ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

REDES NEURONALES ARTIFICIALES PARA EL ESTUDIO DE LOS SISTEMAS COMPLEJOS

CLASIFICACIÓN DE CONFIGURACIONES DEL MODELO DE ISING EN DOS Y TRES DIMENSIONES MEDIANTE REDES NEURONALES CONVOLUCIONALES

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE FÍSICO

ALEJANDRO JAVIER COBO URVINA

alejandro.cobo@epn.edu.ec

DIRECTOR: RAMON XULVI-BRUNET

ramon.xulvi@epn.edu.ec

DMQ, 14 DE FEBRERO 2024

CERTIFICACIONES

Yo, ALEJANDRO JAVIER COBO URVINA, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Alejandro Javier Cobo Urvina

Certifico que el presente trabajo de integración curricular fue desarrollado por Alejandro Javier Cobo Urvina, bajo mi supervisión.

Ramon Xulvi-Brunet

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el(los) producto(s) resultante(s) del mismo, es(son) público(s) y estará(n) a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Alejandro Javier Cobo Urvina

Ramon Xulvi-Brunet

RESUMEN

Una red neuronal convolucional (CNN) es una herramienta de aprendizaje profundo ampliamente utilizada para procesar y clasificar imágenes. En física, un sistema de interés es el modelo de Ising, el cual es ampliamente utilizado para el estudio fenómenos de transición de fase en materiales magnéticos. Este modelo se compone de una red de espines organizados en el espacio. La distribución espacial de los espines da lugar a la formación de patrones visuales caracterizados por regiones donde exhiben una orientación específica. Esta distribución varía dependiendo de si el sistema es bidimensional o tridimensional. La distinción entre los patrones formados por ambas distribuciones no es fácilmente discernible a simple vista. El objetivo de este proyecto es determinar si el sistema descrito por el modelo de Ising es bidimensional o tridimensional a partir de estos patrones, independientemente de si se encuentra en el estado estacionario o en un régimen transitorio. Se emplea una CNN para clasificar imágenes de estos patrones, generadas a través de simulaciones Monte Carlo de un modelo de Ising en dos y tres dimensiones. La red es capaz de clasificar con notable efectividad estos patrones para estados lo suficientemente alejados de un estado aleatorio, lo que sugiere que es posible determinar la dimensionalidad del sistema a partir de estos patrones.

Palabras clave: *Modelo de Ising, Redes Neuronales Convolucionales, Algoritmo de Metrópolis, Equilibrio, Aleatoriedad.*

ABSTRACT

A Convolutional Neural Network (CNN) is a widely used deep learning tool for processing and classifying images. In physics, a system of interest is the Ising model, which is widely used for studying phase transition phenomena in magnetic materials. This model consists of a network of spins organized in space. The spatial distribution of the spins leads to the formation of visual patterns characterized by regions where they exhibit a specific orientation. This distribution varies depending on whether the system is two-dimensional or three-dimensional. Distinguishing between the patterns formed by both distributions is not easily discernible to the naked eye. The aim of this project is to determine whether the system described by the Ising model is two-dimensional or three-dimensional based on these patterns, regardless of whether it is in a steady state or in a transient regime. A CNN is employed to classify images of these patterns, generated through Monte Carlo simulations of a two and three-dimensional Ising model. The network is capable of classifying these patterns with remarkable effectiveness for states sufficiently distant from a random state, suggesting that it is possible to determine the dimensionality of the system from these patterns.

Keywords: *Ising Model, Convolutional Neural Networks, Metropolis Algorithm, Equilibrium, Randomness.*

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Descripción del componente desarrollado

1.1. Marco teórico

1.1.1. Modelo de Ising

El modelo de Ising es un modelo estándar en la rama de la mecánica estadística. Fue propuesto por Ernst Ising en 1925 y sugerido previamente en 1920 por su asesor de tesis, Wilhelm Lenz [2]. Este modelo ha demostrado ser de gran utilidad al proporcionar una descripción de las interacciones magnéticas entre espines dispuestos en una red cristalina, siendo esencial para investigar fenómenos de transición de fase en materiales magnéticos [3]. La representación de este modelo viene dada por un sistema organizado en una red de espines en el espacio, donde cada espín puede adoptar únicamente orientación positiva o negativa. Esta red puede ser unidimensional, bidimensional o tridimensional, dependiendo del sistema de estudio. Además, se postula que las interacciones entre espines son de corto alcance, es decir, la energía de interacción depende exclusivamente de la configuración de los espines vecinos más cercanos [4].

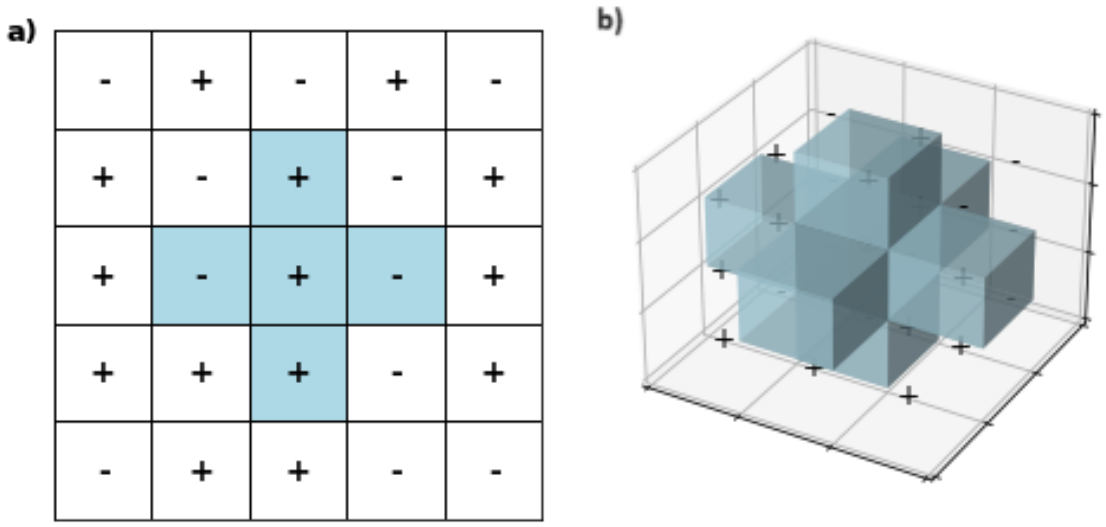

Figura 1.1: Esquema ejemplo del modelo de Ising, a) bidimensional y b) tridimensional. En ambos casos, los cuadros y bloques en tono celeste indican los vecinos más cercanos con los que interactúa el espín central en cada sistema respectivamente.

Las transiciones de fase se pueden definir como cambios en la estructura del sistema o en sus propiedades macroscópicas en respuesta a la variación de algún parámetro físico, como la presión o la temperatura. En el contexto del modelo de Ising, esto puede asociarse a cambios en la tendencia de los espines a alinearse en la misma dirección. Estos cambios en la alineación de los espines conducen a una reorganización global en la configuración del sistema, lo que a su vez afecta sus propiedades macroscópicas, como la magnetización [4].

El notable interés en el modelo de Ising se debe a que, pese a su simplicidad, ha demostrado que las interacciones microscópicas de corto alcance pueden dar lugar a transiciones de fase en el sistema, cuando tradicionalmente a las transiciones de fase se las asociaba a interacciones de larga distancia y fenómenos más complejos. La temperatura a la que se produce una transición de fase se conoce como temperatura crítica [5, 6].

En un inicio, Ernst Ising demostró que las transiciones de fase no ocurren en el modelo de Ising unidimensional, y extrapoló esta idea a di-

mensiones superiores. Sin embargo, esta hipótesis fue refutada por Lars Onsager, quien resolvió de manera analítica el modelo de Ising en dos dimensiones y demostró la existencia de transiciones de fase [2]. Para el caso tridimensional, pese a no tener aún una solución analítica, se ha confirmado mediante el argumento de Peierls la presencia de éstas [7].

Cuando la temperatura del sistema descrito por el modelo de Ising supera la temperatura crítica, los espines adoptan orientaciones aleatorias, lo que resulta en la anulación mutua de los campos magnéticos asociados a los espines y da lugar a un estado paramagnético. Sin embargo, al disminuir la temperatura y alcanzar valores por debajo de la temperatura crítica, los espines experimentan una alineación espontánea. Este fenómeno se manifiesta en la transición hacia un estado ferromagnético en el sistema [5, 2, 8].

A pesar de que inicialmente podría pensarse que la representación simplificada de las fuerzas intermoleculares en la que se fundamenta el modelo de Ising lo haría inaplicable a sistemas reales, se ha demostrado que este modelo emerge como una herramienta que ofrece enfoques sólidos para el estudio de este tipo de sistemas [2]. Un ejemplo destacado de su aplicabilidad se evidencia en los experimentos realizados con K_2CoF_4, donde se han observado predicciones teóricas precisas derivadas del modelo [9].

1.1.2. Redes Neuronales Artificiales

Una Red Neuronal Artificial (ANN, por sus siglas en inglés) constituye un sistema de procesamiento de información compuesto por numerosas unidades de procesamiento simples denominadas nodos o neuronas, las cuales se encuentran interconectadas [10]. Su estructura y funcionamiento están inspirados en el cerebro humano [11, 12]. Las ANN están conformadas por capas de nodos que incluyen una capa de entrada, una o varias capas ocultas y una capa de salida [13]. La red opera mediante el conjunto de nodos para procesar información de manera paralela, lo

que le permite abordar con eficiencia ciertas tareas computacionales específicas [10].

La red es capaz de abordar dos tipos de problemas fundamentales. En primer lugar está la regresión, en la que la red busca predecir valores continuos o de tipo flotante a partir de los datos de entrada. Este proceso resulta particularmente útil para proyectar tendencias a partir de un conjunto de datos. Por otro lado está la clasificación, donde la red busca asignar a cada dato de entrada en una de distintas clases o categorías específicas [12, 14]. Estos procesos permiten diversas aplicaciones de las ANN, desde el reconocimiento de patrones hasta su implementación en campos tan diversos como la medicina y la economía [15].

Pese a la notable eficiencia de las redes neuronales en una amplia gama de aplicaciones, éstas enfrentan diversos desafíos en la obtención de modelos eficientes en determinados casos. Entre ellos se encuentra el diseño de la arquitectura de la red y la selección de hiperparámetros, que son variables externas al modelo que afectan el proceso de entrenamiento. Para abordar esta limitación, los investigadores dedican esfuerzos significativos a la tarea de diseñar cuidadosamente la arquitectura de la red y seleccionar los hiperparámetros adecuados, dependiendo de la naturaleza específica del problema a resolver. A menudo se recurre a enfoques basados en la experiencia para tratar estas dificultades, lo que implica que la selección de un conjunto adecuado de hiperparámetros y un diseño óptimo de la arquitectura de la red requiere de alguien experimentado [16, 17].

