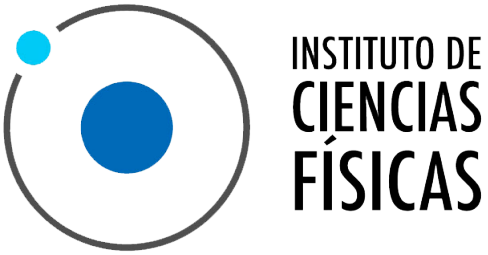


Notas servicio social

Diego Antonio Villalba González
Universidad Nacional Autónoma de México
Facultad de Ciencias
enero 2026

Datos del Alumno:

- Nombre: Diego Antonio Villalba González
- Carrera: Física
- Número de cuenta: 31932070-1

Datos del Servicio Social

- **Programa:** Cosmología Observacional a través del Cómputo Científico
- **Clave:** 2024-12/164-1652
- **Institución:** Universidad Nacional Autónoma de México
- **Dependencia:** Instituto de Ciencias Físicas
- **Fecha de inicio:** 2 de diciembre de 2024
- **Fecha de término:** 1 de agosto de 2025

Asesor Responsable: Dr. José Alberto Vázquez González

Ciudad de México, agosto 2025

Declaración

En el presente documento se exponen los resultados del proyecto realizado como parte de mi servicio social en el Instituto de Ciencias Físicas de la Universidad Nacional Autónoma de México. Declaro que este trabajo es de mi completa autoría y que toda la información proveniente de fuentes externas ha sido debidamente citada de forma permanente. En caso de que se requiera utilizar, citar o acceder a cualquier resultado, figura o contenido incluido en este trabajo, deberá hacerse previa solicitud y con estricto apego a los derechos de autor correspondientes.

Ciudad de México, enero 2026

Índice

Resumen

ste informe presenta las actividades y resultados desarrollados durante el servicio social realizado en el Instituto de Ciencias Físicas de la Universidad Nacional Autónoma de México, dentro del programa Cosmología Observacional a través del Cómputo Científico. El trabajo se centró en el estudio y aplicación de herramientas modernas de análisis de datos, aprendizaje automático y procesamiento multiescala, con énfasis en su relevancia para problemas contemporáneos de la física y la cosmología.

A lo largo del proyecto se abordaron conceptos fundamentales de modelado probabilístico, alta dimensionalidad y renormalización, estableciendo conexiones conceptuales entre la física estadística y las arquitecturas de aprendizaje profundo. Se estudió de manera sistemática el funcionamiento de distintas redes neuronales —incluyendo redes convolucionales, residuales y mecanismos de atención— así como su implementación práctica en problemas de análisis y reconstrucción de datos. En particular, se exploró el problema de la súper resolución de imágenes, revisando desde métodos clásicos de interpolación hasta enfoques basados en aprendizaje profundo, destacando sus limitaciones y ventajas en contextos no naturales como los datos cosmológicos.

Asimismo, se realizó un análisis detallado de técnicas de descomposición multiescala mediante wavelets, incluyendo la Transformada Wavelet Discreta y la Transformada Wavelet Scattering, resaltando sus propiedades de estabilidad, interpretabilidad y capacidad para capturar correlaciones de orden superior en campos no gaussianos. Finalmente, se incluyó el uso de métodos de reducción de dimensionalidad, como el Análisis de Componentes Principales, para la interpretación y compresión eficiente de datos complejos.

En conjunto, este trabajo contribuye a la formación interdisciplinaria en física computacional y aprendizaje automático, proporcionando una base teórica y práctica sólida para el análisis de grandes volúmenes de datos y el modelado estadístico de sistemas físicos complejos, con especial relevancia en aplicaciones cosmológicas.

1

Aprendiendo física desde los datos

A lo largo de la historia de la física como ciencia natural, la capacidad de comprender y procesar la información presente en la naturaleza ha sido un pilar fundamental para el avance científico. Desde las primeras observaciones de Aristóteles sobre la caída de los cuerpos hasta los desarrollos más recientes en la cosmología contemporánea, los científicos han creado una gran variedad de herramientas para analizar datos y así refinar o validar nuestros modelos teóricos.

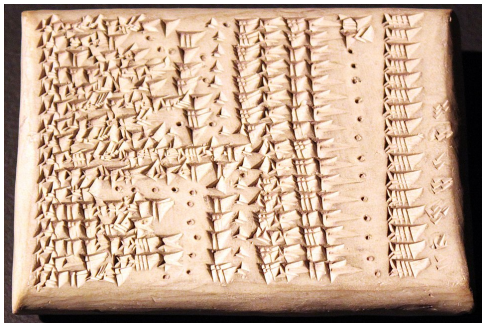


Figura 1.1: Tabla Babilonia con lista de estrellas incluyendo información de distancia en varas [1]

Es en este contexto que entre las distintas ramas de la física, la astronomía ha destacado desde sus inicios por su naturaleza intrínsecamente observacional dadas las grandes dificultades para realizar experimentos controlados. Dependiendo intensamente del procesamiento de grandes volúmenes de datos obtenidos a través de observaciones sistemáticas de la bóveda celeste, con el fin de construir modelos teóricos cada vez más precisos y útiles.

Se tiene registro de que los astrónomos babilonios ya recopilaban vastas tablillas con información detallada sobre los movimientos de los astros. A pesar de sus impresionantes logros predictivos (como la anticipación de eclipses lunares) sus sistemas carecían de un marco físico que explicara estos fenómenos. No fue sino hasta el trabajo de Johannes Kepler, a comienzos del siglo XVII, que el análisis sistemático de datos observacionales alcanzó un nuevo nivel. Al utilizar las detalladas mediciones del astrónomo Tycho Brahe (el conjunto más extenso de datos astronómicos de la época), Kepler logró deducir las leyes que rigen las órbitas planetarias, sentando las bases para la formulación de la teoría de la gravitación universal por parte de Isaac Newton. Este episodio representa uno de los primeros y más notables ejemplos del uso del análisis de datos como herramienta para generar modelos teóricos matemáticamente fundamentados, capaces de describir con precisión los fenómenos naturales.

Hoy, el surgimiento de nuevas técnicas para el procesamiento de grandes volúmenes de datos es más crucial que nunca. Con telescopios y proyectos de mapeo celeste (surveys) produciendo, en una sola noche, una cantidad de información equivalente a tres años de transmisión continua de video, se requieren herramientas computacionales innovadoras para transformar estos gigantescos flujos de información en nuevo conocimiento científico. El futuro de la física y la astronomía está intrínsecamente ligado a nuestra capacidad de dominar y extraer significado de esta avalancha de información.

