



Máster en Física Avanzada

Especialidad Física Teórica



VNIVERSITAT
DE VALÈNCIA

Trabajo de Fin de Máster

Búsqueda de Simetrías en la Naturaleza con Técnicas de
Machine Learning

Gabriel Andy Szalkowski *

Tutora (1): Gabriela Alejandra Barenboim Szuchman

Tutor (2): Johannes Hirn

Tutora (3): Verónica Sanz González

Curso Académico 2020 - 2021

* gandys@alumni.uv.es

Índice

1. Introducción	3
2. Machine Learning: Qué significa que un ordenador aprenda	4
2.1. Ejemplo 1: Gatos y Perros	6
2.2. Ejemplo 2: Titanic	8
2.3. <i>Machine Learning</i> Aplicado a la Física	9
3. Fundamento Teórico	10
3.1. ¿Qué es <i>Machine Learning</i> ? Variantes	10
3.2. Redes Neuronales	11
4. Simetrías y Machine Learning: Estado del Arte	12
5. Búsqueda de Simetrías en 2 Dimensiones	13
5.1. Entorno de trabajo: Python y <i>fastai</i>	13
5.2. Esquema del Procedimiento	14
5.3. Potenciales con Simetría en 2 Dimensiones	14
5.4. Obtención de imágenes usando PCA	16
5.5. Análisis de las imágenes de PCA usando una CNN	18
6. Resultados	20
7. Conclusiones y Futuros Pasos	21
8. Agradecimientos	23
Anexos	23
A. Cuantiles	23
B. <i>Troubleshooting</i>	26

Summary

The objective of this project is to study whether a neural network can be able to identify symmetries in data sets while learning to perform a task on that set, adapting the procedure of the article [5] to the continuous case. In order to do this, we first introduce the fundamentals of the Machine Learning techniques that we used. Subsequently, the procedure followed to achieve a model capable of evaluating the understanding of symmetries by a neural network is detailed. This procedure is based on generating thousands of two-dimensional potentials $V(x, y)$ with the same symmetries as those used in [5], and training a neural network for each of them. The objective of these networks is, given x e y , to predict the value of the potential. Projecting the results from the last hidden layer of these networks, we obtain two-dimensional images, with which we train a neural network dedicated to classifying them according to the symmetry of the potential from which these images come. The smallest and most robust of the resulting networks is capable of identifying the original symmetry in 96 % of the cases, thus showing that the networks that predict the value of the potential encode information about its symmetry. Finally, we analyze the meaning of these results and the possible ways to further develop this concept.

Resumen

El objetivo de este trabajo es estudiar si una red neuronal puede ser capaz de encontrar simetrías en conjuntos de datos mientras aprende a realizar una tarea sobre ese conjunto, adaptando el procedimiento del artículo [5] al caso continuo. Para ello, se introducen primero los fundamentos de las técnicas de *Machine Learning* utilizadas y se detalla el procedimiento seguido para alcanzar un modelo capaz de evaluar la comprensión de la simetría por parte de una red neuronal. Dicho procedimiento se basa en generar miles de potenciales bidimensionales $V(x, y)$ con las mismas simetrías que las utilizadas en [5], y entrenar una red neuronal para cada uno de ellos. El objetivo de estas redes es, dados x e y , predecir el valor del potencial. Proyectando los resultados provenientes de la última capa oculta de estas redes obtenemos imágenes bidimensionales, con las que entrenamos una red neuronal encargada de clasificarlas según la simetría del potencial del que provienen. La red resultante más pequeña y a la vez más robusta es capaz de identificar la simetría original en el 96 % de los casos, mostrando así que las redes que predicen el valor del potencial codifican información acerca de la simetría del mismo. Finalmente, analizamos el significado de estos resultados y las posibles vías para seguir desarrollando este concepto.

1. Introducción

Comencemos con una pregunta sencilla. ¿Puede un ordenador entender qué es una simetría? La simetría es un concepto humano. Todo ser humano tiene cierta idea intuitiva de lo que significa que algo sea simétrico. De hecho, las personas tenemos tendencia a sentirnos atraídos por la simetría, ya sea en el arte, en la ciencia o en la vida diaria. Tanto así que, independientemente de la cultura, existe una preferencia clara por la simetría a la hora de evaluar lo atractivo que es un rostro (refs. [15], [22]). Este concepto humano de simetría es por tanto una herramienta fundamental que utilizamos para navegar y comprender el mundo que nos rodea.

El objetivo de la Ciencia Física es precisamente describir las leyes que rigen el Universo, de forma que los humanos podamos entender su funcionamiento. Todas las leyes de las que

disponemos, desde las leyes de Newton y las leyes de Maxwell hasta la Relatividad General y el Modelo Estándar, que describen infinitud de fenómenos naturales, encuentran su origen en principios de simetría. Es por ello que disponer de una herramienta capaz de encontrar hasta las simetrías que no somos capaces de ver a simple vista sería algo de enorme utilidad. Esta es pues la motivación de la pregunta del principio, y a su vez la motivación de este estudio.

El planteamiento de este trabajo sigue las líneas del artículo [5]: usar Inteligencia Artificial para conseguir un código capaz de reconocer simetrías. Para entenderlo, comenzaremos con una breve introducción a las técnicas de *Machine Learning* con algunos ejemplos para dar una muestra de por qué nos pueden resultar tan útiles, en la sección 2. Más adelante, en la sección 3 explicaremos con algo más de detalle las técnicas utilizadas, centrándonos en las redes neuronales. Al ser la base de este trabajo, comentaremos también algunos detalles del artículo [5] en la sección 4. Posteriormente, en la sección 5 explicaremos el trabajo realizado hasta conseguir un modelo capaz de reconocer ciertas simetrías para un caso concreto. Finalmente, en la sección 6 comentaremos los resultados del aprendizaje de este modelo, valorando la precisión de las predicciones del modelo obtenido, concluyendo en la sección 7 las implicaciones de dichos resultados así como futuras posibilidades para continuar esta investigación.

2. Machine Learning: Qué significa que un ordenador aprenda

Antes de empezar con los detalles técnicos, merece la pena dedicar algo de tiempo a responder a la pregunta de por qué estamos haciendo este tipo de estudio y por qué esto también es hacer ciencia. Para ello, debemos comenzar asumiendo que las capacidades de nuestra mente son asombrosas, aunque asombrosamente limitadas a su vez. En general, nuestro conocimiento acerca de cualquier tema siempre pretende ser analítico (con analítico nos referimos a todo aquello cuya solución exacta conocemos). Al fin y al cabo, podría decirse que el objetivo fundamental de la Ciencia, y de la Física en concreto, es hallar una Teoría del Todo, un hipotético marco teórico único y coherente, que describa completamente (de forma analítica) los aspectos físicos del Universo y sus relaciones [27]. Sin embargo, y estando muy lejos todavía de saber siquiera si semejante teoría podría llegar a existir, nos enfrentamos a diario a problemas cuya solución analítica nos es desconocida.

Ciertos problemas sin solución analítica conocida los resolvemos de forma consciente, especialmente en el ámbito científico. Algunos ejemplos podrían ser la predicción de las condiciones atmosféricas por los meteorólogos, la resolución numérica de las ecuaciones de la Relatividad General que permitieron obtener la primera imagen del agujero negro M87 por el EHT [3] y la resolución de las ecuaciones diferenciales que controlan la trayectoria y el aterrizaje del Rover Perseverance, recientemente desplegado en Marte [9]. Un ejemplo menos familiar para el público general, pero bien conocido en la comunidad de la Física Teórica, son los diagramas de Feynman, que en sí son términos de una expansión (y por tanto una aproximación) perturbativa a una amplitud de transición [25]. El procedimiento para resolver estos problemas por lo general es el mismo: dado un problema cuya solución analítica desconocemos, calcularemos una solución tan aproximada como podamos a base

de combinaciones lineales de términos cuya solución analítica sí que conocemos. Dicho de otra forma, si no conocemos la solución, encontramos algo que se le parezca a base de cosas que sí conocemos. Este procedimiento funciona muy bien para algunos problemas, pero no es aplicable en todos los casos.

Muchos casos a los que no aplica el procedimiento de la combinación lineal son precisamente problemas que resolvemos de forma inconsciente. Para aclarar este punto, supongamos por un momento que un conocido nos muestra una fotografía de su mascota. Suponiendo que no se trata de nada excepcionalmente inusual, sabríamos inmediatamente y sin pensar mucho, que se trata de un gato, un perro, un conejo, etc. La pregunta que podemos hacernos es: ¿cómo hemos podido identificar al animal de la fotografía? ¿Cuál ha sido el procedimiento que hemos seguido?

Para tratar de responderla, un pequeño ejercicio mental que podríamos hacer es el siguiente. Imaginemos que de repente nos podemos comunicar con un alienígena, que nunca ha visto ni un perro ni un gato. El ejercicio consiste en pensar cómo podríamos describirle estos animales para que, una vez este alienígena vea una imagen de uno, sepa distinguir si se trata de un gato o se trata de un perro. Esta tarea, aparentemente sencilla, se complica a medida que lo empezamos a pensar. Primero está la cuestión de enseñarle qué es lo que tiene que buscar en la foto, enseñarle a separar entre lo que es un animal y lo que es por ejemplo un árbol, una manta o un zapato. Cuando sepa encontrar al animal, podríamos describir la apariencia del mismo, enfatizando detalles como la forma de las orejas o los bigotes. Claro está que si los gatos tienen orejas puntiagudas, y nuestro amigo extraterrestre encuentra un perro con orejas puntiagudas, podría confundirse.

Hacer esta distinción no es para nada una tarea sencilla, aunque para nosotros sí que lo parece (tal y como hemos comenzado asumiendo, las capacidades de nuestra mente son asombrosas). Hay multitud de factores a tener en cuenta y procesos que hacemos de forma inconsciente. Distinguir perros de gatos es un ejemplo de tarea que no podemos aproximar por medio de soluciones de tareas más sencillas. Si en vez de ir a un caso extremo como el del alienígena, consideramos un caso más próximo a la realidad, este hecho se hace evidente. Supongamos que queremos hacer un programa informático que distinga gatos de perros. La cantidad de variables y relaciones entre ellas que entran en juego es tan grande que resulta (aparentemente por lo menos) imposible programarlo de forma analítica en un código finito.

En 1966, este problema fue altamente subestimado en el MIT, en su *summer vision project* [20]. Este proyecto tenía como objetivo final identificar objetos y asignarles un nombre de una lista de objetos conocidos. El problema resultó ser mucho más complicado de lo que se esperaba, y no fue hasta 1988, con el desarrollo e implementación de la red neuronal Neocognitron por Fukushima [10], cuando se consiguió algo remotamente similar al objetivo del *summer vision project*. El programa de Fukushima era capaz de reconocer cifras escritas a mano, algo mucho menos ambicioso que un programa de reconocimiento de objetos de carácter general. El detalle importante de este avance es que hacía uso de una red neuronal. No se trataba de una líneas de código dedicadas al reconocimiento de números escritas a priori (lo que podríamos llamar una combinación de soluciones analíticas), sino de un programa que había sido entrenado, a base de ejemplos, para realizar dicha tarea. El programa de Fukushima había “entendido”, tras haber visualizado cientos de cifras escritas a mano, los intrincados detalles de las formas, los tamaños, los patrones, las distintas escrituras de los números... La red había aprendido ciertas relaciones altamente no lineales,

que no podrían expresarse de forma analítica a priori.

