicional $\zeta(z) = \pi/2$, la amplitud compleja del haz gaussiano se aproxima a la de una onda paraboloidal, que a su vez se aproxima a la onda esférica en la aproximación paraxial (Figura 9).

4.1.2. Parámetros que caracterizan un haz gaussiano

Suponiendo que la longitud de onda λ , vamos a ver cuántos parámetros más se requieren para describir una onda plana, una onda esférica y un haz gaussiano.

La onda plana se especifica completamente por su amplitud compleja y dirección. La onda esférica se especifica por su amplitud y la ubicación de su origen.

El haz gaussiano, en contraste, se caracteriza por más parámetros: su amplitud máxima [el parámetro A_0 en (10)], su dirección (el eje del haz), la ubicación de su cintura y un parámetro adicional: el radio de la cintura W_0 o el rango de Rayleigh z_0 , por ejemplo. Por lo tanto, si la amplitud máxima del haz y el eje son conocidos, se requieren dos parámetros adicionales.

Si el número complejo $q(z) = z + jz_0$ es conocido, la distancia z a la cintura del haz y el rango de Rayleigh z_0 se identifican fácilmente como las partes real e imaginaria de $q(z)$. Como ejemplo, si el parámetro q es $3 + j4$ cm en algún punto del eje del haz, concluimos que la cintura del haz se encuentra a una distancia $z = 3$ cm a la izquierda de ese punto y que la profundidad de foco es $2z_0 = 8$ cm. El radio de la cintura W_0 puede determinarse utilizando (14).

El parámetro q , $q(z)$, es por lo tanto suficiente para caracterizar un haz gaussiano de amplitud máxima y eje del haz conocidos. La dependencia lineal del parámetro q con respecto a z nos permite determinar fácilmente q en todos los puntos, dado q en un único punto. Si $q(z) = q_1$ y $q(z + d) = q_2$, entonces $q_2 = q_1 + d$. En el presente ejemplo, en $z = 13$ cm, $q = 13 + j4$.

Si el ancho del haz $W(z)$ y el radio de curvatura $R(z)$ son conocidos en un punto arbitrario del eje, el haz puede identificarse completamente resolviendo (11), (12) y (14) para z , z_0 y W_0 . Alternativamente, el parámetro q puede determinarse a partir de $W(z)$ y $R(z)$ usando la relación:

$$\frac{1}{q(z)} = \frac{1}{R(z)} - j \frac{\lambda}{\pi W^2(z)},$$

a partir de la cual el haz queda identificado.

4.2. Haces Hermite-Gaussianos (HG)

El haz gaussiano no es la única solución en forma de haz de la ecuación de Helmholtz paraxial (5). Puede haber otras soluciones, incluidos haces con distribuciones de intensi-

dad no gaussianas. De particular interés son aquellas soluciones que comparten los frentes de onda paraboloidales del haz gaussiano, pero exhiben diferentes distribuciones de intensidad.

Consideremos un haz gaussiano cuya envolvente compleja, $A_G(x, y, z)$, viene dada por la ecuación (8). El radio del haz $W(z)$ se da en (11) y el radio de curvatura $R(z)$ se da en (12).

Consideremos una segunda onda cuya envolvente compleja es una versión modulada del haz gaussiano:

$$A(x, y, z) = \mathfrak{X} \left(\frac{\sqrt{2}x}{W(z)} \right) \mathcal{Y} \left(\frac{\sqrt{2}y}{W(z)} \right) \exp [j\mathcal{Z}(z)] A_G(x, y, z), \quad (26)$$

donde \mathfrak{X} , \mathcal{Y} y \mathcal{Z} son funciones reales.

Si esta onda existe, tiene las siguientes dos propiedades:

- La fase es la misma que la del haz gaussiano subyacente, excepto por un exceso de fase $\mathcal{Z}(z)$ que es independiente de x e y . Si $\mathcal{Z}(z)$ es una función de z que varía lentamente, las dos ondas tienen frentes de onda paraboloidales con el mismo radio de curvatura $R(z)$. Por lo tanto, estas dos ondas se enfocan mediante lentes y espejos exactamente de la misma manera.
- La magnitud

$$A_0 \mathfrak{X} \left(\frac{\sqrt{2}x}{W(z)} \right) \mathcal{Y} \left(\frac{\sqrt{2}y}{W(z)} \right) \left(\frac{W_0}{W(z)} \right) \exp \left[-\frac{x^2 + y^2}{W^2(z)} \right]. \quad (27)$$

donde $A_0 = A_1/jz_0$ es una función de $x/W(z)$ e $y/W(z)$, cuyos anchos en las direcciones x e y varían con z de acuerdo con el mismo factor de escala $W(z)$. A medida que z aumenta, la distribución de intensidad en el plano transversal permanece fija, excepto por un factor de ampliación $W(z)$. Esta distribución es una función gaussiana modulada en las direcciones x e y por las funciones $\mathfrak{X}(\cdot)$ y $\mathcal{Y}(\cdot)$.

Por lo tanto, la onda modulada representa un haz con una distribución de intensidad no gaussiana, pero con los mismos frentes de onda y la misma divergencia angular que el haz gaussiano.

La existencia de esta onda está asegurada si se pueden encontrar tres funciones reales $\mathfrak{X}(\cdot)$, $\mathcal{Y}(\cdot)$ y $\mathcal{Z}(z)$ tales que la ecuación (26) satisfaga la ecuación de Helmholtz paraxial (5). Sustituyendo (26) en (5), usando el hecho de que A_G satisface (5), y definiendo dos nuevas variables $u = \sqrt{2}x/W(z)$ y $v = \sqrt{2}y/W(z)$, obtenemos:

$$\frac{1}{\mathfrak{X}} \left(\frac{\partial^2 \mathfrak{X}}{\partial u^2} - 2u \frac{\partial \mathfrak{X}}{\partial u} \right) + \frac{1}{\mathcal{Y}} \left(\frac{\partial^2 \mathcal{Y}}{\partial v^2} - 2v \frac{\partial \mathcal{Y}}{\partial v} \right) + kW^2(z) \frac{\partial \mathcal{Z}}{\partial z} = 0. \quad (28)$$

Dado que el lado izquierdo de esta ecuación es la suma de tres términos, y cada uno de ellos es función de una variable independiente única, u , v o z , cada uno de estos términos debe ser constante. Igualando el primer término a la constante $-2\mu_1$ y el segundo a $-2\mu_2$, el tercero debe ser igual a $2(\mu_1 + \mu_2)$.

Esta técnica de "separación de variables" nos permite reducir la ecuación diferencial parcial (28) a tres ecuaciones diferenciales ordinarias para $\mathfrak{X}(u)$, $\mathcal{Y}(v)$ y $\mathcal{Z}(z)$, respectivamente:

$$-\frac{1}{2} \frac{d^2 \mathfrak{X}}{du^2} + u \frac{d\mathfrak{X}}{du} = \mu_1 \mathfrak{X} \quad (29)$$

$$-\frac{1}{2} \frac{d^2 \mathcal{Y}}{dv^2} + v \frac{d\mathcal{Y}}{dv} = \mu_2 \mathcal{Y} \quad (30)$$

$$z_0 \left[1 + \left(\frac{z}{z_0} \right)^2 \right] \frac{d\mathcal{Z}}{dz} = \mu_1 + \mu_2. \quad (31)$$

donde hemos usado la expresión para $W(z)$ dada en (11) y (14).

La ecuación (29) representa un problema de autovalores cuyos valores propios son $\mu_1 = l$, donde $l = 0, 1, 2, \dots$, y cuyas autofunciones son los **polinomios de Hermite** $\mathfrak{X}(u) = H_l(u)$. Estos polinomios están definidos por la relación de recurrencia:

$$H_{l+1}(u) = 2uH_l(u) - 2lH_{l-1}(u) \quad (32)$$

y

$$H_0(u) = 1, \quad H_1(u) = 2u. \quad (33)$$

Por lo tanto,

$$H_2(u) = 4u^2 - 2, \quad H_3(u) = 8u^3 - 12u, \quad \dots \quad (34)$$

De manera similar, las soluciones de (30) son $\mu_2 = m$ y $\mathcal{Y}(v) = H_m(v)$, donde $m = 0, 1, 2, \dots$. Por lo tanto, existe una familia de soluciones etiquetadas por los índices (l, m) .

Sustituyendo $\mu_1 = l$ y $\mu_2 = m$ en (31), e integrando, obtenemos:

$$\mathcal{Z}(z) = (l + m)\zeta(z), \quad (35)$$

donde $\zeta(z) = \tan^{-1}(z/z_0)$. La fase en exceso $\mathcal{Z}(z)$ varía lentamente entre $-(l + m)\pi/2$ y $(l + m)\pi/2$, a medida que z varía entre $-\infty$ y ∞ (ver Figura 8).

Finalmente, sustituimos en (26) para obtener una expresión para la envolvente compleja del haz etiquetado por los índices (l, m) . Reordenando términos y multiplicando por $\exp(-jkz)$, obtenemos la amplitud compleja:

$$U_{l,m}(x, y, z) = A_{l,m} \left[\frac{W_0}{W(z)} \right] G_l \left[\frac{\sqrt{2}x}{W(z)} \right] G_m \left[\frac{\sqrt{2}y}{W(z)} \right] \times \exp \left[-jkz - jk \frac{x^2 + y^2}{2R(z)} + j(l + m + 1)\zeta(z) \right] \quad (36)$$

donde

$$G_l(u) = H_l(u) \exp \left(\frac{-u^2}{2} \right), \quad l = 0, 1, 2, \dots, \quad (37)$$

es conocida como la **función de Hermite-Gauss** de orden l , y $A_{l,m}$ es una constante.

Dado que $H_0(u) = 1$, la función de Hermite-Gauss de orden 0 es simplemente la función gaussiana. $G_1(u) = 2u \exp(-u^2/2)$ es una función impar, $G_2(u) = (4u^2 - 2) \exp(-u^2/2)$ es par, $G_3(u) = (8u^3 - 12u) \exp(-u^2/2)$ es impar, y así sucesivamente. Estas funciones se muestran en la Figura 12.

Una onda óptica con amplitud compleja dada por (36) es conocida como el haz de Hermite-Gauss de orden (l, m) . El haz de Hermite-Gauss de orden $(0, 0)$ es el haz Gaussiano.

Distribución de Intensidad

La intensidad óptica del haz de Hermite-Gaussiano (l, m) es

$$I_{l,m}(x, y, z) = |A_{l,m}|^2 \left[\frac{W_0}{W(z)} \right]^2 G_l^2 \left[\frac{\sqrt{2}x}{W(z)} \right] G_m^2 \left[\frac{\sqrt{2}y}{W(z)} \right]. \quad (38)$$

La Figura 13 ilustra la dependencia de la intensidad en las distancias transversales normalizadas $u = \sqrt{2}x/W(z)$ y $v = \sqrt{2}y/W(z)$ para varios valores de l y m . Los haces de orden superior tienen mayores anchuras que los de orden inferior, como es evidente en la Figura 12.

Independientemente del orden, sin embargo, el ancho del haz es proporcional a $W(z)$, de modo que a medida que z aumenta, el patrón de intensidad se amplifica por el factor $W(z)/W_0$, pero por lo demás mantiene su perfil. Entre la familia de haces Hermite-Gaussianos, el único miembro con simetría circular es el haz Gaussiano.

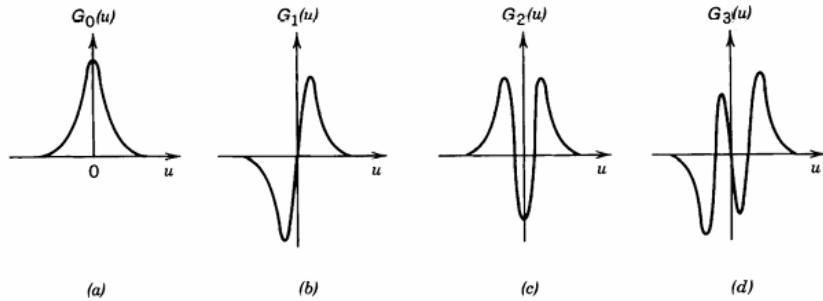


Figura 12: Varias funciones de Hermite-Gauss de bajo orden: (a) $G_0(u)$; (b) $G_1(u)$; (c) $G_2(u)$; (d) $G_3(u)$.[21]

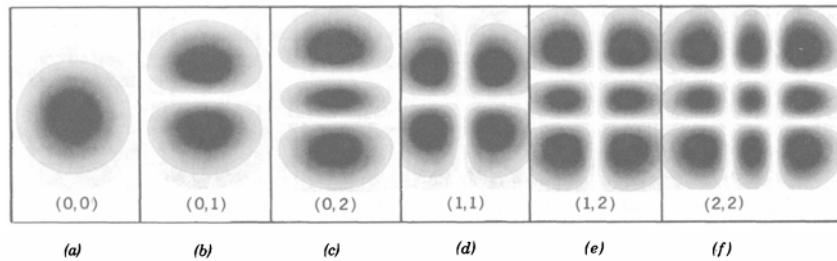


Figura 13: Distribuciones de intensidad de varios haces Hermite-Gaussianos de orden bajo en el plano transversal. El orden (l, m) está indicado en cada caso [21]

5. Redes neuronales

Una red neuronal artificial constituye un modelo computacional cuya inspiración proviene del funcionamiento del cerebro humano. Su finalidad principal es la identificación de patrones y la resolución de problemas de alta complejidad. Este tipo de red se construye a partir de unidades denominadas neuronas artificiales, distribuidas en distintos tipos de capas: de entrada, ocultas y de salida. Las conexiones entre neuronas poseen pesos asociados que se van ajustando a lo largo del proceso de entrenamiento con el objetivo de minimizar el error cometido en las predicciones [22].

Para comprender adecuadamente tanto la dinámica como la estructura de estas redes, resulta útil comenzar analizando arquitecturas básicas e ir incorporando gradualmente elementos de mayor complejidad. Este recorrido nos conducirá hasta las redes convolucionales, que son el tipo particular de red neuronal que utilizaremos en nuestro trabajo, ya que se ajustan mejor al tipo de problema que buscamos resolver.

5.1. Redes neuronales de una capa

El primer tipo de red neuronal artificial que vamos a describir en detalle es el modelo ADALINE [23]. Este modelo surgió poco tiempo después del *Perceptrón* y presenta algunas diferencias sutiles respecto a él, aunque ambos se basan en el mismo principio: una arquitectura con una sola capa de neuronas cuyos pesos se actualizan mediante un proceso iterativo [24].

Para comenzar, nos centraremos en la estructura más básica: un ADALINE compuesto por una única neurona, lo que lo convierte en un sistema de clasificación binaria, capaz de diferenciar entre dos clases o patrones. La neurona artificial funciona como un módulo de cálculo cuya respuesta depende de los valores asignados a sus pesos. Al recibir un vector de entrada $x = (x_1, \dots, x_d)$, cada componente se multiplica por su peso correspondiente w_i . A esta combinación lineal se le suma un valor adicional w_0 , denominado *bias* o sesgo, cuya función es desplazar la función de activación en el eje horizontal⁴.

El resultado de esta suma ponderada y el sesgo es el nivel de activación interno $z(x)$ ⁵. Como el propósito final es asignar una clase, es necesario transformar $z(x)$ en una salida discreta. Para ello se utiliza una función umbral, que en este caso es la función signo, la cual devuelve $+1$ si el valor es positivo y -1 si es negativo. La salida $\hat{y}(x) \in \{-1, 1\}$ representa así la clase asignada a la muestra.

La función de activación es el término que se aplica a la combinación lineal de las entradas. Aunque existen diversas opciones según la red y la tarea concreta, el modelo original de ADALINE emplea la función signo debido a su sencillez. Para mayor comodidad en la notación, suele añadirse una componente extra $x_0 = 1$, lo que permite integrar el sesgo en la suma ponderada. Con esta convención, el nivel de activación interno se define como:

$$z(x) = \sum_{i=0}^d w_i x_i \quad (39)$$

Y el valor de salida de la neurona como:

⁴El sesgo permite al modelo ajustarse a conjuntos de datos cuyo hiperplano separador no pase por el origen.

⁵Este valor no tiene restricciones y puede adoptar cualquier número real.

$$\hat{y}(x) = \text{sgn}(z(x)) = \text{sgn} \left(\sum_{i=0}^d w_i x_i \right) \quad (40)$$

Otra forma de visualizar la analogía entre el funcionamiento de una neurona artificial y el de una biológica es el esquema de la figura 14.

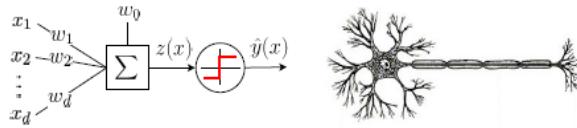


Figura 14: Comparación entre una neurona artificial (ADALINE) y una biológica.[13]

Hasta este punto se ha explicado el mecanismo mediante el cual se genera la salida de una neurona a partir de sus entradas y los pesos asignados. Sin embargo, la principal aportación del modelo ADALINE respecto a propuestas más primitivas como la de McCulloch-Pitts [25] reside en su capacidad para ajustar los pesos automáticamente utilizando un conjunto de entrenamiento.

Este ajuste se formula como un problema de optimización, cuya resolución se lleva a cabo de manera iterativa. En primer lugar, se define una función de pérdida o coste (también conocida como *loss function*) que cuantifica el error cometido por la red sobre el conjunto de entrenamiento. Durante el proceso de aprendizaje, el objetivo es minimizar esta función, es decir, encontrar aquellos valores de los pesos que produzcan el menor error posible.

La función de coste más comúnmente empleada en redes neuronales es el error cuadrático medio o MSE (*mean squared error*). Formalmente, sea $X = \{x^{(1)}, \dots, x^{(n)}\}$ el conjunto de entradas del conjunto de entrenamiento, e $Y = \{y^{(1)}, \dots, y^{(n)}\}$ las etiquetas correspondientes. Entonces, el error cuadrático medio para un ADALINE de una sola neurona se expresa como:

$$L(w) = \frac{1}{2n} \sum_{j=1}^n (y^{(j)} - z(x^{(j)}))^2 \quad (41)$$

Donde se usa el error cuadrado para que la función de coste crezca con los errores cometidos por la red independientemente de su signo. Así mismo, se usa el factor $\frac{1}{n}$ para hacer la magnitud del error independiente del número de muestras presentes en el set de entrenamiento y el $\frac{1}{2}$ para simplificar la derivada.

El procedimiento de optimización más empleado es el conocido como descenso por gradiente. Este método comienza asignando a los pesos del ADALINE valores iniciales aleatorios que suelen ser pequeños y próximos a cero. A continuación, en cada iteración se calcula el gradiente parcial de la función de pérdida respecto a cada peso, con el fin de analizar cómo influye un pequeño cambio en dichos valores sobre el error cometido por la red.

Al ajustar los pesos en la dirección contraria a la del gradiente de la función de coste, se consigue reducir progresivamente el error. La figura 15 ofrece una representación visual de este proceso.

Formalmente, en cada iteración se modifica cada peso w_i en una cantidad Δw_i , que a su vez es calculada multiplicando la derivada parcial de la función de coste respecto a w_i

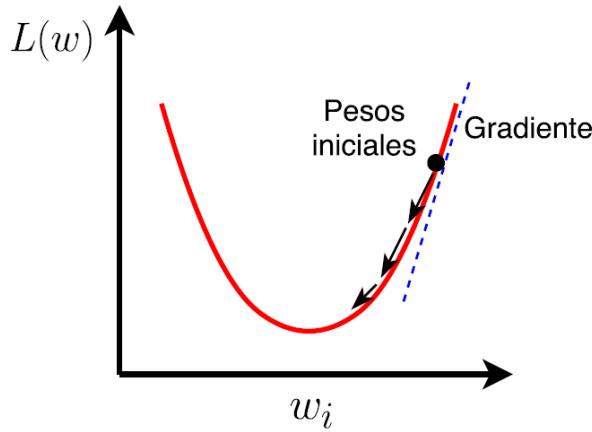


Figura 15: Ilustración del proceso de descenso de gradiente para minimizar la función de coste.[13]

por una constante α (es el *learning rate*, y es uno de los hiperparámetros que debemos ajustar cuando entrenamos una red neuronal) el que controla la velocidad de aprendizaje:

$$w_i \rightarrow w_i + \Delta w_i$$

$$\Delta w_i = -\alpha \frac{\partial L}{\partial w_i}$$

Si el valor elegido para α es demasiado bajo, el proceso de entrenamiento puede requerir un número elevado de iteraciones para lograr que el error se reduzca. En cambio, si se selecciona un α excesivamente alto, la red podría no llegar nunca a un nivel óptimo de error, ya que los pesos tenderían a oscilar alrededor del mínimo de la función de pérdida, y el tamaño de esas oscilaciones dependería directamente de α^6 .

Aunque en redes neuronales con múltiples capas el cálculo del gradiente de la función de pérdida resulta un desafío mayor, en el caso del ADALINE este procedimiento es bastante directo. Es fundamental tener en cuenta que el algoritmo actualiza simultáneamente todos los pesos en cada paso del descenso por gradiente.

5.2. Red multi-clase de una capa

Tal como se ha explicado anteriormente, un ADALINE formado por una única neurona puede abordar problemas de clasificación binaria, pero no es adecuado para problemas que implican más de dos categorías. Por suerte, adaptar este modelo a escenarios multi-clase resulta muy sencillo. En concreto, si el objetivo es clasificar en c clases diferentes, bastará con diseñar una red que disponga de c neuronas, asignando a cada una de ellas una clase concreta. Al presentar un ejemplo de entrada, la neurona correspondiente a la clase correcta debería producir un valor de activación elevado, mientras que el resto deberían generar salidas de menor magnitud. Así, el modelo seleccionará como clase predicha aquella vinculada a la neurona con la mayor activación.