Retomando el segundo problema fundamental de las redes neuronales, éste puede aplicarse directamente a la tarea de clasificación de imágenes. Este proceso implica asignar imágenes a una de varias clases predefinidas por un usuario y constituye un desafío fundamental en el ámbito de la visión por computadora. Además, sirve como base para tareas más complejas como la detección y segmentación de imágenes [18, 19]. En este contexto, se destaca un tipo específico de red neuronal artificial denominado red neuronal convolucional (CNN, por sus siglas en inglés),

reconocido por su eficacia en la clasificación de imágenes [20, 21, 22]. Dichas redes se especializan en identificar y extraer características clave de las imágenes, como texturas y bordes, lo que mejora significativamente su capacidad para clasificarlas de manera más eficiente en comparación con una red neuronal artificial convencional [20, 23].

Volviendo al sistema descrito por el modelo de Ising, la distribución de regiones con orientación positiva y negativa de los espines dan lugar a patrones visuales característicos. Cuando el sistema se encuentra en un estado fuera del equilibrio y a una temperatura específica, las interacciones entre espines provocan cambios en sus orientaciones, lo que conduce al sistema hacia un estado de equilibrio [24]. Los cambios en la orientación de los espines dan lugar a una amplia gama de patrones visuales distintivos que emergen a lo largo de la evolución de sistema. Para simular computacionalmente esta evolución es necesario emplear un algoritmo adecuado que genere los cambio en la orientación de los espines. En este proyecto se implementa el algoritmo de Metrópolis. Este algoritmo requiere realizar una o varias iteraciones para producir el cambio en la orientación de un determinado espín. Los detalles acerca del algoritmo y las iteraciones se describen en el capítulo dos.

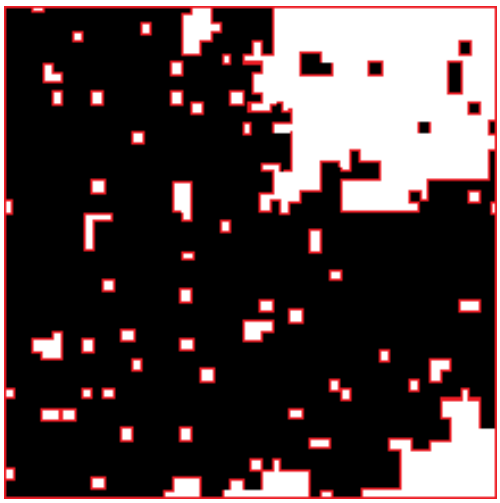

Figura 1.2: Representación visual ejemplo de los patrones generados durante una simulación del modelo de Ising bidimensional en la iteración $1{,}0982 \times 10^7$. Las áreas resaltadas con líneas rojas ilustran regiones donde la orientación de los espines es positiva (blancas) o negativa (negras).

La caracterización de estos patrones es de particular interés, ya que podrían estar intrínsecamente vinculados con información significante del sistema. En particular, resulta relevante determinar si estos patrones dependen de la dimensión del sistema. Con el propósito de abordar esta cuestión, se simula la evolución del modelo de Ising, tanto en dos como en tres dimensiones, y se generan imágenes que capturan dichos patrones en distintas iteraciones del algoritmo de Metrópolis. Posteriormente, se estudia la clasificación de estas imágenes en función de su dimensión.

A simple vista, los patrones asociados a ambas dimensiones no son fácilmente distinguibles entre sí. Por lo tanto, resulta crucial explorar métodos que permitan diferenciar visualmente la dimensionalidad del sistema a partir de estos patrones, independientemente del estado en el que se encuentre. Con ese propósito, se planea diseñar, entrenar y validar una red neuronal convolucional capaz de clasificar las imágenes de los patrones obtenidos en las simulaciones del modelo de Ising en dos y tres dimensiones.

1.2. Objetivo general

Clasificar las imágenes de los patrones obtenidos mediante simulaciones del modelo de Ising, discerniendo si corresponden a un modelo de Ising de dos dimensiones o a uno de tres dimensiones, independientemente de la iteración en la que se extraiga la imagen, es decir, sin importar si el sistema ha alcanzado el estado estacionario o si aún se encuentra en un régimen transitorio.

1.3. Objetivos específicos

1. Desarrollar un programa para simular la evolución del modelo de Ising y extraer imágenes de los patrones formados en diversas iteraciones, tanto para dos como para tres dimensiones.

2. Desarrollar, entrenar y validar una red neuronal convolucional para la clasificación de imágenes extraídas de las simulaciones del mode-

lo de Ising en dos y tres dimensiones.

1.4. Alcance

Mantener la consistencia en las simulaciones del modelo de Ising entre el sistema bidimensional y tridimensional es esencial para garantizar una comparación significativa entre las imágenes generadas en ambos casos. Con ese propósito, se decide emplear arreglos que contengan aproximadamente la misma cantidad de espines. Esto se debe a que la velocidad de evolución del sistema está relacionada con el número total de espines, y es crucial que ambas dimensiones evolucionen de manera sincronizada. De esta manera, se asegura que una imagen extraída en una iteración específica sea equivalente en ambos sistemas. En ese contexto, en las simulaciones del caso bidimensional se emplean arreglos de dimensiones 445×446, equivalentes a 198470 espines, mientras que en el caso tridimensional se utilizan arreglos de dimensiones $80 \times 80 \times 31$, equivalentes a 198400 espines. Aunque la cantidad exacta de espines no es idéntica, se procura que sean lo más parecidos posible. Asimismo, bajo la misma premisa, las simulaciones se llevan a cabo bajo las mismas condiciones de temperatura en ambas dimensiones. La decisión de no emplear arreglos de espines más extensos se justifica en consideraciones de costo computacional y restricciones temporales asociadas al desarrollo de este proyecto.

Las simulaciones destinadas a modelar la evolución del sistema descrito por el modelo de Ising se llevan a cabo empleando el algoritmo de Metrópolis. En cada simulación, se extraen de manera aleatoria 1×10^4 imágenes en diferentes iteraciones. Estas imágenes son representaciones de un sub-arreglo de 50×50 espines dentro del sistema para ambas dimensiones. Los detalles asociados a este sub-arreglo se detallan en el capítulo dos. Para el entrenamiento y validación de la red neuronal convolucional se emplean imágenes obtenidas de 20 simulaciones por cada dimensión. No obstante, debido a limitaciones temporales, no se profundiza en el ajuste detallado de los hiperparámetros de la red. Adicionalmente, la construcción de la red se realiza mediante la implementación de las bibliotecas de Tensorflow y la interfaz Keras.

Capítulo 2

Metodología

2.1. Simulaciones del Modelo de Ising

2.1.1. Descripción del Sistema

La probabilidad de que un sistema con ensamble canónico se encuentre en un estado particular, a una temperatura T y con Hamiltoniano $H(x)$, está dada por la expresión:

$$p = \frac{1}{Z} \exp\left(-\beta H(x)\right), \tag{2.1}$$

donde β se define como $\beta = \frac{1}{k_b T}$, k_b es la constante de Boltzmann, y Z es la función de partición canónica, definida como:

$$Z = \int_{\Omega} \exp\left(-\beta H(x)\right) dx, \tag{2.2}$$

x representa un punto en el espacio de fases asociado a un estado específico del sistema, mientras que Ω denota la hipersuperficie en el espacio de fases que abarca todos los posibles estados del sistema [25, 26].

Cuando se requiere modelar computacionalmente este sistema se busca utilizar un conjunto de puntos o estados aleatorios adecuados de acuerdo con la distribución de probabilidad de Boltzmann (2.1). Para ello,

el algoritmo de Metropolis propone una distribución de puntos obtenida mediante el uso de una cadena de Markov. En esta cadena, cada estado $\eta+1$ se construye a partir de un estado anterior η mediante una probabilidad de transición $W(\eta \to \eta+1)$. Para implementar adecuadamente esta idea, es necesario imponer la condición de equilibrio detallado, la cual, en términos generales, adopta la forma:

$$p(\eta)W(\eta \to \gamma) = p(\gamma)W(\gamma \to \eta), \tag{2.3}$$

siendo $p(\eta)$ y $p(\gamma)$ las probabilidades de que el sistema se encuentre en los estados η y γ respectivamente. Las probabilidades de transición, $W(\eta \to \gamma)$ y $W(\gamma \to \eta)$, indican la probabilidad de que el sistema transite de un estado η a un estado γ y viceversa. Esta condición es la que asegura que es la distribución de probabilidad que alcanza el sistema al estar en el estado de equilibrio es la distribución de Boltzmann [25, 27].

Sustituyendo las probabilidades dadas en la ecuación (2.1) en la ecuación de equilibrio detallado (2.3), se tiene:

$$\frac{W(\eta \to \gamma)}{W(\gamma \to \eta)} = \frac{p_\gamma}{p_\eta} = \exp\left(-\beta\delta H\right), \tag{2.4}$$

donde δH es la variación de energía entre ambos estados. Al observar la ecuación (2.4) es evidente que la razón entre las probabilidades de transición dependen solamente de δH [25, 28]. Sin embargo, esta condición no impone explícitamente una expresión para $W(\eta \to \gamma)$, por lo que se define:

$$W(\eta \to \gamma) = \begin{cases} \exp\left(-\beta\delta H\right), & \text{si } \delta H < 0 \\ 1, & \text{en caso contrario.} \end{cases} \tag{2.5}$$

Se puede demostrar que utilizando esta probabilidad de transición de un estado η a un estado $\eta+1$, la distribución $P(\eta)$ de los estados generados por la evolución dada a partir de una cadena de Markov converge a la distribución de equilibrio, a medida que $\eta \to \infty$ [28].

Por otro lado, el Hamiltoniano del sistema asociado al modelo de Ising,

cuando no está sometido a un campo magnético externo, viene dado por:

$$H = \sum_{\langle i,j \rangle} -J\sigma_i\sigma_j, \tag{2.6}$$

donde $\sum_{\langle i,j \rangle}$ denota la suma sobre los vecinos más cercanos de cada espín, σ_i (σ_j) es la orientación del espín de la partícula i-ésima (j-ésima), y J es el factor que relaciona la energía del sistema y la interacción entre espines [24, 27].

2.1.2. Simulación

Se implementan simulaciones Monte Carlo para modelar la evolución del modelo de Ising a una determinada temperatura, tanto para 2D como para 3D. La temperatura es la misma en ambas dimensiones del modelo. Cada simulación comienza desde un estado inicial con orientaciones completamente aleatorias de los espines y evoluciona hacia un estado de equilibrio mediante la aplicación del algoritmo de Metrópolis. En cada paso o iteración de la simulación se selecciona aleatoriamente un espín en la red, y se propone un cambio en su orientación. La conservación de esta nueva configuración se determina considerando la probabilidad de transición dada en la ecuación (2.5), lo que implica que el estado aleatorio evolucione hacia un estado de equilibrio, a través de una cadena de Markov.