El segundo hecho que hemos comenzado asumiendo es que, a pesar de ser asombrosas, las capacidades de nuestra mente son asombrosamente limitadas. Reconocer patrones es una tarea para la que el cerebro humano está particularmente especializado, hecho que tiene sentido desde una perspectiva evolutiva. Aun así, ante ciertos problemas los patrones no resultan evidentes, especialmente cuando se trata de cantidades enormes de datos que escapan de nuestra intuición cotidiana, como los factores que pueden afectar las ventas de una empresa o las trazas detectadas en un colisionador de partículas. Es en esos casos en los que las técnicas de *Machine Learning* sacan a relucir su potencial superando nuestras capacidades, cosa que sucede incluso en tareas que deberían resultarnos sencillas, como el reconocimiento de caras (ref. [18]). La capacidad de estas técnicas para reconocer relaciones y patrones dentro de un conjunto de datos las convierten en, posiblemente, una de las herramientas más potentes de las que disponemos para el avance a nivel científico y tecnológico. Por supuesto, aunque está claro que esto va mucho más allá de distinguir perros de gatos, curiosamente, ese es un gran punto donde empezar a conocer las técnicas de *Machine Learning*.

2.1. Ejemplo 1: Gatos y Perros

El primer ejercicio que suele plantearse a la hora de aprender cualquier técnica de *Machine Learning* es una clasificación binaria, entrenar una red neuronal para que distinga entre objetos de dos categorías. Siguiendo la temática del experimento mental del alienígena, veremos los resultados del aprendizaje de una red neuronal que distingue perros de gatos (sin entrar en detalle en el código, del que hablaremos en las secciones 3 y 5). Trabajamos con imágenes del *Oxford-IIIT Pet Dataset* (una selección aleatoria de éstas se muestra en la figura 1). Para facilitar el trabajo de clasificación, hemos etiquetado cada imagen con una categoría booleana, *True* si se trata de un gato, *False* si es un perro.



FIGURA 1: Algunas imágenes de la base de datos *Oxford-IIIT Pet Dataset* utilizadas en el entrenamiento de la red neuronal, etiquetadas según si son gatos (True) o perros (False).

De las 7390 imágenes de las que disponemos, entrenaremos una red neuronal (una *Convolutional Neural Network, CNN* concretamente) con el 80 % (*training set*), y reservaremos el 20 % para comprobar si el entrenamiento ha sido satisfactorio (*validation set*). Esa separación es importante para asegurarnos de que la red realmente ha “entendido” la tarea que le hemos propuesto, y no ha optado por memorizar todas las imágenes del conjunto de entrenamiento, que a menudo es el procedimiento más sencillo. A la hora de entrenar la red, le pasamos todas las imágenes de entrenamiento varias veces, y cada vez comprobamos la tasa de error (*error rate*), es decir, cuántas veces se equivoca al mostrarle las imágenes del *validation set*. Nos quedaremos con el modelo que menos se equivoque. En la figura 2 se muestra la tasa de error para cada *epoch* (cada vez que le mostramos a la red todas las imágenes del *training set*). En la *epoch* 8, la tasa de error es mínima, un 0.27 %, es decir, de las 1478 imágenes del *validation set* (20 % del total) se ha equivocado al predecir únicamente 4 de ellas. Estas cuatro imágenes aparecen en la figura 3.

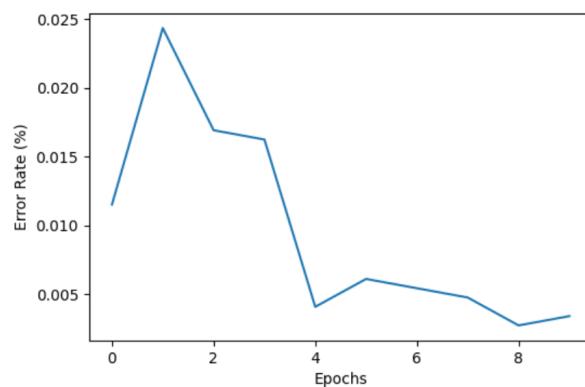


FIGURA 2: Tasa de Error (*Error Rate*) de la red neuronal al hacer una clasificación binaria en gatos y perros de las imágenes del *validation set*, para cada *epoch*.

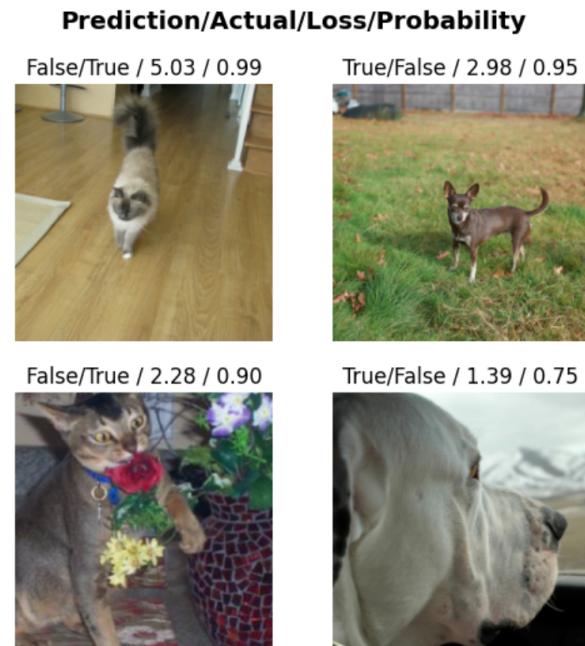


FIGURA 3: Cuatro imágenes cuya categoría la red neuronal ha predicho de forma incorrecta, junto a la probabilidad que le ha asignado la red a la predicción. Nótese que varios de estos fallos se tratan de casos en los que las imágenes muestran tomas peculiares o con objetos, de forma que el error es comprensible.

La consecuencia más importante de estos resultados no reside en que ahora tenemos un

programa capaz de discernir entre perros y gatos. El resultado de mayor relevancia es que tenemos herramientas capaces de “comprender” de alguna forma las diferencias entre varias categorías, y son capaces de, a partir de la experiencia, resolver problemas demasiado complejos para nuestras herramientas analíticas. Este programa concreto para esta clasificación ocupa únicamente 5 líneas de código, desde que las imágenes son importadas hasta obtener un modelo tan bueno que, ante un problema de esta complejidad, es capaz de acertar el 99.7 % de las veces. Con este detalle queda más que claro que tenemos a nuestra disposición herramientas relativamente sencillas que pueden ayudarnos de forma inmensurable a ampliar nuestro conocimiento del mundo.

2.2. Ejemplo 2: Titanic

Es posible que un ejemplo como clasificar mascotas no resulte del todo convincente para algunas personas. Para estos casos, tenemos otro ejemplo: el Titanic. Disponemos de una lista de 891 pasajeros del famoso transatlántico en un fichero con detalles como su edad, sexo, clase económica, si tenían o no familiares a bordo...¹ De esa lista, sabemos quién sobrevivió y quién no lo logró. En base a esta información, dados los datos de un nuevo pasajero, queremos ser capaces de predecir si sobrevivirá o no.

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

FIGURA 4: Fragmento de la lista de pasajeros del Titanic con los datos utilizados para entrenar una red neuronal, cuyo objetivo es predecir si un pasajero sobrevivirá o no. El significado de cada columna puede consultarse en <https://www.kaggle.com/c/titanic/data>.

Para una persona, un fichero con este tipo de datos es algo completamente abstracto. A pesar de poder formular suposiciones sobre el problema, como que quizás una persona de 25 años tiene mayor probabilidad de sobrevivir que una de 75, nuestra intuición sobre ello es muy pobre. Este hecho junto a la gran cantidad de datos a analizar, hace que esta tarea sea prácticamente imposible (o por lo menos muy complicada) para una persona sola. Sin embargo, utilizando muy pocas líneas de código para programar una red neuronal al igual que antes, podemos llegar a hacer predicciones que fácilmente llegan a superar el 85 % de precisión.

Si bien es cierto que la facilidad de hacer este tipo de predicciones sobre temas relativos a vidas humanas llega a ser algo aterrador, dejando cuestiones morales aparte, la potencia de los algoritmos de *Machine Learning* queda bien clara. Tenemos a nuestra disposición un código capaz de analizar y encontrar patrones en una cantidad inmensa de datos completamente abstractos. Desde este punto de vista, resulta obvio que se trata de una herramienta de gran valor, especialmente en el ámbito científico.

¹Los datos utilizados provienen de la competición de Kaggle “Titanic: Machine Learning from Disaster”, <https://www.kaggle.com/c/titanic/overview>

2.3. *Machine Learning* Aplicado a la Física

Las técnicas de *Machine Learning* se extienden a todas las ramas del conocimiento científico, y la Física no es una excepción. Veamos algunos ejemplos.

En la sección 2.2 se puso de manifiesto la capacidad de análisis de grandes cantidades de datos abstractos. En esta situación, lo primero que le viene a la mente a cualquier persona que se dedique a la física es el colosal volumen de información registrado en las colisiones del LHC. Las colisiones registradas consisten de una enorme cantidad de datos de carácter multidimensional, cuyo análisis estadístico de forma tradicional resulta insuficiente (ref. [11]). Es por ello por lo que las técnicas de *Machine Learning* encuentran una potente aplicación en el LHC ([11]) y anteriormente en el LEP([24]). Un objetivo fundamental de estas técnicas en el LHC es la reducción de las medidas de los detectores a un conjunto de datos de alto nivel y menor dimensionalidad, que puedan ser más comprensibles a la hora de analizarlos.

En general las aplicaciones de *Machine Learning* en Física de Altas Energías se basan en clasificación binaria, al igual que el ejemplo de las mascotas. El objetivo puede ser distinguir lo que es fondo (*Background*, eventos sin interés) de lo que es señal (*Signal*, eventos de interés). En base a esto, la búsqueda de nueva física usando *Machine Learning* y concretamente *Deep Learning* ofrece de forma consistente mejores resultados que los métodos tradicionales de *feature engineering* (ref. [4]).

Siguiendo con el campo de la Física de Altas Energías, las técnicas de *Machine Learning* son utilizadas en gran medida para la clasificación de jets, *flavour (jet) tagging*. Puesto que las partículas con, por ejemplo, quarks b tardan en desintegrarse del orden de picosegundos, haces de estas partículas pueden alejarse lo suficiente del centro de la colisión como para ser detectados e identificados. Los productos de la desintegración de estas partículas forman un jet, un haz de partículas altamente colimado, cuya reconstrucción puede arrojar luz sobre procesos más allá del modelo estándar ([7]). Para llevar a cabo esta reconstrucción, es necesario etiquetar las partículas que forman el jet según si provienen de la desintegración de una partícula con contenido en quarks b, y para este etiquetado son de gran ayuda las técnicas de *Machine Learning*, tanto las más sencillas como Regresión Logística [2] como las más complejas, como redes neuronales [1].

Este tipo de técnicas van más allá de la Física de colisionadores. En el caso de la astrofísica, se han utilizado para el estudio de ondas gravitatorias en LIGO ([6]), con el fin de identificar “*glitches*” de los detectores y distinguirlos de señales de ondas gravitatorias verdaderas. También se ha aplicado *Machine Learning* a la detección de agujeros negros ([21]), clasificando *clusters* de estrellas según si contienen o no un agujero negro de masa intermedia.