⁶Es importante señalar que, al tratarse de un modelo tan sencillo como el ADALINE, el problema de optimización que abordamos es convexo. Esto implica que cualquier mínimo local de la función de coste coincide con el mínimo global. Gracias a esta característica, si el conjunto de entrenamiento corresponde a un problema linealmente separable y el *learning rate* se elige suficientemente pequeño, el algoritmo logrará en un número finito de pasos encontrar unos pesos que permitan clasificar correctamente todas las muestras [26]. Esta propiedad es conocida como la convergencia del modelo.

A diferencia del ADALINE binario, que empleaba una función escalón, aquí se requiere una función de activación que permita obtener salidas continuas, ya que de este modo es posible realizar la comparación entre neuronas y determinar la clase. En este caso se utiliza la función identidad, de manera que la salida de cada neurona es igual a la suma ponderada de sus entradas, es decir, no se aplica ninguna transformación adicional. La figura 16 muestra un esquema de este tipo de red neuronal.

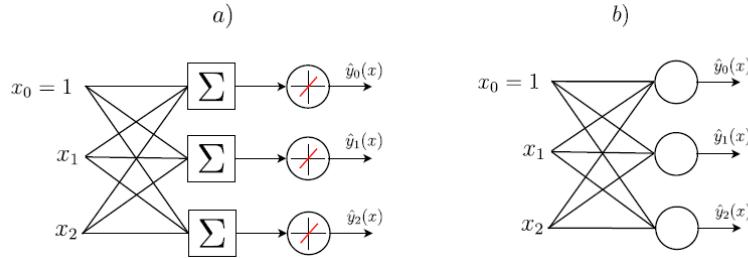


Figura 16: Diagramas arquitectónicos de una red neuronal de una sola capa con varias neuronas. El diagrama a) muestra los diferentes elementos que forman cada neurona y el diagrama b) muestra la simbología simplificada (los bias no aparecen dado que se usa una característica auxiliar $x_0 = 1$). [13].

Cada neurona tiene sus propios pesos asociados y el conjunto de todos puede representarse como una matriz: $W \in \mathbb{R}^{d+1 \times c}$, donde d es el número de características de nuestras muestras (sin contar la auxiliar $x_0 = 1$) y c es el número de neuronas en nuestra red de una capa. La salida de la i -ésima neurona de nuestra red viene dada en la ecuación 42.

$$\hat{y}_i(x) = g\left(\sum_{j=0}^d x_j W_{j,i}\right) \quad (42)$$

A su vez, el vector $\hat{y}(x) \in \mathbb{R}^c$ con las salidas de todas las neuronas se calculará como:

$$\hat{y}(x) = g(xW) \quad (43)$$

donde g representa la función de activación elegida, en este caso $g(x) = x$. La figura (17) muestra esta notación de forma visual.

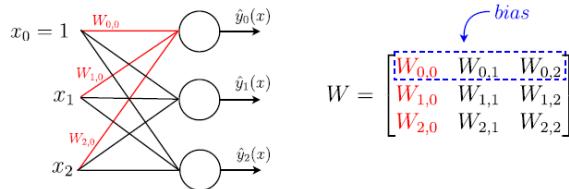


Figura 17: Los pesos de una red de una capa pueden representarse como una matriz W siempre y cuando añadamos a nuestras muestras la característica auxiliar $x_0 = 1$. [13]

Para poder entrenar este tipo de redes, es necesario convertir el vector de etiquetas Y en una matriz que contenga las salidas ideales de la red para cada muestra del conjunto de entrenamiento. En este caso, la salida ideal es un vector de c dimensiones, donde cada componente corresponde al valor esperado en la salida de una de las neuronas de la red. Para realizar esta conversión se emplea habitualmente la técnica conocida como *One-hot encoding*, que asocia a cada muestra de entrenamiento un vector c -dimensional con las salidas deseadas.

Por ejemplo, en un problema de clasificación con cuatro clases, una muestra etiquetada como clase ‘0’ se asociará al vector *One-hot* $[1, -1, -1, -1]$, mientras que una muestra de clase ‘1’ tendrá como salida deseada el vector $[-1, 1, -1, -1]$. De manera similar, las muestras de clase ‘2’ y ‘3’ tendrán los vectores $[-1, -1, 1, -1]$ y $[-1, -1, -1, 1]$ respectivamente. Con este enfoque se consigue que la red aprenda a generar una salida alta únicamente en la neurona que representa la clase correcta, manteniendo bajas las salidas del resto de neuronas.

Cuando ya se han generado las salidas deseadas correspondientes a cada ejemplo del conjunto de entrenamiento, se puede iniciar el proceso de aprendizaje de la red neuronal. En este tipo de redes con una única capa, cada peso influye únicamente en la salida de una neurona específica. Esto permite que el entrenamiento de la red completa se pueda abordar como el entrenamiento independiente de cada una de sus neuronas, del mismo modo que se explicó en la sección anterior.

5.3. Perceptrón multicapa

Aunque las redes neuronales artificiales de una sola capa gozaron de gran aceptación en sus inicios, su uso fue decayendo al hacerse evidentes sus carencias. La más destacada de estas limitaciones era su imposibilidad para abordar tareas de clasificación en las que las clases no son separables mediante un hiperplano, es decir, problemas no linealmente separables. Por suerte, como se mencionó al comienzo, este periodo de estancamiento en el desarrollo de las redes neuronales finalizó en la década de 1980 gracias al surgimiento del algoritmo de *backpropagation*, que hizo posible entrenar redes con múltiples capas y afrontar este tipo de problemas con éxito.

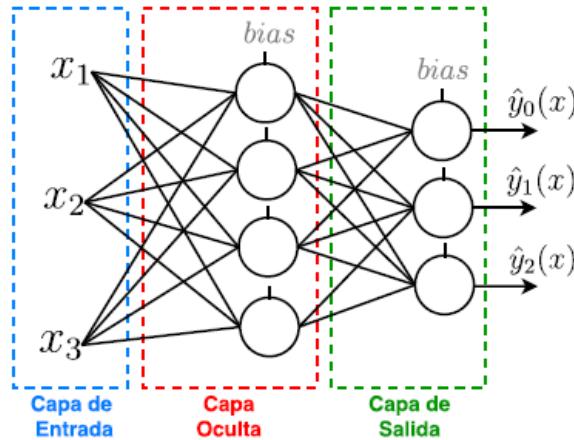


Figura 18: Arquitectura típica de un perceptrón multicapa con una sola capa oculta.[13]

El tipo de red neuronal con múltiples capas más utilizado es el denominado perceptrón multicapa. Este modelo está formado por una capa de entrada (que se encarga de recibir las características de la muestra a procesar), una o varias capas ocultas y una capa de salida (véase la figura 18). En esta estructura, el flujo de información siempre va en un único sentido: desde la capa de entrada, pasando por las ocultas, hasta llegar a la salida; por ello, estas redes se conocen como redes *feedforward*. El propósito de las capas ocultas es construir representaciones no lineales de los datos de entrada que permitan que la salida de la red pueda resolver problemas en los que las clases no sean separables mediante un hiperplano. Por esta razón, las neuronas de las capas ocultas deben emplear funciones de

activación no lineales, ya que si no fuera así, la red no aportaría ninguna ventaja sobre un modelo de una sola capa.

Una característica esencial de estas redes es que esas representaciones intermedias no se especifican de forma explícita, sino que son aprendidas automáticamente a partir del conjunto de entrenamiento mediante el ajuste de los pesos de las neuronas ocultas. Este carácter poco interpretable es el motivo por el que al perceptrón multicapa se le suele describir como un modelo de caja negra: las características que genera en sus capas internas durante el aprendizaje no son fácilmente comprensibles para una persona. Entre las funciones de activación no lineales más empleadas en estos modelos se encuentran las siguientes:

- **Función de activación Tangente Hiperbólica:** Se puede decir que esta función de activación es la versión continua y suavizada de la función escalón o signo que vimos en la sección anterior. Toma valores en el intervalo $[-1, 1]$ y su fórmula es la siguiente:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Además, tiene la útil propiedad de que su derivada puede escribirse en función del valor de la propia función, lo que hace más eficiente la ejecución del algoritmo *backpropagation*. En particular $\tanh'(x) = 1 - \tanh^2(x)$.

- **Función de activación Sigmoide/Logística:** Similar a la función tangente hiperbólica (ver figura 19), toma valores en el intervalo $[0, 1]$ y su fórmula es la siguiente:

$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

Su derivada puede escribirse en términos de la propia función como $\text{sigm}'(x) = \text{sigm}(x)(1 - \text{sigm}(x))$.

Una comparación entre las dos funciones anteriores podemos verla en la figura 19

- **Función de activación Softmax:** En lugar de aplicarse de forma individual en cada neurona, esta función actúa sobre el conjunto de los valores de activación internos de las neuronas de una capa. En particular, toma un vector z con valores arbitrarios y devuelve un vector de la misma dimensión donde todos los elementos están en el rango $[0, 1]$ y su suma es igual a uno. Por este motivo, este tipo de activación se usa frecuentemente en las capas de salida de las redes neuronales, dado que permite interpretar la salida de cada neurona como la probabilidad estimada para la muestra de entrada de pertenecer a la clase asociada a esa neurona. Dicho de otra forma, si usamos la función *softmax* como función de activación para la última capa de nuestra red, la salida de la misma podrá considerarse como un vector de probabilidades estimadas para cada clase. Formalmente, la salida de la i -ésima neurona en una capa de salida con activación *softmax* se calcula como sigue:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

donde el sumatorio del denominador se aplica sobre los niveles de activación internos de cada neurona en la capa de salida.

El desarrollo de una red neuronal con varias capas no difiere demasiado del de una red de capa única. Como se explicó previamente, los pesos de una capa pueden organizarse

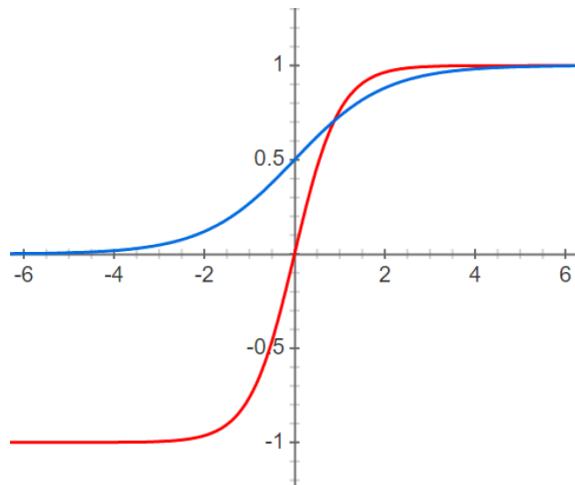


Figura 19: Comparación entre función sigmoide (azul) y tangente hiperbólica (rojo).[13]

en forma de matriz, y procesar una muestra consiste en multiplicar dicha matriz por el vector de entrada y aplicar a continuación la función de activación. En el caso del perceptrón multicapa, basta con disponer de una matriz de pesos por cada capa y aplicar estas transformaciones de forma secuencial hasta obtener la salida correspondiente a una muestra. Sin embargo, la mayor complejidad de este tipo de redes radica en el proceso de aprendizaje. Al hablar de ADALINE y redes de una sola capa, vimos que el descenso por gradiente consiste en calcular la derivada parcial de la función de error con respecto a cada peso, para después ajustar esos pesos en la dirección que reduce el error. En redes de una capa, como ADALINE, este cálculo es sencillo porque la salida de cada neurona depende directamente de sus pesos y de las entradas. En cambio, en redes con varias capas, calcular esas derivadas se convierte en un desafío considerable. De hecho, no fue hasta dos décadas después del auge de las redes de una capa que se desarrolló el algoritmo de *backpropagation*, que permitió realizar estos cálculos de forma eficiente y flexible.

El funcionamiento del algoritmo de *backpropagation* se basa en repetir de manera iterativa dos etapas: la propagación de la entrada a través de las capas hasta producir una salida, y la actualización de los pesos. Al procesar una muestra, se calcula la salida de la red y se compara con el valor deseado. Esto permite obtener el error en la capa de salida, el cual se transmite hacia las capas anteriores mediante la regla de la cadena de derivadas. De este modo, se obtiene el error correspondiente a cada neurona y se calcula el gradiente de la función de pérdida con respecto a cada peso, para luego modificar dichos pesos en la dirección que disminuya el error.

El verdadero potencial del *backpropagation* se manifiesta al combinarlo con herramientas de diferenciación automática, como *TensorFlow*⁷. Estas herramientas permiten definir cualquier arquitectura de red y función de pérdida, y se encargan de aplicar el algoritmo de *backpropagation* para ajustar automáticamente los pesos y minimizar el error en el conjunto de entrenamiento. Gracias a ello, modificar el número de capas, el número de neuronas por capa, las funciones de activación o incluso proponer nuevos tipos de neuronas resulta sencillo, ya que el motor de diferenciación se ocupa de los cálculos necesarios para ajustar los parámetros y reducir el error.

Además, al hacer un uso eficiente de la notación matricial y técnicas como el *broadcasting*, estas herramientas optimizan el modelo para ejecutarse en hardware especializado como las tarjetas gráficas (GPUs), que permiten cálculos paralelos a gran escala.

⁷Este es el entorno que emplearemos en nuestro proyecto.

Actualmente existen numerosos entornos para trabajar con redes neuronales artificiales, organizados en distintos niveles de abstracción:

- **Nivel 0:** Hardware diseñado para acelerar el entrenamiento y ejecución de redes neuronales, como las GPUs o las *Tensor Processing Units* (TPU) de Google.
- **Nivel 1:** Librerías y *drivers* que permiten sacar partido del hardware para cálculos distribuidos. Destaca la tecnología CUDA para las GPUs.
- **Nivel 2:** Librerías de diferenciación automática que facilitan la definición de modelos y la optimización de sus parámetros, como *TensorFlow Core*.
- **Nivel 3:** Librerías que incluyen implementaciones de distintos modelos de redes neuronales listas para su uso, apoyándose en las librerías de diferenciación automática. Un ejemplo destacado es *Keras* (la emplearemos en nuestro trabajo).
- **Nivel 4:** Interfaces gráficas que permiten a usuarios sin conocimientos de programación aplicar redes neuronales a problemas reales, como *Google AutoML*.

Más ejemplos de las herramientas de cada uno de los niveles se muestran en la figura 20.

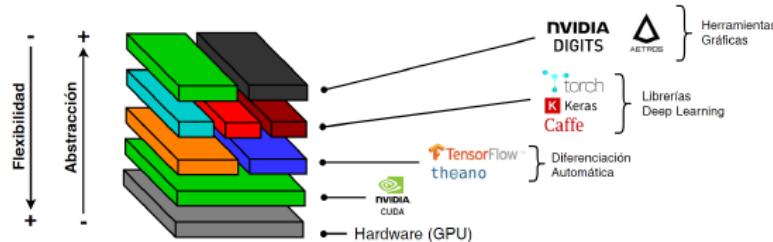


Figura 20: Herramientas para la implementación y uso de redes neuronales artificiales a diferentes niveles de abstracción.[13]

5.4. El paradigma del Deep-Learning

El último tipo de red neuronal artificial que abordaremos son las denominadas redes neuronales convolucionales profundas (*Deep Convolutional Neural Networks*, DCNN). Estas redes representan una de las arquitecturas más representativas dentro del ámbito del *deep learning*, y muchos de los modelos más avanzados actuales las integran junto con otras estructuras. Antes de profundizar en las DCNN, conviene repasar los conceptos clave que sustentan el enfoque del *deep learning*, lo que nos ayudará a comprender el origen de su éxito.

Aunque el desarrollo y la adopción del algoritmo de *backpropagation* en los años ochenta reavivaron el interés por las redes neuronales artificiales, el surgimiento de modelos como las máquinas de soporte vectorial hizo que las redes volvieran a perder protagonismo. Si bien el *backpropagation* permitía teóricamente entrenar redes con numerosas capas, en la práctica se observó que añadir más de dos o tres capas conducía a resultados insatisfactorios. En 1991, Hochreiter *et al.*[27] identificaron la causa principal: las funciones de activación habituales, como la tangente hiperbólica, generan gradientes en el intervalo $(-1, 1)$. Como el *backpropagation* aplica repetidamente la regla de la cadena, se terminan

multiplicando estos valores pequeños, lo que provoca que los gradientes disminuyan exponencialmente al aumentar la profundidad de la red. Este fenómeno, conocido como el *vanishing gradient problem*, ralentizaba el aprendizaje en las capas iniciales.

Por un lado, los intentos de entrenar redes más profundas resultaban ineficaces, y por otro, las redes poco profundas no lograban superar a los modelos alternativos en rendimiento. Esto llevó a un segundo periodo de desinterés en la investigación sobre redes neuronales. Sin embargo, como sabemos, hoy en día las redes neuronales viven un momento de esplendor, con una enorme presencia en aplicaciones comerciales y una comunidad científica muy activa. Esto plantea la pregunta de qué factores contribuyeron al renacimiento de las redes neuronales y al auge del *deep learning*.

El éxito del aprendizaje profundo no se debe a un único avance, sino a la combinación de múltiples factores que permitieron a las redes superar a otros modelos y facilitar su aplicación en entornos reales gracias a nuevas herramientas de *software* y *hardware*. Uno de los hitos iniciales fue el trabajo de Hinton *et al.*[19], quienes propusieron un preentrenamiento no supervisado capa a capa seguido de un ajuste fino mediante *backpropagation*. No obstante, hoy en día la mayoría de los modelos se entranan directamente con variantes optimizadas de *backpropagation*, gracias a la confluencia de varios elementos:

- **Avances en hardware.** El desarrollo de GPUs, impulsado en gran medida por la industria de los videojuegos, proporcionó el hardware ideal para realizar operaciones matriciales de forma paralela, lo que resultó fundamental para el entrenamiento eficiente de redes profundas y contribuyó a mitigar el *vanishing gradient problem*.
- **Disponibilidad de grandes conjuntos de datos.** La proliferación de dispositivos digitales y la digitalización masiva generaron volúmenes de datos sin precedentes, lo que permitió entrenar redes complejas sin que aparecieran problemas graves de *overfitting*, alcanzando además altos niveles de precisión.
- **Mejoras en los algoritmos.** El éxito del aprendizaje profundo se debe también al trabajo acumulado de numerosos investigadores que lograron optimizar diferentes aspectos de las redes. Entre las innovaciones más relevantes destacan técnicas como el *dropout*⁸ y la función de activación *ReLU*, que reduce el impacto del *vanishing gradient problem*.

Comprendido este contexto, podemos adentrarnos en los detalles de las principales arquitecturas del *deep learning*. La esencia de este enfoque es transformar progresivamente las entradas a lo largo de múltiples capas, generando representaciones cada vez más abstractas que faciliten la tarea a resolver.

Aunque podríamos pensar que una red neuronal profunda es simplemente una extensión del perceptrón multicapa, añadiendo más capas ocultas (representado en la figura 21), lo cierto es que el modelo más utilizado en el *deep learning* son las redes convolucionales, diseñadas específicamente para procesar datos estructurados como las imágenes.

⁸El *dropout* es una técnica que consiste en desactivar aleatoriamente un porcentaje de neuronas durante el entrenamiento, lo que ayuda a prevenir el sobreajuste y mejora la capacidad de generalización del modelo.

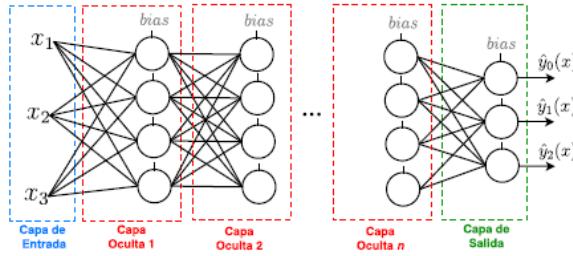


Figura 21: Arquitectura de un perceptrón multicapa con un número arbitrario de capas ocultas.[13]

5.5. Redes neuronales convolucionales (CNN)

Las redes estudiadas hasta este punto se componen de neuronas dispuestas en capas, donde cada neurona recibe como entrada el conjunto completo de características generadas por la capa previa y produce un único valor como salida. Por este motivo, cuando se quiere introducir una imagen en la red, es necesario transformar la matriz de intensidades de los píxeles en un vector unidimensional. Por ejemplo, una imagen en escala de grises de tamaño 28×28 debe representarse como un vector de 784 elementos para que pueda ser procesada por un perceptrón multicapa.

En cambio, las redes convolucionales (CNN, por sus siglas en inglés) están formadas por un tipo de neuronas diseñadas específicamente para trabajar con imágenes directamente, sin necesidad de convertirlas en vectores. Estas neuronas reciben una imagen como entrada y generan como salida otro conjunto de datos que se conoce como mapa de características. A diferencia de las neuronas tradicionales, cuyo funcionamiento se basa en un conjunto de pesos, las neuronas convolucionales emplean matrices denominadas filtros de convolución, que suelen ser bastante más pequeñas que las imágenes de entrada. Si tomamos una imagen I con c canales (por ejemplo, los tres correspondientes a RGB), cada uno se representa como una matriz bidimensional I_1, I_2, \dots, I_c . Una neurona convolucional aplicará sobre esta imagen un conjunto de c filtros, uno por canal, que llamaremos K_1, K_2, \dots, K_c , normalmente de tamaño 3×3 . Además, incluye un término de sesgo b , al igual que las neuronas estándar. La salida generada, un mapa de características, se expresa como una matriz M que se obtiene mediante:

$$M = g \left(b + \sum_c I_c \otimes K_c \right) \quad (44)$$

donde \otimes denota la operación de convolución, el *bias* b se suma a cada elemento de la matriz resultante del sumatorio de convoluciones y $g(\cdot)$ es la función de activación elegida que se aplica elemento a elemento sobre la matriz que recibe. En el contexto de las redes convolucionales, la función de activación más frecuentemente utilizada es la *rectified linear unit* o *relu*:

$$\text{relu}(x) = x^+ = \max(0, x) \quad (45)$$

A pesar de su aparente sencillez, la función de activación *relu* ha demostrado ser muy efectiva y tener numerosas ventajas, siendo hoy en día el estándar de facto en el campo de las redes convolucionales. Esta función retorna cero si la entrada es negativa, y retorna la entrada misma si es positiva. Una de sus principales ventajas es que introduce no linealidad sin saturar la salida (como lo hacen la sigmoide o tangente hiperbólica),

lo cual mitiga el problema del desvanecimiento del gradiente y acelera el aprendizaje. Sin embargo, puede sufrir del problema conocido como “*neurona muerta*”, cuando una neurona queda atascada en la región en la que su salida es siempre cero. La figura 22 muestra visualmente el efecto de esta función de activación.