1.1. Modelos probabilísticos

Un desafío persistente en la física es la estimación de la distribución de probabilidad $p(x)$ para datos de alta dimensionalidad, especialmente cuando se dispone de un número limitado de observaciones y aún más cuando los datos presentan relaciones no gaussianas. Si bien los sistemas físicos suelen describirse mediante ecuaciones diferenciales, a menudo resulta impráctico caracterizar completamente sus soluciones de manera analítica, haciendo que contar con modelos probabilísticos realistas para este tipo de campos permita aplicaciones significativas como caracterizar y comparar con precisión procesos no lineales, o separar distintas fuentes de información y resolver problemas inversos.

Para formalizar esta idea supongamos que existe un campo o estado del sistema $\varphi \in \mathbb{R}^d$, del cual sabemos que en el equilibrio la probabilidad de encontrar el sistema en una configuración φ particular se describe mediante la **distribución de Boltzmann-Gibbs**.

$$p(\varphi) = \frac{1}{Z} e^{-U(\varphi)}$$

Donde:

- $U(\phi)$ es la **energía de Gibbs** asociada a una configuración específica φ del sistema.
- Z es la **función de partición** que encapsula toda la información termodinámica del sistema:

$$Z = \int e^{-U(\varphi)} d\varphi$$

Al modelar y estudiar $U(\phi)$ podemos comprender la física detrás del proceso y proponer modelos teóricos que nos permitan estimar sus parámetros a partir de datos, además de tener la capacidad de generar muestras de configuraciones de ϕ que sigan la distribución $p(\phi)$ que nos permtan explorar el espacio de estados del sistema.

En nuestro caso buscamos es estimar la distribución de probabilidad $p(\varphi)$ a partir de un conjunto de muestras observadas $\{\phi^{(i)}\}_{i=1}^{N}$. Sin embargo abordar esta tarea tenemos dos tipos de problemas interrelacionados:

1. **Definición del Modelo:**
   Necesitamos elegir un modelo para $U(\phi)$ que sea lo suficientemente flexible para capturar la complejidad de la distribución real ya que la complejidad de muchos sistemas radica en que sus componentes interactúan de manera diferente a nivel microscópico, mesoscópico y macroscópico. Una buena aproximación implica desarrollar modelos que capturen estas interacciones multi-escala de manera efectiva, regularmente se inicia con una suposición inicial (ansatz) que parte de algún modelo teórico, sin embargo en la actualidad las **redes** neuronales surgen como candidatas por su capacidad de aprender representaciones complejas y no lineales, en este ultimo caso la definición del modelo viene de la estructura con la que dotamos a la red neuronal. El objetivo es que la distribución modelada, $P_\theta(\phi) = Z_\theta^{-1} e^{-U_\theta(\phi)}$, donde θ son los parámetros del modelo, se aproxime lo más posible a la distribución de equilibrio "verdadera"del sistema.

2. **Optimización y Muestreo:**
   Una vez definido el modelo, el siguiente paso es estimar sus parámetros θ. Esto se logra minimizando una función de "pérdida"que cuantifica la diferencia entre la distribución modelada y los datos observados. Una vez que el modelo $P_\theta(\phi)$ ha sido encontrado y sus parámetros estimados, el siguiente paso es generar nuevas muestras de datos a partir de esta distribución aprendida. Esto es crucial para la simulación, la predicción y para explorar el espacio de configuraciones del sistema.

Ejemplo: Modelado probabilístico con el modelo de Ising

Una aplicación concreta de los conceptos presentados es el modelo de Ising bidimensional, un sistema físico clásico en mecánica estadística que captura interacciones de espines sobre una red. El sistema está compuesto por $N = L \times L$ espines $\varphi_i \in \{-1, +1\}$ dispuestos sobre una retícula cuadrada. La energía asociada a una configuración $\varphi = \{\varphi_i\}$ está dada por:

$$U(\varphi) = -J \sum_{\langle i,j \rangle} \varphi_i \varphi_j - h \sum_i \varphi_i,$$

donde J es la constante de acoplamiento (positiva en el caso ferromagnético), h es el campo magnético externo, y la suma $\langle i, j \rangle$ se realiza sobre pares de sitios vecinos. La distribución de probabilidad correspondiente está dada por la distribución de Boltzmann:

$$p(\varphi) = \frac{1}{Z} e^{-\beta U(\varphi)}.$$

Nuestro objetivo es aproximar esta distribución a partir de un conjunto de configuraciones $\{\varphi^{(i)}\}_{i=1}^{N}$ generadas mediante simulaciones Monte Carlo. Para ello, introducimos un modelo paramétrico $U_\theta(\varphi)$, por ejemplo, una red neuronal convolucional, que permita estimar la energía de una configuración sin conocer explícitamente los parámetros físicos J y h.

Definición del modelo

Podemos representar $U_\theta(\varphi)$ mediante una red neuronal convolucional que capture interacciones locales entre espines. Una arquitectura simple podría ser:

- Una capa convolucional con múltiples filtros para detectar patrones locales.
- Una función de activación no lineal (ReLU).
- Una segunda capa convolucional que refina la representación.
- Una capa lineal final que colapsa la información en un único valor escalar $U_\theta(\varphi)$.

El objetivo del aprendizaje es que la distribución modelada

$$p_\theta(\varphi) = \frac{1}{Z_\theta} e^{-U_\theta(\varphi)}$$

se aproxime a la distribución verdadera del sistema.

Optimización del modelo

La optimización de los parámetros θ puede realizarse mediante *contrastive divergence*, aproximando el gradiente de la función de pérdida:

$$\mathcal{L}(\theta) = \mathbb{E}_{\varphi \sim p_{\text{data}}} \left[U_\theta(\varphi) \right] - \mathbb{E}_{\varphi \sim p_\theta} \left[U_\theta(\varphi) \right].$$

Dado que el segundo término es intratable, se estima mediante muestreo usando métodos como *Langevin Dynamics*:

$$\varphi_{t+1} = \varphi_t - \frac{\eta}{2} \nabla_\varphi U_\theta(\varphi_t) + \sqrt{\eta} \cdot \epsilon_t,$$

donde ϵ_t es una perturbación gaussiana y η es el tamaño de paso.

Muestreo y validación

Una vez entrenado el modelo, se pueden generar nuevas configuraciones $\varphi \sim p_\theta$ y comparar sus propiedades estadísticas con las configuraciones reales del modelo de Ising:

- Distribución de la magnetización total $M = \sum_i \varphi_i$.
- Correlaciones espaciales $\langle \varphi_i \varphi_j \rangle$.
- Histograma de energías $U(\varphi)$.

Este enfoque permite aprender una representación efectiva del sistema físico directamente desde datos simulados, lo cual es fundamental en escenarios donde los parámetros microscópicos son desconocidos o los modelos analíticos son intratables.