Sin duda esta breve lista de ejemplos se queda muy corta, pero es suficiente para mostrar la infinidad de aplicaciones de interés científico de las técnicas de *Machine Learning*, que de cierta forma justifican este estudio.

3. Fundamento Teórico

En esta sección estudiamos las bases técnicas necesarias para el desarrollo de este Trabajo de Fin de Máster. Hablaremos primero del concepto de *Machine Learning* para centrarnos concretamente en las redes neuronales. También describiremos el entorno en el que se ha llevado la tarea de programación y entrenamiento de la redes utilizadas tanto para los ejemplos de las secciones 2.1 y 2.2 como para la parte principal del TFM.

3.1. ¿Qué es *Machine Learning*? Variantes

Machine Learning es un término “paraguas” que engloba las técnicas utilizadas para programar un ordenador con el fin de optimizar algún criterio de rendimiento en base a ejemplos o experiencia previa (ref. [14]). Por tanto, las técnicas de *Machine Learning* están basadas en un conjunto de datos sobre algún tipo de proceso y un algoritmo que sea capaz de aprender de dichos datos para hacer predicciones sobre futuros sucesos y/o describir los sucesos observados. Así pues, el *Machine Learning* es un subconjunto de lo que se conoce como inteligencia artificial, todo programa capaz de razonar y actuar de forma adaptativa. Dentro del conjunto de *Machine Learning* tenemos las técnicas de *Deep Learning* que hacen uso de redes neuronales profundas (ver figura 5).

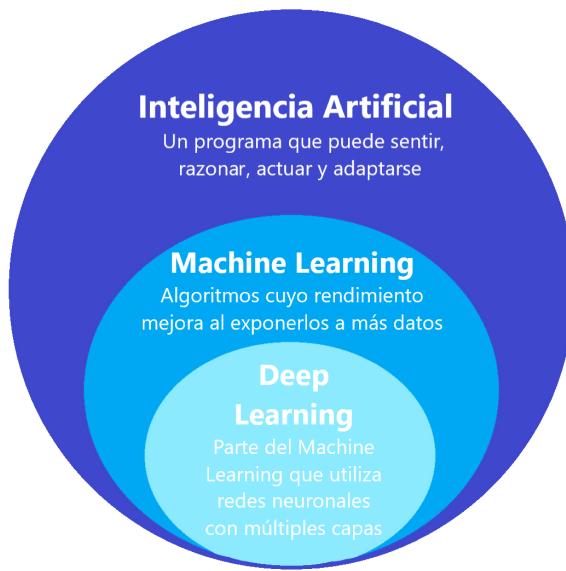


FIGURA 5: Representación pictórica de los conjuntos de Inteligencia Artificial, *Machine Learning* y *Deep Learning*

Existen varias formas de que los algoritmos aprendan a partir de datos. Las principales son *supervised learning* (aprendizaje supervisado), *unsupervised learning* (aprendizaje sin supervisión) y *reinforcement learning* (aprendizaje por refuerzo) [16]. El aprendizaje supervisado, usado para tareas como clasificación y regresión, consiste en aprender de datos etiquetados, como el conjunto de imágenes de gatos y perros usadas en la sección 2.1. Trataremos principalmente con este tipo de aprendizaje. El aprendizaje sin supervisión sirve

para encontrar patrones que permitan la clasificación y agrupación de sucesos similares, dentro de un conjunto sin etiquetar. Algunos ejemplos son la reducción dimensional y el *clustering* de patrones, como cuando se trata de estudiar el comportamiento de consumidores de un servicio (con el fin, por ejemplo, de enfocar cierto tipo de anuncios a cierto tipo de consumidor). El *reinforcement learning* tiene como fundamento maximizar una recompensa a medida que el algoritmo interacciona con el entorno. Estos algoritmos se pueden usar, entre otras aplicaciones, para enseñar a un ordenador a jugar a un videojuego, como Mario Bros. ([23]), de modo que las acciones que acerquen a Mario al final del nivel aumentan la recompensa.

3.2. Redes Neuronales

Dentro de las diversas técnicas de *Machine Learning* destacan las redes neuronales (NNs, *Neural Networks*), que nos permiten explorar el mundo del *Deep Learning* (ver figura 5). Puesto que este proyecto trata principalmente con redes neuronales, describiremos brevemente el fundamento de una. Esto nos permitirá comprender mejor el procedimiento seguido en la sección 5.

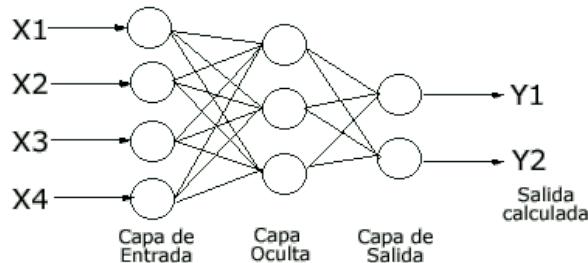


FIGURA 6: Esquema de una red neuronal. Fuente: <https://unpocodejava.com/2011/03/29/redes-de-neuronas-artificiales-con-netbeans/>

Una red neuronal es un algoritmo no lineal dedicado al aprendizaje supervisado, inspirado en las neuronas biológicas [16]. Consta en general de una capa de *input*, una capa de *output* y varias capas intermedias, llamadas *deep layers*, *hidden layers* o capas ocultas (ver figura 6). Cada neurona es un elemento que lleva a cabo una transformación no lineal de una combinación lineal de *inputs*. En otras palabras, una neurona recibe como entrada un vector, con cuyos elementos obtiene un valor por medio de una combinación lineal, para efectuar sobre dicho valor una transformación no lineal (conocida como función de activación). Dado un vector $\chi = (x_1, x_2, \dots, x_n)$, una neurona lleva a cabo una transformación del tipo

$$z = \mathbf{w} \cdot \chi + b = \mathbf{x}^T \cdot \mathbf{W} ,$$

donde \mathbf{w} es un vector de pesos y b es un escalar (coeficientes de la transformación lineal). Hemos definido también $\mathbf{x} = (1, \chi)$ y $\mathbf{W} = (b, \mathbf{w})$, de modo que z , el resultado de la transformación lineal, es el resultado del producto escalar de dos vectores. A este resultado se aplica una transformación no lineal, una función de activación σ , de modo que la salida de la neurona es $a = \sigma(z)$. Algunas de las funciones de activación más utilizadas son ReLU, $\tanh(x)$ y sigmoidea o logística.

Cuando hablamos de entrenar una red neuronal, nos referimos a hallar los valores de los

pesos \mathbf{W} asociados a cada neurona que minimizan una cierta función de coste. Se trata pues de un problema clásico de minimizar una función respecto de varios parámetros. Con el fin de optimizar el cálculo, en vez de calcular el gradiente como se haría de forma habitual, se hace uso de métodos iterativos como *Stochastic gradient descent*, puesto que la cantidad de parámetros y de puntos donde calcular el gradiente puede llegar a ser muy grande. Este método consiste en aproximar el gradiente total de esta función de coste por el gradiente calculado en un subconjunto de los datos, escogido de forma aleatoria (ref. [17]). Al calcular el gradiente con un subconjunto aleatorio, el promedio resultante es una aproximación del gradiente de la función, lo que permite encontrar el mínimo de forma más rápida.

Una idea común a la hora de hablar de *Deep Learning* es que, conforme se añaden más capas a una red neuronal, ésta es capaz de “aprender” detalles más abstractos. Se dice entonces que cada capa de la red logra un mayor nivel de abstracción [8]. En el ejemplo de los gatos y los perros, la primera capa de la red neuronal respondería ante formas sencillas, como bordes de formas. Una capa más profunda se encargaría de reconocer estructuras más complejas, como el hocico, las patas, etc. Las últimas capas distinguirían entonces entre conceptos altamente abstractos y complejos, diferenciando el concepto de gato del concepto de perro.

En base a esta idea de abstracción, podemos suponer que la última oculta (penúltima capa de la red) organiza la información aprendida de la manera más compacta. Esto significa que cualquier concepto con cierto nivel de abstracción que la red haya aprendido estará codificado de alguna forma en esa última capa. Si extraemos la información de esta última capa podemos estudiar si hay algún tipo de información en esa capa sobre dicho concepto abstracto. Esta es la idea fundamental de este estudio, así como el artículo que le da origen (ref. [5]) del que hablaremos en la sección 4.

4. Simetrías y Machine Learning: Estado del Arte

De la misma forma que cualquier estudio, para comprender el que presentamos aquí, es muy oportuno hablar de su origen: el estudio de simetrías con *Deep Learning*, basado en el artículo [5]. Como se menciona en dicho artículo, las simetrías, concebidas como invariancias bajo una transformación, son el elemento central para entender la Naturaleza, puesto que permiten la deducción de toda ley física que la describe. Así pues, identificar una simetría presente en cierto proceso permite lograr una mayor comprensión del mismo. Este hecho que parece a priori algo abstracto tiene aplicaciones muy claras en casos como la cristalografía o física de partículas.

A pesar de ser las simetrías un concepto fundamental y a pesar de que las técnicas de *Machine Learning* hayan estado presentes desde hace décadas, la pregunta que se plantea en el artículo [5] es verdaderamente novedosa: ¿Puede un ordenador aprender el concepto de simetría o la ausencia de ella? Para responderla, Barenboim, Hirn y Sanz entrenan una red neuronal a partir de imágenes con varias simetrías e imágenes sin simetría, a las que se les han sustraído algunos píxeles. El objetivo de la red es ser capaz de completar esos píxeles que faltan a partir del resto de píxeles. Por cada imagen, se entrena una red neuronal dedicada a completarla. Suponiendo que la red ha podido extraer el concepto abstracto detrás de la simetría de la misma, la información de ésta aparecerá codificada en la última capa

oculta. Usando la técnica de análisis de las componentes principales (PCA), se proyecta la información de las neuronas de esta capa en una imagen bidimensional. Sabiendo a qué simetría corresponde cada imagen resultante, entranan una red neuronal de convolución que distinga entre unas simetrías u otras a partir de ellas.

Los resultados para las propias imágenes de prueba, imágenes con simetrías definidas, son considerablemente buenos, logrando una precisión del 73 % en la clasificación. Esto significa que las redes iniciales, al predecir los píxeles de las imágenes, han logrado aprender algo. Esto lo sabemos debido a que los valores de los pesos de la última capa se comportan de formas distintas dependiendo de la simetría en cuestión, compartiendo rasgos comunes entre imágenes de PCA del mismo tipo de simetría, lo que permite clasificarlas de alguna forma.

Aunque los resultados de clasificación de simetrías sean prometedores, la red resultante puede ponerse a prueba de formas algo más creativas, como hacen en el artículo [5]. Ya que estudiar la idea innovadora de la comprensión de las simetrías por un ordenador hace que dicho artículo sea el *estado del arte* en su campo, ¿por qué no aplicar los resultados a otro ámbito, donde la simetría juega también un papel fundamental? ¿Por qué no aplicarlo al Arte en sí? Tras aplicar el algoritmo a diversos cuadros de artistas de fama mundial, éste es capaz de reconocer simetrías en ellos, como la clara simetría de translación discreta en cuadros de Kandinsky o de Miró. Una de las predicciones posiblemente más sorprendentes del algoritmo es que en los cuadros de Pollock, como *Number 1, 1948* mostrado en el artículo, domina la ausencia de simetría, como era de esperar. Esto nos lleva a la conclusión de que, aunque sea de forma aproximada, la red neuronal fue capaz de aprender los conceptos de distintas simetrías o de su ausencia.