Figura 22: Efecto de la función de activación *relu* sobre el resultado de una operación de convolución.[13]

En cuanto a la operación de convolución, esta consiste en ir aplicando un filtro sobre la matriz de entrada, desplazando el mismo por toda la superficie de la imagen para obtener la matriz de salida. En cada posición, se multiplican los valores del filtro por los valores de la matriz de entrada que quedan debajo del mismo, para luego sumarlos dando lugar a un único valor que se anota en la matriz de salida de la convolución. A medida que el filtro se desplaza sobre la imagen original, se van llenando los huecos hasta completar la matriz de salida (ver figura 23). Volviendo a la ecuación 44, vemos que cuando la imagen de entrada tiene varios canales, el filtro de convolución debe estar formado por un número igual de canales, y la salida de la operación es la suma de aplicar cada canal del filtro de convolución a cada canal de la imagen de entrada (ver figura 24).

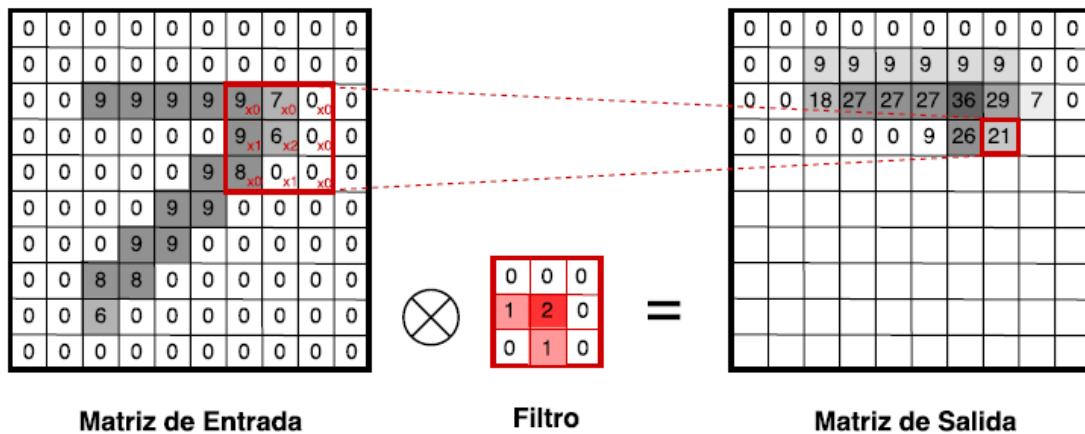


Figura 23: Ejemplo de operación de convolución actuando sobre una imagen de un solo canal. En este caso la matriz de entrada corresponde a una imagen de un dígito escrito a mano en escala de grises. Se usa un filtro 3×3 .[13]

Si nos fijamos bien, veremos que cuando el filtro se desplaza cerca de los bordes de la matriz de entrada, algunos de los elementos del filtro quedan fuera de la matriz. Existen diferentes opciones para manejar este inconveniente. Una opción es simplemente no desplazar el filtro tan lejos del centro de la imagen como para que los bordes del filtro salgan fuera de los límites de la matriz de entrada. Como resultado, la matriz de salida no tendrá las mismas dimensiones que las de entrada, sino que será ligeramente menor. Otra opción más frecuentemente utilizada consiste en considerar el espacio exterior de la matriz de entrada como lleno de ceros, de forma que cuando el filtro se aplique parcialmente fuera de la matriz, los valores faltantes serán simplemente completados con ceros, asegurando de esta forma que la matriz de salida tiene el mismo tamaño que la de entrada. A esta técnica se le conoce como *zero-padding* (ver figura 25).

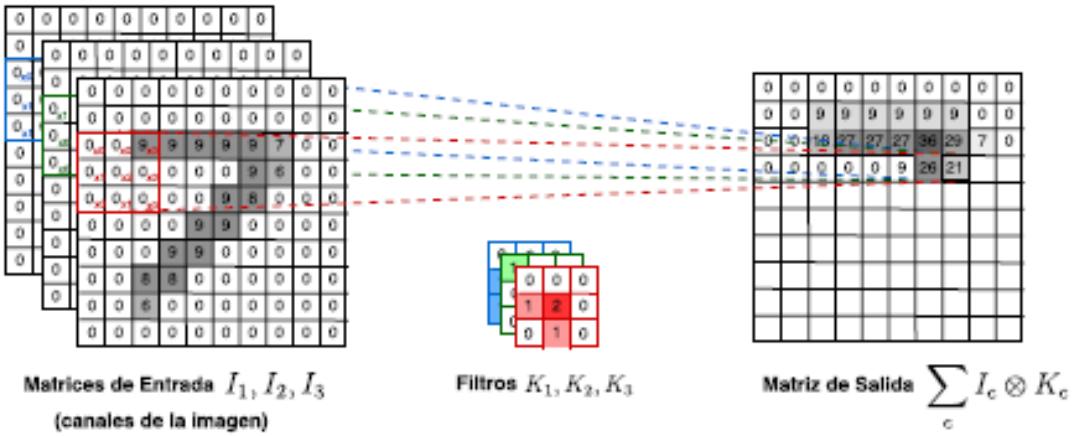


Figura 24: Ejemplo de operación de convolución actuando sobre varios canales. Se usan filtros 3×3 .[13]

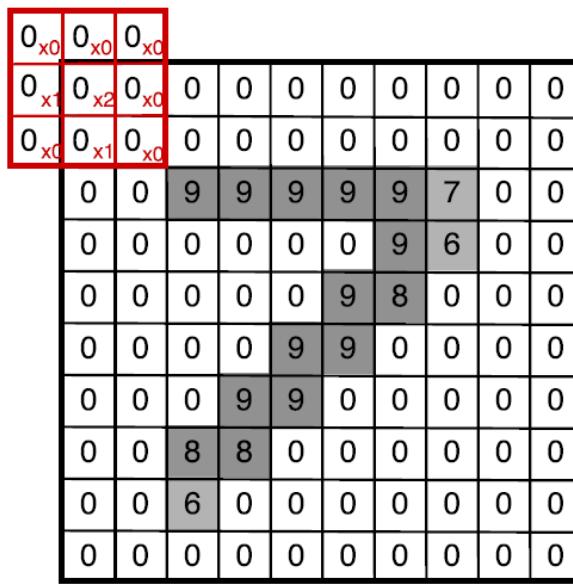


Figura 25: Ejemplo del funcionamiento del *zero-padding*. Cuando parte del filtro queda fuera de la matriz de entrada, los valores faltantes se consideran como ceros.[13]

Es posible considerar los mapas de características generados por una capa como los diferentes canales de una misma imagen y alimentar así la entrada de la siguiente capa convolucional. De esta forma, se van apilando las capas para formar una red convolucional profunda. Además de capas de convolución, se suele hacer uso de otros tipos de capas. En particular, se suelen intercalar las capas de neuronas convolucionales con las llamadas capas de *Pooling*.

El objetivo de las capas de tipo *pooling* es reducir de forma progresiva el tamaño espacial de las imágenes procesadas por la red a medida que progresan hacia las capas más profundas. De esta forma, se reduce la cantidad de parámetros que la red debe mantener y con ello el coste computacional. Además, la paulatina reducción de la dimensión espacial de las imágenes permite a la red extraer características de nivel creciente de abstracción, tal como describíamos al principio de la sección. Las capas de *pooling* operan de forma independiente en cada mapa de características generado por la capa anterior, redimensionándolos a un tamaño menor. Esto se consigue aplicando filtros especiales 2×2 con un *stride* de 2 unidades sobre los mapas de características, de forma que cada filtro devuelve

únicamente el valor máximo de los cuatro elementos sobre los que se mueve. Lo mejor para entender esta operación es observar un ejemplo visual (figura 26).

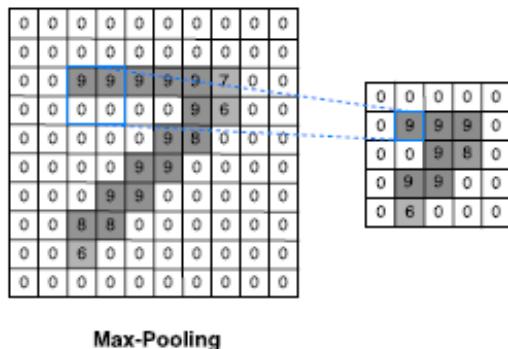


Figura 26: Efecto de una capa tipo *max-pooling* con *stride* (2,2) sobre un mapa de características individual.[13]

La red neuronal convolucional típica se construye alternando parejas de capas de convolución con capas de tipo *max-pooling*. Adicionalmente, las redes convolucionales suelen incluir algunas capas finales formadas por neuronas convencionales, que permiten usar la función de activación *softmax* ya estudiada para emitir predicciones acerca de la categoría a la que pertenecen las *imágenes de entrada*. En el argot del *deep learning* a este tipo de capas se les conoce como *capas densas* o *completamente conectadas*. La figura 27 muestra la arquitectura típica de este tipo de redes.

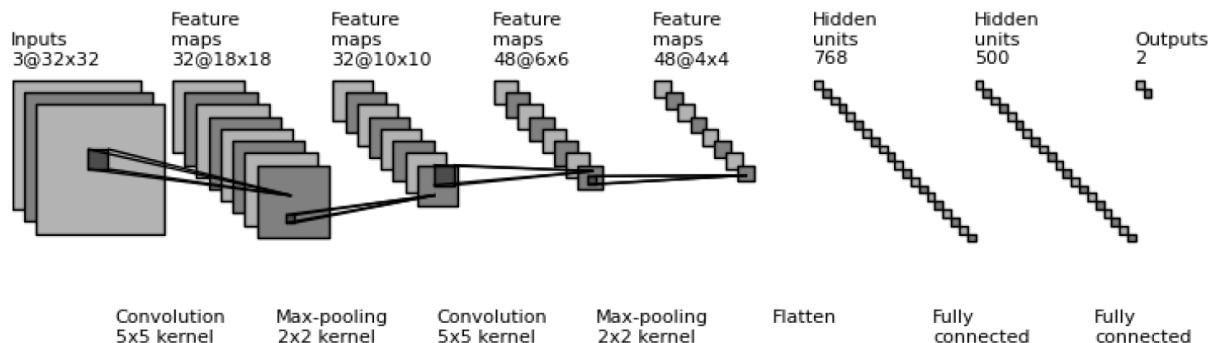


Figura 27: Arquitectura típica de una red neuronal convolucional.[13]

Si nos fijamos en la imagen, veremos que las anotaciones de la parte de arriba nos indican la salida de cada una de las capas del modelo. Para los mapas de características se usa la notación $3@32 \times 32$ para indicar que se tienen tres canales, cada uno de los cuales representado por una matriz 32×32 . Las anotaciones de la parte de abajo nos indican el tipo de capa que actúa en cada caso y sus hiperparámetros más importantes.

Finalmente, debemos mencionar una técnica que por su eficacia se ha vuelto esencial para entrenar de forma eficaz modelos del aprendizaje profundo: el *dropout*. Esta técnica busca evitar el sobre-entrenamiento de las redes, permitiendo una correcta generalización de los modelos. En esencia, la técnica de regularización *dropout* consiste en fijar a cero la salida de cada una de las neuronas elegidas al azar durante cada iteración del entrenamiento de la red neuronal (ver figura 28). Una vez finalizado el entrenamiento, estas neuronas vuelven a funcionar de forma habitual.

De esta forma, se previene la co-adaptación de unas neuronas con otras, manteniendo la generalidad del modelo (es decir, su capacidad de funcionar correctamente sobre datos no vistos durante su entrenamiento). La mayoría de librerías de redes neuronales artificiales nos permiten elegir si deseamos aplicar *dropout* en cada una de las capas de nuestros modelos, estableciendo en cada caso el porcentaje de neuronas a desactivar en cada iteración. Convenientemente, esta técnica puede aplicarse tanto sobre capas del tipo convolución/pooling como sobre capas de neuronas convencionales.

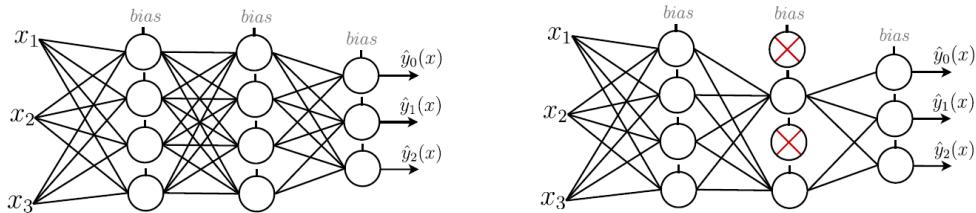


Figura 28: Representación visual del efecto de *dropout* en la segunda capa oculta de un perceptrón multi-capa.[13]

6. Red neuronal para decodificar los modos presentes en una superposición

Como se ha comentado al principio del trabajo, el fin de este es conseguir crear una red neuronal que a partir de imágenes (simuladas en este caso) en las cuales se superponen diferentes modos HG, concretamente desde el (0,0) hasta el (5,5), esta sea capaz de identificar los modos que componen cada imagen, ademas de su peso relativo en estas.

En la sección anterior se habla de los redes convolucionales profundas (DCNN) y del importante lugar que ocupan en la resolución de problemas con imágenes. Es por ello que las redes neuronales con las que se trabajará en este trabajo serán convolucionales.

6.1. Simulación

Como se acaba de mencionar, las imágenes que componen nuestro dataset son simuladas, por lo que tendremos que tener un programa responsable de su creación.

Como es sabido, en un laboratorio lo que se mide es la intensidad de los haces, así que nuestro dataset serán imágenes de la intensidad de nuestros haces. Luego la expresión importante con la que ha de trabajar nuestro programa es la de la ecuación (38):

$$I_{l,m}(x, y, z) = |A_{l,m}|^2 \left[\frac{W_0}{W(z)} \right]^2 G_l^2 \left[\frac{\sqrt{2}x}{W(z)} \right] G_m^2 \left[\frac{\sqrt{2}y}{W(z)} \right]. \quad (46)$$

La "base"de modos HG con la que se va a trabajar será la de la figura 29. Las imágenes de nuestro dataset serán superposiciones de estos modos.

Ejemplos de las superposiciones creadas serán dados posteriormente en sus apartados correspondientes.

6.1.1. Superposición de dos modos simples

En esta simulación, se generan imágenes a partir de modos Hermite-Gaussianos puros y superposiciones lineales de dos modos diferentes, $\psi_{l_1m_1}$ y $\psi_{l_2m_2}$, sin introducir un desfase relativo entre ellos.

La superposición se construye como:

$$\Psi(x, y) = w_1 \psi_{l_1m_1}(x, y) + w_2 e^{i\phi} \psi_{l_2m_2}(x, y) \quad (47)$$

donde:

- No se añade ningún desfase entre los dos modos ($\phi = 0$).
- Los coeficientes w_1 y w_2 son pesos reales positivos, generados aleatoriamente a partir de una distribución de Dirichlet.
- En el caso de los modos puros, se tiene $w_1 = 1$ y $w_2 = 0$.
- La intensidad de cada imagen se calcula como $I(x, y) = |\Psi(x, y)|^2$.

Algunos ejemplos se muestran en la figura 30⁹.

⁹Cabe mencionar que aunque veamos que $w_1^2 + w_2^2 \neq 1$, eso no implica que la intensidad no esté

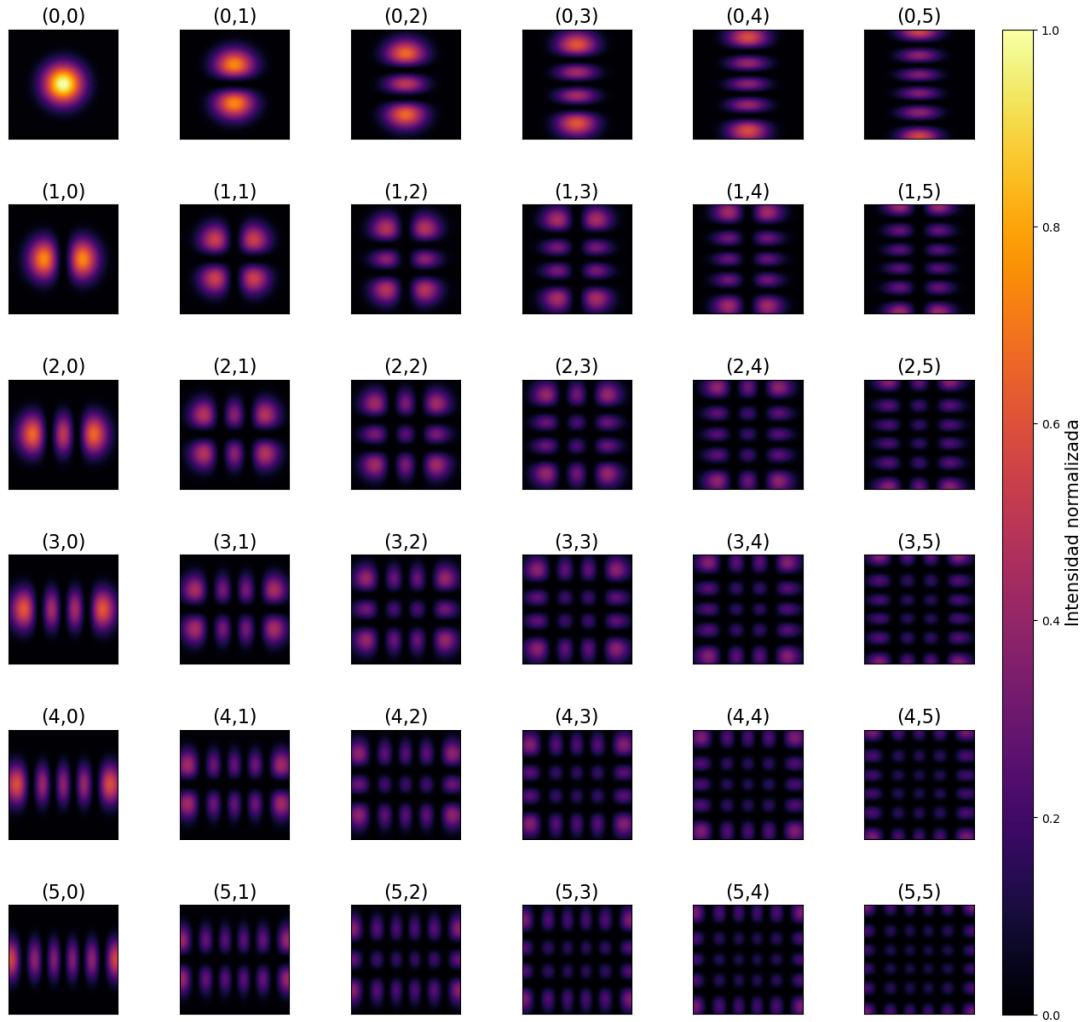


Figura 29: Modos HG del (0,0) al (5,5).

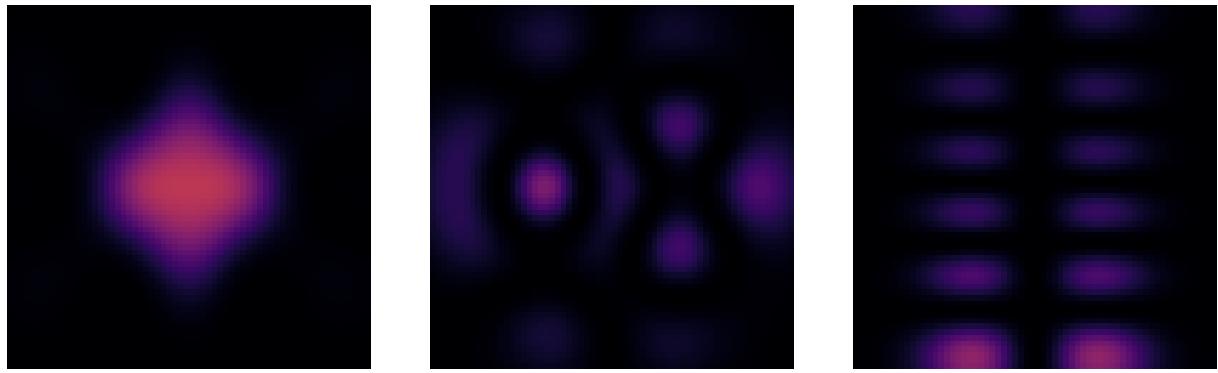
(a) (0,0) con $w_1=0.7960$ y (4,2) con $w_2=0.2040$.(b) (1,4) con $w_1=0.4436$ y (4,0) con $w_2=0.5564$.(c) (1,4) con $w_1=0.2255$ y (1,5) con $w_2=0.7745$.

Figura 30: Ejemplos superposición de dos modos HG.