1.2. El reto de la alta Dimensionalidad

Un obstáculo significativo en este tipo de problemas es la llamada **maldición de la dimensionalidad**. Supongamos que queremos aprender una función $y = f(\phi)$ (ya sea para clasificación, regresión u otras tareas) a partir de un conjunto de ejemplos observados $\{\phi_i, y_i = f(\phi_i)\}$. Nuestra intuición nos diría que con suficientes ejemplos, podríamos interpolar para aproximar la función. Sin embargo este razonamiento presenta un problema fundamental; típicamente, en espacios de alta dimensión, **no existe ningún ejemplo φ_i que sea lo suficientemente cercano a una φ arbitraria** que queramos evaluar. Esto significa que los datos disponibles son extremadamente dispersos.

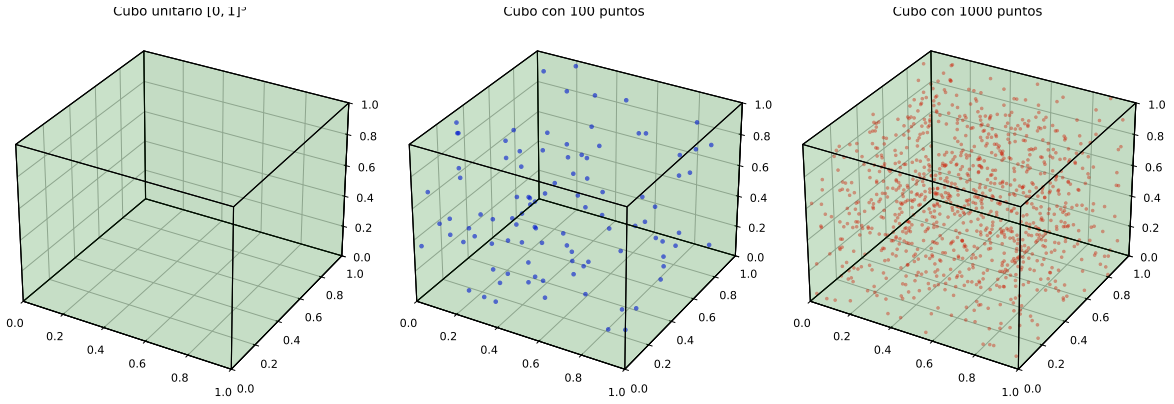


Figura 1.2: Ejemplo de la maldición de la dimensionalidad en un cubo de tres dimensiones

Para ilustrar esto, consideremos la densidad de puntos en un cubo unitario $[0, 1]^d$:

- Si queremos que nuestros puntos de datos estén distribuidos de tal manera que cualquier punto en el espacio esté a una distancia ϵ de al menos un punto de datos (es decir, cubrir el espacio con esferas de radio ϵ), el número de puntos n necesarios crece exponencialmente con la dimensión d. Específicamente, necesitaríamos aproximadamente $n \sim (1/\epsilon)^d$ puntos.
- Por ejemplo, si $\epsilon = 1/10$ (es decir, queremos una resolución del 10 %) y $d = 10$, necesitaríamos $n \sim (10)^{10}$ puntos. Esta cantidad es astronómica e inalcanzable en la práctica.

Es así que es imperativo encontrar métodos que nos permitan trabajar con representaciones de menor dimensión de los datos manteniendo la mayor cantidad de información estadística del conjunto, estas técnicas se denominan **reducción de dimensionalidad**.

En física, este problema se aborda analizando las **interacciones globales** o colectivas de un sistema, en lugar de intentar modelar cada interacción individual. Esto significa que, en lugar de centrarnos en la interacción entre pares de partículas o componentes, buscamos identificar y modelar las **interacciones de grupo** o las propiedades emergentes del sistema en su conjunto. Este enfoque colectivo permite simplificar drásticamente la complejidad inherente a los sistemas de alta dimensión, buscando principios más generales que rigen el comportamiento macroscópico o las propiedades de campo.

1.3. Análisis multidimensional

Como se mencionó anteriormente, un enfoque fundamental en el análisis físico de diversos fenómenos consiste en simplificar las interacciones a distintas escalas con el fin de comprender el comportamiento global del sistema. Este principio es clave en múltiples áreas de la física, como en la física estadística, o en los algoritmos de simulación de N cuerpos, que aproximan las interacciones lejanas para reducir la complejidad computacional. Entender el comportamiento de los sistemas físicos a múltiples escalas no solo es esencial para una descripción precisa, sino también para desarrollar modelos eficientes y escalables.

1.3.1. Renormalización: Abordando las Múltiples Escalas

El concepto de renormalización (pionero por Kenneth G. Wilson en la década de 1970) es una herramienta conceptual y computacional poderosa para abordar el problema de la aproximación y la maldición de la dimensionalidad, especialmente en sistemas con interacciones multi-escala [2].

- **Idea Central:** Si tenemos un sistema definido a una escala muy fina (microscópica), lo que buscamos hacer es **reducir su dimensionalidad** promediando o coarse-graining (grueso-granulando) las propiedades del sistema en escalas más grandes. Esto nos permite entender el comportamiento emergente a escalas mayores.
- **Proceso Iterativo:** La renormalización es un proceso iterativo donde se promedian, se renormalizanz se reducen las dimensiones de los datos a través de diferentes escalas. El objetivo es calcular la probabilidad a través de estas escalas, revelando cómo las propiedades macroscópicas emergen de las microscópicas.
- **Grupo de Renormalización (RG):** La aplicación de un grupo de renormalización (RG) actúa sobre los parámetros de acoplamiento del sistema, mostrando cómo estos evolucionan a medida que cambiamos la escala de observación. Este es un concepto fundamental en la teoría de transiciones de fase y fenómenos críticos.

Ejemplo: Distribución Gaussiana Multi-escala de Largo Alcance Consideremos un ejemplo específico de un modelo probabilístico que exhibe interacciones multi-escala. Un modelo gaussiano de largo alcance puede tener la siguiente forma para la distribución de probabilidad $p(\phi)$:

$$p(\phi) = Z^{-1} e^{-U(\phi)} \quad \text{con} \quad U(\phi) = \frac{1}{2}\phi^T K \phi$$

Aquí, K es la matriz de acoplamiento. Para una distribución gaussiana, la inversa de esta matriz, K^{-1}, es la matriz de covarianza.

- **Propiedad Clave:** Para una distribución gaussiana, K^{-1} es la covarianza.
- **Estacionariedad:** Si el campo es estacionario (sus propiedades estadísticas no cambian con la posición), K y K^{-1} son diagonalizables en el espacio de Fourier. Esto simplifica el análisis de las correlaciones en diferentes frecuencias (escalas).
- **Espectro de Potencia:** El espectro de potencia (que representa la varianza de las componentes de Fourier del campo) corresponde a los valores propios de la matriz de covarianza K^{-1}.
- **Correlaciones de Largo Alcance:** En muchos sistemas físicos, como la turbulencia o las estructuras cosmológicas, el espectro de potencia sigue una **ley de potencias** a bajas frecuencias:

  $$|\omega|^{-\eta}$$

  Donde ω son las frecuencias (inversas a las escalas) y η es un exponente. Una ley de potencias a bajas frecuencias indica **correlaciones de largo alcance** o memoria en el sistema.