5. Búsqueda de Simetrías en 2 Dimensiones

El primer objetivo de este trabajo es conseguir un modelo capaz de encontrar simetrías en un conjunto de datos de carácter continuo, empezando por datos bidimensionales, del tipo $[x, y]$ con un resultado $V(x, y)$. Aunque las aplicaciones prácticas de este modelo son muy diversas, iremos paso a paso partiendo de un problema sencillo, donde $V(x, y)$ es un potencial conocido, una función matemática concreta, hasta obtener un modelo que cumpla nuestro objetivo.

5.1. Entorno de trabajo: Python y fastai

Como paso previo a la explicación del procedimiento seguido para el estudio de simetrías, conviene definir el entorno de trabajo. Todo el trabajo, tanto de estudio de ejemplos como el de programación en sí (véase sección 2), fue llevado a cabo en el lenguaje de programación Python principalmente a través de la plataforma *Google Colab*², cuyo formato es el de *Jupyter Notebooks*. La mayoría del trabajo de entrenamiento de redes neuronales en este estudio se basa en las herramientas del paquete *fastai* [26], uno de cuyos requisitos es tener

²<https://colab.research.google.com>

disponible una GPU. Es por esta razón por la que se ha escogido el entorno *Google Colab*, puesto que ofrece de forma gratuita el acceso a GPUs de manera remota. Toda herramienta utilizada en este estudio es de código libre.

5.2. Esquema del Procedimiento

Si bien es cierto que el procedimiento seguido es muy similar al del artículo [5], que ya ha sido descrito brevemente en la sección 4, hay cierta cantidad de conceptos que merece la pena destacar. Al ser este procedimiento una parte fundamental del estudio realizado aquí, analizaremos en detalle cada paso. La explicación siguiente puede seguirse conjuntamente con el correspondiente *Notebook* en Google Colab: https://colab.research.google.com/drive/1l_WxjeCyxS7n8uP1piDdt3z59G4ABNBC?usp=sharing

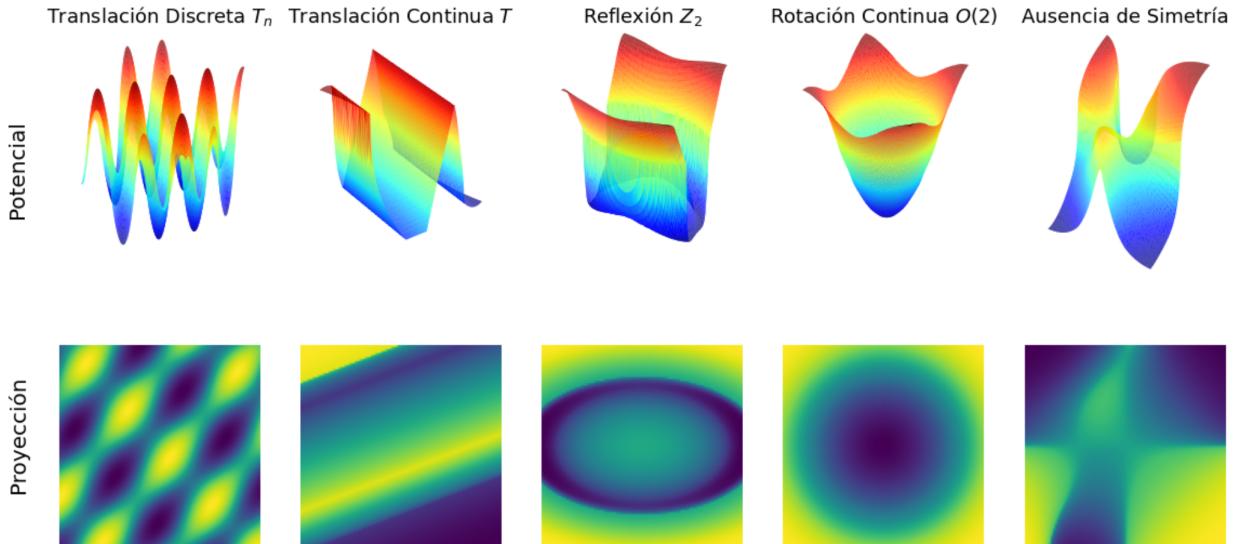
En términos generales, el objetivo de esta fase es desarrollar una forma de generalizar el trabajo del artículo [5] para un conjunto de datos de carácter continuo, puesto que el código original estaba especializado en una clasificación binaria. Los pasos llevados a cabo para lograrlo, explicados en las secciones siguientes, se resumen en:

1. Generar potenciales $V(x, y)$ con diferentes simetrías.
2. Separar en dos categorías los valores del potencial generados, escogiendo de forma aleatoria un 20% de ellos para el conjunto de validación.
3. Para cada potencial, entrenar una red neuronal de tipo FCNN (*Fully Connected Neural Network*) con los puntos de entrenamiento y escoger el modelo que ofrece las mejores predicciones del valor del potencial en los puntos de validación.
4. Para cada red entrenada, proyectamos los resultados de la última capa de neuronas a una imagen bidimensional, por medio del análisis de componentes principales (PCA).
5. Con todas las imágenes resultantes del PCA, entrenamos una CNN para que las clasifique según la simetría del potencial original.

5.3. Potenciales con Simetría en 2 Dimensiones

El primer paso es obtener un conjunto de potenciales simétricos pero con cierta variabilidad. Las simetrías utilizadas son $O(2)$ (Rotación Continua), T_n (Translación Discreta), T (Translación Continua) y Z_2 (Reflexión). A su vez, fueron generados potenciales aleatorios sin simetría (\emptyset). Algunos ejemplos de potenciales con estas simetrías pueden verse en la Figura 7. Estas simetrías han sido escogidas debido a que son las más comunes en física y son las simetrías a las que más estamos acostumbrados, puesto que pueden verse a simple vista con muy poco esfuerzo. Además, el resto de simetrías en dos dimensiones pueden obtenerse a base de una combinación de las utilizadas aquí.

Los potenciales utilizados son los mismos que sirvieron de base para el artículo [5], generados según el siguiente procedimiento:



- Definimos una función $f : \mathbb{R} \rightarrow \mathbb{R}^5/f(x) = (x, x^2, x^3, x^4, x^5)$. Esta función puede ser definida desde cualquier potencia de x hasta cualquier otra potencia mayor, pero elegimos $x^1 \rightarrow x^5$ con el fin de usar los mismos potenciales de entrenamiento que en el artículo [5].
- Generamos potenciales aleatorios para cada simetría usando la función f :
 - $O(2) : V(x, y) = f(\sqrt{x^2 + y^2}) \cdot \mathbf{c}$
 - $T : V(x, y) = f(k_1x + k_2y) \cdot \mathbf{c}$
 - $T_n : V(x, y) = k_1 \sin(k_2x + k_3y) + k_4 \cos(k_5x + k_6y)$
 - $Z_2 : V(x, y) = f(k_1x^2 + k_2y^2) \cdot \hat{\mathbf{C}} \cdot f(k_3x^2 + k_4y^2)^t$
 - $\emptyset : V(x, y) = f(x) \cdot \hat{\mathbf{C}} \cdot f(y)^t$

donde k_i son números aleatorios entre 0 y 1, \mathbf{c} es un vector de 5 componentes aleatorias entre 0 y 1 y $\hat{\mathbf{C}}$ es una matriz 5×5 de componentes aleatorias con valores entre 0 y 1.

- Generamos una red de puntos espaciados uniformemente: 100 valores de x entre -1 y 1, y para cada valor de x , 100 valores de y entre -1 y 1.
- Calculamos el valor del potencial en cada punto de la red. Para evitar una acumulación de puntos en algunas zonas y una escasez de puntos en otras (realizando la simetría), así como posibles zonas donde el potencial puede divergir, redefinimos los valores del potencial usando cuantiles (ver anexo A).

Siguiendo este procedimiento podemos generar un conjunto de potenciales suficientemente grande (por lo menos 200 de cada simetría, idealmente más de 1000) como para proceder a los siguientes pasos.

5.4. Obtención de imágenes usando PCA

Una vez generados y procesados los potenciales, para cada uno de ellos procedemos de la siguiente forma. Primero escogemos de manera aleatoria un 20 % de los puntos del potencial (parejas de coordenadas x, y y valor del potencial $V(x, y)$) y los sepáramos del resto. Estos puntos, que podrían verse como huecos en una superficie como las de la figura 7, serán el conjunto de validación de ese potencial, y el resto serán los puntos de entrenamiento. Con ellos entrenaremos una red neuronal (FCNN) para que prediga el valor del potencial dados x e y , y nos quedaremos con el modelo de red que mejor predice los valores $V(x, y)$ para los puntos del conjunto de validación.

Puesto que para cada potencial hay que entrenar una red neuronal, conviene hacer un estudio de los hiper-parámetros que ofrecen mejores predicciones en la reproducción del potencial de forma consistente. Para medir la calidad de una predicción usamos la raíz del error cuadrático medio, definido según

$$RMSE = \sqrt{\sum_{i=0}^N \frac{\text{Predicción}_i - \text{Valor Real}_i}{N}} , \quad (1)$$

de modo que escogeremos la selección de hiper-parámetros (parámetros cuyo valor utilizamos para controlar el proceso de aprendizaje) que obtenga un valor mínimo de $RMSE$. Para poder analizar los resultados independientemente del rango de valores del potencial, podemos normalizar el $RMSE$,

$$NRMSE = \frac{RMSE}{\text{máx. de } V(x, y) - \text{mín. de } V(x, y)} . \quad (2)$$

El comportamiento esperado es que cuanto más compleja se haga la red y más tiempo se entrene, mejores resultados obtendrá, hasta llegar a un punto donde aumentar la expresividad (complejidad) empeore los resultados. Esto es debido a una sobre-especialización de la red en el conjunto de entrenamiento (en inglés *Overfitting*), que dificulta la generalización al conjunto de validación. El estudio de los hiper-parámetros de la FCNN se resume en los siguientes puntos, y de forma más gráfica en la tabla 1.

- Número de Capas Ocultas de la Red. Posibilidades: 3, 5, 10, 20. Manteniendo constante el resto de parámetros, el menor valor de $NRMSE$ para distintos potenciales fue obtenido para 10 capas ocultas, obteniendo valores del orden de 0.1 % en el mejor de los casos. El modelo de 20 capas ocultas presentaba *overfitting*.
- Número de Neuronas por capa. Posibilidades: 50, 100, 200, 300, 400, 500. Con el resto de parámetros constantes, la cantidad de neuronas por capa oculta que minimizaba en promedio el valor de $NRMSE$ fue 200, puesto que más neuronas producen *overfitting* y menos neuronas no consiguen una buena precisión.
- Número de épocas (*epochs*) de entrenamiento, es decir, el número de veces que la red procesa el conjunto de entrenamiento. Posibilidades: 50, 100, 200, 300, 500. Al aumentar el número de *epochs*, el valor de $NRMSE$ disminuye. En este caso, debe buscarse un equilibrio de tiempo de entrenamiento frente a la mejora de precisión

que ofrece entrenar durante más *epochs*. En promedio, al aumentar de 50 a 100, el *NRMSE* disminuyó un 0.3 %. Al pasar de 100 a 200, el cambio fue de 0.15 %, y de 200 a 500 el cambio significaba únicamente una mejora del 0.03 %. Puesto que esta última mejora es prácticamente insignificante pero influye de forma muy significativa en el tiempo de entrenamiento, consideramos que el número óptimo de *epochs* en este caso es de 200.