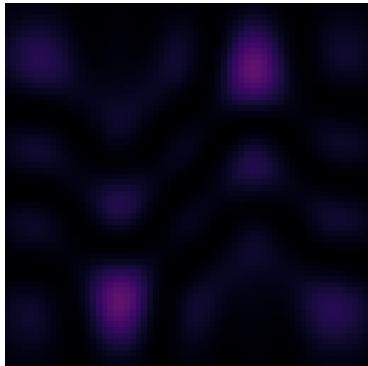
normalizada entre 0 y 1, pues a la hora de normalizar lo que se ha hecho es dividir entre la intensidad máxima de todas las superposiciones de nuestro dataset. Simplemente para los pesos se ha utilizado el criterio de normalización L1 ($w_1 + w_2 = 1$).

6.1.2. Superposición de dos modos desfasados

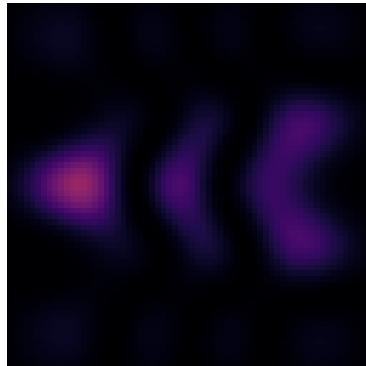
En la generación de las imágenes, se considera la superposición coherente de dos modos Hermite-Gaussianos, $\psi_{l_1 m_1}$ y $\psi_{l_2 m_2}$, con pesos w_1 y w_2 , y un desfase relativo ϕ aleatorio en el intervalo $[0, \pi]$. La superposición se construye de igual forma que la anterior (ecuación 47). Solo que ahora:

- Existe un desfase relativo entre los dos modos ($\phi \neq 0$). Este es elegido al azar para cada superposición entre $[0 : 2\pi]$.
- w_1 y w_2 son los pesos de cada modo, generados aleatoriamente según una distribución de Dirichlet,
- La intensidad de la imagen se calcula como $I(x, y) = |\Psi(x, y)|^2$ y se normaliza entre 0 y 1.

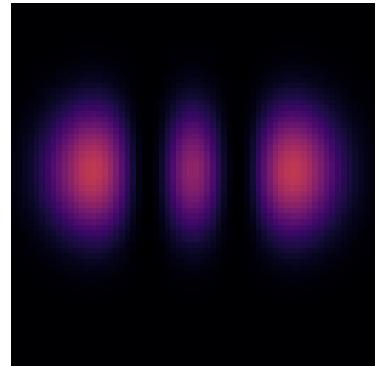
Se pueden ver algunos ejemplos en la figura 31.



(a) (1,2) con $w_1=0.4619$ y (4,3)
con $w_2=0.5381$, $\phi = 0,60$ rad.



(b) (2,0) con $w_1=0.6207$ y (3,4)
con $w_2=0.3793$, $\phi = 2,87$ rad.



(c) (2,0) con $w_1=0.8602$ y (2,1)
con $w_2=0.1398$, $\phi = 3,09$ rad.

Figura 31: Ejemplos superposición de dos modos HG desfasados.

6.1.3. Superposición de dos modos con ruido gaussiano

En esta simulación, se generan imágenes de superposiciones de modos Hermite-Gaussianos, pero además se introduce **ruido gaussiano** en las intensidades para simular condiciones más realistas de detección experimental.

La superposición se define al igual que en la ecuación (47) y en este caso:

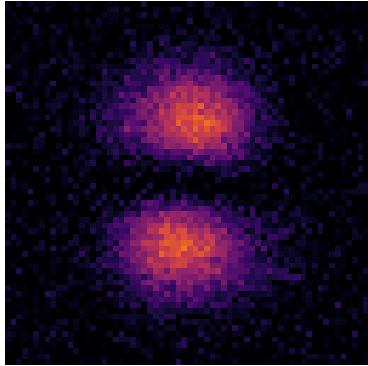
- No se añade desfase relativo entre los modos ($\phi = 0$).
- Los pesos w_1 y w_2 son números reales positivos, extraídos aleatoriamente de una distribución de Dirichlet.
- En los modos puros, $w_1 = 1$ y $w_2 = 0$.
- La intensidad se calcula como $I(x, y) = |\Psi(x, y)|^2$.

- A la intensidad $I(x, y)$ se le añade ruido gaussiano y se normaliza:

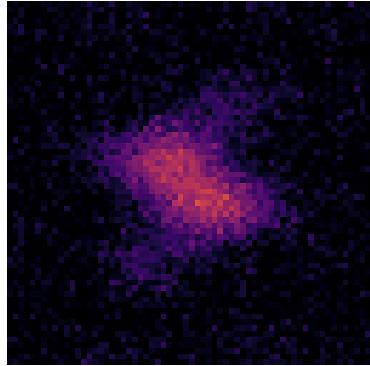
$$I_{\text{ruido}}(x, y) = \left(I(x, y) + \mathcal{N}(0, \sigma^2) \right) \in [0, 1]$$

donde $\mathcal{N}(0, \sigma^2)$ representa una distribución normal con media 0 y desviación estándar σ .

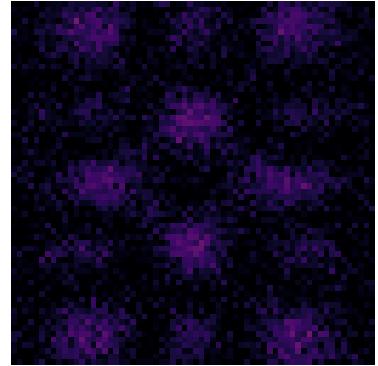
Se pueden ver algunos ejemplos con ruido gaussiano en las figuras 32 (con $\sigma = 0,05$), 33 (con $\sigma = 0,10$), 34 (con $\sigma = 0,15$) y 35 (con $\sigma = 0,20$).



(a) $(0,1)$ con $w_1=0.8985$ y $(3,0)$
con $w_2=0.1015$

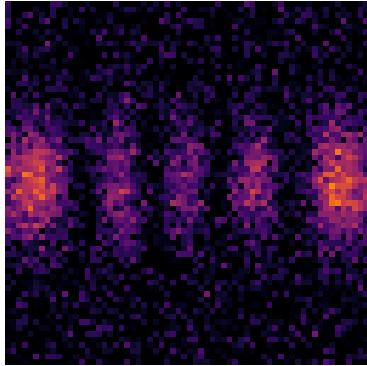


(b) $(0,0)$ con $w_1=0.6859$ y $(1,5)$
con $w_2=0.3141$

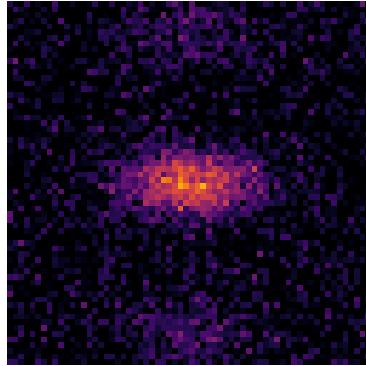


(c) $(0,0)$ con $w_1=0.3086$ y $(2,4)$
con $w_2=0.6914$

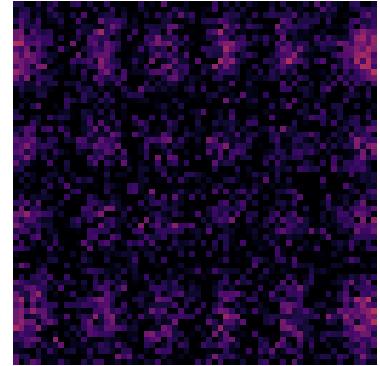
Figura 32: Ejemplos superposición de dos modos HG con ruido gaussiano de $\sigma = 0,05$.



(a) $(1,2)$ con $w_1=0.0534$ y $(4,4)$
con $w_2=0.9466$

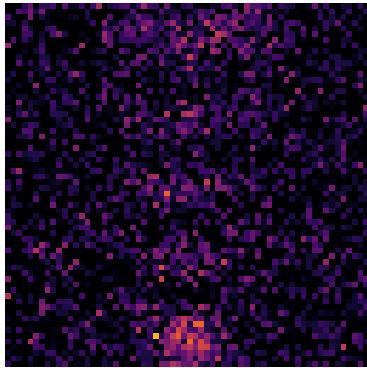


(b) $(0,0)$ con $w_1=0.5169$ y $(0,4)$
con $w_2=0.4831$

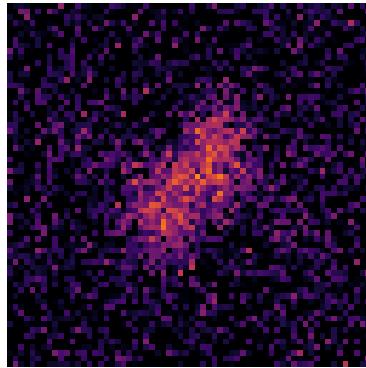


(c) $(0,3)$ con $w_1=0.0211$ y $(5,3)$
con $w_2=0.9789$

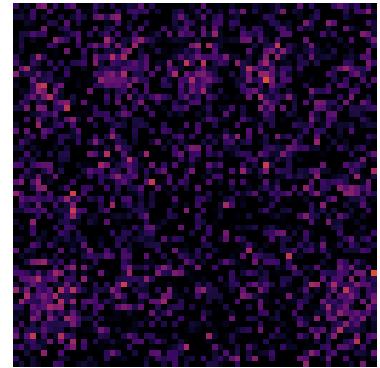
Figura 33: Ejemplos superposición de dos modos HG con ruido gaussiano de $\sigma = 0,10$.



(a) $(0,4)$ con $w_1=0.6649$ y $(4,5)$
con $w_2=0.3351$

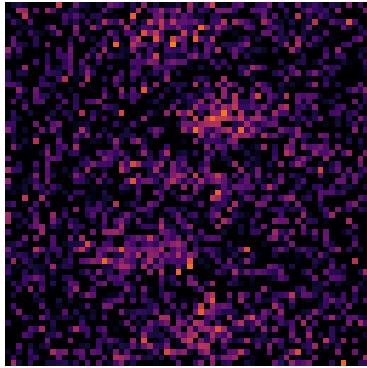


(b) $(0,0)$ con $w_1=0.6634$ y $(3,2)$
con $w_2=0.3366$

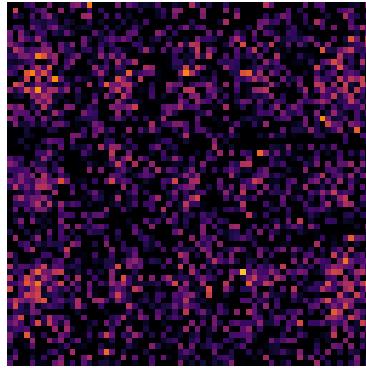


(c) $(2,3)$ con $w_1=0.3342$ y $(4,2)$
con $w_2=0.6658$

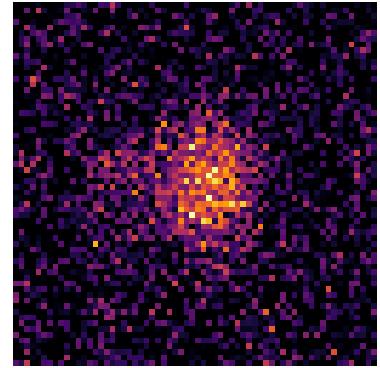
Figura 34: Ejemplos superposición de dos modos HG con ruido gaussiano de $\sigma = 0,15$.



(a) $(0,4)$ con $w_1=0.6634$ y $(1,1)$
con $w_2=0.3366$



(b) $(4,2)$ con $w_1=0.9221$ y $(5,0)$
con $w_2=0.0779$



(c) $(3,5)$ con $w_1=0.7864$ y $(5,0)$
con $w_2=0.2136$

Figura 35: Ejemplos superposición de dos modos HG con ruido gaussiano de $\sigma = 0,20$.

6.1.4. Superposición de más de dos modos

Se generan imágenes de superposiciones de tres modos Hermite-Gaussianos (HG) de órdenes $l, m \in \{0, 1, 2, 3, 4, 5\}$ sobre una malla 128×128 en $z = 0$. La superposición viene descrita por la expresión siguiente:

$$\Psi(x, y) = w_1 \psi_{l_1 m_1}(x, y) + w_2 \psi_{l_2 m_2}(x, y) + w_3 \psi_{l_3 m_3}(x, y) \quad (48)$$

- Se precomputan todos los modos HG posibles.
- Se calcula la intensidad $I(x, y)$ normalizada en el rango $[0, 1]$.
- Cada imagen es una superposición lineal de tres modos HG con pesos aleatorios.
- Las 10000 imágenes se guardan en escala de color global para todas ellas y resolución exacta 128×128 píxeles.

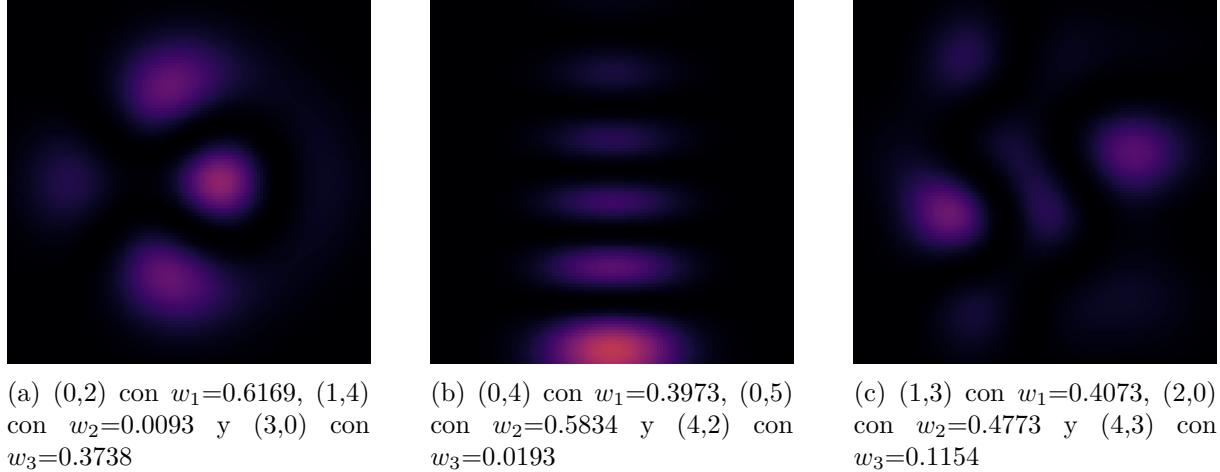


Figura 36: Ejemplos superposición de 3 modos HG.

6.2. Redes empleadas para cada tipo de superposición

6.2.1. Superposiciones de dos modos

La red empleada para todos los tipos de superposiciones de 2 modos (simples, desfasados y con ruido) ha sido la misma. El desfase no ha supuesto un gran problema para la red original (la creada para la superposición de 2 modos simples) y el ruido tampoco (a bajas escalas):

Esta red neuronal convolucional tiene como objetivo identificar combinaciones de dos modos Hermite-Gaussian (HG) a partir de imágenes en escala de grises (64×64) y estimar sus pesos relativos. La arquitectura y la función de pérdida están cuidadosamente diseñadas para respetar la simetría en la superposición de modos, penalizar predicciones inválidas y combinar clasificación con regresión. Esto se debe a cómo están definidas tanto la arquitectura de la red como la función de coste:

- Simetría en la superposición de modos: el formato de las etiquetas y el diseño de la función de pérdida respaldan explícitamente la simetría inherente al problema. Para cada par de modos (l_1, m_1) y (l_2, m_2) , la combinación se representa mediante un vector *one-hot* de 36 componentes, en el que los modos presentes se marcan con un 1 y los ausentes con un 0. Además, los pares de modos se ordenan de forma canónica (ascendente) antes de generar la etiqueta. Esto garantiza que la red no distinga combinaciones equivalentes como $(l_1, m_1) + (l_2, m_2)$ y $(l_2, m_2) + (l_1, m_1)$, evitando así redundancias en el aprendizaje.

- Penalización de predicciones inválidas: la función de pérdida incluye un término de penalización que castiga explícitamente la predicción de una superposición donde los dos modos sean idénticos con el fin de disuadir a la red de generar predicciones físicamente inconsistentes.
- Combinación de clasificación con regresión: la función de pérdida combina dos componentes principales: una entropía cruzada binaria que supervisa la clasificación de los modos presentes, mientras que el segundo es un error cuadrático medio para los pesos, ponderado para que solo se compute cuando los modos predichos coinciden con los reales (independientemente del orden). Además se equilibra la importancia relativa de los pesos frente a la clasificación de los modos (se le da menos importancia a un fallo en los pesos que a uno en los modos).

Características principales

- **Entrada:** 5000 imágenes 64x64 normalizadas.
- **Salida:** Vector de 38 elementos (36 para modos HG en one-hot y 2 para pesos relativos).
- **Etiquetas:** Generadas con formato one-hot para modos ordenados + pesos w_1, w_2 .
- **Pérdida personalizada:**
 - *Binary crossentropy*¹⁰ para clasificación de modos.
 - *Error cuadrático medio (MSE)* para pesos, pero sólo si los modos son correctos.
 - Penalización extra si la red predice el mismo modo dos veces.
- **Arquitectura**(ver figura 37):
 - 2 capas Conv2D + MaxPooling2D
 - Flatten → Dense(256) → Dropout(0.3)
 - Capa de salida: Dense(38, activation='sigmoid')
- **Optimizador:** Adam (LR=0.0001) con EarlyStopping con patience=10¹¹
- **División de datos:** 70 % entrenamiento, 10 % validación y 20 % test.
- **Evaluación:** Pérdida total y error medio absoluto (MAE) sobre conjunto de prueba.

Una visión más esquemática de la arquitectura de esta red puede verse en la figura 37.

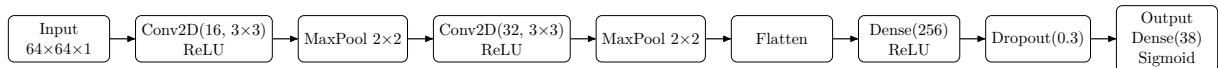


Figura 37: Esquema de la arquitectura de la red neuronal.

¹⁰Función de pérdida utilizada en clasificación binaria que mide la discrepancia entre las etiquetas reales (0 o 1) y las probabilidades predichas; penaliza más fuertemente cuanto más se aleja la predicción del valor verdadero.

¹¹Esto significa que, durante el entrenamiento, espera 10 épocas consecutivas sin mejora en la métrica monitoreada (por ejemplo la *pérdida en validación* o el *mae*) antes de parar el entrenamiento.

Proceso de elección de parámetros

- Tamaño del dataset y resolución: inicialmente se comenzó con un tamaño de 1000 imágenes de resolución 32x32. Dado que con este dataset tan pequeño no se pudo conseguir que el MAE bajase de un 0.2565 para el caso de menor complejidad (dos modos simples), fue necesario aumentar este a 5000 imágenes con una resolución de 64x64 para ver mejoras importantes en los valores de evaluación estadísticos (MAE baja a 0.0345)
- Etiquetas: realmente el formato *one-hot* fue el utilizado desde el principio puesto que este es el más común en este tipo de problemas de clasificación multietiqueta.
- Función pérdida: a la hora de crear la función pérdida esta fue modificándose considerando las distintas dificultades a superar para cada caso de forma iterativa, tratando de minimizar su propio valor siempre observando uno o dos ejemplos de los modos predichos y reales por pantalla así como la evolución del MAE y de la pérdida durante el entrenamiento y la validación para asegurarnos de que todo funcionaba como debía. Cabe mencionar que cada consideración incluida en la función pérdida para los casos de mayor complejidad (desfase y ruido) aplica también a las de menor complejidad, por lo tanto utilizando la misma red se disminuye aún más la pérdida en estos casos, resultando en poder utilizar una misma función pérdida para todos los casos considerados de dos modos. En qué consisten las dificultades mencionadas se explicó anteriormente.
- Arquitectura: en qué consiste cada tipo de capa se explica en sección 5. La arquitectura concreta de nuestra red fue elegida también de forma iterativa, probando con cuál conseguíamos un contraste óptimo entre tiempo de ejecución y error o pérdida.
- Optimizador: desde un principio se optó por el Adam por ser uno de los más utilizados y adaptarse bien a nuestro problema. En cuanto al *learning rate*, el valor elegido fue el que minimizaba el error y la pérdida sin suponernos un tiempo de ejecución exagerado ni *overfitting*
- División de los datos: el proceso de elección de los porcentajes fue el mismo que para los dos parámetros anteriores.

6.2.2. Superposición de más de dos modos

Esta red neuronal convolucional tiene como objetivo identificar combinaciones de tres modos Hermite-Gaussian (HG) a partir de imágenes en escala de grises (128x128) y estimar sus pesos relativos. La arquitectura y la función de pérdida están diseñadas para respetar la simetría de la superposición de modos, penalizar predicciones inválidas (como la repetición de modos) y combinar clasificación multiclas con regresión de pesos. La red busca maximizar tanto la exactitud de los modos predichos como la precisión de los pesos asignados a cada modo.

Características principales

- **Entrada:** 10000 imágenes 128x128 en escala de grises, normalizadas.
- **Salida:** Vector de 39 elementos: 36 para modos (*one-hot*) y 3 para pesos relativos.

- **Etiquetas:** modos representados en formato one-hot (36 clases) y pesos relativos w_1, w_2, w_3 como valores continuos.
- **Pérdida personalizada:** *Binary crossentropy* para clasificación de modos y *Mean Squared Error (MSE)* para pesos relativos.
- **Arquitectura** (una visión más esquemática es mostrada en la figura 38)
 - 3 bloques convolucionales:
 - Conv2D(32, kernel 3×3) → MaxPooling(2×2)
 - Conv2D(64, kernel 3×3) → MaxPooling(2×2)
 - Conv2D(128, kernel 3×3)
 - Capa *Flatten* →
 - Dense(128) + ReLU → Dropout(0.4)
 - Dense(128) + ReLU → Dropout(0.4)
 - Salidas:
 - Dense(36, activation='sigmoid') para modos.
 - Dense(3, activation='softmax') para pesos.
 - Concatenación de ambas salidas.