- **Aproximación Gaussiana (limitaciones):** Aunque las distribuciones gaussianas son útiles, a menudo son una **mala aproximación** para campos físicos complejos que exhiben fenómenos no gaussianos como intermitencia, vórtices o filamentos (como se ve en las imágenes adjuntas, si las hubiera). Las imágenes suelen mostrar la complejidad de los campos reales versus una aproximación gaussiana.

1.4. Principios de renormalización en el aprendizaje automático

Una de las similitudes más notables entre el RG y el aprendizaje profundo radica en el proceso de **extracción jerárquica de características**. Los modelos de aprendizaje profundo, en particular las *redes neuronales convolucionales* (CNNs), se caracterizan por su arquitectura multinivel, donde cada capa procesa la salida de la anterior para construir representaciones cada vez más abstractas y complejas de los datos de entrada. Esta abstracción capa por capa en el aprendizaje profundo es directamente análoga al procedimiento iterativo de *coarse-graining* central en el Grupo de Renormalización en física.

En el RG, el *coarse-graining* consiste en simplificar un sistema agregando variables microscópicas en bloques macroscópicos o integrando las fluctuaciones a corta distancia. Este proceso revela de forma sistemática cómo cambian las correlaciones y las interacciones efectivas al aumentar la escala. De manera similar, en las redes profundas, las primeras capas pueden detectar patrones locales finos (como bordes en una imagen), mientras que las capas posteriores combinan estas características simples en patrones más complejos a gran escala (como texturas, partes de objetos o incluso objetos completos). Esta abstracción progresiva actúa como un proceso de *coarse-graining* sobre los datos de entrada, reduciendo su dimensionalidad mientras preserva la información esencial para tareas de alto nivel.

La capacidad de los modelos de aprendizaje profundo para descubrir descripciones análogas a la renormalización en teorías físicas, como el modelo de Ising, ha llevado a la hipótesis de que este mecanismo podría explicar su eficacia al modelar distribuciones generales con correlaciones altamente dependientes de la escala, como las que se encuentran en problemas de visión por computadora.

2 Redes Neuronales

Durante los últimos años hemos sido testigos de un crecimiento acelerado y sin precedentes en la integración de diversas herramientas computacionales en nuestra vida cotidiana. Uno de los campos más destacados en esta transformación es el de la inteligencia artificial, cuyo objetivo principal es desarrollar sistemas capaces de realizar tareas comúnmente asociadas con el razonamiento humano [3].

Dentro de este ámbito, una de las áreas con mayor desarrollo ha sido el **aprendizaje automático** (*machine learning*), el cual, mediante una variedad de técnicas estadísticas y computacionales, busca construir sistemas complejos que sean capaces de *aprender* el comportamiento de fenómenos no triviales a partir de un conjunto dado de *datos de entrenamiento*, con el propósito de reproducirlos, modelarlos o extraer conocimiento de ellos, siendo ampliamente usadas para el reconocimiento de imágenes, texto, predicción de fenómenos físicos, etc.

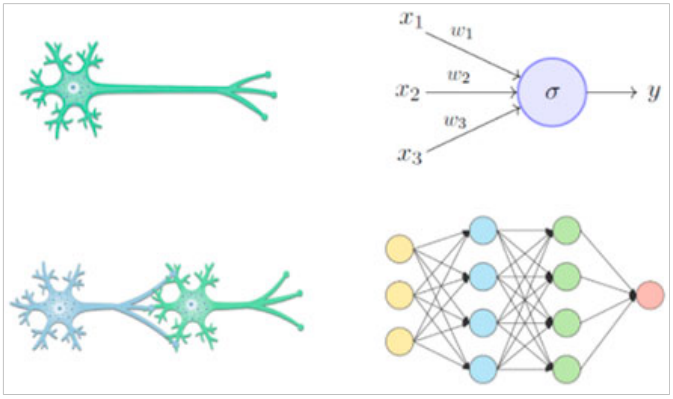


Figura 2.1: Diagrama que muestra la analogia entre una neurona biológica y una sintética

Es en este contexto que surgen las **redes neuronales**, las cuales en su forma más simple pueden ser entendidas como **aproximadores universales de funciones**, tomando como bloque funcional una neurona emulando el funcionamiento conjunto de una red de neuronas biológicas de manera sintética, tomando datos de entrada y aplicando diferentes operaciones matemáticas con el fin de, mediante una variedad de técnicas de entrenamiento, las redes aprenden a mapear los datos de entrada en los resultados que buscamos a su salida. En la actualidad estas redes neuronales tienen una gran cantidad de capas, es por esto que se denominan redes neuronales profundas (**Deep learning** [3])

2.1. Arquitecturas de redes neuronales

La conexión entre los diferentes elementos de las redes neuronales (también denominados nodos o neuronas) representa un paradigma computacional abierto, donde diferentes estructuras de redes neuronales tienen un mejor rendimiento en diferentes tareas para las que se apliquen. En los últimos años una gran variedad de arquitecturas han surgido para responder a la resolución de problemas mas complejos, donde las entradas de cada red neuronal pueden ser del orden de millones y los conjuntos de entrenamiento aun mayores,. A continuación, se presentan las principales estructuras identificadas en la literatura y explicadas en [4].