- *Learning Rate* (LR) o Tasa de Aprendizaje Máxima. Posibilidades: 0.1, 0.01, 0.001. En la librería `fastai`, la tasa de aprendizaje crece desde un mínimo a un máximo para luego volver a bajar al mínimo (ref. [12]). Definir el máximo de LR significa definir el tamaño máximo de los pasos del algoritmo que minimiza la función de coste para entrenar la red neuronal. Cuanto más grandes sean los pasos que da, más fácilmente encontrará un mínimo pero más le costará converger al valor óptimo de los parámetros de la red. El valor del máximo de LR = 0.1 ofrece una convergencia más rápida, es decir, reduce el número de *epochs* necesarias para obtener resultados aceptables ($NRMSE < 1\%$), pero no logra una precisión tan buena como los otros valores. El valor del máximo de LR = 0.001 requería cerca de 500 *epochs* para lograr una buena precisión, aunque los resultados no fueron muy distintos al valor LR = 0.01. Este último, para el equilibrio entre número de *epochs* y precisión, resultó ser óptimo.

TABLA 1: Resumen de los resultados del estudio para optimizar los hiper-parámetros de las FCNN utilizadas para predecir el valor de un potencial bidimensional. Los colores están relacionados con el *NRMSE* de las predicciones sobre el conjunto de validación, de modo que colores más intensos representan mejores predicciones y por tanto menor valor de *NRMSE*. El tiempo de entrenamiento de las FCNN aumenta al moverse hacia las columnas de la derecha.

LR Máxima	0.1		0.01		0.001
Capas Ocultas	3	5	10	20	
Neuronas / Capa	50	100	200	300	500
Epochs	50	100	200	300	500
Tiempo de Entrenamiento					

Los resultados de hacer un balance entre mejora de precisión y tiempo de entrenamiento nos llevan finalmente a escoger una configuración de 10 capas ocultas, con 200 neuronas por capa, que será entrenada durante 200 *epochs* con una LR máxima de 0.01. Con estos hiper-parámetros, considerados a priori óptimos para la tarea³, entrenamos una red para cada potencial. Ahora queremos extraer la información codificada en la última capa oculta de esta red y obtener una imagen con ella. Para ello registramos la respuesta de esta última capa usando un *hook* [19], es decir, guardamos los valores de salida de las neuronas de dicha capa, resultantes de pasarle a la red los puntos $[x, y]$ para obtener una predicción del potencial. Partiendo de 100 puntos de x y 100 puntos de y para cada punto de x , tenemos en total 10000 puntos. Si tratamos con una capa de 200 neuronas, al procesar cada punto registraremos 200 valores, resultantes de la activación de cada una de las neuronas. Esos 200

³Aunque los hiper-parámetros elegidos predigan de la mejor forma el valor del potencial incluso en los puntos de validación, no tienen por qué ser los valores óptimos para la tarea de comprender una simetría. Podría darse el caso de que esta tarea es demasiado sencilla o demasiado complicada para la red utilizada, de modo que esté haciendo *overfitting* o *underfitting* respectivamente.

valores que obtenemos para una pareja de $[x, y]$ se pueden entender como las coordenadas de un punto en un espacio de 200 dimensiones. Haciendo este cálculo para los 10000 puntos del potencial, tenemos una distribución en ese sistema, a la que aplicamos el análisis de componentes principales (PCA).

Aunque obtener la PCA es una sola línea de código, básicamente se trata de encontrar el centro de la distribución, hacerlo coincidir con el origen y calcular las componentes principales (PC, *principal component*). La primera PC será la recta en el espacio que, pasando por el origen, ofrece el mejor ajuste a la distribución, minimizando la distancia de los puntos a dicha recta. La siguiente PC será aquella que, siendo perpendicular a la anterior y pasando por el origen, ajusta mejor los datos de la distribución. Este proceso puede repetirse hasta obtener un nuevo sistema de coordenadas ortogonales formado a base de combinaciones lineales de las originales, que en nuestro caso serían 200 dimensiones, 200 ejes de coordenadas. Los dos primeros ejes son por construcción los ejes de mayor importancia, de modo que si los usamos de ejes de coordenadas 2-D para representar los puntos de la distribución, es posible obtener cierta información de patrones según los que se agrupan dichos puntos, en forma de imagen bidimensional.

El último paso antes de generar las imágenes es añadir color. Concretamente, se trata de asignar a cada punto, obtenido de una pareja concreta de valores $[x, y]$, un color en función del valor del potencial para dicha pareja de valores. La intensidad del color estará directamente relacionada con la cantidad de puntos acumulados en cada píxel de la imagen. Algunos ejemplos pueden verse en la figura 8. Una vez completas, estas imágenes serán utilizadas posteriormente para entrenar una CNN cuyo objetivo es diferenciarlas según la simetría del potencial del que provienen.

Es interesante destacar que, en vista de los ejemplos de la figura 8, las imágenes generadas no presentan información de las simetrías que pudiera ser identificada a simple vista. De hecho, al igual que se menciona en el artículo [5], aparentemente existe la misma variabilidad entre imágenes de distintas simetrías como entre imágenes de la misma. Al igual que las figuras 8a y 8c son muy distintas, perteneciendo a simetrías diferentes, las figuras 8a y 8b también lo son, a pesar de representar el mismo tipo de simetría. Aunque no se pueda diferenciar a simple vista la simetría que representan estas imágenes, un algoritmo basado en una CNN podría ser capaz de distinguir las sutiles diferencias entre ellas. La obtención de dicho algoritmo se detalla en la sección 5.5.

5.5. Análisis de las imágenes de PCA usando una CNN

El último paso para conseguir un modelo que pueda encontrar simetrías en conjuntos de datos es entrenar el modelo en sí con las imágenes obtenidas usando PCA. En el caso en que las redes que predicen el valor del potencial hayan *entendido* la simetría de dicho potencial, las imágenes procedentes de la misma simetría presentarán rasgos similares. Entonces, una red de convolución especializada en reconocimiento de imágenes podría ser capaz de encontrar dichas similitudes, pudiendo así clasificarlas según el tipo de simetría del potencial con el que fueron generadas.

Una posibilidad es entrenar una CNN de cero, estudiando posibles arquitecturas de

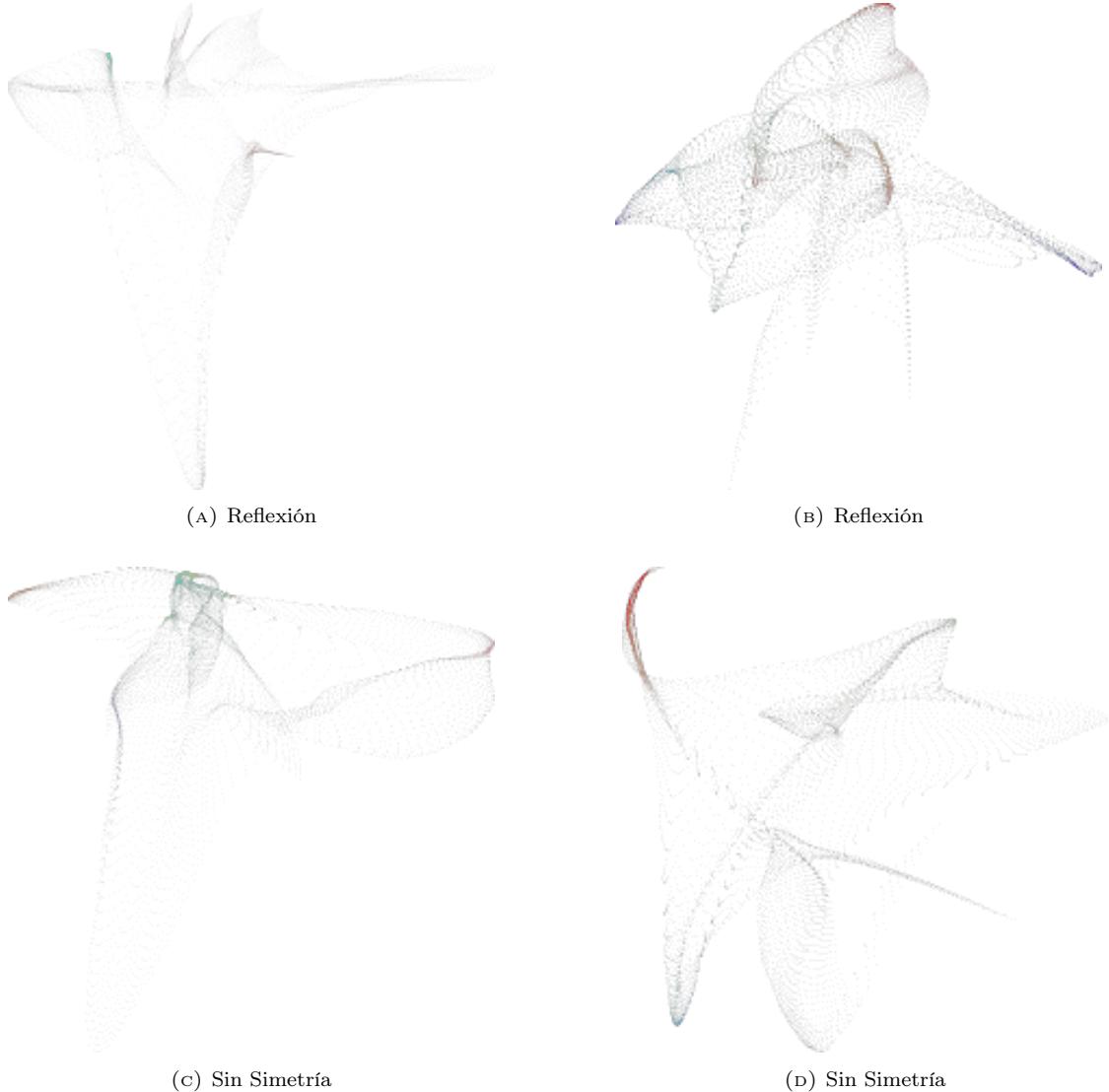


FIGURA 8: Ejemplos de imágenes obtenidas por medio de la técnica de análisis de componentes principales (PCA), a partir de la activación de la última capa oculta de una FCNN dedicada a la predicción del valor de un potencial con simetría de reflexión [figuras (A) y (B)], y un potencial sin simetría [figuras (C) y (D)].

la red y haciendo múltiples pruebas a fin de escoger los hiper-parámetros. Sin embargo, existe la opción de hacer *transfer learning*, usar un modelo ya entrenado para una tarea de clasificación de imágenes y adaptarlo a la tarea en cuestión (*fine tuning*). Decidimos usar esta segunda opción, al igual que se hizo en el artículo [5]. La red elegida para esto fue ResNet18 [13], aunque probamos también con otros tamaños, incluyendo ResNet34, ResNet50 y ResNet101.