Durante cada simulación se capturan imágenes de la configuración del sistema en iteraciones específicas. Se procura que las iteraciones seleccionadas se encuentren en su mayoría en estados cercanos al estado inicial aleatorio. Esto debido al interés de evaluar la capacidad de la red para clasificar imágenes extraídas sobre todo en estados más cercanos al estado inicial. En tales condiciones, la tarea de discernir entre dimensiones se vuelve más desafiante. Además, la mayoría de las primeras iteraciones de la simulación resultan en cambios en la orientación del espín seleccionado. Estos cambios generan nuevos y diversos patrones visuales. A medida que el sistema se aproxima al estado de equilibrio, la frecuencia de cambios en la orientación de espines disminuye conside-

rablemente, lo que resulta en una menor diversidad de patrones en las imágenes. Por lo tanto, no es necesario almacenar un gran número de imágenes en estas etapas, ya que es probable que se repitan con mayor frecuencia.

En particular, en este proyecto, las iteraciones se seleccionan aleatoriamente a partir de una distribución exponencial. La función de distribución tiene la forma:

$$F(x) = \begin{cases} \frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right), & \text{si } x \geq 0 \\ 0, & \text{si } x < 0, \end{cases} \tag{2.7}$$

siendo x la iteración en la que se selecciona la imagen y β el parámetro conocido como factor de escala [29]. En las simulaciones se utiliza un factor de escala determinado por la ecuación:

$$\beta = \alpha \cdot iteraciones, \tag{2.8}$$

donde α es un valor definido, e $iteraciones$ representa el número máximo de iteraciones que puede realizar la simulación. Mientras menor sea el valor de β, mayor cantidad de imágenes cercanas al estado inicial son extraídas.

Cada imagen obtenida durante las simulaciones es una representación de subarreglos bidimensionales de espines de tamaño 50×50. Si asignamos una posición en el espacio (con coordenadas cartesianas) a cada espín del sistema, en el caso bidimensional, donde el sistema está conformado por un arreglo de 445×446, el subarreglo abarca el plano que va desde el espín ubicado en la posición $x = y = 198$ hasta el espín en la posición $x = y = 248$. Mientras que, para el caso tridmiensional, que está conformado por un arreglo $80 \times 80 \times 31$, el subarreglo se sitúa en el plano $Z = 16$ y se extiende desde el espín ubicado en $x = y = 15$ hasta el espín en $x = y = 65$. La elección de estos subarreglos de espines asegura que sean comparables entre sí, ya que tienen las mismas dimensiones, y su posición centrada ayuda a mitigar posibles efectos de borde.

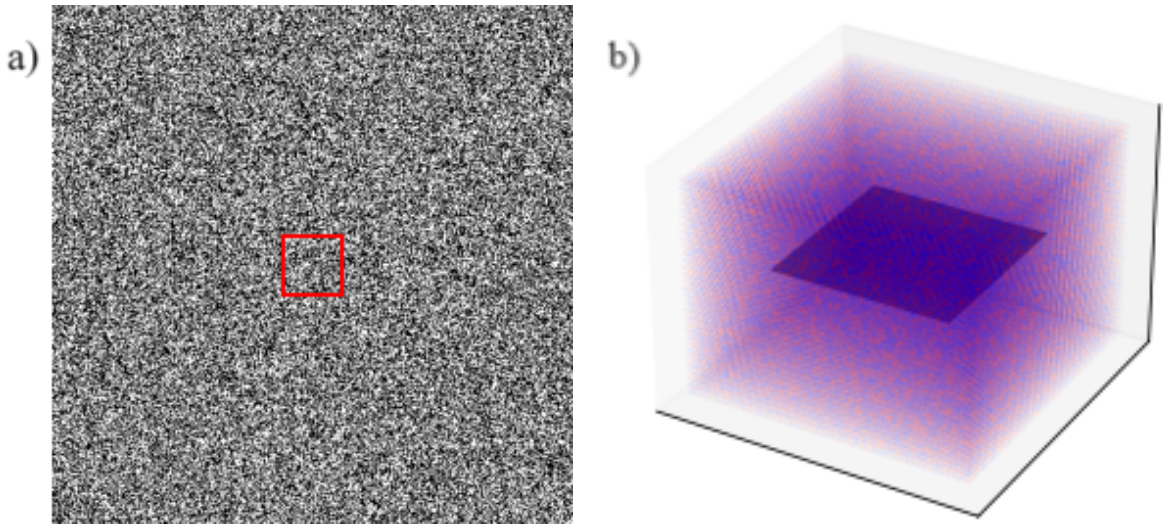

Figura 2.1: Visualización ejemplo de los arreglos extraídos de las redes bidimensional y tridimensional, ambos con dimensiones de 50×50. a) Caso bidimensional, el arreglo se señala con un cuadrado de color rojo. b) Caso tridimensional, el arreglo se resalta con un tono más oscuro que el entorno.

El código utilizado en este proyecto está escrito en lenguaje Python. La estructura general del código para simular el modelo de Ising (Anexo 1 para el caso bidimensional y 2 para el caso tridimensional) se presenta a continuación:

1. **Definición de parámetros:** En esta fase, se definen los parámetros esenciales de la simulación que abarcan la dimensión de la red, el número máximo de iteraciones a simular, la cantidad de imágenes a almacenar en cada ejecución y, los parámetros relacionados con la distribución exponencial que determina en qué iteraciones se capturan las imágenes. Además, se establece la lista de iteraciones seleccionadas para dicho propósito. *(Ver líneas 7-14 del anexo 1 y 8-16 del anexo 2)*

2. **Inicialización del sistema:** En esta fase, se genera un arreglo bidimensional o tridimensional de números aleatorios distribuidos uniformemente en el rango de 0 a 1. Posteriormente, se asigna el valor de -1 a aquellos elementos cuyo valor sea menor a 0.5, y el valor 1 a aquellos cuyo valor sea mayor a 0.5. Con ello, se define un esta-

do inicial con orientación aleatoria de espines. *(Ver líneas 16-20 del anexo 1 y 18-22 del anexo 2)*

3. **Algoritmo de Metrópolis:** Se establece el algoritmo de Metrópolis dentro de una subrutina para simular la evolución del sistema de espines mediante una cadena de Markov. Esta subrutina recibe como parámetro el arreglo de espines y la temperatura del sistema, y devuelve el arreglo actualizado debido a una iteración del algoritmo de Metropolis. La temperatura utilizada en las simulaciones es de $2{,}1\ \left[\frac{J}{K_b}\right]$, aunque no es un parámetro relevante para el objetivo de estudio del proyecto. La inclusión de $@numba.njit(nopython = True, nogil = True)$ permite una ejecución optimizada al compilar la función en código de máquina, mejorando significativamente la eficiencia al eliminar la dependencia del intérprete de Python [30]. Su estructura se describe detalladamente a continuación *(Ver líneas 23-52 del anexo 1 y 24-60 del anexo 2)* [26]:

   a) Selecciona aleatoriamente un espín en la red.

   b) Propone un cambio en la orientación de dicho espín.

   c) Calcula la energía de interacción de ese espín con sus vecinos más cercanos en dos escenarios: uno en el que su orientación no ha cambiado y otro en el que sí ha cambiado. Posteriormente, determina la variación de energía entre ambos casos. No es necesario realizar el cálculo de la energía de interacción entre el espín y todos sus vecinos en el sistema, ya que en el algoritmo de Metrópolis, solo nos interesa la diferencia de energía entre estos dos escenarios, como se mencionó al presentar la ecuación (2.5).

   d) Luego, se presentan dos posibles escenarios, como lo dicta la ecuación (2.5):

      - En el caso de que la energía de la nueva configuración sea mayor, la configuración se conservará solamente si se cumple la condición:

$$\exp\left(-\beta J \delta H\right) > \omega, \tag{2.9}$$

        siendo ω un número aleatorio entre 0 y 1, y δH la variación en la energía de ambos estados.

- Si la energía de esta nueva configuración es inferior a la del estado inicial, entonces se conserva el nuevo estado.

e) La función devuelve la nueva configuración del sistema, considerando el posible cambio en la orientación del espín.

4. **Almacenamiento de imágenes:** En esta fase se ejecuta el algoritmo de Metrópolis dentro del programa. En paralelo, se incorpora la funcionalidad de capturar y almacenar imágenes de la configuración del sistema en las iteraciones específicas previamente seleccionadas. Estas imágenes se almacenan en ficheros que tienen de nombre la fecha y hora de la máquina en la que se realiza la simulación en particular y de nombre de imagen el número total de espines en la red y la iteración en la que se almacena. Esto con el único propósito de identificar unívocamente cada una de las imágenes. Además, debido a que el enfoque se centra exclusivamente en la clasificación de imágenes mediante una CNN, no es necesario simular la evolución del sistema hasta la última iteración. Por esta razón, una vez que se ha almacenado el número predeterminado de imágenes la simulación concluye. Esta decisión permite optimizar recursos computacionales. *(Ver líneas 54-74 del anexo 1 y 63-84 del anexo 2)*. La estructura se analiza a detalle:

   a) Se define el número total de espines en la red.

   b) Se crea una carpeta en la que se almacenan las imágenes obtenidas en la simulación. Para ello, utiliza como nombre la fecha y hora en la que se ejecutó la simulación en particular.

   c) Realiza una copia de la configuración de la red en su estado inicial.

   d) Crea un bucle para iterar el algoritmo de metrópolis sobre el sistema. El límite de iteraciones viene dado por el parámetro definido al inicio del código.

   e) Ejecuta el algoritmo de metrópolis utilizando la red inicial y almacena la nueva configuración en otra variable.

   f) Reasigna la nueva configuración a la red inicial para repetir la iteración.

g) Dentro del mismo bucle, verifica si la iteración actual se encuentra dentro de la lista de iteraciones seleccionadas para almacenar imágenes.

h) Si la condición mencionada anteriormente se satisface, procede a definir un pequeño arreglo de 50×50 espines centrado en la red. Posteriormente, lo almacena como una imagen en formato PNG, asignándole de nombre la iteración actual.

i) Por último, verifica si la iteración actual coincide con la última de la lista definida en los parámetros al inicio del código. Si es así, se detiene el bucle, ya que ya se han almacenado el número de imágenes definidas.

2.2. Red Neuronal

Una Red Neuronal Artificial (ANN) consta de un conjunto de unidades de procesamiento simples (nodos o neuronas) que se comunican a través de un gran número de conexiones ponderadas. Su estructura está organizada en capas, siendo la primera para la entrada de información, posteriormente se ubican capas intermedias o ocultas que procesan dicha información y por último una de salida que predice resultados [13, 31].

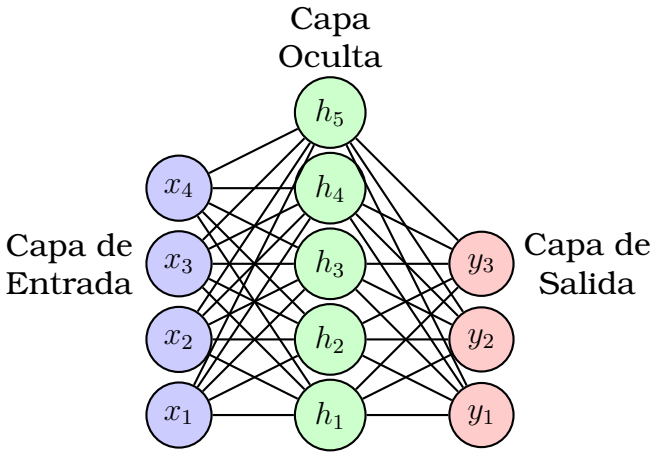

Figura 2.2: Estructura general ejemplo de una red neuronal artificial.