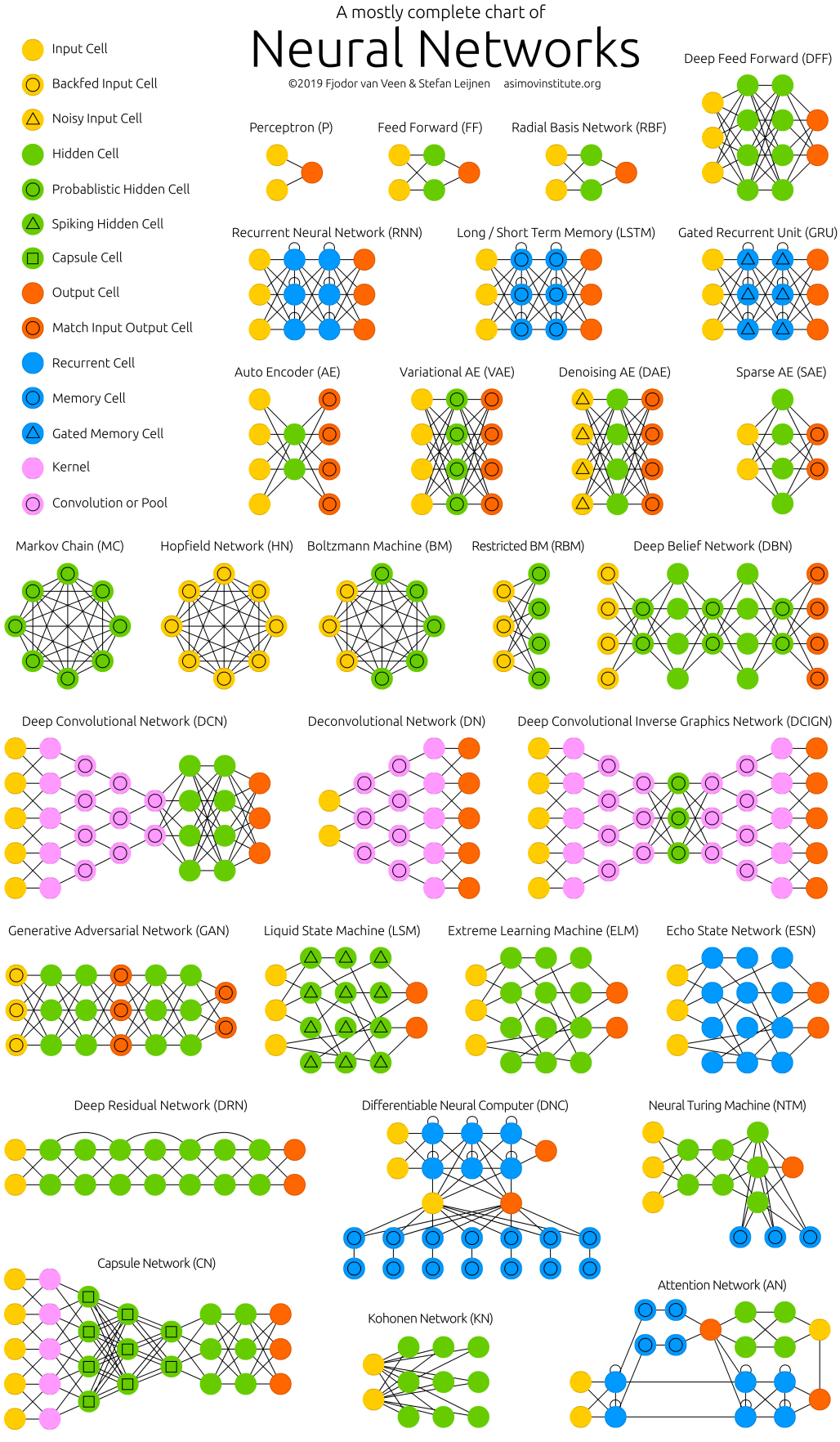


Figura 2.2: Diferentes arquitecturas de redes neuronales, tomado de [5]

Redes Neuronales Feedforward (FF) Las redes **feedforward** son las más tradicionales y conectan las capas de entrada y salida mediante capas ocultas sin ciclos. La información fluye en una sola dirección: desde la entrada hasta la salida. Son ampliamente utilizadas en clasificación supervisada y regresión. Una subclase importante son las redes profundas (*deep feedforward networks*), que incluyen múltiples capas ocultas [4].

Redes Neuronales Recurrentes (RNN) Las **RNN** están diseñadas para procesar datos secuenciales mediante conexiones recurrentes entre sus nodos, lo cual les proporciona memoria temporal. Variantes avanzadas incluyen:

- **LSTM (Long Short-Term Memory):** optimizadas para largas dependencias temporales.
- **GRU (Gated Recurrent Unit):** una versión más simple y eficiente en algunos casos.
- **ESN (Echo State Network):** redes con conexiones internas fijas y entrenamiento sólo en la salida.

Redes Convolucionales Profundas (DCNN) Las **DCNN** son redes neuronales aplicadas a tareas de visión computacional. Usan filtros locales y capas de agrupamiento donde por medio de convoluciones extraen características espaciales. Han sido fundamentales en clasificación de imágenes y segmentación semántica. Arquitecturas relacionadas incluyen:

- **Deconvolutional Networks (DN):** redes inversas que permiten reconstrucción espacial jerárquica.
- **DCIGN (Deep Convolutional Inverse Graphics Network):** combinan DCNN con autoencoders variacionales.

Redes Generativas Adversarias (GAN) Las **GAN** consisten en dos redes: un *generador*, que produce datos sintéticos, y un *discriminador*, que evalúa su autenticidad. Entrenan compitiendo una con otra. Se han usado exitosamente en la generación de imágenes, audio y en simulaciones físicas [4].

Redes Autoencoder (AE) Un **autoencoder** aprende una representación comprimida de los datos. Existen múltiples variantes:

- **VAE (Variational Autoencoder):** introduce regularización probabilística para modelado generativo.
- **DAE (Denoising Autoencoder):** entrenados para reconstruir datos a partir de versiones ruidosas.
- **SAE (Stacked Autoencoder):** autoencoders profundos apilados para aprendizaje no supervisado.

Otras arquitecturas destacadas

- **RBM (Restricted Boltzmann Machine):** redes estocásticas para reducción de dimensión y filtrado colaborativo.
- **DBN (Deep Belief Network):** composición de múltiples RBMs o autoencoders.
- **DRN (Deep Residual Network):** incluye conexiones residuales que permiten redes muy profundas.
- **KN (Kohonen Network):** también llamadas mapas auto-organizables para clustering y visualización.

- **NTM (Neural Turing Machine):** integra una red neuronal con una memoria externa diferenciable.
- **LSM/ELM (Liquid State Machine / Extreme Learning Machine):** redes rápidas con conexiones aleatorias y entrenamiento mínimo.
- **GNN (Graph Neurl Networks):** redes aplicadas al procesamiento de grafos relacionales, explorando sus conexiones y propiedades individuales.

La figura 2.2 se presenta un "zoológico de redes neuronales" tomado de [5] destacando la diversidad estructural y funcional de estas y otras arquitecturas.

Redes Físicamente Informadas El uso de arquitecturas de redes neuronales no se ha limitado únicamente al ámbito industrial. En la actualidad, su aplicación en el desarrollo de ciencia de frontera ha crecido de forma acelerada. Un ejemplo destacado es **AlphaFold**, desarrollado por *Google DeepMind*, un sistema capaz de predecir la estructura tridimensional de una proteína a partir de su secuencia de aminoácidos, con una precisión comparable a la obtenida mediante técnicas experimentales [6].

Este avance evidencía el impacto creciente de las redes neuronales en la ciencia, en el caso de la física se ha impulsado el desarrollo de las llamadas **redes neuronales físicamente informadas** (*Physics-Informed Neural Networks*, PINNs). Estas redes incorporan restricciones físicas directamente en su arquitectura o durante el entrenamiento, lo que les permite aprovechar su capacidad de aproximación universal para resolver sistemas regidos por leyes físicas conocidas.

Entre sus aplicaciones se encuentran la predicción del comportamiento de fluidos, la simulación de materiales, y contribuciones recientes en el diseño y control de reactores de *fusión nuclear*, entre otras áreas. A continuación, se presentan algunos ejemplos destacados de su uso.