El criterio para decidir entre las diversas redes pre-entrenadas fue la capacidad de generalizar los resultados del aprendizaje a imágenes de PCA nuevas, que no pertenecieran al conjunto de entrenamiento ni al de validación. Como queremos un modelo lo más robusto posible, probamos a generar imágenes con un número distinto de puntos a los utilizados en el entrenamiento. Esto significa que, ya que entrenamos usando 100 valores de x y 100 valores de y para cada valor de x , 100×100 , probamos con 200×200 o con 50×50 puntos). También fueron modificados los hiper-parámetros de las FCNN descritos en la sección 5.4, generando imágenes usando un número distinto de capas ocultas (5 capas), un número menor de neuronas por capa oculta (100 y 50). El modelo que mejores predicciones obtuvo fue el modelo basado en ResNet18, por lo que fue elegido como modelo final.

Al hacer *transfer learning* con el paquete `fastai`, a la red pre-entrenada se le añade una capa al final adaptada a la tarea específica de clasificación. Existen dos posibilidades en este caso. La primera se basa en entrenar un cierto número de *epochs* variando únicamente los pesos de esta última capa añadida, para luego entrenar la red completa. Este procedimiento suele ser el más adecuado en la mayoría de los casos, puesto que adapta la nueva capa al problema en cuestión. Sin embargo, en nuestro caso, resultó ser más conveniente entrenar directamente la red completa, ya que al entrenar la última capa al principio incurría en mínimo local que reducía la precisión final de la red.

6. Resultados

Los resultados principales de este trabajo son los modelos entrenados para distinguir las simetrías de los potenciales a base de imágenes, obtenidas por medio de PCA de la activación de la última capa oculta de neuronas de una red (FCNN), entrenada para predecir los valores de dicho potencial. Para distintos modelos de red, entrenados con un total de 1800 imágenes, compararemos la precisión tanto en el conjunto de validación como para imágenes generadas después de entrenar (conjunto test), usando los mismos hiper-parámetros y usando unos distintos. Para cada uno de los conjuntos de test, usamos un conjunto de 50 imágenes para cada simetría. La comparación de los mejores modelos obtenidos se presenta en la tabla 2.

Con el fin de escoger el modelo que mejor *entiende* las simetrías, nos quedaremos con el modelo que mayor capacidad de generalización tiene. En nuestro caso se trata de ResNet18, que a pesar de tener menor precisión en el conjunto de validación que ResNet50, obtiene resultados considerablemente mejores en los casos de imágenes generadas con 5 capas ocultas. Como puede verse en la tabla 2, cuando la precisión de otros modelos baja considerablemente, a valores cercanos al 75 %, ResNet18 mantiene una precisión de 98 % o superior.

TABLA 2: Comparativa de los mejores modelos de CNN entrenados usando *transfer learning* sobre ResNet18, ResNet34 y ResNet50 [13], evaluando si precisión (porcentaje de aciertos) en varios conjuntos de imágenes obtenidas usando PCA según el procedimiento de la sección 5.4. En los conjuntos de test, la notación $m \times n$ significa que las imágenes fueron generadas con redes neuronales de m capas ocultas con n neuronas en cada capa oculta.

	ResNet18	ResNet34	ResNet50
Validación	96 %	96 %	98 %
Test: 10×200	90 %	85 %	91 %
Test: 5×200	98 %	70 %	75 %
Test: 5×100	99 %	74 %	76 %

7. Conclusiones y Futuros Pasos

La pregunta de si un ordenador puede llegar a comprender un concepto abstracto como la simetría es, como poco, llamativa. Si bien es cierto que las capacidades computacionales actuales no tienen precedentes, cuando hablamos de simetría hablamos de un concepto humano, una forma que usamos para describir la complejidad de la Naturaleza basada en principios con la mayor sencillez posible. La búsqueda de un programa que sea capaz de entender estos principios es por tanto la búsqueda de una herramienta de gran valor, que podría ayudarnos a desvelar algunos misterios del mundo que nos rodea.

A lo largo de este trabajo hemos presentado una breve explicación de los fundamentos de las herramientas de aprendizaje de un ordenador: *Machine Learning* y *Deep Learning*, con ejemplos para ilustrar las capacidades de estas técnicas como herramientas científicas. Concretamente, hemos contemplado las bases de las redes neuronales y su capacidad de abstracción. Dicha capacidad es el corazón de nuestro estudio, basado en el supuesto de que una red neuronal puede ser capaz de comprender el concepto de simetría. Más concretamente, aplicando un procedimiento similar al del artículo [5], hemos estudiado la capacidad de una red neuronal para comprender la simetría de un potencial continuo.

Con el fin de generalizar el procedimiento del artículo [5] al caso continuo, hemos generado en total 1800 potenciales bidimensionales con cada una de las simetrías utilizadas en dicho artículo: Translación Discreta, Translación Continua, Rotación Continua, Reflexión y Ausencia de Simetría. Para cada potencial, hemos entrenado una red neuronal que tiene como objetivo predecir su valor. Hemos extraído la información de la activación de la última capa oculta de la red, la capa con mayor nivel de abstracción, por medio del análisis de componentes principales (PCA), para generar una imagen por cada potencial. Con esas imágenes hemos entrenado varias redes de convolución dedicadas a clasificar las imágenes según la simetría de la que provienen.

De todas las CNN entrenadas, la más sencilla (ResNet18) fue la red con mayor capacidad de generalización de lo aprendido. Llegamos a esta conclusión tras probar varios modelos de diferente complejidad sobre imágenes nuevas generadas con varias combinaciones de hiper-parámetros. Donde las redes más complejas perdían precisión considerablemente, el modelo obtenido usando ResNet18 mantenía una buena precisión. Sobre el conjunto de validación, el mejor modelo conseguido logró una precisión del 96 %. El significado de dicho rendimiento es que, en efecto, en la última capa de las redes neuronales que predicen el

valor del potencial, existe información codificada acerca de la simetría de ese potencial que han aprendido. Esto es así porque la CNN ha sido capaz de encontrar patrones comunes en imágenes provenientes del mismo tipo de potencial. La existencia de estos patrones es lo que prueba que las FCNN tratan de forma distinta los datos dependiendo de la simetría de los mismos, es decir, reaccionan de una forma concreta en función de la simetría del problema.

Este resultado, aunque esperanzador, no implica directamente que nuestra red neuronal sea capaz de entender el concepto abstracto de simetría. Este es únicamente un pequeño paso adelante en el camino hacia un código que pueda comprenderlo. Como primer paso, viene acompañado de una serie de posibles caminos para seguir avanzando. Por un lado, podría generalizarse nuestro procedimiento a un caso de n dimensiones, utilizando potenciales $V(x_1, x_2, \dots, x_n)$ con simetrías definidas en ese espacio n -dimensional. En ese caso surgiría el problema de definir las simetrías independientes en dicho espacio, es decir, simetrías que no puedan simplificarse proyectando en dos dimensiones.

Otro camino podría consistir en centrar nuestra atención en las simetrías bidimensionales y estudiar, por ejemplo, si un conjunto de datos tiene alguna de estas simetrías en sus dos componentes principales. Un caso interesante sería añadir una variable redundante z a los datos x, y y $V(x, y)$. En caso de ser capaces de distinguir esta variable redundante de las variables que definen el potencial, podríamos buscar directamente la simetría del potencial $V(x, y)$ en función de x e y , como hemos hecho en este trabajo. Sería interesante también hacer una búsqueda de las componentes principales que influyen en el valor del potencial, de modo que un potencial podría tener una simetría de cierto tipo si consideramos una combinación lineal de las variables presentes. Un ejemplo podría ser un potencial con simetría de rotación de la forma $V(x, y + 2z)$, que no presenta simetría de rotación estudiándolo frente a las parejas $[x, y]$, $[x, z]$ o $[y, z]$ por separado, pero sí la presenta cuando tenemos en cuenta la combinación $[x, y + 2z]$.

Este último procedimiento podría ser de enorme utilidad para la búsqueda de simetrías en bases de datos. Encontrar simetrías por ejemplo en patrones de tráfico en una ciudad podría ayudar a predecirlo de forma más robusta. Ya no tendríamos únicamente los datos de una ciudad, sino que podríamos tener los de miles de ciudades virtuales nuevas pero similares a la original, ya que seguirían la misma simetría. Un ejemplo algo más físico es el mencionado en el artículo [5]: dados los datos de un detector de partículas como Gargamelle, ¿podría este programa encontrar las simetrías que conducen al Modelo Estándar? ¿Podría este programa, con los datos de los colisionadores modernos, reconocer alguna simetría más allá de las que hemos identificado nosotros?

En conclusión, la comprensión de las simetrías por un ordenador es un tema con gran potencial para aportarnos nuevos conocimientos, y no sólo a nivel informático. Una herramienta así podría convertirse en el pilar de una revolución ontológica, en tanto que podría ser capaz de desvelar misterios de la Naturaleza que hasta el momento han permanecido ocultos, escondidos tras una simetría que todavía no hemos sido capaces de desentrañar.

8. Agradecimientos

Agradezco la inmensurable ayuda recibida por parte de mis tutores Gabriela Barenboim, Johannes Hirn y Verónica Sanz. Su apoyo tanto a nivel técnico como conceptual y la enorme motivación transmitida han sido, sin lugar a duda, la pieza más importante de este trabajo. Trabajar con ellos ha sido una experiencia de valor incalculable.

Agradezco a Karina su apoyo en los momentos más difíciles del desarrollo de este trabajo.

Anexos

A. Cuantiles

Para generar los potenciales en el procedimiento de la sección 5.3 hemos procesado el valor del potencial haciendo uso de cuantiles. Dividir una cierta distribución de valores en cuantiles (percentiles, cuartiles...) consiste en encontrar los valores de dicha distribución que dejan por debajo una cierta fracción de los datos. Así, la mediana es un ejemplo de cuantil, que deja por debajo el 50 % de los valores. Si quisiéramos encontrar los percentiles de una distribución, tendríamos que buscar aquel valor que justo es mayor que el 1 % de los datos, el valor que deja por debajo el 2 % de los datos, el que deja por debajo el 3 %, etc. Si en vez de tener 100 intervalos con el mismo número de puntos (1 % en cada intervalo) tenemos una cantidad arbitraria de intervalos con el mismo número de valores, estamos tratando con cuantiles.

La utilidad de este método es que preserva la forma de la distribución acentuando las diferencias locales. Veamos esto con dos ejemplos. Digamos que queremos estudiar la altura de las personas en una sala para un experimento, y resulta que se trata de una clase de infantil con 99 niños, de unos 3 años que miden alrededor de 1 metro, de los que se ocupa un profesor de 2 metros de altura. Tenemos 100 personas en total. Si hacemos un histograma de las alturas con bines uniformes, veríamos que hay unos pocos bines cerca de 1 metro y un bin en 2 metros. Viendo esta gráfica (Figura 9 (b)) está claro que hemos perdido mucha información acerca de las diferencias de altura entre los niños, debido a que hemos dividido las edades en intervalos regulares. No podemos saber el tipo de distribución que sigue ni podemos inferir nada de ella, solo sabemos que los niños miden cerca de 1 metro. Por ejemplo, los niños que miden 90 cm y los que miden 99 cm están agrupados en la misma categoría, y vemos que hay muchos, pero no podemos decir nada de las diferencias individuales en ese grupo.