▪ Entrenamiento

- **Optimizador:** Adam (LR=0.0001) con *EarlyStopping* (patience=5).
- **División de datos:** 80 % entrenamiento, 10 % validación y 10 % test.
- **Evaluación:**
 - Error de clasificación (modo incorrecto).
 - Error de regresión (MAE) para pesos, sólo si los modos predichos son correctos.

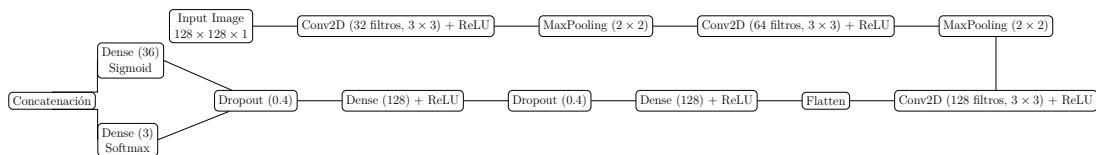


Figura 38: Esquema de la arquitectura de la red neuronal (disposición horizontal en zigzag).

Proceso de elección de parámetros

- Tamaño del dataset y resolución: inicialmente se comenzó con un tamaño de 5000 imágenes de resolución 64x64. De nuevo con este dataset (pequeño para la complejidad del problema) el MAE y la pérdida se quedaban estancados en un valor relativamente alto en comparación con el que obteníamos para los casos de dos modos. Debido a nuestra experiencia previa, recurrimos a aumentar este a 10000 imágenes con una resolución de 128x128 y en efecto se consiguieron mejoras en los valores de evaluación estadísticos aunque también aumentó el tiempo de entrenamiento, como era de esperar.

- Etiquetas: el motivo de la elección *one-hot* es el mismo que en el caso de la anterior red.
- Función pérdida: la función pérdida para esta red no cambia respecto a la otra más allá del hecho de que en esta se tienen que considerar tres modos y tres pesos.
- Arquitectura: la arquitectura concreta de nuestra red fue elegida de forma iterativa, probando con cuál conseguíamos un balance óptimo entre tiempo de ejecución y valor de MAE o pérdida.
- Optimizador: el criterio fue el mismo que para la red de los casos de combinación de dos modos.
- División de los datos: el proceso de elección de los porcentajes fue el mismo que para los dos parámetros anteriores.

El código utilizado en Python (tanto el de las simulaciones como el de las redes neuronales) se encuentra en el anexo que encontramos al final de este trabajo.

6.3. Resultados

6.3.1. Superposición de dos modos simples

Los resultados de la red sobre este conjunto de modos se muestran en el cuadro 1¹² y la evolución de la pérdida y del MAE en la figura 39. Por otra parte, en la figura 40 se muestran dos ejemplos de modos reales y predichos.

Pérdida total	MAE total	Accuracy modos	MAE pesos	Desv. pesos
0.0620	0.0245	0.9876	0.1466	0.2119

Cuadro 1: Resultados de nuestra red entrenada con un dataset formado por imágenes de superposiciones de 2 modos al resolver el conjunto reservado para evaluación.

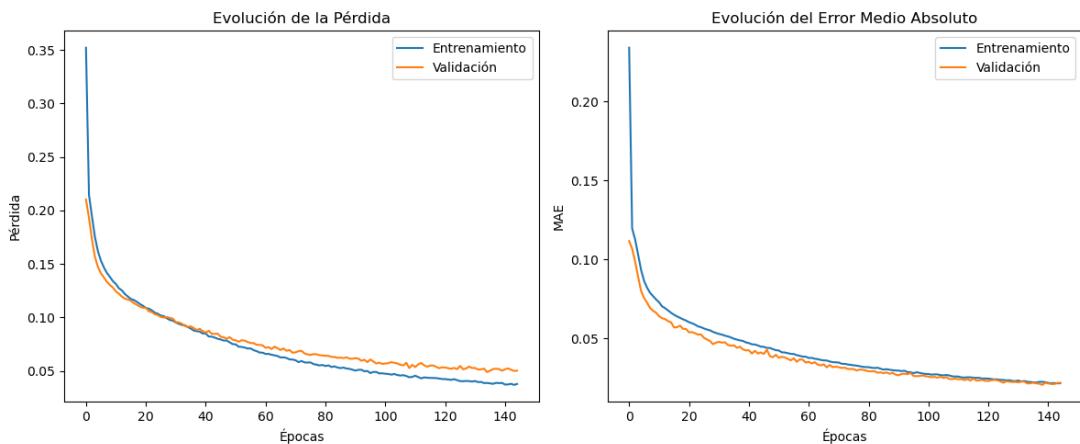
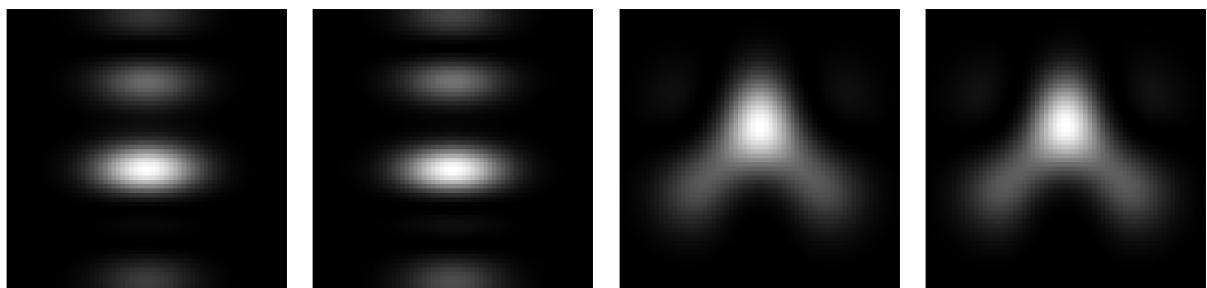


Figura 39: Evolución de las funciones pérdida y MAE a lo largo del entrenamiento sobre los conjuntos de entrenamiento y validación.



(a) (0,0) con $w_1=0.52$ y (b) (0,0) con $w_1=0.53$ y (c) (0,0) con $w_1=0.63$ y (d) (0,0) con $w_1=0.61$ y
(0,5) con $w_2=0.48$ (0,5) con $w_2=0.47$ (2,1) con $w_2=0.37$ (2,1) con $w_2=0.39$

Figura 40: En las imágenes 40a y 40c se muestran dos ejemplos aleatorios de superposiciones del dataset con sus modos y pesos. En 40b y 40d las simulaciones de los modos y pesos predichos por la red para cada uno de los ejemplos, respectivamente.

¹²El término "accuracy" nos da una idea del porcentaje de acierto. Siendo 1 el 100 %.

6.3.2. Superposición de dos modos desfasados

Los resultados de la red sobre este conjunto de modos se muestran en el cuadro 2 y la evolución de la pérdida y del MAE en la figura 41. Por otra parte, en la figura 42 se muestran dos ejemplos de modos reales y predichos.

Pérdida total	MAE total	Accuracy modos	MAE pesos	Desv. pesos
0.0770	0.0354	0.9835	0.1802	0.2455

Cuadro 2: Resultados de nuestra red entrenada con un dataset formado por imágenes de superposiciones de dos modos desfasados al resolver el conjunto reservado para evaluación.

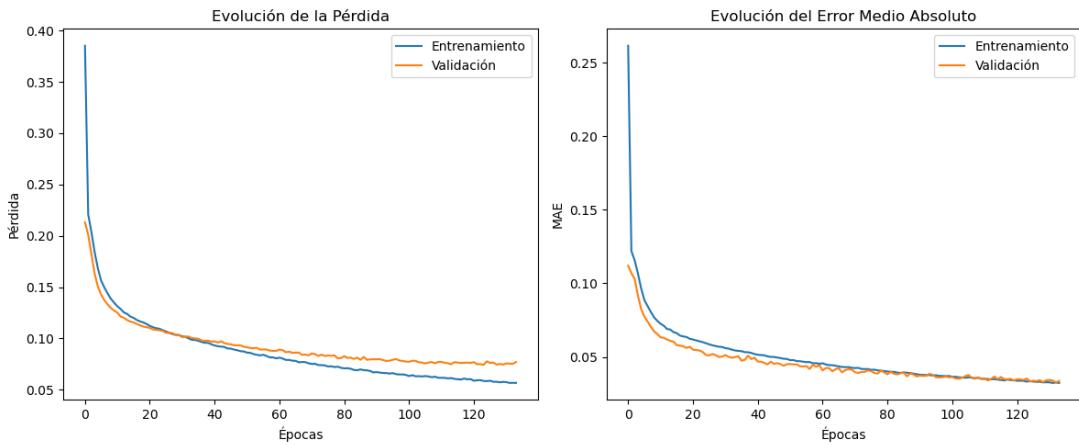
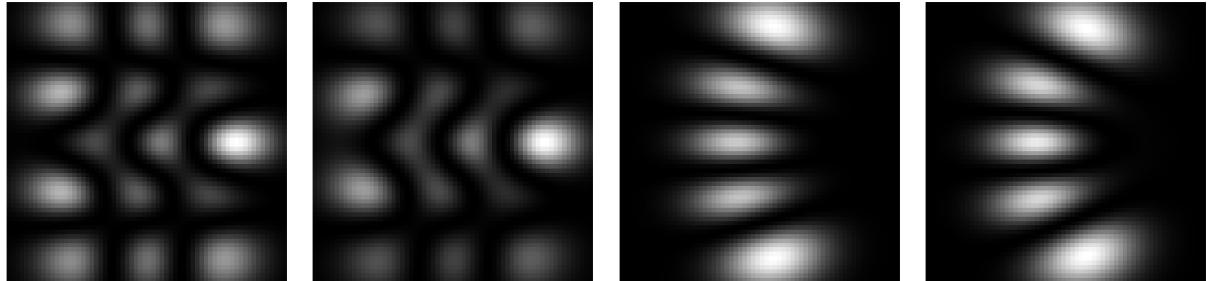


Figura 41: Evolución de las funciones pérdida y MAE a lo largo del entrenamiento sobre los conjuntos de entrenamiento y validación.



(a) (2,4) con $w_1=0.67$ y (b) (2,4) con $w_1=0.55$ y (c) (0,4) con $w_1=0.72$ y (d) (0,4) con $w_1=0.63$ y
(3,0) con $w_2=0.33$ (3,0) con $w_2=0.45$ (1,2) con $w_2=0.28$ (1,2) con $w_2=0.37$

Figura 42: En las imágenes 42a y 42c se muestran dos ejemplos aleatorios de superposiciones del dataset con sus modos y pesos. En 42b y 42d las simulaciones de los modos y pesos predichos por la red para cada uno de los ejemplos, respectivamente.

6.3.3. Superposición de dos modos con ruido

- $\sigma = 0,05$

Los resultados de la red sobre este conjunto de modos se muestran en el cuadro 3 y la evolución de la pérdida y del MAE en la figura 43. Por otra parte, en la figura ?? se muestran dos ejemplos de modos reales y predichos.

Pérdida total	MAE total	Accuracy modos	MAE pesos	Desv. pesos
0.0712	0.0315	0.9836	0.1726	0.2298

Cuadro 3: Resultados de nuestra red entrenada con un dataset formado por imágenes de superposiciones de dos modos con ruido gaussiano ($\sigma = 0,05$) al resolver el conjunto reservado para evaluación

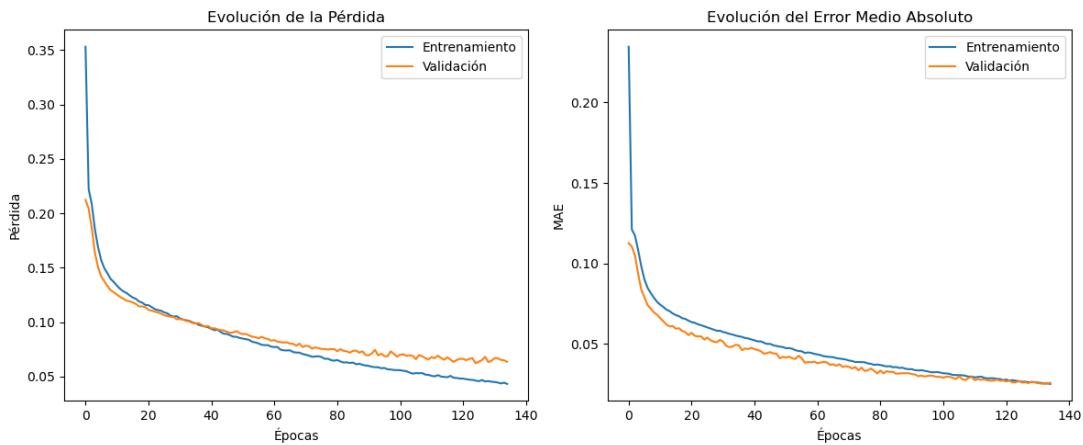


Figura 43: Evolución de las funciones pérdida y MAE a lo largo del entrenamiento sobre los conjuntos de entrenamiento y validación

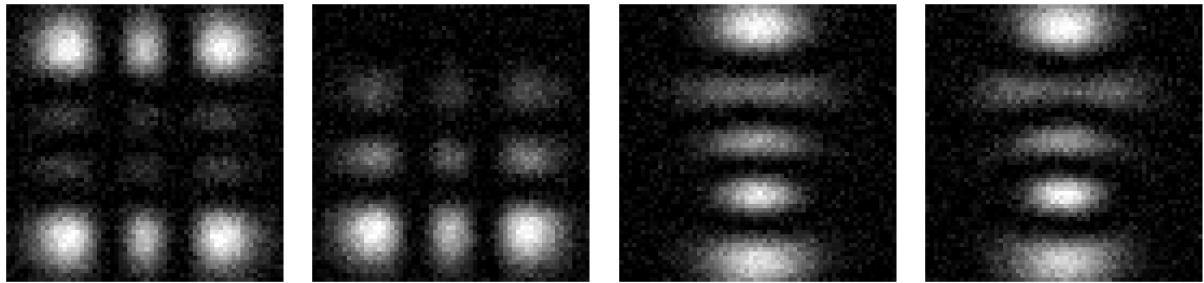


Figura 44: En las imágenes 44a y 44c se muestran dos ejemplos aleatorios de superposiciones del dataset con sus modos y pesos. En 44b y 44d las simulaciones de los modos y pesos predichos por la red para cada uno de los ejemplos, respectivamente.

- $\sigma = 0,10$

Los resultados de la red sobre este conjunto de modos se muestran en el cuadro 4 y la evolución de la pérdida y del MAE en la figura 45. Por otra parte, en la figura ?? se muestran dos ejemplos de modos reales y predichos.

Pérdida total	MAE total	Accuracy modos	MAE pesos	Desv. pesos
0.0972	0.0437	0.9768	0.1916	0.2400

Cuadro 4: Resultados de nuestra red entrenada con un dataset formado por imágenes de superposiciones de dos modos con ruido gaussiano ($\sigma = 0,10$) al resolver el conjunto reservado para evaluación

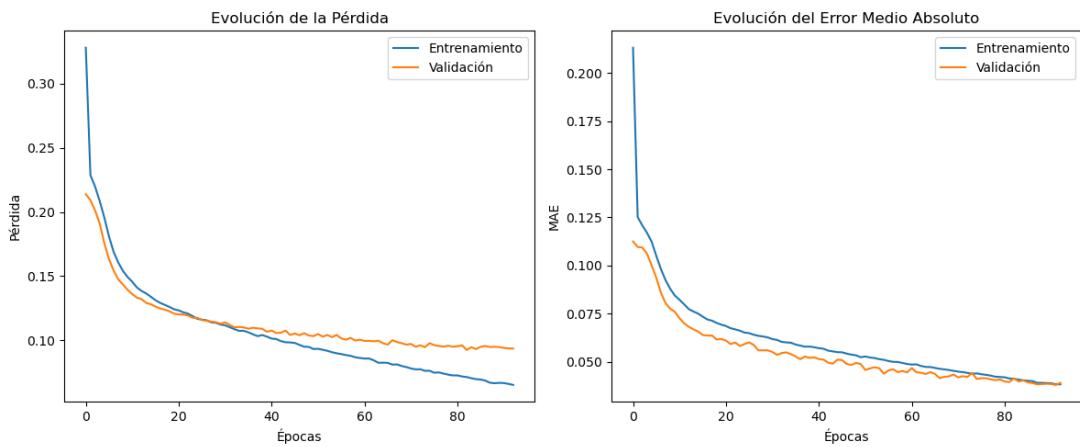
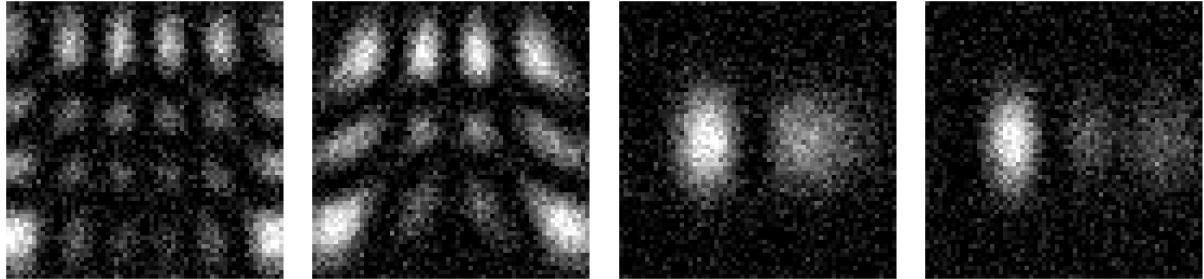


Figura 45: Evolución de las funciones pérdida y MAE a lo largo del entrenamiento sobre los conjuntos de entrenamiento y validación



(a) (5,3) con $w_1=0.75$ y (b) (5,3) con $w_1=0.36$ y (c) (1,0) con $w_1=0.84$ y (d) (1,0) con $w_1=0.64$ y
(3,2) con $w_2=0.25$ (3,2) con $w_2=0.64$ (4,0) con $w_2=0.16$ (4,0) con $w_2=0.36$

Figura 46: En las imágenes 46a y 46c se muestran dos ejemplos aleatorios de superposiciones del dataset con sus modos y pesos. En 46b y 46d las simulaciones de los modos y pesos predichos por la red para cada uno de los ejemplos, respectivamente.

- $\sigma = 0,15$

Los resultados de la red sobre este conjunto de modos se muestran en el cuadro 5 y la evolución de la pérdida y del MAE en la figura 47. Por otra parte, en la figura 48 se muestran dos ejemplos de modos reales y predichos.

Pérdida total	MAE total	Accuracy modos	MAE pesos	Desv. pesos
0.1104	0.0509	0.9751	0.2091	0.2542

Cuadro 5: Resultados de nuestra red entrenada con un dataset formado por imágenes de superposiciones de dos modos con ruido gaussiano ($\sigma = 0,15$) al resolver el conjunto reservado para evaluación

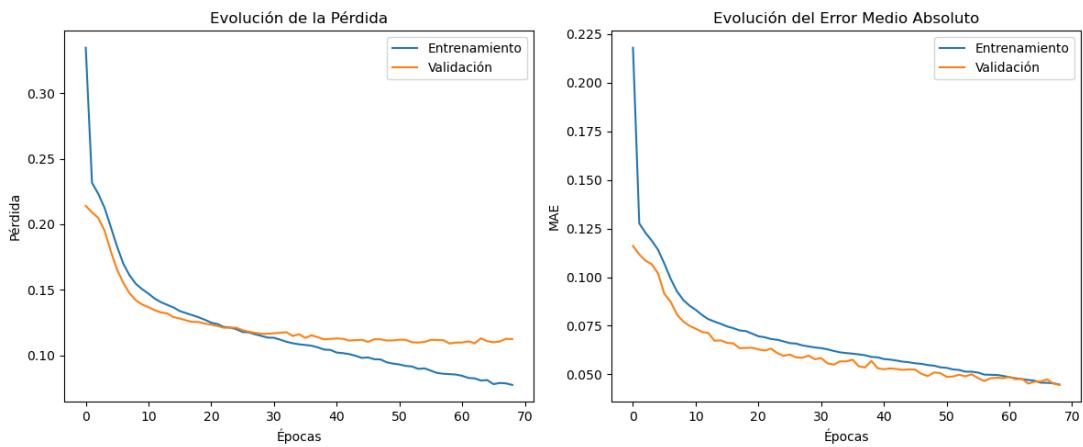


Figura 47: Evolución de las funciones pérdida y MAE a lo largo del entrenamiento sobre los conjuntos de entrenamiento y validación

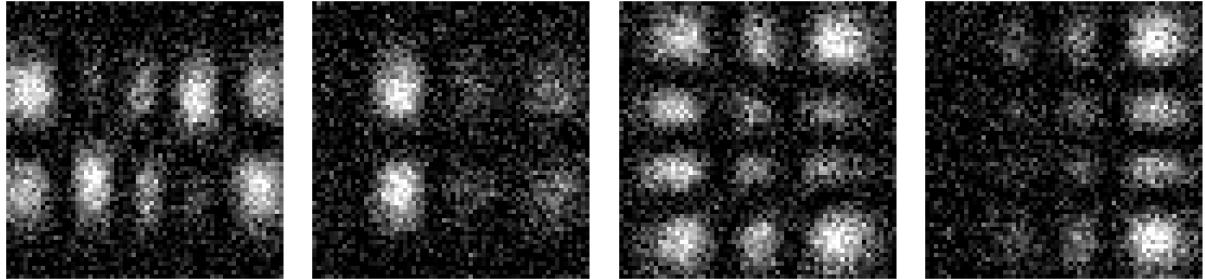


Figura 48: En las imágenes 48a y 48c se muestran dos ejemplos aleatorios de superposiciones del dataset con sus modos y pesos. En 48b y 48d las simulaciones de los modos y pesos predichos por la red para cada uno de los ejemplos, respectivamente.