1. **Mecánica de Fluidos:** Resolución de flujos supersónicos mediante la incorporación de las ecuaciones de Navier-Stokes en la función de pérdida de la red neuronal, permitiendo predecir campos de velocidad y presión [7].
2. **Mecánica de Sólidos:** Solución de problemas de elasticidad plana mediante un ensamble de PINNs que modelan desplazamientos y esfuerzos en materiales elásticos [8].
3. **Geofísica:** Inversión de formas de onda sísmicas completas utilizando PINNs para resolver la ecuación de onda acústica en medios heterogéneos [9].
4. **Física de Materiales:** Simulación de procesos termo-mecánicos en fluidos no newtonianos, como el calandrado de caucho, usando PINNs para abordar tanto problemas directos como inversos [10].
5. **Fisia Nuclear:** Solución de las ecuaciones cinéticas puntuales que describen la dinámica de reactores nucleares, modelando con precisión el comportamiento transitorio del sistema [11].
6. **Cosmología Computacional:** Uso de redes neuronales basadas en grafos (GNNs) para reconstruir el campo de velocidades cósmico a partir de la distribución espacial de galaxias, capturando relaciones locales y no locales entre halos [12].

2.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNNs, por sus siglas en inglés) son una clase especializada de redes neuronales profundas diseñadas para procesar datos con estructura en forma de rejilla, como imágenes bidimensionales o mapas tridimensionales en simulaciones físicas. Su arquitectura está inspirada en el sistema visual de los seres vivos, que analiza información de manera jerárquica, enfocándose primero en patrones locales para posteriormente analizar estructuras más abstractas [13].

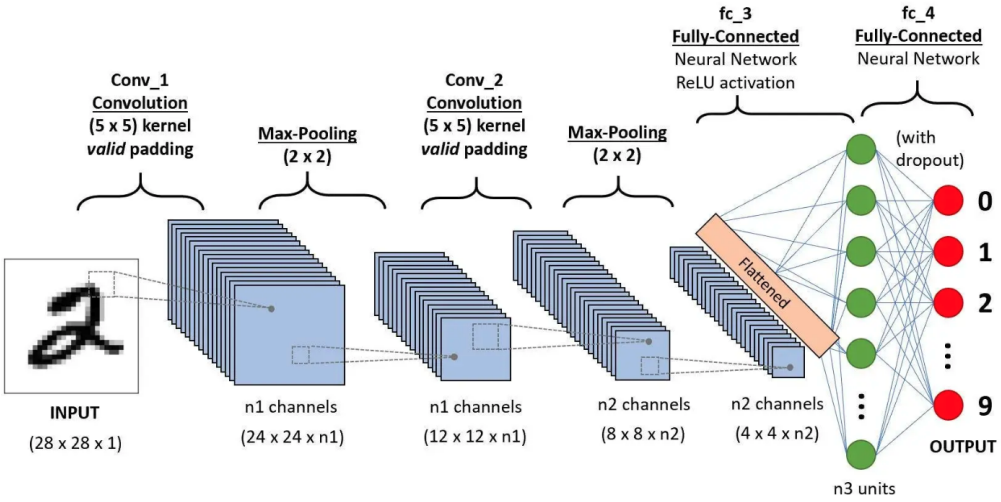


Figura 2.3: Arquitectura de red neuronal convolucional para clasificación en el MNIST dataset, tomada de [14]

A diferencia de las redes neuronales densamente conectadas, las CNNs aprovechan la correlación espacial de los datos mediante filtros llamados *kernels* (estos pueden ser fijos o aprendidos) que se aplican de forma local y compartida en toda la entrada, permitiendo:

- **Reducción de parámetros:** se evita conectar cada neurona con todas las anteriores, lo que disminuye el número total de pesos entrenables.
- **Captura de estructuras espaciales:** se identifican patrones como bordes, texturas, simetrías o regiones específicas.
- **Invarianza ante traslaciones:** gracias a la operación de *pooling* y a la naturaleza local de los filtros.

2.2.1. Arquitectura Típica de una CNN

Una red convolucional como la mostrada en la figura2.3 puede dividirse en los siguientes bloques funcionales:

- **Capas de convolución (*Conv layers*):** La **convolución bidimensional** es una operación fundamental en procesamiento de imágenes; en estas capas se aplican filtros que por medio de esta operación extraen características locales de la imagen de entrada. Matemáticamente dada una imagen de entrada $I(x, y)$ y un kernel $K(i, j)$ de tamaño $m \times n$, la convolución se define como:

$$O(x, y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} K(i, j) \cdot I(x + i - a, y + j - b)$$

  donde:

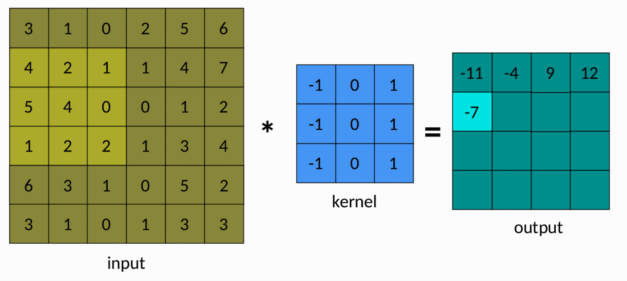


Figura 2.4: Ejemplo de aplicación de kernel en imagen

- $O(x, y)$: valor de salida en la posición (x, y),
- $K(i, j)$: valor del kernel en la posición (i, j),
- $I(x + i - a, y + j - b)$: valor de la imagen centrado respecto al kernel,
- $a = \lfloor \frac{m}{2} \rfloor$, $b = \lfloor \frac{n}{2} \rfloor$: centro del kernel.

Al aplicar esta operación, la dimensión de nuestro tensor de salida se ve reducido, con lo cual a menudo se aplica un *Padding*, que consiste en agregar bordes (normalmente ceros) de tal manera que la salida mantenga la misma dimensión. Un parámetro igual de importante en la aplicación de esta operación es el *Stride*, el cual define el número de píxeles que se mueve el kernel después de cada aplicación.

Al modificar estos parámetros así como el contenido del kernel producimos salidas denominadas *mapas de caracteristicas* que contienen información sobre bordes, texturas o patrones específicos, los cuales son pasados a otra capa de la red con el fin de ejecutar acciones que van desde la clasificación de objetos hasta la síntesis de imágenes nuevas.

- **Capas de *pooling*:** Estas capas nos permiten reducir la dimensionalidad de los mapas de características obtenidos, reteniendo la mayor cantidad de información posible con el objetivo de introducir invarianzas en la traslación, definir la importancia de las características y prevenir el overfitiing. Uno de los tipos de capas mas usadas es ***max pooling*** que retiene el valor máximo de cada subregión.