Hagamos ahora 10 cuantiles, de modo que el primer grupo contendrá a los 10 niños de menor estatura, el segundo, a los 10 niños siguientes, etc. Si hacemos un histograma de estos grupos, obtenemos una forma uniforme (en el eje y) (Figura 9 (a)), con unos bines que no están distribuidos de forma uniforme (en el eje x). Ahora podemos ver que en un intervalo de 4 centímetros, el intervalo 101 ± 2 cm, hay 20 niños, un detalle que en el histograma de la figura 9 (b) era imposible ver. Este hecho se vería más claro todavía en un caso fantástico

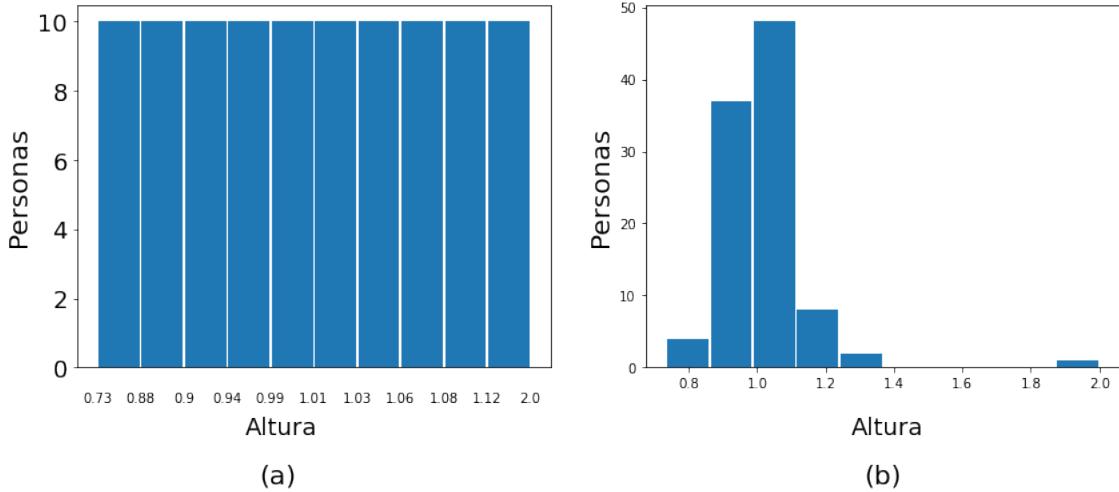


FIGURA 9: Histograma de alturas de una clase de 99 niños con alturas alrededor de 1 metro junto a un profesor de 2 metros. (a) Histograma utilizando cuantiles, donde en cada bin hay un 10 % de las personas de la clase, uniforme en el eje y . (b) Histograma con intervalos uniformes de altura (eje x).

en el que el profesor fuera un gigante de 20 metros⁴. Los detalles de las diferencias de altura en los niños se perderían por completo en el histograma, mientras que con los cuantiles estas diferencias seguirían estando presentes, ya que obtendríamos los mismos bines que en la figura 9 (a) cambiando únicamente el límite del último de 2 a 20 (ver Figura 10).

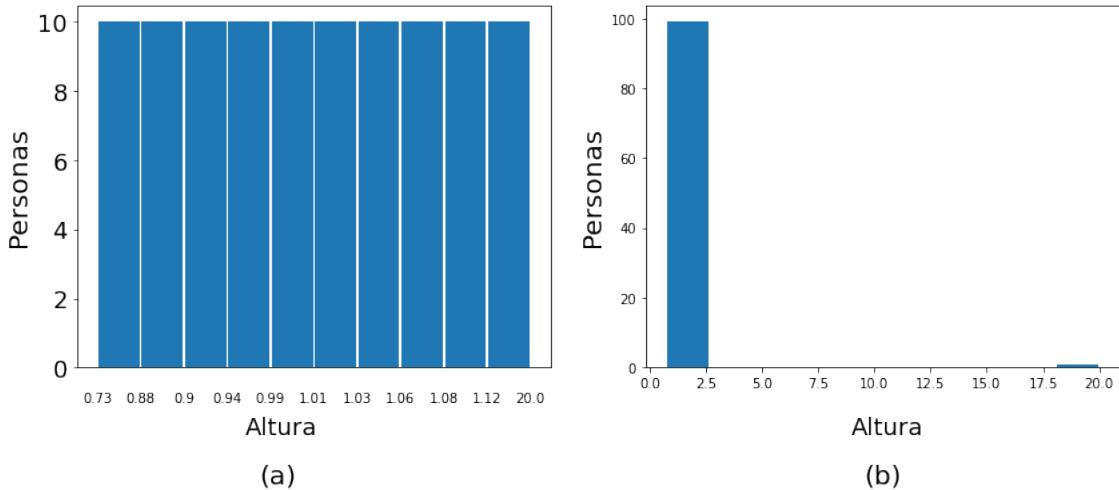


FIGURA 10: Histograma de alturas de una clase de 99 niños con alturas alrededor de 1 metro junto a un profesor que en este caso es un gigante de 20 metros. (a) Histograma utilizando cuantiles, donde en cada bin hay un 10 % de las personas de la clase, uniforme en el eje y . (b) Histograma con intervalos uniformes de altura (eje x). Nótese que toda la información de las diferencias de altura entre los niños se ha perdido en la Figura (b), mientras que en la (a), usando cuantiles, permanece inalterada.

El segundo ejemplo está enfocado al uso que se le da a los cuantiles en la sección 5.3. Supongamos que tenemos una función, un potencial como $V(x) = x^4 + 3 \cos(x) - 2$. Por simplicidad, vamos a escoger únicamente 10 puntos de este potencial. Viendo la gráfica entre $x = -10$ y $x = +10$ (Figura 11 (a)), la mayoría de los puntos escogidos (puntos rojos) aparentan estar en una recta. Sin embargo, si nos acercamos al rango $[-2.5, 2.5]$, podemos

⁴Aunque esta situación sea más bien absurda, es posible imaginar el caso económico en el que se compara por ejemplo la riqueza de un multimillonario con la de un grupo de personas promedio. La diferencia en tal caso se acentuaría mucho más. Sin embargo, como opinión personal, resulta considerablemente más animado imaginar una clase de infantil cuyo profesor es un amable gigante de 20 metros.

ver que en realidad la distribución, lejos de ser una recta presenta varios extremos, máximos y mínimos.

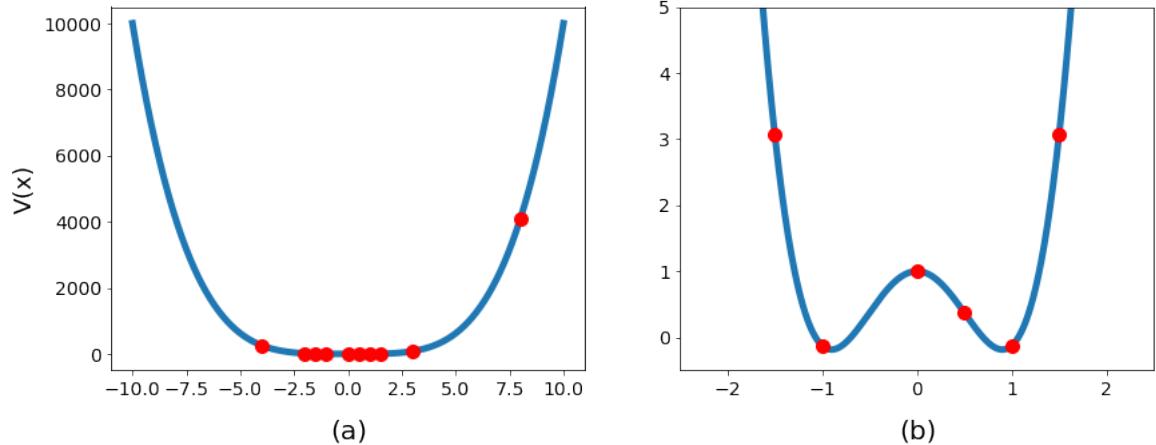


FIGURA 11: Función $V(x) = x^4 + 3 \cos(x) - 2$ junto a 10 puntos de la misma escogidos de forma arbitraria. (a) Función y conjunto de 10 puntos en el rango $[-10, 10]$. (b) Ampliación de la función al rango $[-2.5, 2.5]$ que revela una estructura interna oculta en la figura (a).

Al representar la figura completa, hemos perdido mucha información acerca de las diferencias locales entre los puntos escogidos. Hemos perdido detalle acerca de la forma del potencial. La función con la que estamos tratando es simétrica bajo reflexión en el eje y, hecho que se aprecia en ambas escalas, grande y pequeña, pero es en esta última en la que tenemos una mayor cantidad de detalles sobre esta simetría. En nuestro caso, nos interesa estudiar en profundidad la simetría de la función, de modo que nos importa principalmente la forma de la misma. Por ello necesitamos una forma de darle más importancia a los detalles de la pequeña escala sin eliminar los de la grande. Esto podemos conseguirlo haciendo uso de los cuantiles, transformando la función de modo que al punto $V(x)$ con el valor más alto le asignamos el valor 1, al siguiente le asignamos 0.9, al siguiente 0.8, etc. De esta manera preservamos la forma de la función dando importancia a las variaciones locales. La diferencia en la gráfica obtenida con los 10 puntos elegidos usando la transformación de cuantiles y sin utilizarla puede verse en la Figura 12.

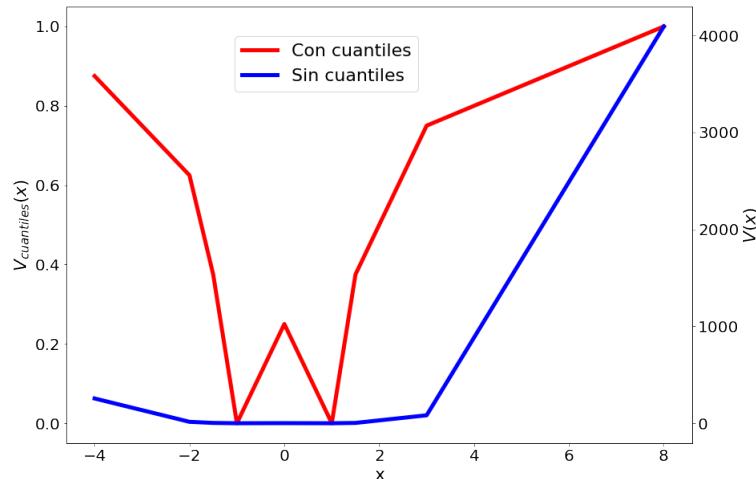


FIGURA 12: 10 puntos de la función $V(x) = x^4 + 3 \cos(x) - 2$ escogidos anteriormente, tanto los puntos originales como los redefinidos utilizando cuantiles (asignando el valor de 1 al punto más alto, 0.9 al siguiente, etc.).