- $\sigma = 0,20$

Los resultados de la red sobre este conjunto de modos se muestran en el cuadro 6 y la evolución de la pérdida y del MAE en la figura 49. Por otra parte, en la figura 50 se muestran dos ejemplos de modos reales y predichos.

Pérdida total	MAE total	Accuracy modos	MAE pesos	Desv. pesos
0.1172	0.0559	0.9706	0.2124	0.2606

Cuadro 6: Resultados de nuestra red entrenada con un dataset formado por imágenes de superposiciones de dos modos con ruido gaussiano ($\sigma = 0,20$) al resolver el conjunto reservado para evaluación

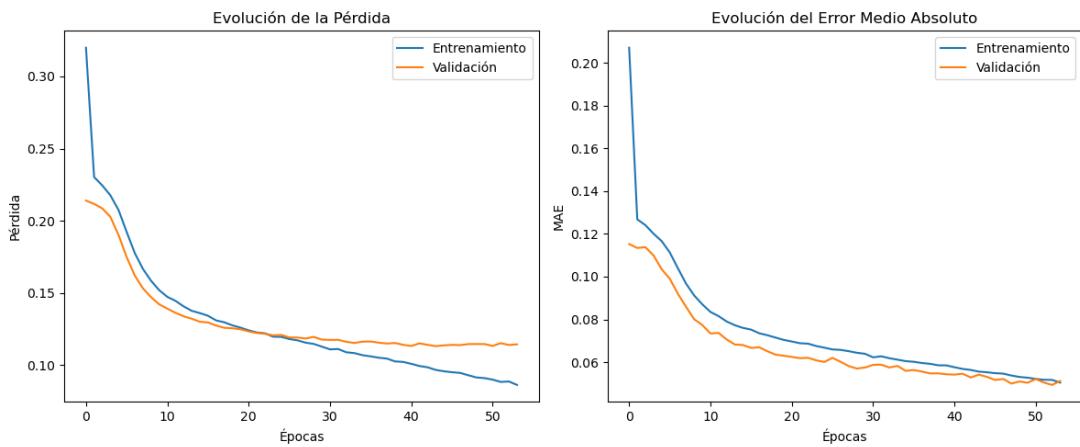
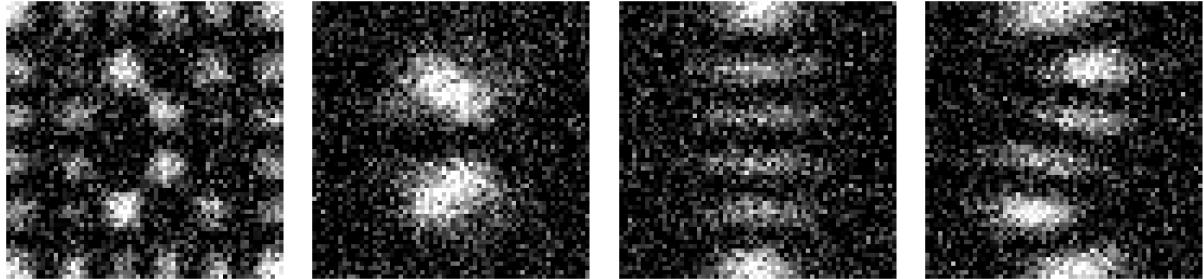


Figura 49: Evolución de las funciones pérdida y MAE a lo largo del entrenamiento sobre los conjuntos de entrenamiento y validación



(a) (5,5) con $w_1=0.75$ y (b) (5,5) con $w_1=0.26$ y (c) (0,5) con $w_1=0.84$ y (d) (0,5) con $w_1=0.69$ y (0,1) con $w_2=0.25$ (0,1) con $w_2=0.74$ (4,3) con $w_2=0.16$ (1,2) con $w_2=0.31$

Figura 50: En las imágenes 50a y 50c se muestran dos ejemplos aleatorios de superposiciones del dataset con sus modos y pesos. En 50b y 50d las simulaciones de los modos y pesos predichos por la red para cada uno de los ejemplos, respectivamente.

6.3.4. Superposición de más de dos modos

Los resultados de la red sobre este conjunto de modos se muestran en el cuadro 7 y la evolución de la pérdida y del MAE en la figura 51. Por otra parte, en la figura 52 se muestran dos ejemplos de modos reales y predichos.

Pérdida total	MAE total	Accuracy modos	MAE pesos	Desv. pesos
0.1406	0.0811	0.9652	0.1987	0.2376

Cuadro 7: Resultados de nuestra red entrenada con un dataset formado por imágenes de superposiciones de tres modos al resolver el conjunto reservado para evaluación

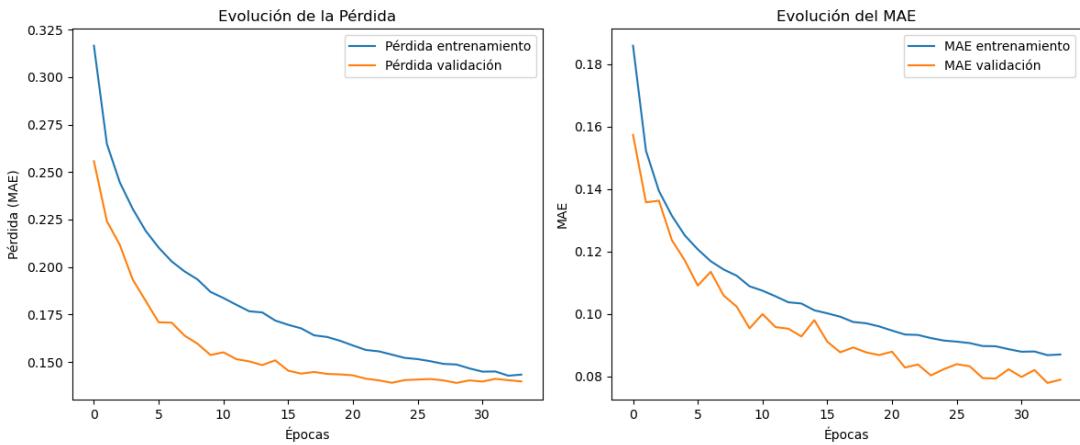
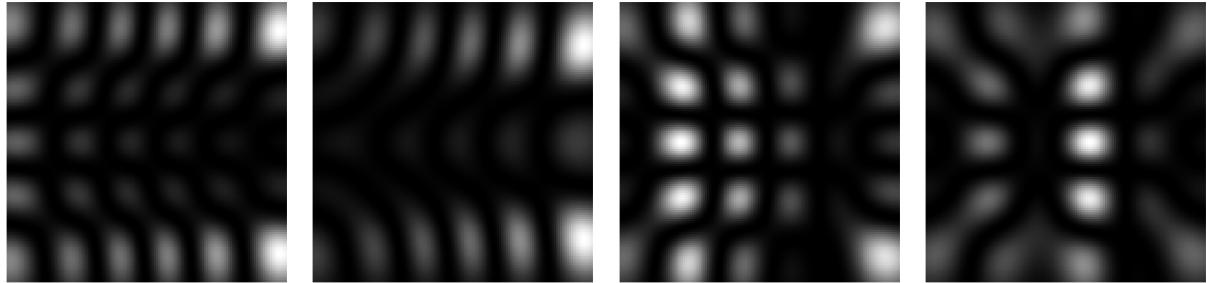


Figura 51: Evolución de las funciones pérdida y MAE a lo largo del entrenamiento sobre los conjuntos de entrenamiento y validación



(a) (4,2) con $w_1=0.22$, (b) (4,2) con $w_1=0.30$, (c) (2,4) con $w_1=0.30$, (d) (0,4) con $w_1=0.34$, (5,2) con $w_2=0.18$, (5,4) (5,2) con $w_2=0.39$, (5,4) (3,2) con $w_2=0.24$, (5,4) (3,2) con $w_2=0.30$, (5,4) con $w_3=0.60$ (b) con $w_3=0.31$ (c) con $w_3=0.46$ (d) con $w_3=0.36$

Figura 52: En las imágenes 52a y 52c se muestran dos ejemplos aleatorios de superposiciones del dataset con sus modos y pesos. En 52b y 52d las simulaciones de los modos y pesos predichos por la red para cada uno de los ejemplos, respectivamente.

7. Discusión de resultados

Como resumen de nuestros resultados para cada caso (tabla 8):

Casos	Pérdida	MAE	Accuracy modos	MAE pesos	Desv. pesos
2 modos	0.0620	0.0245	0.9876	0.1466	0.2119
2 modos desfasados	0.0770	0.0354	0.9835	0.1802	0.2455
2 modos ruido ($\sigma = 0.05$)	0.0712	0.0315	0.9836	0.1726	0.2298
2 modos ruido ($\sigma = 0.10$)	0.0972	0.0437	0.9768	0.1916	0.2400
2 modos ruido ($\sigma = 0.15$)	0.1104	0.0509	0.9751	0.2091	0.2542
2 modos ruido ($\sigma = 0.20$)	0.1172	0.0559	0.9706	0.2124	0.2606
3 modos	0.1406	0.0811	0.9652	0.1987	0.2376

Cuadro 8: Resultados de todos los casos estudiados.

En la Tabla 8, se resumen los principales indicadores de rendimiento de los casos considerados:

Los resultados obtenidos muestran un rendimiento sólido del modelo en la tarea de clasificación y regresión de modos Hermite–Gauss bajo distintas configuraciones. El caso más sencillo, correspondiente a la superposición de dos modos sin desfase ni ruido, ofrece los mejores resultados globales. En concreto, se alcanza una precisión de identificación de modos (*accuracy modos*) del 98.76 %, junto con los valores mínimos de pérdida (0.0620) y error absoluto medio (MAE) general (0.0245). Esto resulta completamente esperable, ya que se trata del escenario con menor complejidad estructural, donde los perfiles de intensidad son más distinguibles y menos afectados por ambigüedades internas.

Cuando se introduce un desfase entre los modos, la precisión disminuye levemente hasta el 98.35 % y el MAE aumenta hasta 0.0354. Esta degradación sugiere que el desfase introduce interferencias y patrones locales más complejos, lo que dificulta parcialmente la identificación precisa por parte de la red. A pesar de ello, el modelo mantiene un rendimiento muy aceptable, lo cual indica una buena capacidad de generalización.

Uno de los aspectos más relevantes en este estudio es el análisis del comportamiento de la red ante distintos niveles de ruido gaussiano. A medida que el parámetro σ del ruido se incrementa de 0.05 a 0.20, se observa una degradación progresiva del rendimiento. La *accuracy* de los modos desciende desde el 98.36 % hasta el 97.06 %, y el MAE global aumenta de 0.0315 a 0.0559. Asimismo, se aprecia un aumento tanto en el MAE de los pesos como en su desviación estándar, lo que indica que la red tiene más dificultades para estimar correctamente la contribución relativa de cada modo conforme se incrementa el nivel de ruido. Sin embargo, incluso con los niveles más altos de ruido considerados en este estudio, el modelo sigue arrojando resultados notablemente buenos, lo que pone de manifiesto su robustez. Aun así, se observa que a partir de $\sigma = 0.20$ la estructura de los perfiles de intensidad se degrada visualmente de tal forma que resulta prácticamente imposible distinguir los modos de forma fiable. Por tanto, no se considera útil analizar casos con niveles de ruido superiores a este umbral.

En cuanto al caso de superposición de tres modos, se trata del escenario con mayor complejidad modal entre los considerados. Aquí, la precisión baja hasta el 96.52 %, mientras que el MAE alcanza su valor más alto (0.0811) y la pérdida también se incrementa notablemente hasta 0.1406. Estos resultados, aunque todavía aceptables, reflejan que la red tiene mayores dificultades para descomponer correctamente la imagen de entrada en tres componentes modales distintas. Este aumento del error sugiere que, para abordar correctamente superposiciones de más de dos modos, podría ser necesario recurrir a arquitecturas más complejas o a conjuntos de entrenamiento significativamente más grandes.

De hecho, se ha comprobado que al pasar de dos a tres modos y duplicar tanto la resolución de las imágenes como el tamaño del conjunto de datos, se logra una mejora en el MAE en test de 0.0987 a 0.0811 y en la pérdida de 0.1659 a 0.1406, junto con un incremento en la precisión de 95.63 % a 96.52 %. Sin embargo, esta mejora implica también un aumento sustancial en el coste computacional: el tiempo de entrenamiento en un equipo sin GPU pasa de aproximadamente ~ 31 minutos a ~ 119 minutos. Esto plantea la cuestión de si merece la pena seguir aumentando la resolución o el tamaño del dataset, teniendo en cuenta la mejora relativamente modesta obtenida frente al coste computacional asociado.

Otra posible vía de mejora sería rediseñar la arquitectura de la red neuronal, aumentando el número de capas ocultas o el número de neuronas por capa, con el fin de que la red sea capaz de aprender representaciones más abstractas y detectar patrones más complejos. No obstante, al igual que en el caso anterior, esta modificación debe ser evaluada en función del coste computacional que implique y del beneficio real que pueda aportar en términos de precisión y generalización.

Por último, cabe mencionar que se intentó incluir en el estudio el caso de superposición de dos modos rotados aleatoriamente. Para ello, se consideraron únicamente las mitades triangulares de la matriz de modos HG hasta el índice (5,5), y se impuso un ángulo de rotación máximo de π para evitar confusiones debidas a simetrías intrínsecas de los modos. Sin embargo, los resultados obtenidos para estos casos fueron claramente insatisfactorios, incluso tras aumentar la resolución y el tamaño del conjunto de entrenamiento. En vista de ello, se decidió no incluir este caso en el análisis final del trabajo, ya que la red no fue capaz de aprender con fiabilidad la descomposición modal bajo estas condiciones.

En resumen, los resultados presentados ponen de manifiesto la efectividad del modelo en condiciones razonables de complejidad y ruido, a la vez que señalan con claridad los límites prácticos del enfoque propuesto. Si bien el rendimiento puede mejorarse marginalmente mediante ajustes en los datos o en la arquitectura, cualquier mejora debe valorarse cuidadosamente frente al incremento de recursos que conlleva.

Cabe mencionar también las nuevas perspectivas de futuro que aportan los resultados obtenidos, pues no solo validan la viabilidad del uso de redes neuronales convolucionales para la descomposición modal de haces Hermite–Gauss. Por un lado, la capacidad del modelo para mantener un rendimiento alto en presencia de ruido sugiere que podría ser integrado en sistemas experimentales reales, donde las condiciones no siempre son ideales. Por otro lado, el hecho de que incluso los casos más complejos —como la superposición de tres modos— puedan ser tratados con una precisión aceptable invita a explorar arquitecturas más avanzadas que puedan escalar mejor en complejidad sin comprometer la eficiencia computacional. Además, estos avances podrían extenderse a tareas más ambiciosas (por ejemplo la caracterización de haces en presencia de turbulencia atmosférica) abriendo así la puerta a aplicaciones en comunicaciones ópticas, metereología...

8. Conclusiones

Tras haber estudiado en profundidad la naturaleza de la óptica paraxial, sus ecuaciones y las soluciones a estas procedimos a centrarnos en una de ellas -los modos Hermite-Gauss (HG)- y utilizarla como base en la que codificar y decodificar un haz de luz a partir de su perfil de intensidad.

La finalidad del trabajo desde un principio fue utilizar técnicas de IA para decodificar estos perfiles de intensidad. Para ello mediante simulación numérica fue creado todo un dataset que luego fue dividido en distintos subconjuntos cada uno con una finalidad: entrenamiento, validación y test. Se crearon distintos modelos de redes: cada una para cada caso a estudiar. Tras múltiples pruebas y modificaciones (tanto en parámetros de la red como en su propia arquitectura) finalmente nos quedamos con dos redes (una para estudiar los casos de superposición de dos modos y otra para los de tres).

En general, los resultados demuestran que el modelo de red neuronal convolucional propuesto es capaz de identificar y descomponer perfiles de intensidad con una alta precisión en una amplia variedad de condiciones. El peor porcentaje de acierto de la red no baja del 96,5 % y el mejor supera el 98,7 %, correspondientes dichos porcentajes a los casos de mayor y menor complejidad, respectivamente.

Las ligeras caídas de rendimiento observadas en presencia de ruido o en escenarios de mayor complejidad confirman las limitaciones naturales del modelo, pero no comprometen su utilidad práctica. Incluso se demuestra que en superposiciones en las que podría parecer que careciera de sentido tratar de decodificar, la red lo hace con un porcentaje bastante bueno (vease el cuadro 6).

No hay que olvidar que aunque el porcentaje de acierto sea alto existe un error apreciable que ha de tenerse siempre en cuenta. De hecho, en este trabajo ha habido casos (como el de las superposiciones de modos rotados) que no hemos sido capaces de solventar al menos con un tamaño de dataset similar al de los otros casos y en tiempos similares.

A pesar de esto, se puede afirmar que los resultados respaldan firmemente el uso de técnicas de aprendizaje profundo como herramientas viables y eficaces para la caracterización automática de haces láser. En particular, demuestran que modelos de redes neuronales convolucionales (CNN) u otras arquitecturas de deep learning son capaces de extraer patrones espaciales complejos, identificar características del perfil del haz (como su intensidad, forma, simetría, o posibles aberraciones) y clasificar o predecir parámetros relevantes con alta precisión y eficiencia.

Este enfoque automatizado representa una alternativa poderosa frente a los métodos tradicionales, que suelen depender de procesamiento manual, ajustes ópticos delicados o análisis estadísticos convencionales, muchas veces más lentos y susceptibles a errores humanos. Además, la capacidad de estas redes para generalizar frente a variaciones en las condiciones experimentales, el ruido de adquisición o el tipo de fuente óptica, las convierte en herramientas especialmente útiles en entornos reales.

En el contexto de aplicaciones experimentales y de comunicación óptica, donde se requiere una caracterización precisa y rápida para mantener la calidad del sistema, la implementación de aprendizaje profundo no solo acelera los tiempos de análisis, sino que también abre la puerta a sistemas inteligentes capaces de ajustar parámetros en tiempo real, detectar fallos automáticamente o adaptarse dinámicamente a cambios en el entorno del haz.

Conclusions

After thoroughly studying the nature of paraxial optics, its equations, and their corresponding solutions, we proceeded to focus on one of them—Hermite-Gaussian (HG) modes—and used it as the foundation for encoding and decoding a light beam based on its intensity profile.

The objective of this work from the outset was to use AI techniques to decode these intensity profiles. To that end, a complete dataset was created through numerical simulation. This dataset was then divided into different subsets, each serving a specific purpose: training, validation, and testing. Several neural network models were developed, each tailored to a particular case under study. After numerous tests and modifications—both to the network parameters and to its architecture—we ultimately selected two models: one for analyzing cases involving the superposition of two modes, and another for three-mode superpositions.

In general, the results demonstrate that the proposed convolutional neural network model is capable of identifying and decomposing intensity profiles with high accuracy under a wide range of conditions. The lowest accuracy recorded did not fall below 96,5 %, while the highest exceeded 98,7 %, corresponding to the most complex and least complex scenarios, respectively.

The slight performance drops observed in the presence of noise or in more complex scenarios highlight the model's natural limitations but do not undermine its practical usefulness. It is even shown that in superpositions where decoding might appear meaningless, the network still performs with a fairly high success rate (see table 6).

It is important to remember that, although the accuracy percentage is high, there is a noticeable error that must always be taken into account. In fact, in this work, there were cases—such as the superpositions of rotated modes—that we were not able to resolve, at least not with a dataset size comparable to that of the other cases and within similar time constraints.

Nevertheless, It is fair to say that our results strongly support the use of deep learning techniques as viable and effective tools for the automatic characterization of laser beams. In particular, they demonstrate that convolutional neural network (CNN) models—or other deep learning architectures—are capable of extracting complex spatial patterns, identifying key features of the beam profile (such as intensity, shape, symmetry, or possible aberrations), and classifying or predicting relevant parameters with high precision and efficiency.

This automated approach represents a powerful alternative to traditional methods, which often rely on manual processing, delicate optical adjustments, or conventional statistical analysis—methods that are typically slower and more prone to human error. Furthermore, the ability of these networks to generalize across variations in experimental conditions, acquisition noise, or optical source types makes them especially useful in real-world environments.

In the context of experimental and optical communication applications, where fast and precise beam characterization is essential to maintain system quality, the implementation of deep learning not only speeds up analysis times but also opens the door to intelligent systems capable of real-time parameter adjustment, automatic fault detection, or dynamic adaptation to changes in the beam environment.