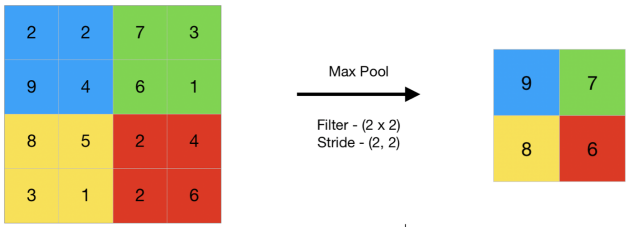


Figura 2.5: Ejemplo de aplicación de Max Pooling

- **Funciones de activación:** Una vez se obtienen los mapas de características de cada imagen el siguiente paso es alimentar esta información en una arquitectura de red neuronal, para esto es necesario que las neuronas individuales aprendan **relaciones complejas no lineales** entre los datos, es ahí donde surge la importancia de las funciones de activación, algunos ejemplos son *sigmoide, tanh, softmax y ReLU*. En el caso de las CNN la funcion mas usada es ***ReLU (Rectified Linear Unit)***, definida maetmáticamente como:

$$\text{ReLU}(x) = \text{máx}(0, x)$$

- **Capas densas o totalmente conectadas:** suelen encontrarse al final de la red y permiten realizar clasificación o regresión.

- **Capa de salida:** depende del problema: una neurona para regresión o múltiples salidas para clasificación.

2.2.2. Ejemplo: Clasificación de Dígitos

Uno de los ejemplos más estudiados es el conjunto de datos MNIST, que contiene imágenes en escala de grises de dígitos escritos a mano (28×28 píxeles). Una CNN típica para clasificar estos dígitos puede estructurarse como sigue[3]:

- Capa convolucional 1: 32 filtros de tamaño 3×3, ReLU
- Max pooling 1: 2×2
- Capa convolucional 2: 64 filtros de tamaño 3×3, ReLU
- Max pooling 2: 2×2
- Capa densa: 128 neuronas, ReLU
- Capa de salida: 10 neuronas (softmax para cada dígito del 0 al 9)

Este modelo alcanza una precisión mayor al 98 % en clasificación, con un tiempo de entrenamiento moderado gracias al uso eficiente de parámetros.

A continuación se muestra una implementación básica en PyTorch:

```
import torch.nn as nn
import torch.nn.functional as F

class MNIST_CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)   # Conv 1 -> 32 canales
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)  # Conv 2 -> 64 canales
        self.pool = nn.MaxPool2d(2, 2)                 # MaxPool 2x2
        self.fc1 = nn.Linear(64 * 5 * 5, 128)          # FC oculta
        self.fc2 = nn.Linear(128, 10)                  # FC salida 10 clases

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))           # [1,28,28] -> [32,13,13]
        x = self.pool(F.relu(self.conv2(x)))           # -> [64,5,5]
        x = x.view(-1, 64 * 5 * 5)                     # Aplanar
        x = F.relu(self.fc1(x))                        # FC oculta
        x = self.fc2(x)                                # FC salida
        return x
```

Listing 2.1: Modelo CNN para clasificacion de MNIST

La red se entrena usando entropía cruzada como función de pérdida y un optimizador como `Adam`:

```
model = MNIST_CNN().to(device)                    # Inicializa modelo
criterion = nn.CrossEntropyLoss()                 # Funcion de perdida
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)  # Optimizador

for epoch in range(10):                           # Epocas de entrenamiento
    for images, labels in dataloader:             # Itera sobre lotes
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()                     # Resetea gradientes
        outputs = model(images)                   # Forward pass
        loss = criterion(outputs, labels)         # Calcula perdida
        loss.backward()                           # Backpropagation
        optimizer.step()                          # Actualiza pesos
```

Listing 2.2: Entrenamiento basico

2.3. Redes Neuronales Residuales

A lo largo de los años, con el aumento del poder computacional, el uso de arquitecturas de redes neuronales cada vez más profundas era la norma; sin embargo esto hizo que las redes fueran cada vez más difíciles de entrenar. La profundidad de las redes hacía que el gradiente para el aprendizaje se hiciera muy pequeño o tendiera a infinito, además de que de manera regular la señal de entrada se perdía en las últimas capas de la red. Fue en estas circunstancias que en el 2015 investigadores He et al. de Microsoft [15] propusieron un nuevo tipo de arquitectura que sobrepasó el rendimiento de las redes convolucionales profundas.

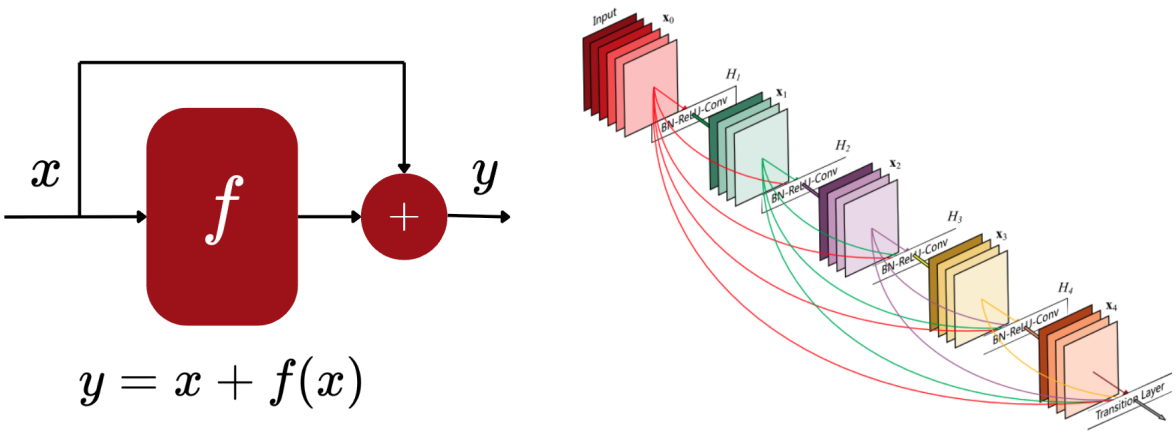


Figura 2.6: Arquitectura de red neuronal residual con bloque funcional

Estas redes funcionan de manera similar a muchas aproximaciones en la física, donde se parte de una aproximación general de entrada y se aprende a modelar la corrección entre el objetivo y la aproximación, denominado ***residuo***. Esto se consigue introduciendo conexiones directas entre capas no contiguas a través de bloques denominados **bloques residuales**.

Cada bloque residual aprende una transformación de la forma:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

donde $\mathbf{x}$ es la entrada del bloque, $\mathcal{F}$ representa una función de transformación (típicamente dos convoluciones con activaciones intermedias), y $\mathbf{y}$ es la salida del bloque. Esta estructura permite al modelo aprender directamente las diferencias o *residuos* entre la entrada y la salida deseada, facilitando así el flujo del gradiente durante el entrenamiento.