Al hacer uso de cuantiles resaltamos detalles sobre la simetría del potencial, cosa que permite una identificación más fácil de la misma. Por tanto, una red neuronal a la que se le presentan unos potenciales simétricos modificados de esta manera tendrá más posibilidades de aprender relaciones más complejas entre los puntos de la imagen, acercándonos así al objetivo de que esta red comprenda la existencia de una simetría. En la Figura 13 se puede comprobar que efectivamente existe esta diferencia en los potenciales utilizados para este proyecto: la simetría es la misma, pero usando cuantiles los detalles de ésta son más marcados.

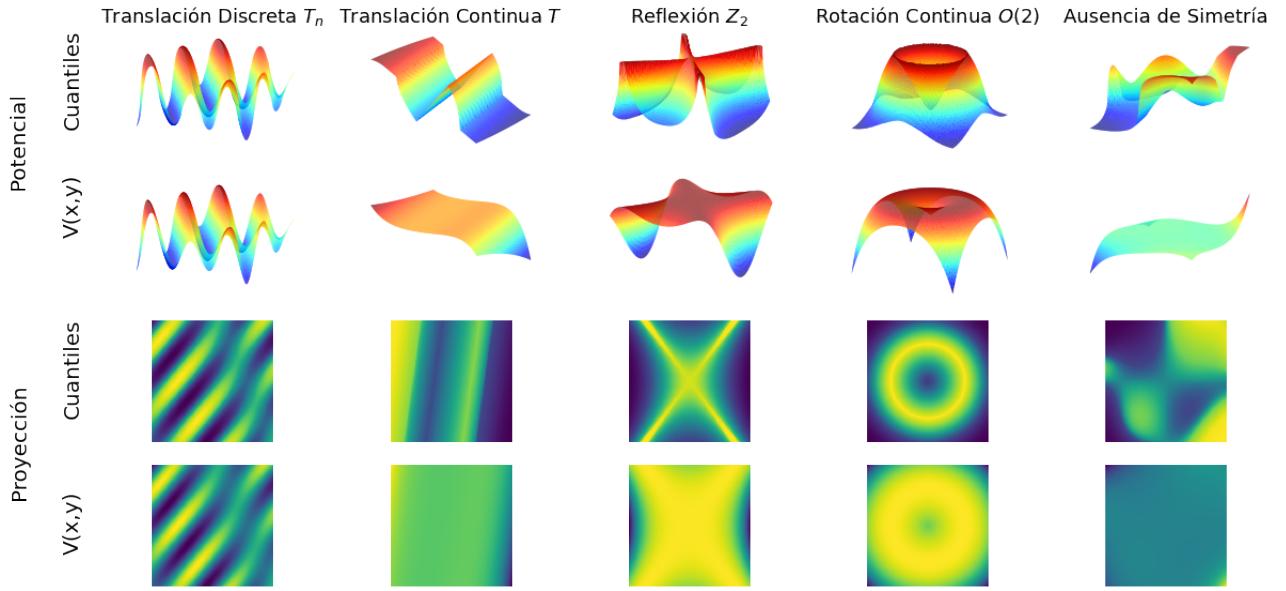


FIGURA 13: Ejemplos de potenciales de cada una de las clases de simetría, utilizados para el entrenamiento de la FCNN, junto a sus proyecciones, comparando la forma que presentan al utilizar la transformación de cuantiles y sin ella. Al usar cuantiles, los detalles de las simetrías en ciertas zonas aparecen más destacados.

B. *Troubleshooting*

A lo largo del desarrollo de este trabajo nos encontramos frente a una serie de problemas técnicos (problemas de librerías, del entorno de programación, etc.), algunos de ellos aparentemente sin documentar. Por ello, resulta conveniente hacer un registro de los mismos así como de sus soluciones.

1.
 - **Problema:** Para usar `fastai` V2 es necesario disponer de una GPU de alta gama.
 - **Solución:** Existen servicios gratuitos (aunque limitados) que ponen a disposición del usuario una GPU compatible con `fastai` V2, como es el caso de Google Colab.
2.
 - **Problema:** Al importar en Google Colab la librería `fastai`, no importa la versión más actualizada.
 - **Solución:** Google Colab importa por defecto la versión 1.0.61. La solución es instalar la versión deseada, utilizando el comando `!pip install -q fastai==2.3.0` por ejemplo, para la versión 2.3.0. Aparecerá un error durante la instalación,

“*ERROR: torchtext 0.10.0 has requirement torch==1.9.0, but you'll have torch 1.7.1 which is incompatible.*”, que no es importante y puede ignorarse a menos que se quiera hacer uso de `torchtext`.

3.
 - **Problema:** En Google Colab, no funcionan las redes de convolución para clasificación de imágenes usando `fastai`. La red no es capaz de aprender nada.
 - **Solución:** Algunas versiones de `fastai` presentan incompatibilidades con Google Colab. Las versiones 2.0.x, 2.1.x y 2.2.x presentaron este problema durante la realización de este trabajo, que fue solucionado en la versión 2.3.0.
4.
 - **Problema:** La sesión de Google Colab se cierra pasado un tiempo incluso cuando hay código en ejecución, y cada vez que la reanuda funciona más despacio.
5.
 - **Problema:** No es posible entrenar una CNN usando *transfer learning* en `fastai` con la función `learn.fine_tune()` sin entrenar la nueva capa primero, es decir, la función no está definida para `freeze_epochs = 0`.
 - **Solución:** Para ese caso, debe *descongelarse* la red (`learn.unfreeze()`) para poder variar todos los parámetros y se debe utilizar la función `learn.fit_one_cycle()`.
6.
 - **Problema:** Al eliminar un *learner* de `fastai` y crear uno nuevo, queda información residual en la memoria que inicializa los pesos del nuevo *learner*.
 - **Solución:** Puesto que eliminar la variable y limpiar el caché de la GPU no elimina la información residual, la solución es reiniciar el entorno de ejecución.
7.
 - **Problema:** Al importar un modelo de clasificación de imágenes guardado y utilizarlo, clasifica todas las imágenes en la misma categoría. El modelo parece haber perdido toda la información que tenía al guardarlo.
 - **Solución:** Hay veces en las que al importar un *learner*, éste parece ser un modelo en blanco. Sin embargo, si se entrena aunque sea una sola *epoch*, el modelo vuelve a un estado parecido al que tenía cuando fue guardado. Si al importar un modelo parece que no funciona bien, la solución es entrenar el *learner* una pequeña cantidad de *epochs* para que vuelva a funcionar.

Referencias

- [1] JA Aguilar-Saavedra, Jack Collins, and Rashmish K Mishra. «A generic anti-QCD jet tagger». In: *Journal of High Energy Physics* 2017.11 (2017), pp. 1–30.
- [2] JA Aguilar-Saavedra and B Zaldívar. «Jet tagging made easy». In: *The European Physical Journal C* 80.6 (2020), pp. 1–16.
- [3] Kazunori Akiyama et al. «First M87 event horizon telescope results. IV. Imaging the central supermassive black hole». In: *The Astrophysical Journal Letters* 875.1 (2019), p. L4.
- [4] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. «Searching for exotic particles in high-energy physics with deep learning». In: *Nature communications* 5.1 (2014), pp. 1–9.
- [5] Gabriela Barenboim, Johannes Hirn, and Veronica Sanz. «Symmetry meets AI». In: *arXiv preprint arXiv:2103.06115* (2021).
- [6] Rahul Biswas et al. «Application of machine learning algorithms to the study of noise artifacts in gravitational-wave data». In: *Physical Review D* 88.6 (2013), p. 062003.
- [7] Serguei Chatrchyan et al. «Search for physics beyond the standard model in events with a Z boson, jets, and missing transverse energy in pp collisions at $s=7$ TeV». In: *Physics Letters B* 716.2 (2012), pp. 260–284.
- [8] Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. «Image processing with neural networks—a review». In: *Pattern recognition* 35.10 (2002), pp. 2279–2301.
- [9] Kenneth A Farley et al. «Mars 2020 mission overview». In: *Space Science Reviews* 216.8 (2020), pp. 1–41.
- [10] Kunihiko Fukushima. «Neocognitron: A hierarchical neural network capable of visual pattern recognition». In: *Neural networks* 1.2 (1988), pp. 119–130.
- [11] Dan Guest, Kyle Cranmer, and Daniel Whiteson. «Deep learning and its application to LHC physics». In: *Annual Review of Nuclear and Particle Science* 68 (2018), pp. 161–181.
- [12] Sylvain Gugger. *The 1cycle policy*. Apr. 2018. URL: <https://sgugger.github.io/the-1cycle-policy.html>.
- [13] Kaiming He et al. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [14] Shahzad Khan. «Ethem Alpaydin. Introduction to Machine Learning (Adaptive Computation and Machine Learning Series). The MIT Press, 2004». In: *Natural Language Engineering* 14.1 (2008), p. 133.
- [15] Anthony C Little, Coren L Apicella, and Frank W Marlowe. «Preferences for symmetry in human faces in two cultures: data from the UK and the Hadza, an isolated group of hunter-gatherers». In: *Proceedings of the Royal Society B: Biological Sciences* 274.1629 (2007), pp. 3113–3117. DOI: [10.1098/rspb.2007.0895](https://doi.org/10.1098/rspb.2007.0895).
- [16] Pankaj Mehta et al. «A high-bias, low-variance introduction to machine learning for physicists». In: *Physics reports* 810 (2019), pp. 1–124.
- [17] Grégoire Montavon, G Orr, and Klaus-Robert Müller. «Neural networks-tricks of the trade second edition». In: *Springer, DOI 10* (2012), pp. 978–3.

- [18] A. J. O'Toole et al. «Face Recognition Algorithms Surpass Humans Matching Faces Over Changes in Illumination». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.9 (2007), pp. 1642–1646. DOI: [10.1109/TPAMI.2007.1107](https://doi.org/10.1109/TPAMI.2007.1107).
- [19] Frank Odom. *How to Use PyTorch Hooks*. Dec. 2020. URL: <https://medium.com/the-dl/how-to-use-pytorch-hooks-5041d777f904>.
- [20] Seymour Papert. *Sumer Vision Project*. <https://dspace.mit.edu/bitstream/handle/1721.1/6125/AIM-100.pdf>. Artificial Intelligence Group, July 1966.
- [21] Mario Pasquato. «Detecting intermediate mass black holes in globular clusters with machine learning». In: *arXiv preprint arXiv:1606.08548* (2016).
- [22] Gillian Rhodes et al. «Attractiveness of Facial Averageness and Symmetry in Non-Western Cultures: In Search of Biologically Based Standards of Beauty». In: *Perception* 30.5 (2001), pp. 611–625. DOI: [10.1080/p3123](https://doi.org/10.1080/p3123).
- [23] Jyh-Jong Tsay, Chao-Cheng Chen, and Jyh-Jung Hsu. «Evolving Intelligent Mario Controller by Reinforcement Learning». In: *2011 International Conference on Technologies and Applications of Artificial Intelligence*. 2011, pp. 266–272. DOI: [10.1109/TAAI.2011.54](https://doi.org/10.1109/TAAI.2011.54).
- [24] Ph Vannerem et al. «Classifying lep data with support vector algorithms». In: *arXiv preprint hep-ex/9905027* (1999).
- [25] Martinus Veltman. *Diagrammatica: the path to Feynman diagrams*. 4. Cambridge University Press, 1994.
- [26] Ver página web de la herramienta en <https://www.fast.ai/>.
- [27] Steven Weinberg. *Dreams of a final theory*. Vintage, 1994.