Referencias

- [1] A. E. Siegman, *Lasers*. University Science Books, 1986.
- [2] H. Luan, D. Lin, K. Li, W. Meng, M. Gu, and X. Fang, “768-ary lg-mode shift keying free-space optical communication based on convolutional neural networks,” *Optics Express*, vol. 29, pp. 19807–19818, 2021.
- [3] S. Avramov-Zamurovic, J. M. Esposito, and C. Nelson, “Classifying beams carrying orbital angular momentum with machine learning: tutorial,” *Journal of the Optical Society of America A*, vol. 40, pp. 64–77, 2023.
- [4] Z. Zhang, S. Zhao, W. He, Y. Gao, X. Wang, Y. Jie, X. Li, Y. Wang, and C. Zhao, “Hermite-gaussian-mode coherently composed states and deep learning based free-space optical communication link,” 2022.
- [5] Real Academia Española, “Definición de láser en el diccionario de la lengua española.” Disponible en <https://dle.rae.es/láser>, 2024.
- [6] Universal Laser Systems, “Historia del láser.” Available at {<https://www.ulssinc.com/es/conocer/historia-del-láser>}, note= {Accedido el 15 de junio de 2025}, s.f.
- [7] A. Einstein, “Zur quantentheorie der strahlung,” *Physikalische Zeitschrift*, vol. 18, pp. 121–128, 1917.
- [8] S. G. Lukishova, “Valentin a. fabrikant: Negative absorption, amplification of light and generation of electromagnetic radiation by quantum systems far from thermal equilibrium (a historical review),” *Journal of the European Optical Society - Rapid Publications*, vol. 5, p. 10045s, 2010.
- [9] G. Gould, “The laser: Light amplification by stimulated emission of radiation.” Laboratory notebook, Columbia University, 1957. Notarized notebook describing the concept of LASER, November 1957.
- [10] T. H. Maiman, “Stimulated optical radiation in ruby,” *Nature*, vol. 187, pp. 493–494, 1960.
- [11] H. R. Laboratories, “First working ruby laser developed.” Available at <https://www.hrl.com>, 1960. Accessed: 26 Feb. 2025.
- [12] H. Kogelnik and T. Li, “Laser beams and resonators,” *Applied Optics*, vol. 5, no. 10, pp. 1550–1567, 1966.
- [13] D. L. Sánchez, *Introducción a la Inteligencia Artificial. Aprendizaje Automático y Redes Neuronales*. Unpublished, Marzo 2021.
- [14] A. IA, “Diagrama de una red neuronal artificial.” Available at https://aprendeia.com/wp-content/uploads/2021/11/51592643870_7db15ffff6_o.png, 2021. Accessed: 26 Feb. 2025.
- [15] W. Pitts and W. McCulloch, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

- [16] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.
- [17] M. Minsky and S. Papert, *Perceptrons*. Cambridge, MA: MIT Press, 1969. Revised edition, 1988.
- [18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [19] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [20] T. A. I. Team, “Artificial intelligence index 2021 report.” Available at [urlhttps://aiindex.stanford.edu/wp-content/uploads/2021/05/2021-AI-Index-Spanish-Edition.pdf](https://aiindex.stanford.edu/wp-content/uploads/2021/05/2021-AI-Index-Spanish-Edition.pdf), 2021. Accessed: 2025-02-26.
- [21] B. E. Saleh and M. C. Teich, *Fundamentals of Photonics*. Hoboken, NJ: Wiley-Interscience, 2nd ed., 2007.
- [22] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1998.
- [23] B. Widrow, M. E. Hoff, *et al.*, “Adaptive switching circuits,” in *IRE WESCON Convention Record*, vol. 4, (New York), pp. 96–104, 1960.
- [24] S. Raschka, “Difference between a perceptron, adaline, and a single-layer neural network.” Available at <https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html>, n.d. Accessed: 2025-04-21.
- [25] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
- [26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [27] S. Hochreiter, “Vanishing gradients in multilayer neural networks,” Technical Report Technical Report FKI-222-91, Institut für Informatik, Technische Universität München, Munich, Germany, August 1991.

Anexo

En este anexo son mostrados de forma explícita los códigos utilizados para llevar a cabo este trabajo.

#SIMULACIÓN 2 MODOS SIMPLES

```
import numpy as np
import matplotlib.pyplot as plt
import os
from scipy.special import hermite
import scipy.special as sp
from itertools import combinations
import random
from skimage.io import imsave

# Crear carpeta de salida
output_folder = "superposiciones"
os.makedirs(output_folder, exist_ok=True)

# Parámetros
lambda_ = 632.8e-9
k = 2 * np.pi / lambda_
W0 = 1e-3
z_R = np.pi * W0**2 / lambda_

# Funciones auxiliares (se mantienen igual)
def A_lm(l, m, W_z):
    return np.sqrt(2 / (np.pi * W_z**2)) * (1 / (2**(l + m)
                                                   * sp.factorial(l)
                                                   * sp.factorial(m)))

def W(z):
    return W0 * np.sqrt(1 + (z / z_R)**2)

def R(z):
    return z * (1 + (z_R / z)**2) if z != 0 else np.inf

def xi(z):
    return np.arctan(z / z_R)

def hermite_gaussian(x, y, z, l, m):
    H_l = hermite(l)
    H_m = hermite(m)
    Wz = W(z)
    Alm = A_lm(l, m, Wz)
    G_l = H_l(np.sqrt(2) * x / Wz)
    G_m = H_m(np.sqrt(2) * y / Wz)
    fase = np.exp(-1j * k * z - 1j * k * (x**2 + y**2) / (2 * R(z))
                  + 1j * (l + m + 1) * xi(z))
    amplitud = Alm * G_l * G_m * np.exp(-(x**2 + y**2) / Wz**2)
    return amplitud * fase

def intensidad(hermite_gaussian, W_0, W_z):
    factor_W = (W_0 / W_z)**2
    return factor_W * (abs(hermite_gaussian))**2

# Crear malla de puntos
gridsize = 64
x_vals = np.linspace(-2e-3, 2e-3, gridsize)
y_vals = np.linspace(-2e-3, 2e-3, gridsize)
X, Y = np.meshgrid(x_vals, y_vals)
z = 0

# 1. Precomputar todos los modos HG para mayor eficiencia
modos = {}
```

```

for l in range(6):
    for m in range(6):
        modo = np.zeros(X.shape, dtype=complex)
        for i in range(X.shape[0]):
            for j in range(X.shape[1]):
                modo[i,j] = hermite_gaussian(X[i,j], Y[i,j], z, l, m)
        modos[(l,m)] = modo

# 2. Calcular intensidad máxima global (incluyendo modos puros y superposiciones)
print("Calculando intensidad máxima global...")
intensidad_maxima_global = 0

# Primero modos puros
for (l,m), modo in modos.items():
    I = intensidad(modo, W0, W(z))
    intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

# Luego combinaciones de modos (muestreo representativo)
for (l1,m1), (l2,m2) in combinations(modos.keys(), 2):
    for w1 in [0.2, 0.5, 0.8]: # Pesos representativos
        w2 = 1 - w1
        superposicion = w1 * modos[(l1,m1)] + w2 * modos[(l2,m2)]
        I = intensidad(superposicion, W0, W(z))
        intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

print(f"Intensidad máxima global: {intensidad_maxima_global}")

# 3. Generar 5000 imágenes con la estructura solicitada
total_imagenes = 5000
np.random.seed(42)

# Lista de todas las combinaciones únicas de 2 modos (630)
combinaciones_modos = list(combinations(modos.keys(), 2))
random.shuffle(combinaciones_modos) # Mezclar para aleatorizar

# Seleccionar 36 posiciones aleatorias para modos puros
posiciones_modos_puros = np.random.choice(total_imagenes, size=36, replace=False)

#guardado

for idx in range(total_imagenes):
    if idx in posiciones_modos_puros:
        # Modo puro
        modo_idx = np.where(posiciones_modos_puros == idx)[0][0]
        (l1,m1) = list(modos.keys())[modo_idx]
        (l2,m2) = (0,0)
        w1, w2 = 1.0, 0.0
    else:
        # Superposición
        combo_idx = (idx - 36) % len(combinaciones_modos)
        (l1,m1), (l2,m2) = combinaciones_modos[combo_idx]
        w1, w2 = np.random.dirichlet([1,1])

    superposicion = w1 * modos[(l1,m1)] + w2 * modos[(l2,m2)]
    I = intensidad(superposicion, W0, W(z)) / intensidad_maxima_global

    # Guardado con matplotlib (asegurando 64x64 exactos)
    filename = f"{idx:04d}_{l1}_{m1}_{w1:.4f}_{l2}_{m2}_{w2:.4f}.png"
    filepath = os.path.join(output_folder, filename)

    plt.figure(figsize=(0.64, 0.64), dpi=100) # Tamaño exacto para 64x64 píxeles

```

```
plt.imshow(I, cmap='inferno', vmin=0, vmax=1)
plt.axis('off')

# Ajustes críticos para eliminar bordes
plt.subplots_adjust(left=0, right=1, top=1, bottom=0)
plt.gca().set_axis_off()
plt.margins(0,0)
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())

plt.savefig(filepath, dpi=100, bbox_inches='tight', pad_inches=0)
plt.close()

print(f"Imagen {idx+1}/{total_imagenes} guardada: {filename}")
```

#SIMULACIÓN 2 MODOS DESFASADOS

```
import numpy as np
import matplotlib.pyplot as plt
import os
from scipy.special import hermite
import scipy.special as sp
from itertools import combinations
import random
from skimage.io import imsave

# Crear carpeta de salida
output_folder = "superposiciones"
os.makedirs(output_folder, exist_ok=True)

# Parámetros
lambda_ = 632.8e-9
k = 2 * np.pi / lambda_
W0 = 1e-3
z_R = np.pi * W0**2 / lambda_

# Funciones auxiliares (se mantienen igual)
def A_lm(l, m, W_z):
    return np.sqrt(2 / (np.pi * W_z**2)) * (1 / (2**(l + m)
                                                   * sp.factorial(l)
                                                   * sp.factorial(m)))

def W(z):
    return W0 * np.sqrt(1 + (z / z_R)**2)

def R(z):
    return z * (1 + (z_R / z)**2) if z != 0 else np.inf

def xi(z):
    return np.arctan(z / z_R)

def hermite_gaussian(x, y, z, l, m):
    H_l = hermite(l)
    H_m = hermite(m)
    Wz = W(z)
    Alm = A_lm(l, m, Wz)
    G_l = H_l(np.sqrt(2) * x / Wz)
    G_m = H_m(np.sqrt(2) * y / Wz)
    fase = np.exp(-1j * k * z - 1j * k * (x**2 + y**2) / (2 * R(z))
                  + 1j * (l + m + 1) * xi(z))
    amplitud = Alm * G_l * G_m * np.exp(-(x**2 + y**2) / Wz**2)
    return amplitud * fase

def intensidad(hermite_gaussian, W_0, W_z):
    factor_W = (W_0 / W_z)**2
    return factor_W * (abs(hermite_gaussian))**2

# Crear malla de puntos
gridsize = 64
x_vals = np.linspace(-2e-3, 2e-3, gridsize)
y_vals = np.linspace(-2e-3, 2e-3, gridsize)
X, Y = np.meshgrid(x_vals, y_vals)
z = 0

# 1. Precomputar todos los modos HG para mayor eficiencia
modos = {}
```

```

for l in range(6):
    for m in range(6):
        modo = np.zeros(X.shape, dtype=complex)
        for i in range(X.shape[0]):
            for j in range(X.shape[1]):
                modo[i,j] = hermite_gaussian(X[i,j], Y[i,j], z, l, m)
        modos[(l,m)] = modo

# 2. Calcular intensidad máxima global (incluyendo modos puros y superposiciones)
print("Calculando intensidad máxima global...")
intensidad_maxima_global = 0

# Primero modos puros
for (l,m), modo in modos.items():
    I = intensidad(modo, W0, W(z))
    intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

# Luego combinaciones de modos (muestreo representativo)
for (l1,m1), (l2,m2) in combinations(modos.keys(), 2):
    for w1 in [0.2, 0.5, 0.8]: # Pesos representativos
        w2 = 1 - w1
        superposicion = w1 * modos[(l1,m1)] + w2 * modos[(l2,m2)]
        I = intensidad(superposicion, W0, W(z))
        intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

print(f"Intensidad máxima global: {intensidad_maxima_global}")

# 3. Generar 5000 imágenes con la estructura solicitada
total_imagenes = 5000
np.random.seed(42)

# Lista de todas las combinaciones únicas de 2 modos (630)
combinaciones_modos = list(combinations(modos.keys(), 2))
random.shuffle(combinaciones_modos) # Mezclar para aleatorizar

# Seleccionar 36 posiciones aleatorias para modos puros
posiciones_modos_puros = np.random.choice(total_imagenes, size=36, replace=False)

#guardado

for idx in range(total_imagenes):
    if idx in posiciones_modos_puros:
        # Modo puro
        modo_idx = np.where(posiciones_modos_puros == idx)[0][0]
        (l1,m1) = list(modos.keys())[modo_idx]
        (l2,m2) = (0,0)
        w1, w2 = 1.0, 0.0
        phi = 0.0
        superposicion = modos[(l1,m1)]
    else:
        # Superposición con fase aleatoria
        combo_idx = (idx - 36) % len(combinaciones_modos)
        (l1,m1), (l2,m2) = combinaciones_modos[combo_idx]
        w1, w2 = np.random.dirichlet([1,1])
        phi = np.random.uniform(0, np.pi) # Fase aleatoria entre 0 y pi
        superposicion = w1 * modos[(l1,m1)] + w2 * np.exp(1j * phi) * modos[(l2,m2)]

    I = intensidad(superposicion, W0, W(z)) / intensidad_maxima_global

```

```
# Guardado con matplotlib (asegurando 64x64 exactos)
filename = f"{idx:04d}_{l1}_{m1}_{w1:.4f}_{l2}_{m2}_{w2:.4f}_phi{phi:.2f}.png"
filepath = os.path.join(output_folder, filename)

plt.figure(figsize=(0.64, 0.64), dpi=100)
plt.imshow(I, cmap='inferno', vmin=0, vmax=1)
plt.axis('off')
plt.subplots_adjust(left=0, right=1, top=1, bottom=0)
plt.gca().set_axis_off()
plt.margins(0,0)
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())

plt.savefig(filepath, dpi=100, bbox_inches='tight', pad_inches=0)
plt.close()

print(f"Imagen {idx+1}/{total_imagenes} guardada: {filename}")
```

#SIMULACIÓN 2 MODOS CON RUIDO

```
import numpy as np
import matplotlib.pyplot as plt
import os
from scipy.special import hermite
import scipy.special as sp
from itertools import combinations
import random
from skimage.io import imsave

# Crear carpeta de salida
output_folder = "superposiciones"
os.makedirs(output_folder, exist_ok=True)

# Parámetros
lambda_ = 632.8e-9
k = 2 * np.pi / lambda_
W0 = 1e-3
z_R = np.pi * W0**2 / lambda_

# Funciones auxiliares
def A_lm(l, m, W_z):
    return np.sqrt(2 / (np.pi * W_z**2)) * (1 / (2**(l + m)
                                                   * sp.factorial(l)
                                                   * sp.factorial(m)))

def W(z):
    return W0 * np.sqrt(1 + (z / z_R)**2)

def R(z):
    return z * (1 + (z_R / z)**2) if z != 0 else np.inf

def xi(z):
    return np.arctan(z / z_R)

def hermite_gaussian(x, y, z, l, m):
    H_l = hermite(l)
    H_m = hermite(m)
    Wz = W(z)
    Alm = A_lm(l, m, Wz)
    G_l = H_l(np.sqrt(2) * x / Wz)
    G_m = H_m(np.sqrt(2) * y / Wz)
    fase = np.exp(-1j * k * z - 1j * k * (x**2 + y**2) / (2 * R(z))
                  + 1j * (l + m + 1) * xi(z))
    amplitud = Alm * G_l * G_m * np.exp(-(x**2 + y**2) / Wz**2)
    return amplitud * fase

def intensidad(hermite_gaussian, W_0, W_z):
    factor_W = (W_0 / W_z)**2
    return factor_W * (abs(hermite_gaussian))**2

# Crear malla de puntos
gridsize = 64
x_vals = np.linspace(-2e-3, 2e-3, gridsize)
y_vals = np.linspace(-2e-3, 2e-3, gridsize)
X, Y = np.meshgrid(x_vals, y_vals)
z = 0

# Precomputar modos HG
modos = {}
```

```

for l in range(6):
    for m in range(6):
        modo = np.zeros(X.shape, dtype=complex)
        for i in range(X.shape[0]):
            for j in range(X.shape[1]):
                modo[i, j] = hermite_gaussian(X[i, j], Y[i, j], z, l, m)
        modos[(l, m)] = modo

# Calcular intensidad máxima global
print("Calculando intensidad máxima global...")
intensidad_maxima_global = 0

# Modos puros
for (l, m), modo in modos.items():
    I = intensidad(modo, W0, W(z))
    intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

# Superposiciones representativas
for (l1, m1), (l2, m2) in combinations(modos.keys(), 2):
    for w1 in [0.2, 0.5, 0.8]:
        w2 = 1 - w1
        superposicion = w1 * modos[(l1, m1)] + w2 * modos[(l2, m2)]
        I = intensidad(superposicion, W0, W(z))
        intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

print(f"Intensidad máxima global: {intensidad_maxima_global}")

# Generar imágenes
total_imagenes = 5000
np.random.seed(42)
combinaciones_modos = list(combinations(modos.keys(), 2))
random.shuffle(combinaciones_modos)
posiciones_modos_puros = np.random.choice(total_imagenes, size=36, replace=False)

for idx in range(total_imagenes):
    if idx in posiciones_modos_puros:
        # Modo puro
        modo_idx = np.where(posiciones_modos_puros == idx)[0][0]
        (l1, m1) = list(modos.keys())[modo_idx]

        # Evitar superposición consigo mismo en nombre y consistencia
        if (l1, m1) == (0, 0):
            (l2, m2) = (1, 0) # Cualquier otro modo, w2 = 0 así que no afecta
        else:
            (l2, m2) = (0, 0)

        w1, w2 = 1.0, 0.0
    else:
        # Superposición
        combo_idx = (idx - 36) % len(combinaciones_modos)
        (l1, m1), (l2, m2) = combinaciones_modos[combo_idx]
        w1, w2 = np.random.dirichlet([1, 1])

    superposicion = w1 * modos[(l1, m1)] + w2 * modos[(l2, m2)]
    I = intensidad(superposicion, W0, W(z)) / intensidad_maxima_global

    # Agregar ruido gaussiano
    ruido = np.random.normal(loc=0.0, scale=0.20, size=I.shape)
    I_ruido = np.clip(I + ruido, 0, 1)

```

```
# Guardar imagen
filename = f"{idx:04d}_{l1}_{m1}_{w1:.4f}_{l2}_{m2}_{w2:.4f}.png"
filepath = os.path.join(output_folder, filename)

plt.figure(figsize=(0.64, 0.64), dpi=100)
plt.imshow(I_ruido, cmap='inferno', vmin=0, vmax=1)
plt.axis('off')
plt.subplots_adjust(left=0, right=1, top=1, bottom=0)
plt.gca().set_axis_off()
plt.margins(0, 0)
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.savefig(filepath, dpi=100, bbox_inches='tight', pad_inches=0)
plt.close()

print(f"Imagen {idx+1}/{total_imagenes} guardada: {filename}")
```

#SIMULACIÓN 3 MODOS

```
import numpy as np
import matplotlib.pyplot as plt
import os
from scipy.special import hermite
import scipy.special as sp
from itertools import combinations
import random
from skimage.io import imsave

# Crear carpeta de salida
output_folder = "superposiciones"
os.makedirs(output_folder, exist_ok=True)

# Parámetros
lambda_ = 632.8e-9
k = 2 * np.pi / lambda_
W0 = 1e-3
z_R = np.pi * W0**2 / lambda_

# Funciones auxiliares
def A_lm(l, m, W_z):
    return np.sqrt(2 / (np.pi * W_z**2)) * (1 / (2**(l + m)
                                                   * sp.factorial(l)
                                                   * sp.factorial(m)))

def W(z):
    return W0 * np.sqrt(1 + (z / z_R)**2)

def R(z):
    return z * (1 + (z_R / z)**2) if z != 0 else np.inf

def xi(z):
    return np.arctan(z / z_R)

def hermite_gaussian(x, y, z, l, m):
    H_l = hermite(l)
    H_m = hermite(m)
    Wz = W(z)
    Alm = A_lm(l, m, Wz)
    G_l = H_l(np.sqrt(2) * x / Wz)
    G_m = H_m(np.sqrt(2) * y / Wz)
    fase = np.exp(-1j * k * z - 1j * k * (x**2 + y**2)
                  / (2 * R(z)) + 1j * (l + m + 1) * xi(z))
    amplitud = Alm * G_l * G_m * np.exp(-(x**2 + y**2) / Wz**2)
    return amplitud * fase

def intensidad(hermite_gaussian, W_0, W_z):
    factor_W = (W_0 / W_z)**2
    return factor_W * (abs(hermite_gaussian))**2

# Crear malla de puntos
gridsize = 128
x_vals = np.linspace(-2e-3, 2e-3, gridsize)
y_vals = np.linspace(-2e-3, 2e-3, gridsize)
X, Y = np.meshgrid(x_vals, y_vals)
z = 0

# 1. Precomputar todos los modos HG
modos = {}
```

```

for l in range(6):
    for m in range(6):
        modo = np.zeros(X.shape, dtype=complex)
        for i in range(X.shape[0]):
            for j in range(X.shape[1]):
                modo[i, j] = hermite_gaussian(X[i, j], Y[i, j], z, l, m)
        modos[(l, m)] = modo

# 2. Calcular intensidad máxima global
print("Calculando intensidad máxima global...")
intensidad_maxima_global = 0

for (l, m), modo in modos.items():
    I = intensidad(modo, W0, W(z))
    intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

for (l1, m1), (l2, m2), (l3, m3) in combinations(modos.keys(), 3):
    for pesos in [[0.2, 0.3, 0.5], [0.1, 0.4, 0.5], [1/3, 1/3, 1/3]]:
        w1, w2, w3 = pesos
        superposicion = w1 * modos[(l1, m1)] + w2 * modos[(l2, m2)]
        + w3 * modos[(l3, m3)]
        I = intensidad(superposicion, W0, W(z))
        intensidad_maxima_global = max(intensidad_maxima_global, np.max(I))

print(f"Intensidad máxima global: {intensidad_maxima_global}")

# 3. Generar imágenes
total_imagenes = 10000
np.random.seed(42)

# Combinaciones únicas de 3 modos (sin importar orden)
combinaciones_modos = list(set(
    tuple(sorted([a, b, c]))
    for a, b, c in combinations(modos.keys(), 3)
))
random.shuffle(combinaciones_modos)

# Selección aleatoria de 36 posiciones para insertar modos puros
posiciones_modos_puros = list(np.random.choice(total_imagenes, size=36,
                                                 replace=False))
random.shuffle(posiciones_modos_puros) # Esto rompe la correlación ordenada

# Selección aleatoria de 36 modos puros distintos
modos_puros_disponibles = sorted(modos.keys())
modos_puros_seleccionados = random.sample(modos_puros_disponibles, 36)

# Mapeo de índice a modo puro (l, m)
mapa_modos_puros = {
    idx: modos_puros_seleccionados[i]
    for i, idx in enumerate(posiciones_modos_puros)
}

for idx in range(total_imagenes):
    if idx in mapa_modos_puros:
        (l1, m1) = mapa_modos_puros[idx]
        w1 = 1.0

        if (l1, m1) == (0, 0):
            (l2, m2), w2 = (0, 2), 0.0
            (l3, m3), w3 = (0, 1), 0.0
        elif (l1, m1) == (0, 1):