Para comprender el funcionamiento de una Red Neuronal Artificial, es esencial introducir conceptos básicos. En ese contexto, se define como

atributos a los nodos ubicados en la capa de entrada. Se los conoce como atributos porque proporcionan características de los datos con los que la red va a trabajar. Por ejemplo, en un proceso de clasificación de imágenes, cada atributo representa un píxel de la imagen. Generalmente, la red está compuesta por varios atributos para realizar sus predicciones. Por otro lado, se conoce como etiqueta al valor que la red intenta predecir. La predicción de la red para determinar la etiqueta se genera en la capa de salida. En el mismo ejemplo de calsificación de imágenes, las etiquetas representan las clases a la que puede pertenecer una determinada imagen [14].

Otro concepto relevante son las muestras, las cuales son un conjunto específico de datos que incluyen tanto atributos como etiquetas. Estas muestras se subdividen en dos categorías: una destinada a la fase de entrenamiento y otra a la fase de validación. Durante la etapa de entrenamiento, el modelo aprende gradualmente dicha relación a partir de los datos de entrenamiento. Posteriormente, en la etapa de validación, el modelo se aplica sobre los datos de validación para comparar las predicciones realizadas por la red con las etiquetas reales de los datos [14].

Por otro lado, se define como función de pérdida a la función que mide la discrepancia entre las predicciones realizadas por el modelo y las etiquetas reales de las muestras [14]. Por último, se define la función de activación. Esta es una función que desempeña un papel crucial en todo el proceso, ya que introduce no linealidad al modelo. Esta función aplica transformaciones no lineales a las funciones lineales o multilineales del modelo generado por la red. Al introducir no linealidad le permite a la red aprender a separar clases de datos que no son linealmente separables. Sin esta función las aplicaciones de una red neuronal serían muy reducidas [14, 32].

Una vez aclarados estos conceptos básicos, se puede pensar en una neurona artificial como un modelo de regresión lineal con una función de activación final. Una neurona de la capa j tomará la salida de todas las neuronas de la capa $j-1$ como entradas, calculará la suma ponderada

y le sumará un sesgo (término independiente). Finalmente, se aplicará una función de activación para introducir no linealidad. El modelo tiene la forma:

$$x_k^j = f\left(\sum_i^n m_i x_i^{j-1} + b\right), \tag{2.10}$$

donde x_k^j es el valor de una neurona k-ésima de la capa j, f es la función de activación, n es el número de neuronas en la capa $j-1$, m_i es el peso asociado a la conexión entre la neurona x_k^j con la neurona i-ésima en la capa $j-1$, x^{j-1} es la neurona en la capa $j-1$ y b es el sesgo. Tal como se muestra en la figura [13, 14]:

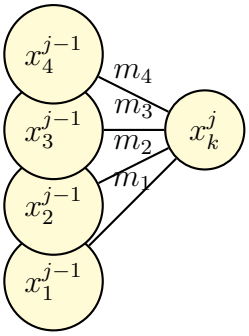

Figura 2.3: Esquema ejemplo de interacción entre la capa $j-1$ y j, siendo $n=4$ en la ecuación (2.10).

En la etapa de entrenamiento la red disminuye gradualmente la función de pérdida, variando así los valores de los pesos y sesgos de la conexión entre todas las neuronas. Se puede pensar en el entrenamiento como un ajuste de regresión que utiliza los datos de entrenamiento para variar los parámetros del modelo [14, 31].

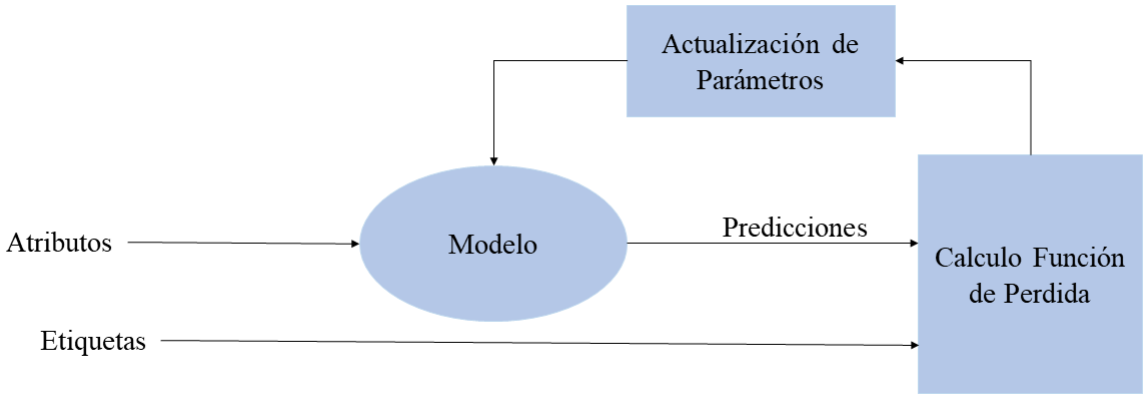

Figura 2.4: Esquema iterativo de la reducción de la función de pérdida en una red neuronal.

2.3. Red Neuronal Convolucional

Para entender el funcionamiento de una red neuronal convolucional, es necesario revisar cierta metodología básica. Primeramente, se define el proceso de convolución, el cual consta de aplicar un filtro o kernel sobre la imagen. Un filtro es una matriz de números que se aplica sobre la matriz de píxeles asociada a una determinada imagen. El filtro se desliza sobre la cuadricula de la imagen un píxel a la vez y, a cada paso, se multiplica los valores del filtro por los valores de la imagen en cada punto de superposición. La suma de los productos se almacena en una nueva imagen, llamada imagen convolucionada, la cual puede tener un tamaño diferente al de la imagen original [20, 23].

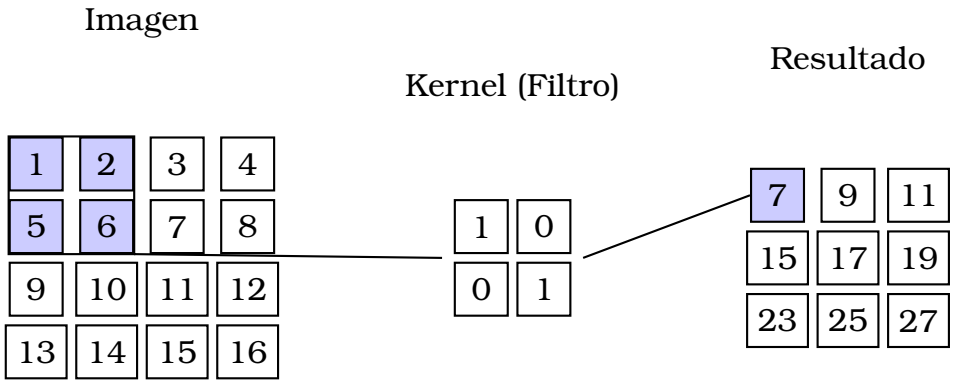

Figura 2.5: Ejemplo del proceso de convolución con kernel 2x2 y matriz de imagen 4x4.

Keras y TensorFlow

TensorFlow es una biblioteca de aprendizaje automático de código abierto que ofrecen interfaces de programación de aplicaciones (APIs) para el desarrollo en diversas aplicaciones. Entre sus principales APIs se encuentra Keras, el cual proporciona una interfaz de alto nivel para el diseño de modelos de aprendizaje profundo, como redes neuronales. Keras es una biblioteca de código abierto que se integra fácilmente en el entorno de Python [33].

Época

En el contexto de las redes neuronales, una época se define como el período durante el cual todos los elementos del conjunto de datos de la muestra han participado en el proceso de entrenamiento. Es decir, han contribuido a la actualización de los parámetros del modelo generado por la red, ajustando así los pesos y sesgos de las conexiones entre nodos de la red. El número de épocas es un hiperparámetro que determina cuántas veces el algoritmo de aprendizaje recorre todo el conjunto de datos de entrenamiento para disminuir la función de perdida y generar el modelo final de la red [34].

Función de Activación ReLu

La función de activación ReLU (Rectified Linear Unit) es ampliamente empleada en redes neuronales. Se define como:

$$f(x) = max(0, x) \tag{2.11}$$

Es decir, para cualquier valor de entrada, devuelve cero si es negativo y el mismo valor si es positivo. Este enfoque, a pesar de parecer ser sencillo, ha demostrado ser esencial en una amplia gama de aplicaciones [35].

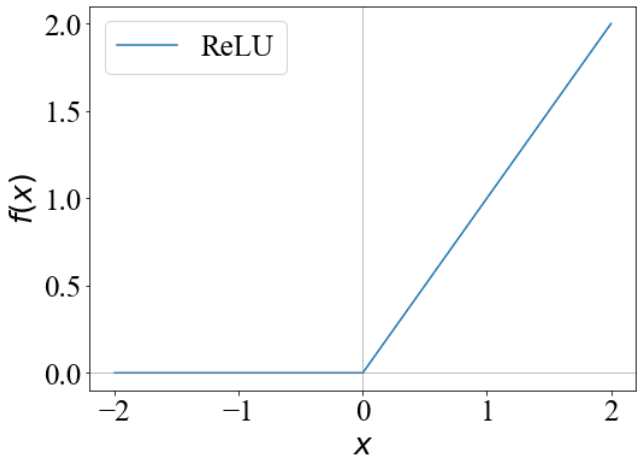

Figura 2.6: Función de activación ReLu.

Función de Activación Sigmoide

Es una función de activación comúnmente utilizada en redes neuronales debido a su forma sigmoidal. Convierte cualquier valor real en un valor que se encuentre en el rango de 0 a 1. Se define como:

$$f(x) = \frac{1}{1 + \exp(-x)}. \tag{2.12}$$

Es una herramienta útil para utilizarse en la capa de salida de modelos de clasificación binaria [36].

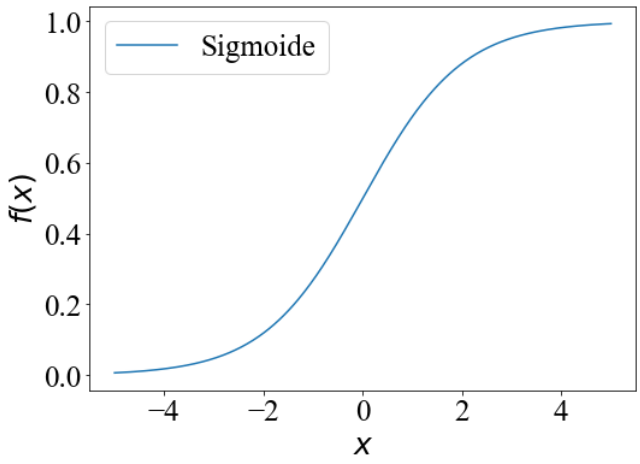

Figura 2.7: Función de Activación Sigmoide.