```

```

        (l2, m2), w2 = (0, 0), 0.0
        (l3, m3), w3 = (0, 2), 0.0
    else:
        (l2, m2), w2 = (0, 0), 0.0
        (l3, m3), w3 = (0, 1), 0.0
else:
    combo_idx = (idx - len(mapa_modos_puros)) % len(combinaciones_modos)
    (l1, m1), (l2, m2), (l3, m3) = combinaciones_modos[combo_idx]
    w1, w2, w3 = np.random.dirichlet([1, 1, 1])

# Ordenar por (l, m)
modos_y_pesos = sorted(
    [((l1, m1), w1), ((l2, m2), w2), ((l3, m3), w3)],
    key=lambda x: x[0]
)
(l1, m1), w1 = modos_y_pesos[0]
(l2, m2), w2 = modos_y_pesos[1]
(l3, m3), w3 = modos_y_pesos[2]

# Crear superposición
superposicion = (
    w1 * modos[(l1, m1)] +
    w2 * modos[(l2, m2)] +
    w3 * modos[(l3, m3)]
)
I = intensidad(superposicion, W0, W(z)) / intensidad_maxima_global

# Guardar imagen
filename =
f"{idx:04d}_{l1}_{m1}_{w1:.4f}_{l2}_{m2}_{w2:.4f}_{l3}_{m3}_{w3:.4f}.png"
filepath = os.path.join(output_folder, filename)

plt.figure(figsize=(1.28, 1.28), dpi=100)
plt.imshow(I, cmap='inferno', vmin=0, vmax=1)
plt.axis('off')
plt.subplots_adjust(left=0, right=1, top=1, bottom=0)
plt.gca().set_axis_off()
plt.margins(0, 0)
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
plt.savefig(filepath, dpi=100, bbox_inches='tight', pad_inches=0)
plt.close()

print(f"Imagen {idx + 1}/{total_imagenes} guardada: {filename}")

```

```

#RED NEURONAL 2 MODOS
import os
import logging
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # 0=all, 1=no info, 2=no warnings
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
logging.getLogger('tensorflow').setLevel(logging.ERROR)
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow as tf

# 1. Función para generar las etiquetas en el nuevo formato
def generate_new_labels(labels):
    new_labels = []
    for label in labels:
        l1, m1, w1, l2, m2, w2 = label
        l1, m1, l2, m2 = int(round(l1*5)), int(round(m1*5)), int(round(l2*5)),
        int(round(m2*5))

        # --- Ordenar los modos para evitar redundancias ---
        model1 = (l1, m1)
        mode2 = (l2, m2)
        if model1 > mode2: # Ordenar por (l,m) ascendente
            model1, mode2 = mode2, model1
            w1, w2 = w2, w1 # Ajustar pesos correspondientes

        # Crear one-hot
        mode_vector = np.zeros(36)
        idx1 = model1[0] * 6 + model1[1]
        idx2 = mode2[0] * 6 + mode2[1]
        mode_vector[idx1] = 1.0
        mode_vector[idx2] = 1.0

        new_label = np.concatenate([mode_vector, [w1, w2]])
        new_labels.append(new_label)

    return np.array(new_labels)

# 2. Función de pérdida personalizada
def custom_loss(y_true, y_pred):
    # 1. Separar modos y pesos
    y_true_modes = y_true[:, :36]
    y_true_weights = y_true[:, 36:]
    y_pred_modes = y_pred[:, :36]
    y_pred_weights = y_pred[:, 36:]

    # 2. Pérdida estándar (clasificación + regresión)
    mode_loss = tf.keras.losses.binary_crossentropy(y_true_modes, y_pred_modes)

    # Solo calcular el error de pesos si los modos predichos son correctos
    top2_pred_idx = tf.argsort(y_pred_modes, axis=1)[:, -2:]
    top2_true_idx = tf.argsort(y_true_modes, axis=1)[:, -2:]

    # Comparar si los índices predichos y verdaderos coinciden (sin importar orden)
    def unordered_match(a, b):

```

```

    return tf.reduce_all(tf.sort(a, axis=-1) == tf.sort(b, axis=-1), axis=-1)

correct_modes = tf.cast(unordered_match(top2_pred_idx, top2_true_idx), tf.float32)

weight_loss = tf.keras.losses.MSE(y_true_weights, y_pred_weights)
weight_loss = correct_modes * weight_loss # solo cuenta si los modos son correctos

# 3. Penalización por superposición de un modo consigo mismo
top2_modes = tf.math.top_k(y_pred_modes, k=2).indices
l1_pred, m1_pred = top2_modes[:, 0] // 6, top2_modes[:, 0] % 6
l2_pred, m2_pred = top2_modes[:, 1] // 6, top2_modes[:, 1] % 6

same_mode = tf.cast(
    tf.logical_and(
        tf.equal(l1_pred, l2_pred),
        tf.equal(m1_pred, m2_pred)
    ),
    tf.float32
)

penalty = same_mode * 20.0

# 4. Combinar todas las pérdidas
return mode_loss + 0.5 * weight_loss + penalty

def mae_pesos(y_true, y_pred):
    y_true_weights = y_true[:, 36:] # Solo pesos
    y_pred_weights = y_pred[:, 36:] # Solo pesos
    return tf.reduce_mean(tf.abs(y_true_weights - y_pred_weights))

def std_metric(y_true,y_pred):
    y_true = y_true[:, 36:] # Solo pesos
    y_pred = y_pred[:, 36:] # Solo pesos
    error=y_true-y_pred
    mean_error=tf.reduce_mean(error)
    squared_diff=tf.square(error-mean_error)
    variance=tf.reduce_mean(squared_diff)
    std_dev=tf.sqrt(variance)
    return std_dev

def acc_modos(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
    true = tf.round(y_true[:, :36])
    pred = tf.round(y_pred[:, :36])
    return tf.reduce_mean(tf.cast(tf.equal(true, pred), tf.float32))

# 3. Carga y preparación de datos
data_folder = "superposiciones"
images = []
old_labels = []

for filename in os.listdir(data_folder):
    if filename.endswith(".png"):
        img = Image.open(os.path.join(data_folder, filename)).convert('L')
        images.append(np.array(img) / 255.0)

    parts = filename.replace(".png", "").split("_")
    old_labels.append([

```

```

        float(parts[1])/5, float(parts[2])/5, float(parts[3]),
        float(parts[4])/5, float(parts[5])/5, float(parts[6])
    ))
}

images = np.array(images)
old_labels = np.array(old_labels)
labels = generate_new_labels(old_labels)

# Separación original
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2,
random_state=42)

# Ahora saca validación desde el conjunto de entrenamiento
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.125,
random_state=42)
# → 0.125 = 0.125 * 0.8 = 0.1 total → Queda 70% train, 10% val, 20% test

X_train = X_train.reshape(-1, 64, 64, 1)
X_test = X_test.reshape(-1, 64, 64, 1)
X_val=X_val.reshape(-1,64,64,1)

# Visualización de imágenes aleatorias para ver como las procesa el programa
plt.figure(figsize=(15, 5))
num_samples = 5
random_indices = np.random.choice(len(X_train), num_samples, replace=False)

for i, idx in enumerate(random_indices):
    plt.subplot(1, num_samples, i+1)
    # Añade vmin y vmax para mantener la escala global
    plt.imshow(
        X_train[idx].reshape(64, 64),
        cmap='inferno',
        vmin=0,
        vmax=1 # Esto es crítico
    )
    plt.axis('off')

    true_modes = y_train[idx][:36]
    true_weights = y_train[idx][36:]
    true_idx = np.where(true_modes > 0.5)[0]
    l1, m1 = divmod(true_idx[0], 6)
    l2, m2 = divmod(true_idx[1], 6)
    w1, w2 = true_weights[0], true_weights[1]

    plt.title(f"({l1},{m1},{w1:.2f})\n({l2},{m2},{w2:.2f})", fontsize=8)

plt.suptitle("Ejemplos aleatorios (Escala global: 0-1)", y=1.05)
plt.tight_layout()
plt.show()

=====
from collections import Counter

# Ver combinaciones únicas de modos en las etiquetas originales e ignorar orden
combinations = []
for label in old_labels:
    l1, m1, _, l2, m2, _ = label
    mode1 = (int(round(l1*5)), int(round(m1*5)))
    mode2 = (int(round(l2*5)), int(round(m2*5)))
    combination = frozenset({mode1, mode2}) # Usar frozenset para ignorar orden

```

```

combinations.append(combination)

# Crear el contador
contador = Counter(combinations)

# 4. Modelo
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', padding='same', input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.3),
    #Dense(128, activation='relu'),
    #Dropout(0.4),
    Dense(38, activation='sigmoid')
])
model.compile(optimizer=Adam(learning_rate= 0.0001), loss=custom_loss, metrics=['mae',
acc_modos, mae_pesos, std_metric])

# Verifica que todas las imágenes usan la misma escala
print("Máximo en X_train:", np.max(X_train)) # Debe ser ~1.0
print("Mínimo en X_train:", np.min(X_train)) # Debe ser ~0.0

# 5. Entrenamiento por 200 épocas
history = model.fit(
    X_train, y_train,
    epochs=200, # Cambiar a 150 épocas o poner un call early stopping
    batch_size=32,
    validation_data=(X_val, y_val),
    callbacks=[EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
]

# 6. Evaluación del error medio total
test_loss, test_mae, ACCmodos, MAEpesos, DESVpesos = model.evaluate(X_test, y_test,
verbose=0)

print(f'\nPérdida total en el conjunto de prueba (pesos+modos): {test_loss:.4f}')
print(f'Error medio absoluto en el conjunto de prueba (pesos+modos): {test_mae:.4f}')
print(f'Acc modos en el conjunto de prueba: {ACCmodos:.4f}')
print(f'MAE pesos en el conjunto de prueba: {MAEpesos:.4f}')
print(f'Desviación estandar pesos en el conjunto de prueba: {DESVpesos:.4f}')

# 7. Gráficos
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Evolución de la Pérdida')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Entrenamiento')
plt.plot(history.history['val_mae'], label='Validación')
plt.title('Evolución del Error Medio Absoluto')
plt.xlabel('Épocas')
plt.ylabel('MAE')

```

```

plt.legend()
plt.tight_layout()
plt.show()

# 8. Función para visualizar ejemplos
def plot_examples(i1, i2):
    predictions = model.predict(X_test)

    fig, ax = plt.subplots(1, 2, figsize=(12, 6))

    for i, idx in enumerate([i1, i2]):
        ax[i].imshow(X_test[idx].reshape(64, 64), cmap='gray')
        ax[i].axis('off')

        # Procesar predicción
        pred_modes = predictions[idx][:36]
        pred_weights = predictions[idx][36:]
        top2_idx = np.argsort(pred_modes)[-2:]
        l1, m1 = divmod(top2_idx[1], 6)
        l2, m2 = divmod(top2_idx[0], 6)
        w1, w2 = pred_weights[0], pred_weights[1]
        w1, w2 = w1/(w1+w2), w2/(w1+w2)

        # Valores reales
        true_modes = y_test[idx][:36]
        true_weights = y_test[idx][36:]
        true_idx = np.where(true_modes > 0.5)[0]
        rl1, rm1 = divmod(true_idx[0], 6)
        rl2, rm2 = divmod(true_idx[1], 6)
        rw1, rw2 = true_weights[0], true_weights[1]

        ax[i].text(2, 60, f'Real: ({rl1},{rm1},{rw1:.2f}) ({rl2},{rm2},{rw2:.2f})',
                   color='yellow', fontsize=10, bbox=dict(facecolor='black', alpha=0.5))
        ax[i].text(2, 56, f'Pred: ({l1},{m1},{w1:.2f}) ({l2},{m2},{w2:.2f})',
                   color='red', fontsize=10, bbox=dict(facecolor='black', alpha=0.5))

        print(f"Imagen {idx} - Real: ({rl1},{rm1},{rw1:.2f}) ({rl2},{rm2},{rw2:.2f})")
        print(f"Predicción: ({l1},{m1},{w1:.2f}) ({l2},{m2},{w2:.2f})\n")

    plt.show()

# 9. Ejemplos de visualización
plot_examples(58, 199)

```

#RED NEURONAL 3 MODOS

```

import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import load_img, img_to_array
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
import tensorflow.keras.backend as K

optimizer = Adam(learning_rate=0.001)

# -----
# CONFIG
# -----
IMG_SIZE = 128 #128 el bueno
DATASET_PATH = "superposiciones"
EPOCHS = 100
BATCH_SIZE = 32

# -----
# FUNCIONES DE UTILIDAD
# -----
def parse_filename(filename):
    name = os.path.splitext(filename)[0]
    parts = name.split('_')[1:]

    mode_mask = np.zeros(36)
    weights = []

    seen_modes = set()

    for i in range(0, 9, 3):
        l = int(parts[i])
        m = int(parts[i+1])
        w = float(parts[i+2])

        if not (0 <= l <= 5 and 0 <= m <= 5):
            raise ValueError(f"Modo fuera de rango permitido (0-5): ({l}, {m}) en {filename}")
        if (l, m) in seen_modes:
            raise ValueError(f"Modo repetido ({l}, {m}) en {filename}")
        seen_modes.add((l, m))

        idx = l * 6 + m # Modo (l,m) se codifica como indice 6*l + m
        mode_mask[idx] = 1.0
        weights.append(w)

    return np.concatenate([mode_mask, weights])

def load_data(path):
    X, y = [], []
    for fname in os.listdir(path):
        if fname.endswith('.png'):

```

```

        img = load_img(os.path.join(path, fname), color_mode='grayscale',
        target_size=(IMG_SIZE, IMG_SIZE))
        img = img_to_array(img) / 255.0
        X.append(img)
        label = parse_filename(fname)
        y.append(np.array(label).flatten())
    return np.array(X, dtype=np.float32), np.array(y, dtype=np.float32)

def mae_pesos(y_true, y_pred):
    y_true_weights = y_true[:, :36] # Solo pesos
    y_pred_weights = y_pred[:, :36] # Solo pesos
    return tf.reduce_mean(tf.abs(y_true_weights - y_pred_weights))

def acc_modos(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
    true = tf.round(y_true[:, :36])
    pred = tf.round(y_pred[:, :36])
    return tf.reduce_mean(tf.cast(tf.equal(true, pred), tf.float32))

def std_metric(y_true,y_pred):
    y_true = y_true[:, :36] # Solo pesos
    y_pred = y_pred[:, :36] # Solo pesos
    error=y_true-y_pred
    mean_error=tf.reduce_mean(error)
    squared_diff=tf.square(error-mean_error)
    variance=tf.reduce_mean(squared_diff)
    std_dev=tf.sqrt(variance)
    return std_dev

# 2. Función de pérdida personalizada
def custom_loss(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
    # Separar modos y pesos
    y_true_modes = y_true[:, :36]
    y_true_weights = y_true[:, :36]
    y_pred_modes = y_pred[:, :36]
    y_pred_weights = y_pred[:, :36]

    # Pérdida de clasificación (binary crossentropy)
    mode_loss = tf.reduce_mean(tf.keras.losses.binary_crossentropy(y_true_modes,
    y_pred_modes))

    # Comparar top 3 modos predichos con los reales
    top3_pred_idx = tf.argsort(y_pred_modes, axis=1)[:, -3:]
    top3_true_idx = tf.argsort(y_true_modes, axis=1)[:, -3:]

    def unordered_match(a, b):
        return tf.reduce_all(tf.sort(a, axis=-1) == tf.sort(b, axis=-1), axis=-1)

    correct_modes = tf.cast(unordered_match(top3_pred_idx, top3_true_idx), tf.float32)

    # Pérdida de pesos (solo si modos son correctos)
    weight_loss = tf.reduce_mean(tf.keras.losses.MSE(y_true_weights, y_pred_weights))
    weight_loss = correct_modes * weight_loss

    # Penalización por repetir el mismo modo (misma combinación l,m en top 3)
    lms_top3 = tf.stack([top3_pred_idx // 6, top3_pred_idx % 6], axis=-1)
    # shape: (batch, 3, 2)

```

```

# Comparar pares de modos para detectar duplicados
def is_same(a, b):
    return tf.reduce_all(tf.equal(a, b), axis=-1)

same12 = is_same(lms_top3[:, 0], lms_top3[:, 1])
same13 = is_same(lms_top3[:, 0], lms_top3[:, 2])
same23 = is_same(lms_top3[:, 1], lms_top3[:, 2])

same_mode = tf.cast(tf.logical_or(tf.logical_or(same12, same13), same23), tf.float32)
penalty = same_mode * 10.0 # Puedes ajustar el peso de la penalización

return mode_loss + 0.5 * weight_loss + penalty


def create_model():
    input_layer = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 1))

    x = layers.Conv2D(32, (3, 3), activation='relu')(input_layer)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.Flatten()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.4)(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.4)(x)
    output_modes = layers.Dense(36, activation='sigmoid')(x)
    output_weights = layers.Dense(3, activation='softmax')(x)

    output = layers.concatenate([output_modes, output_weights])

    model = keras.Model(inputs=input_layer, outputs=output)

    model.compile(optimizer=optimizer,
                  loss=custom_loss,
                  metrics=['mae', mae_pesos, acc_modos, std_metric])

    return model

def plot_metrics(history):
    plt.figure(figsize=(12,5))

    # Pérdida
    plt.subplot(1,2,1)
    plt.plot(history.history['loss'], label='Pérdida entrenamiento')
    plt.plot(history.history['val_loss'], label='Pérdida validación')
    plt.xlabel('Épocas')
    plt.ylabel('Pérdida (MAE)')
    plt.legend()
    plt.title("Evolución de la Pérdida")

    # MAE
    plt.subplot(1,2,2)
    plt.plot(history.history['mae'], label='MAE entrenamiento')
    plt.plot(history.history['val_mae'], label='MAE validación')
    plt.xlabel('Épocas')
    plt.ylabel('MAE')

```

```

plt.legend()
plt.title("Evolución del MAE")

plt.tight_layout()
plt.show()

def show_predictions(model, X_test, y_test, num_examples=2):
    preds = model.predict(X_test[:num_examples])
    for i in range(num_examples):
        plt.imshow(X_test[i].squeeze(), cmap='gray')
        plt.axis('off')
        plt.title("Imagen de prueba")
        plt.show()

    y_true = y_test[i]
    y_pred = preds[i]

    true_mask = y_true[:36]
    true_weights = y_true[36:]

    pred_mask = y_pred[:36]
    pred_weights = y_pred[36:]

    true_modes = np.argwhere(true_mask > 0.5)
    pred_modes = np.argsort(pred_mask)[-3:][::-1]

    print(f"--- Ejemplo {i+1} ---")
    print("Modos reales:")
    for j, idx in enumerate(true_modes.flatten()):
        l, m = divmod(idx, 6)
        print(f"  l={l}, m={m}, peso={true_weights[j]:.2f}")

    print("Modos predichos:")
    for j, idx in enumerate(pred_modes):
        l, m = divmod(idx, 6)
        peso = pred_weights[j] if j < 3 else 0.0
        print(f"  l={l}, m={m}, peso={peso:.2f}")
    print()

# -----
# EJECUCIÓN
# -----
X, y = load_data(DATASET_PATH)
X, y = shuffle(X, y, random_state=42)
model = create_model()
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# División manual (80% train, 10% val, 10% test)
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.1,
                                                random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp,
                                                test_size=0.111, random_state=42)

# Crear datasets eficientes
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train)) \
    .shuffle(buffer_size=1000) \
    .batch(BATCH_SIZE) \
    .prefetch(tf.data.AUTOTUNE)

val_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val)) \
    .batch(BATCH_SIZE) \

```

```

.prefetch(tf.data.AUTOTUNE)

test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test)) \
    .batch(BATCH_SIZE)

history = model.fit(train_dataset, epochs=EP0CHS, validation_data=val_dataset, callbacks

# Evaluación
results = model.evaluate(test_dataset)
loss = results[0]
mae = results[1]
maepesos=results[2]
accmodos=results[3]
DESVest=results[4]

# Evaluación

print(f"\nPérdida en test: {loss:.4f}")
print(f"MAE en test: {mae:.4f}")
print(f"Accmodos en test: {accmodos:.4f}")
print(f"MAE pesos en test: {maepesos:.4f}")

print(f"DESVestandar pesos en test: {DESVest:.4f}")
# Mostrar predicciones
show_predictions(model, X_test, y_test)

# Gráficas de entrenamiento
plot_metrics(history)

```