Max-Pooling

El Max-Pooling es una técnica comúnmente utilizada después del proceso de convolución en las redes neuronales convolucionales. Su objetivo principal es reducir la dimensionalidad espacial de las capas ocultas. Para lograr este propósito, se aplica una matriz de tamaño $\alpha \times \alpha$ sobre la matriz resultante del proceso de convolución. Esta matriz divide las regiones cuadradas en bloques de tamaño $\alpha \times \alpha$ y conserva exclusivamente el valor máximo de cada una de estas regiones. Este enfoque permite preservar las características más relevantes de la imagen, al tiempo que reduce significativamente la cantidad de parámetros sobre los que la red va a calcular [37].

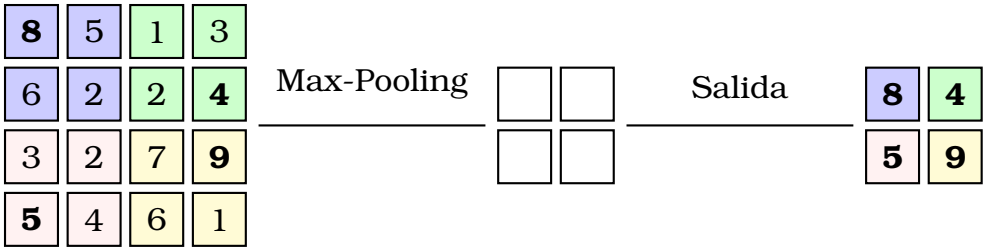

Figura 2.8: Esquema ejemplo del proceso Max-Pooling con $\alpha = 2$. Se resalta en negrita el valor máximo en cada región, definida por colores.

Sobreajuste y Dropout

El sobreajuste es un fenómeno en el cual el modelo de una red neuronal se ajusta de manera excesiva a los datos de entrenamiento. Esto provoca que el modelo sea altamente sensible a ciertas características específicas del conjunto de datos de entrenamiento, pero incapaz de generalizar de manera efectiva para datos no vistos previamente. Como consecuencia, el rendimiento en tareas de predicción o clasificación puede resultar deficiente [38, 39].

Para abordar esta limitación, se introduce la técnica conocida como dropout. El dropout consiste en desactivar temporal y aleatoriamente ciertos nodos o neuronas de la red, utilizando una probabilidad definida. Esto implica que las contribuciones de estas unidades no se consideran en la actualización de los pesos y sesgos durante el entrenamiento.

El dropout introduce variabilidad de manera estratégica, evitando que las neuronas dependan excesivamente de características particulares del conjunto de datos de entrenamiento. Además, promueve la generalización al obligar al modelo a aprender características útiles de manera más independiente [38, 39].

Función de Pérdida: Binary crossentropy

La función Binary Cross-Entropy es una herramienta matemática esencial para evaluar la discrepancia entre las predicciones generadas por un modelo de clasificación y las etiquetas reales asociadas a los datos de entrenamiento. En el ámbito de problemas de clasificación binaria, donde las etiquetas pueden tomar dos valores posibles (0 o 1), esta función se define como:

$$L(y, \hat{y}) = -[y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})], \tag{2.13}$$

donde y representa la etiqueta real (0 o 1), y $\hat{y}$ es la predicción del modelo, calculada como la probabilidad de pertenencia a una de las dos clases. En cada instancia, la función asigna un valor escalar que mide la discrepancia entre la predicción y la verdad. Es importante destacar la forma convexa de esta función, que facilita la búsqueda de un mínimo. Este método es ampliamente empleado en modelos de clasificación binaria [40].

Tasa de Aprendizaje

La tasa de aprendizaje constituye un hiperparámetro crítico que armoniza la rapidez y estabilidad del proceso de aprendizaje de una red neuronal, siendo determinante en la magnitud de los pasos que orientan al algoritmo hacia el punto mínimo de la función de pérdida. Funciona como un factor multiplicativo aplicado al vector gradiente. Cuando la tasa de aprendizaje es excesivamente grande, puede resultar en oscilaciones que se alejan del mínimo de la función de pérdida, mientras que, si es demasiado pequeña, el aprendizaje tiende a volverse notablemente lento [40, 41].

Optimizador: RMSprop

Un optimizador es un algoritmo que ajusta los parámetros del modelo de manera iterativa durante el proceso de entrenamiento. RMSprop es una herramienta de optimización ampliamente utilizada para el entrenamiento de redes neuronales. Su función principal radica en ajustar la tasa de aprendizaje de manera individual para cada parámetro del modelo. Esta técnica destaca por su capacidad para mejorar la eficiencia y velocidad de entrenamiento. En lugar de utilizar una tasa de aprendizaje constante, RMSProp mantiene una media móvil ponderada de los cuadrados de los gradientes anteriores. El ajuste de la tasa de aprendizaje para cada parámetro m_j viene dado por:

$$m_j = m_j - \eta \cdot \frac{1}{\sqrt{v_{m_j,t}^-}} \frac{\partial J}{\partial m_j}, \tag{2.14}$$

donde η es la tasa de aprendizaje original, $\frac{\partial J}{\partial m_j}$ representa el gradiente de la función de pérdida respecto al parámetro m_j, y $v_{m_j,t}^-$ es la actualización de la media móvil ponderada, que viene dado por la ecuación:

$$v_{c_j,t}^- = \gamma \cdot v_{c_{j-1},t}^- + (1-\gamma) \left(\frac{\partial J}{\partial m_j} \right)^2, \tag{2.15}$$

donde γ es el factor de olvido, un hiperparámetro que determina la importancia relativa de los gradientes anteriores en comparación con los nuevos durante la actualización [42].

2.3.1. Matriz de Confusión

La matriz de confusión es una herramienta utilizada para evaluar modelos de clasificación, permitiendo cuantificar el rendimiento predictivo del modelo. Su función principal es organizar las clasificaciones predichas por el modelo y compararlas con las clases reales a las que pertenecen los datos, ofreciendo así una visión didáctica de los resultados obtenidos. La matriz de confusión resume los resultados de esta clasificación, destacando cuatro elementos: Verdaderos Positivos, Verdaderos Negativos, Falsos Positivos y Falsos Negativos [43].

La arquitectura de la red neuronal convolucional empleada en este proyecto se deriva de una simplificación de arquitecturas estándar comúnmente utilizadas en tareas de clasificación imágenes [44]. Específicamente, presenta la siguiente estructura:

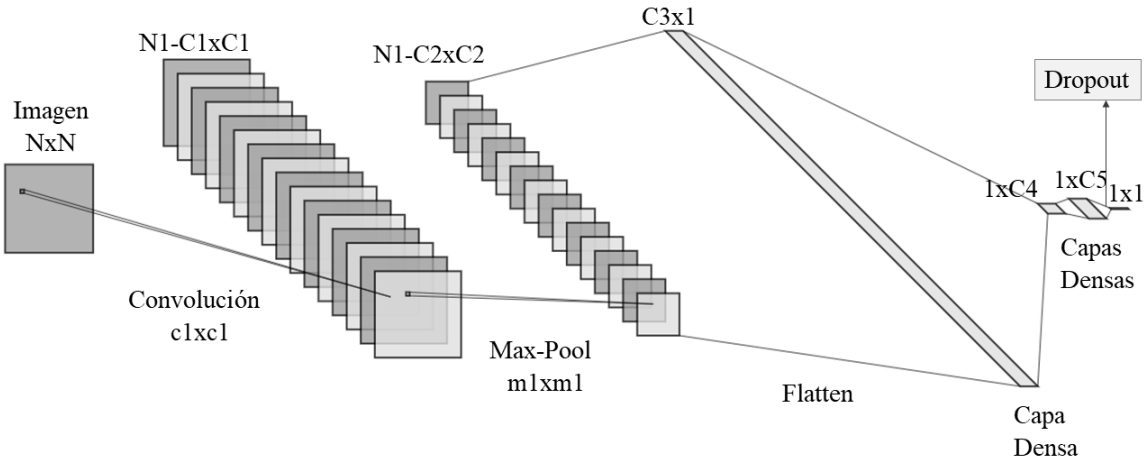

Figura 2.9: Esquema general de la arquitectura de la Red Neuronal Convolucional empleada [1].

El procedimiento explicado que se muestra en la figura 2.9 es que a cada imagen se la convierte en matrices de $N \times N$, donde cada componente de la matriz representa un píxel de la imagen. En este proyecto, se trabaja únicamente con imágenes de color blanco y negro. Cada píxel tiene una determinada tonalidad, que en este caso es solamente o blanco o negro y le asigna un determinado número dependiendo del color del píxel. A cada una de estas matrices se le aplica $N1$ filtros o convoluciones de tamaño $c1 \times c1$, dando como resultado $N1$ matrices de tamaño $C1 \times C1$. Posterior a ello, se aplica un proceso de Max-Pooling a través de matrices de tamaño $m1 \times m1$, dando como resultado $N1$ matrices de dimensión menor llamada $C2 \times C2$. Posterior a ello se aplica el proceso de Flatten, donde a todas las matrices se las organiza tal que posean de dimensión una única columna y $C3$ filas que se comportan como nodos que se conectan con otra capa de $C4$ nodos. Esta capa, a su vez, se conecta con una capa de $C5$ nodos, para finalmente presentar una salida de tamaño 1×1, ya que se trata de un problema de clasificación binaria.

En las capas $C4$ y $C5$ se emplea la función de activación ReLU, mientras que en la capa de salida se utiliza una función de activación Sigmoide. Además, entre la capa $C5$ y la de salida, se implementa un Dropout para mitigar el riesgo de sobreajuste. Por otro lado, en la compilación de la red, se utiliza la función de pérdida Binary Crossentropy, y como optimizador, se elige RMSprop con una tasa de aprendizaje $\eta = 0{,}001$.

2.3.2. Código para clasificación de imágenes

El código utilizado (anexo 3) para la clasificación de imágenes en este proyecto también está escrito en Python, y construye la red a partir de las librerías de TensorFlow y Keras. Este inicia cargando las imágenes generadas por las simulaciones del modelo de Ising, asignándoles etiquetas según la dimensionalidad del sistema correspondiente. Luego, divide los datos en conjuntos de entrenamiento y validación en un $80\,\%$ y $20\,\%$ respectivamente. Posteriormente, se incluye la arquitectura de la red y se procede al entrenamiento del modelo con el conjunto de entrenamiento. La evaluación del rendimiento se lleva a cabo utilizando el conjunto de validación, y los resultados se visualizan mediante gráficos de matrices de confusión. La estructura del código (anexo 3) detallada es:

1. **Carga de imágenes:** Se define una función que recibe como parámetros el directorio que almacena los resultados de las simulaciones dependiendo de la dimensión, la etiqueta asignada a esas imágenes y el tamaño al que se redimensionarán. Esta función itera a través de todas las subcarpetas del directorio principal definido en los parámetros, abriendo cada imagen para redimensionarla, convertirla a escala de grises y transformarla en un arreglo. Estos arreglos se almacenan en una lista, mientras que los nombres de las etiquetas se guardan en otra. Finalmente, la función retorna las dos listas, una que contiene las imágenes y otra sus etiquetas correspondientes. Utilizando esta función, se generan listas separadas para las imágenes y etiquetas tanto de las simulaciones 2D como de las 3D. Posteriormente, se combinan las listas de imágenes 2D con las de 3D, al igual que las listas de etiquetas, consolidándolas en una sola lista. *(Ver líneas 15-39 del anexo 3)*

2. **División de datos:** Las imágenes se normalizan al dividir sus valores para 255. Este es un paso común en el procesamiento de imágenes y tiene que ver con la representación de colores en el espacio RGB. Cada píxel tiene un valor que va desde 0 (negro) hasta 255 (blanco). Posterior a ello, se procede a dividir los datos en conjuntos de entrenamiento y validación. *(Ver líneas 41-42 del anexo 3)*

3. **Inclusión de la arquitectura de la red:** Se define la arquitectura descrita en la figura 2.9. Se establece una entrada de imágenes de tamaño definido, y un solo canal (escala de grises). Adicionalmente, se imprime el resumen de la arquitectura del modelo. *(Ver líneas 45-55 del anexo 3)*

4. **Entrenamiento del modelo:** El modelo se compila utilizando la función de pérdida de binary crossentropy, el optimizador RMSprop con una con una determinada tasa de aprendizaje de η.

   Se emplea Early Stopping como una técnica de regularización, la cual monitorea la pérdida en el conjunto de validación y detiene el entrenamiento si los resultados no mejoran luego de un cierto número de épocas. Además, se define también el número máximo de épocas. Finalmente, imprime el valor de pérdida obtenido en la validación del modelo final de la red. *(Ver líneas 58-64 del anexo 3)*

5. **Visualización de Matriz de Confusión:** Se realiza una predicción sobre el conjunto de validación y, al compararla con las etiquetas reales, se crea una matriz de confusión. Esta matriz muestra las predicciones correctas e incorrectas para cada clase (“2D” y “3D”). Todas las imágenes generadas son almacenadas con un nombre que refleja la fecha y hora exactas en las que se ejecutó el programa. *(Ver líneas 67-87 del anexo 3*)

Capítulo 3

Resultados, conclusiones y recomendaciones

3.1. Resultados

Los parámetros de la función de distribución exponencial empleada para la extracción de imágenes de las simulaciones del modelo de Ising (2.8) son $\alpha = 0{,}1$ e $iteraciones = 8 \times 10^7$. Se utiliza este valor de $iteraciones$ debido a que previamente se determinó que ese es el mínimo necesario de iteraciones que debe realizar el sistema simulado para que alcance el estado estacionario, donde ya no presenta variaciones significativas en su magnetización y energía. Las imágenes son generadas a partir de 20 simulaciones del modelo de Ising tanto en dos como en tres dimensiones. Los resultados de la extracción de imágenes se presentan a través de histogramas que muestran la frecuencia de imágenes obtenidas en función de las iteraciones realizadas por el algoritmo de Metrópolis en todas las simulaciones.

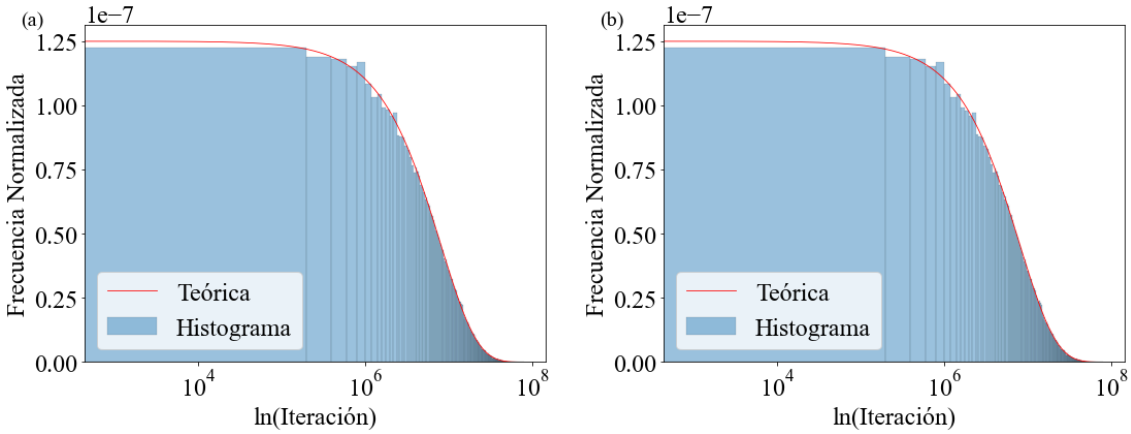

Figura 3.1: Número de imágenes extraídas en función del número de iteraciones realizadas en las simulaciones. La figura (a) corresponde al caso bidimensional. La figura (b) corresponde al caso tridimensional. Cada intervalo del histograma tiene un ancho de NT iteraciones, donde NT es igual 198470 en 2D y 198400 en 3D, y estos son los valores del número total de espines utilizados en cada sistema simulado. La curva roja representa la función de distribución exponencial teórica empleada en la extracción de imágenes en ambos casos. El histograma está normalizado. El eje x está representado en escala logarítmica.

Como se mencionó previamente, el objetivo de las simulaciones es extraer una mayor cantidad de imágenes en estados cercanos al inicial, o estado aleatorio, lo que se observa en la figura 3.1. Asimismo, la elección del ancho de barra específico de los histogramas se fundamenta en la premisa de que, para cada uno de los sistemas, se espera que permanezcan en el régimen aleatorio después de al menos esa cantidad de iteraciones. Esto se debe a que la evolución del sistema se simula seleccionando un espín al azar en cada iteración y proponiendo un cambio en su orientación. Si en la simulación se han realizado menos iteraciones que la cantidad de espines en el sistema, entonces la simulación no ha recorrido sobre todos esos espines al menos una vez. Además, ya que en varias iteraciones el algoritmo de Metrópolis deja invariante al sistema, se estima que al menos se debería experimentar varias veces esa cantidad de iteraciones para salir del régimen completamente aleatorio. Por el mismo motivo, se presenta a continuación el porcentaje total de imágenes extraídas en los intervalos de una y dos veces ese valor de iteraciones (cuadro 3.1), junto con el porcentaje de imágenes extraídas en las iteraciones restantes.

Cuadro 3.1: Porcentaje de imágenes extraídas (P) en distintos intervalos de iteraciones (I) realizadas por simulaciones con $\alpha = 0{,}1$ e $iteraciones = 8 \times 10^7$, tanto para el caso bidimensional como para el caso tridimensional.

2D		**3D**	
I [iteraciones]	P [%]	I [iteraciones]	P [%]
[0,NT)	2.44	[0,NT)	2.43
[NT, 2NT)	2.4	[NT, 2NT)	2.36
[2NT, restante)	95.17	[2NT, restante)	95.21

En particular, para estos parámetros de α e $iteraciones$ utilizados en la extracción de imágenes con la que se obtiene la figura 3.1, se presenta que menos del $5\,\%$ de las imágenes totales se encuentran en estados que "intuitivamente" están caracterizados por la aleatoriedad.

Por otro lado, recordando la red neuronal convolucional discutida en el capítulo anterior y con estructura presentada en la figura 2.9, proporcionamos los parámetros específicos empleados en la configuración de la red para el presente proyecto:

Cuadro 3.2: Configuración de parámetros específicos de la red neuronal convolucional utilizada en el proyecto.

Parámetro	**Valor**
N	50
c1	2
N1	16
C1	49
m1	2
C2	24
C3	9216
C4	8
C5	16
Dropout	0.5

Cabe destacar que la configuración proporcionada para la red, según se muestra en el cuadro 3.2, resulta en un total de 73,977 parámetros con los cuales la red genera el modelo para la clasificación. No está de

más mencionar que, si bien 74,000 parámetros para un modelo en física pueden parecer un número significativamente grande, para un modelo generado a partir de redes neuronales es en realidad un número relativamente pequeño de parámetros. Al comparar esta cantidad con el número de imágenes, que es aproximadamente 2×10^5 por cada dimensión, es decir, 4×10^5 en total, se observa que el número de parámetros es menor que la mitad del número de datos de entrenamiento. Este hecho es relevante, ya que, de manera similar a un ajuste que utiliza puntos para realizar una regresión, se requiere que el número de puntos o datos sea significativamente mayor que el número de parámetros con los que se genera el modelo para la regresión, o en este caso, la clasificación.

Los resultados obtenidos al aplicar esta red neuronal convolucional para la clasificación de las imágenes extraídas en las simulaciones con parámetros $\alpha = 0{,}1$ e $iteraciones = 8 \times 10^7$ son:

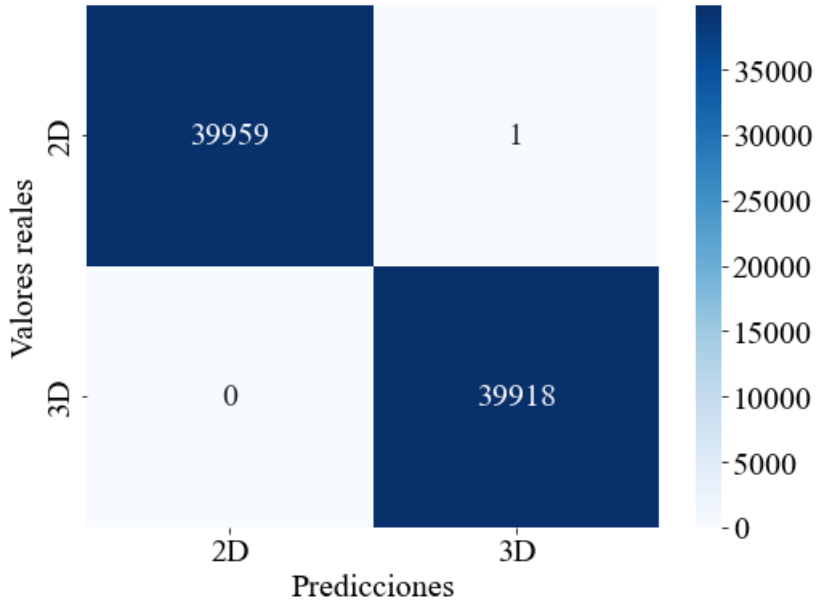

Figura 3.2: Matriz de confusión.

La figura 3.2 muestra el total de predicciones correctas e incorrectas en la clasificación, utilizando el conjunto de datos de validación. En los elementos de la diagonal se ubican las predicciones correctas, mientras que los elementos fuera de la diagonal representan las predicciones incorrectas. La precisión final obtenida en la clasificación de estas imágenes

es de $0{,}999$.

En base a los resultados obtenidos se puede afirmar que el modelo ha alcanzado una capacidad de clasificación efectiva de los patrones presentes en las imágenes obtenidas de las simulaciones realizadas. Sin embargo, a pesar de los resultados satisfactorios presentados en la figura 3.2, no estaría de más cuestionar si la imagen que clasificó mal la red se encuentra en los estados iniciales del sistema, donde ambos sistemas son completamente aleatorios.

Para explicar si la red, como se podría anticipar, es incapaz de clasificar proceso aleatorios (o quasi-aleatorios), repetimos nuevamente el estudio con parámetros $\alpha = 0{,}001$ e $iteraciones = 5 \times 10^7$. Los histogramas asociados a la extracción de imágenes obtenidos al utilizar estos parámetros son:

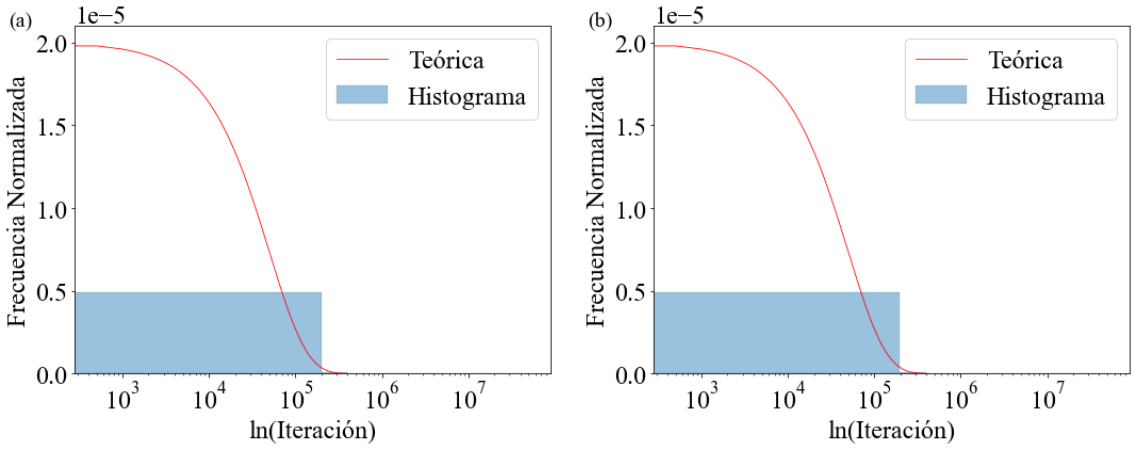

Figura 3.3: Número de imágenes extraídas en función del número de iteraciones realizadas en las simulaciones. La figura (a) corresponde al caso bidimensional. La figura (b) corresponde al caso tridimensional. Cada intervalo del histograma tiene un ancho de NT iteraciones, donde NT es igual 198470 en 2D y 198400 en 3D, y estos son los valores del número total de espines utilizados en cada sistema simulado. La curva roja representa la función de distribución exponencial teórica empleada en la extracción de imágenes en ambos casos. El histograma está normalizado. El eje x está representado en escala logarítmica.

Mientras que, el porcentaje de imágenes extraídas en los mismos in-

tervalos presentados en el cuadro 3.3 son:

Cuadro 3.3: Porcentaje de imágenes extraídas (P) en distintos intervalos de iteraciones (I) realizadas por simulaciones con $\alpha = 0{,}001$ e $iteraciones = 5 \times 10^7$, tanto para el caso $2D$ como para el caso $3D$.

2D		**3D**	
I [iteraciones]	P [%]	I [iteraciones]	P [%]
[0,NT)	98.02	[0,NT)	97.97
[NT, 2NT)	1.94	[NT, 2NT)	2
[2NT, restante)	0.04	[2NT, restante)	0.03

El cuadro 3.3 revela que más del $99\,\%$ de las imágenes se extraen antes de que la simulación alcance NT iteraciones, y prácticamente el $100\,\%$ de las imágenes se extraen antes de las $2NT$ iteraciones al utilizar estos nuevos parámetros. Esto sugiere que la gran mayoría de las imágenes extraídas pertenecen al régimen aleatorio. Al intentar clasificar estas imágenes con la red se obtiene:

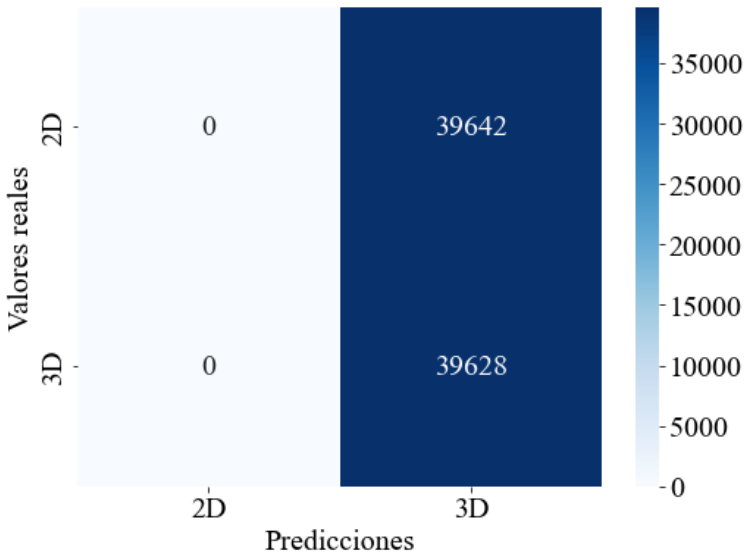

Figura 3.4: Matriz de Confusión.

La precisión final obtenida en la clasificación de imágenes con este conjunto de datos de validación es de $0{,}505$. Tal y como se intuía, se observa que la red simplemente hizo una clasificación aleatoria ya que todos los patrones que presentan las imágenes también lo son.

3.2. Conclusiones y recomendaciones

3.2.1. Conclusiones

Al considerar imágenes que se encuentran lo suficientemente alejadas del estado inicial, donde todos los espines del sistema presentan orientaciones aleatorias, se destaca la capacidad casi perfecta de clasificación de la red neuronal convolucional. Sin embargo, al examinar estados cercanos al estado inicial, las imágenes muestran patrones completamente aleatorios, lo que resulta en la incapacidad de la red para distinguir entre el caso bidimensional y tridimensional del sistema al cual pertenece cada imagen. Se puede intuir que los pocos errores que presenta la clasificación de la red se debe a imágenes cercanas a este régimen aleatorio.

En relación a lo antes expuesto, los patrones generados por la distribución de áreas con orientación positiva y negativa ofrecen información crucial sobre la dimensionalidad del sistema descrito por el modelo de Ising. Una red neuronal convolucional es una herramienta capaz de clasificar estos patrones entre el caso bidimensional y tridimensional.

3.3. Recomendaciones

Un área primordial a investigar en este proyecto radica en la optimización del rendimiento de la red neuronal convolucional empleada. Para ello, se puede extender el estudio a la búsqueda de un conjunto acertado de hiperparámetros y el diseño eficiente de la red. Esta exploración podría resultar en una reducción considerable del número de parámetros de la red.

Además, considerando la limitación identificada en el funcionamiento de la red para estados cercanos al estado inicial aleatorio, sería prudente extender este estudio para identificar un punto crítico en el que la red ya no puede clasificar con precisión entre el caso bidimensional y tridimensional. Este punto crítico coincidiría con el momento en el cual las

imágenes comienzan a exhibir patrones únicos que son discernibles en función de la dimensión del sistema.

Referencias bibliográficas

[1] LeNail. Nn-svg: Publication-ready neural network architecture schematics. journal of open source software, 4(33), 747,. https://alexlenail.me/NN-SVG/. [Online; accessed 27-January-2024].

[2] STEPHEN G. BRIJSH. History of the lenz-ising model. *Reviews of Modern Physics*, 1967.

[3] Wolfgang Paul Johannes J. Schneider Tobias Preis, Peter Virnau. Gpu accelerated monte carlo simulation of the 2d and 3d ising model. *Journal of Computational Physics*, 2009.

[4] Barry A. Cipra. *An Introduction to the Ising Model*. Mathematical Association of America, 1987.

[5] G. GALLAVOTTI. Instabilities and phase transitions in the ising model. a review. *RIVISTA DEL NUOVO CIMENTO*, 1972.

[6] R P Gammag L A Aviles. Critical temperature in 2d square ising model with anisotropic coupling constant using kramers-wannier duality. *IOP*, 2023.

[7] Claudio Bonati. The peierls argument for higher dimensional ising models. *European Journal of Physics*, 2014.

[8] Ciencias ULisboa. Transições de fase. https://fenix.ciencias.ulisboa.pt/courses/tf-2254879305237353/ver-artigo/modelo-de-ising-831. [Online; accessed 16-January-2024].

[9] E. Betz and U. Dürr. Mn2+ induced magnon gap mode in the 2-d antiferromagnet k2cof4. *Physica B+C*, 1980.

[10] Neural networks – state of art, brief history, basic models and architecture. 2016.

[11] Thomas Dean David Daniel Conx. Neural networks and neuroscience-inspired computer vision. *Current Biology*, 2014.

[12] Michael Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[13] IBM. What are neural networks? https://www.ibm.com/topics/neural-networks#:~:text=Neural%20networks%2C%20also%20known%20as,neurons%20signal%20to%20one%20another. [Online; accessed 27-December-2023].

[14] Google. Framing: Key ml terminology. https://developers.google.com/machine-learning/crash-course/framing/ml-terminology. [Online; accessed 30-December-2023].

[15] Patrick K. Simpson. *Neural Networks Applications*. IEEE Press, 1997.

[16] Hong Zhu Tong Yu. Hyper-parameter optimization: A review of algorithms and applications. *ArXiv*, 2020.

[17] Abdallah Shami Li Yang. *On hyperparameter optimization of machine learning algorithms: Theory and practice*. Neurocomputing, 2020.

[18] Wang Z Rawat, W. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 2017.

[19] A. Karpathy. Cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/classification/, 2016. [Online; accessed 24-January-2024].

[20] Jahani Heravi E Habibi Aghdam, H. *Guide to Convolutional Neural Networks*. Springer, 2017.

[21] Vibhor J. Mishra A. Sharma, N. An analysis of convolutional neural networks for image classification. *International Conference on Computational Intelligence and Data Science*, 2018.

[22] Dong J. Li H. Gao Y. Guo, T. Simple convolutional neural network on image classification. *IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, 2017.

[23] Google. Ml practicum: Image classification. https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks. [Online; accessed 30-December-2023].

[24] Martin Grant. Thermal physics lecture notes, 2018.

[25] M. E. J. Newman and G. T. Barkema. *Monte Carlo Methods in Statistical Physics*. Oxford University Press Inc, 1999.

[26] L. Polson. The ising model in python: Statistical mechanics and permanent magnets. https://github.com/lukepolson/youtube_channel/blob/main/Python%20Metaphysics%20Series/vid14.ipynb. [Online; accessed 20-October-2023].

[27] Jindal S. *Monte Carlo Simulation of the Ising Model*. University of California, 2007.

[28] Jacques Kotze. Introduction to monte carlo methods for an ising model of a ferromagnet. *ArXiv*, 2008.

[29] NumPy documentation. numpy.random.exponential. https://numpy.org/doc/stable/reference/random/generated/numpy.random.exponential.html. [Online; accessed 31-January-2024].

[30] Numba-User Manual. What is nopython mode? https://numba.pydata.org/numba-doc/latest/user/5minguide.html#what-is-nopython-mode. [Online; accessed 31-January-2024].

[31] Patrick van der Smagt Ben Krose. *An introduction to neural networks*. University of Amsterdam, 1996.

[32] Christopher Olah. Neural networks, manifolds, and topology. https://colah.github.io/posts/

2014-03-NN-Manifolds-Topology/. [Online; accessed 04-Febuary-2024].

[33] Kannan R. Alexander S. A. Kanagachidambaresan G. R. Prakash, K. B. *Advanced Deep Learning for Engineers and Scientists*. Springer, 2021.

[34] Jason Brownlee. What is the difference between a batch and an epoch in a neural network? *Deep Learning*, 2018.

[35] Eduardo Pasiliao Chaity Banerjee, Tathagata Mukherjee. An empirical study on generalizations of the relu activation function. *ACMSE*, 2019.

[36] Sridhar Narayan. The generalized sigmoid activation function: Competitive supervised learning. *Information Sciences*, 1997.

[37] Benjamin Graham. Fractional max-pooling. *arXiv*, 2015.

[38] Nitish Srivastava. Improving neural networks with dropout, 2013.

[39] Google. Generalization: Peril of overfitting. https://developers.google.com/machine-learning/crash-course/generalization/peril-of-overfitting. [Online; accessed 01-February-2024].

[40] Geraldine Bessie Amali Dc Venkataraman Muthiah-Nakarajand Mathew Mithra Noela, Arindam Banerjeeb. Alternate loss functions for classification and robust regression can improve the accuracy of artificial neural networks. *arXiv*, 2023.

[41] Google. Reducing loss: Learning rate. https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate. [Online; accessed 01-February-2024].

[42] Thomas Kurbiel and Shahrzad Khaleghia. Training of deep neural networks based on distance measures using rmsprop. *arXiv*, 2017.

[43] Stephen V. Stehman. *Selecting and interpreting measures of thematic classification accuracy*. Remote Sensing of Environment, 1997.

[44] Google. Práctica de aa: Clasificación de imágenes. https://developers.google.com/machine-learning/practica/image-classification/exercise-3?hl=es-419. [Online; accessed 10-February-2024].

Anexos

Anexos

Modelo de Ising 2D

Listing 1: Ising 2D

```
import numpy as np
import matplotlib.pyplot as plt
import numba
import os
from datetime import datetime

N = 445
M = 446
times=80000
scale=0.1*times

num_img=10000
iteraciones=np.random.exponential(scale=scale,
    size=num_img).astype(int)
iteraciones.sort()

init_random = np.random.random((N,M))
red = np.zeros((N, M))
limits=0.5
red[init_random<=limits] = 1
red[init_random>limits] = -1


@numba.njit(nopython=True, nogil=True)
def metropolis(spin_arr, BJ):
        spin_arr = spin_arr.copy()
        x = np.random.randint(0,N)
        y = np.random.randint(0,M)
        spin_i = spin_arr[x,y]
        spin_f = spin_i*-1

        E_i = 0
```

```
        E_f = 0
        if x>0:
            E_i += -spin_i*spin_arr[x-1,y]
            E_f += -spin_f*spin_arr[x-1,y]
        if x<N-1:
            E_i += -spin_i*spin_arr[x+1,y]
            E_f += -spin_f*spin_arr[x+1,y]
        if y>0:
            E_i += -spin_i*spin_arr[x,y-1]
            E_f += -spin_f*spin_arr[x,y-1]
        if y<M-1:
            E_i += -spin_i*spin_arr[x,y+1]
            E_f += -spin_f*spin_arr[x,y+1]

        dE = E_f-E_i
        if (dE>0)*(np.random.random() < np.exp(-BJ*dE)):
            spin_arr[x,y]=spin_f
        elif dE<=0:
            spin_arr[x,y]=spin_f

        return spin_arr

NT=N*M
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
folder_name = f'resultados_{NT}_{timestamp}'
os.makedirs(folder_name, exist_ok=True)

Sinitial=red.copy()

with open(os.path.join(folder_name, f'{NT}_spins.txt'), 'w') as
    file:
     for t in range(0,times-1):
        spins_arr = metropolis(Sinitial, 1/2.1)
        Sinitial=spins_arr

        if t in iteraciones:
            small_spins = spins_arr[198:248, 198:248]
            plt.imshow(small_spins, cmap='binary')
```

```
            plt.axis('off')
            image_path = os.path.join(folder_name,
                f'{NT}-Spins_{t}.png')
            plt.savefig(image_path, bbox_inches='tight',
                pad_inches=0)
            plt.close()
        if t == iteraciones[-1]:
            break
```

Modelo de Ising 3D

Listing 2: Ising 3D

```
import numpy as np
import matplotlib.pyplot as plt
import numba
import os
from datetime import datetime


N = 80
M = 80
P = 31
times=8e7
scale=0.1*times

num_img=10000
iteraciones=np.random.exponential(scale=scale,
    size=num_img).astype(int)
iteraciones.sort()

init_random = np.random.random((N,M,P))
red = np.zeros((N, M, P))
limits=0.5
red[init_random<=limits] = 1
red[init_random>limits] = -1

@numba.njit(nopython=True, nogil=True)
```

```
25 def metropolis(spin_arr, BJ):
26         spin_arr = spin_arr.copy()
27         x = np.random.randint(0,N)
28         y = np.random.randint(0,M)
29         z = np.random.randint(0,P)
30         spin_i = spin_arr[x,y,z]
31         spin_f = spin_i*-1
32 
33         E_i = 0
34         E_f = 0
35         if x>0:
36             E_i += -spin_i*spin_arr[x-1,y,z]
37             E_f += -spin_f*spin_arr[x-1,y,z]
38         if x<N-1:
39             E_i += -spin_i*spin_arr[x+1,y,z]
40             E_f += -spin_f*spin_arr[x+1,y,z]
41         if y>0:
42             E_i += -spin_i*spin_arr[x,y-1,z]
43             E_f += -spin_f*spin_arr[x,y-1,z]
44         if y<M-1:
45             E_i += -spin_i*spin_arr[x,y+1,z]
46             E_f += -spin_f*spin_arr[x,y+1,z]
47         if z > 0:
48             E_i += -spin_i * spin_arr[x, y, z-1]
49             E_f += -spin_f * spin_arr[x, y, z-1]
50         if z<P-1:
51             E_i += -spin_i * spin_arr[x, y, z+1]
52             E_f += -spin_f * spin_arr[x, y, z+1]
53 
54         dE = E_f-E_i
55         if (dE>0)*(np.random.random() < np.exp(-BJ*dE)):
56             spin_arr[x,y,z]=spin_f
57         elif dE<=0:
58             spin_arr[x,y,z]=spin_f
59 
60         return spin_arr
61 
62 
```

```
NT=N*M*P
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
folder_name = f'resultados_{NT}_{timestamp}'
os.makedirs(folder_name, exist_ok=True)


Sinitial=red.copy()

with open(os.path.join(folder_name, f'{NT}_spins.txt'), 'w') as
    file:
     cont=0
     for t in range(0,times-1):
         spins_arr = metropolis(Sinitial,1/2.1)
         Sinitial=spins_arr
         if t in iteraciones:
             small_spins = spins_arr[15:65, 15:65, int(P/2)]
             plt.imshow(small_spins, cmap='binary')
             plt.axis('off')
             image_path = os.path.join(folder_name,
                f'{NT}-Spins_{t}.png')
             plt.savefig(image_path, bbox_inches='tight',
                pad_inches=0)
             plt.close()
         if t == iteraciones[-1]:
              break
```

Red Neuronal Convolucional

Listing 3: Red Neuronal Convolucional

```
import os
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.callbacks import EarlyStopping
```

```
from tensorflow.keras.optimizers import RMSprop
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tqdm import tqdm
import seaborn as sns

def load_images_from_folder(folder, label, target_size):
    data = []
    labels = []
    for subfolder in os.listdir(folder):
        subfolder_path = os.path.join(folder, subfolder)
        if os.path.isdir(subfolder_path):
            for archive in tqdm(os.listdir(subfolder_path),
               desc=f'Cargando Im genes en {subfolder}'):
                if archive.endswith('.png') and
                   (archive.startswith('198470-Spins_') or
                   archive.startswith('198400-Spins_')):
                    archive_path = os.path.join(subfolder_path,
                       archive)
                    image =
                       Image.open(archive_path).resize(target_size).convert('
                    image = np.array(image)
                    data.append(image)
                    labels.append(label)
    return data, labels

train_2D_dir = 'Ising 2D/'
train_3D_dir = 'Ising 3D/'
target_size = (50, 50)
data_2D, labels_2D = load_images_from_folder(train_2D_dir,
    label=0, target_size=target_size)
data_3D, labels_3D = load_images_from_folder(train_3D_dir,
    label=1, target_size=target_size)
train_data = []
labels = []
train_data = data_2D + data_3D
labels = labels_2D + labels_3D
labels = np.array(labels)
```

```
40
41 train_data_n = np.array(train_data)/ 255.0
42 X_train, X_val, y_train, y_val = train_test_split(train_data_n,
       labels, test_size=0.2, random_state=42)
43
44
45 img_input = layers.Input(shape=(50, 50, 1))
46 x=layers.Conv2D(16, 2,activation='relu')(img_input)
47 x=layers.MaxPooling2D(2)(x)
48 x=layers.Flatten()(x)
49 x=layers.Dense(8, activation='relu')(x)
50 x=layers.Dense(16, activation='relu')(x)
51 x=layers.Dropout(0.5)(x)
52 output=layers.Dense(1, activation='sigmoid')(x)
53
54 model=Model(img_input, output)
55 model.summary()
56
57
58 model.compile(loss='binary_crossentropy',
59               optimizer = RMSprop(learning_rate=0.001),
60               metrics=['acc'])
61 early_stopping = EarlyStopping(monitor='val_loss', patience=10,
       verbose=1, restore_best_weights=True)
62 history = model.fit(X_train, y_train, epochs=100,
       validation_data=(X_val, y_val), verbose=1,
       callbacks=[early_stopping])
63 test_loss,test_acc = model.evaluate(X_val, y_val)
64 print(f'Loss␣on␣test␣set:␣{test_loss}')
65
66
67 y_pred = model.predict(X_val)
68 y_pred_classes = (y_pred > 0.5).astype(int)
69 conf_matrix = confusion_matrix(y_val, y_pred_classes)
70 class_labels = ['2D', '3D']
71 plt.figure(figsize=(8, 6))
72 ax = sns.heatmap(conf_matrix, fmt='g', cmap='Blues',
73             xticklabels=class_labels,
```

```
            yticklabels=class_labels)

for i in range(len(class_labels)):
    for j in range(len(class_labels)):
        color = "white" if i == j else "black"
        text = ax.text(j + 0.5, i + 0.5, conf_matrix[i, j],
                       ha="center", va="center", color=color,
                          fontsize=12)

current_datetime = datetime.now().strftime("%Y%m%d_%H%M%S")
plt.xlabel('Predicciones', fontsize=12)
plt.ylabel('Valores reales', fontsize=12)
confusion_matrix_filename =
    f'confusion_matrix_{current_datetime}.png'
plt.savefig(confusion_matrix_filename)
plt.close()
```