Además, es común que la función $\mathcal{F}$ incorpore operaciones de **Batch Normalization** (BN) entre las capas convolucionales. La normalización por lotes estabiliza y acelera el entrenamiento al reducir la covariancia interna (internal covariate shift), es decir, las variaciones en la distribución de las activaciones a lo largo del tiempo. BN estandariza cada mini-lote para tener media cero y varianza unitaria, y luego aplica un reescalado aprendido. Esto ayuda a mantener las activaciones en un rango controlado, lo cual favorece una propagación de gradientes más robusta, permitiendo entrenar redes más profundas de manera eficiente.

2.3.1. Ejemplo Práctico

En este ejemplo el bloque residual `ResidualBlock` toma una entrada x, la pasa por dos capas convolucionales con activación intermedia, y luego suma el resultado con la entrada original.

$$y = \text{ReLU}(\mathcal{F}(x) + x) \tag{2.1}$$

donde x es la entrada, $\mathcal{F}(x)$ es el residuo aprendido, y la suma es una *conexión residual* que facilita la propagación del gradiente. Esta suma permite que el modelo aprenda únicamente los cambios necesarios (el residuo) y no la transformación completa, haciendo que el entrenamiento sea más estable y eficiente.

Implementación en PyTorch

En el código `SmallResNet`, la imagen de entrada pasa por una convolución inicial para aumentar el número de canales, luego por dos bloques residuales que refinan progresivamente las representaciones, y finalmente por una convolución de salida que produce la imagen reconstruida.

```
class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.conv1 = nn.Conv2d(channels, channels, 3, padding=1)  # 1a convolucion
        self.bn1 = nn.BatchNorm2d(channels)                       # BN
        self.conv2 = nn.Conv2d(channels, channels, 3, padding=1)  # 2a convolucion
        self.bn2 = nn.BatchNorm2d(channels)                       # BN

    def forward(self, x):
        identity = x                                # Conexion residual
        out = F.relu(self.bn1(self.conv1(x)))       # Conv + BN + ReLU
        out = self.bn2(self.conv2(out))             # Conv + BN
        return F.relu(out + identity)               # Suma residual y activacion

class SmallResNet(nn.Module):
    def __init__(self, in_channels=1, base_channels=64):
        super().__init__()
        self.input_conv = nn.Conv2d(in_channels, base_channels, 3, padding=1)
        self.res1 = ResidualBlock(base_channels)     # 1er bloque residual
        self.res2 = ResidualBlock(base_channels)     # 2do bloque residual
        self.output_conv = nn.Conv2d(base_channels, 1, 3, padding=1) # Salida final

    def forward(self, x):
        x = F.relu(self.input_conv(x))      # Activacion inicial
        x = self.res1(x)                    # Primer paso residual
        x = self.res2(x)                    # Segundo paso residual
        return self.output_conv(x)          # Generacion de salida
```

Listing 2.3: Ejemplo de una red residual simple en PyTorch

Una vez definida podemos crear un codigo simple para entrenarla como el que se muestra a continuación.

```
x_train = torch.randn(100, 1, 28, 28)               # Imagenes entrada
y_train = x_train + 0.1 * torch.randn_like(x_train) # Objetivo con ruido
loader = DataLoader(TensorDataset(x_train, y_train), batch_size=10)

model = SmallResNet()                               # Modelo
loss_fn = nn.MSELoss()                              # Perdida MSE
opt = torch.optim.Adam(model.parameters(), lr=1e-3) # Optimizador

for epoch in range(10):
    for x, y in loader:
        opt.zero_grad()                             # Reset gradientes
        loss = loss_fn(model(x), y)                 # Calculo perdida
        loss.backward(); opt.step()                 # Backprop y actualizacion
    print(f"Epoch {epoch+1}: {loss.item():.4f}")    # Mostrar perdida
```

Listing 2.4: Entrenamiento simple de red residual

Esta arquitectura puede integrarse fácilmente en arquitecturas más complejas como U-Nets o generadores de GANs.

2.4. Mecanismos de atención

Los mecanismos de atención han demostrado ser herramientas poderosas en redes neuronales profundas, al permitir que el modelo seleccione dinámicamente las características más relevantes para una tarea dada. Inspirados por mecanismos biológicos de enfoque visual selectivo, estos métodos buscan realzar las representaciones informativas y suprimir aquellas menos útiles [16, 17].

Una de las formas más eficaces y computacionalmente ligeras de implementar atención en redes convolucionales es a través de los bloques *Squeeze-and-Excitation* (SE), introducidos por Hu et al. [18]. Estos bloques permiten a la red recalibrar dinámicamente los canales de sus mapas de activación, modelando explícitamente las interdependencias entre canales.

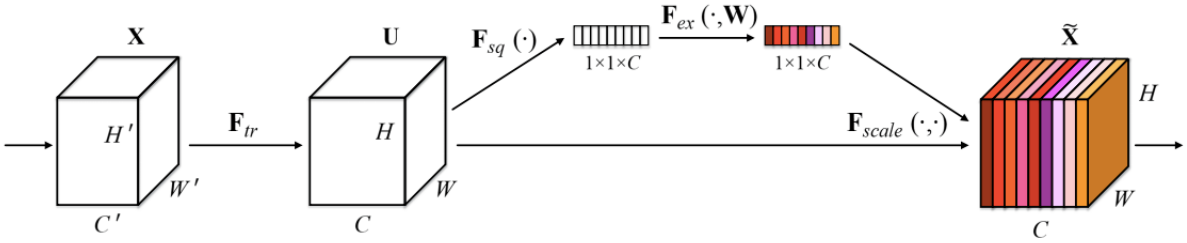


Figura 2.7: Proceso de Squeeze and Excite

2.4.1. Fundamentos Matemáticos del Bloque SE

Dado un tensor de entrada $\mathbf{X} \in \mathbb{R}^{C' \times H' \times W'}$, este primero se transforma a un nuevo tensor de características $\mathbf{U} = F_{tr}(\mathbf{X}) \in \mathbb{R}^{C \times H \times W}$, por medio de una operación convolucional o residual previa.

El bloque Squeeze-and-Excitation aplica atención canal-dependiente a $\mathbf{U}$ mediante las siguientes etapas:

1. **Squeeze (F_{sq}):** Se aplica un promedio global por canal a $\mathbf{U}$, generando un descriptor de contexto global:

$$z_c = \frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{U}_{c,i,j}, \quad \text{para } c = 1, \ldots, C$$

   Esto produce un vector $\mathbf{z} \in \mathbb{R}^{1 \times 1 \times C}$.

2. **Excitation (F_{ex}):** El vector $\mathbf{z}$ es transformado mediante una red neuronal de dos capas completamente conectadas con activación intermedia ReLU y sigmoide al final:

$$\mathbf{s} = \sigma(W_2 \cdot \delta(W_1 \cdot \mathbf{z}))$$

   donde $W_1 \in \mathbb{R}^{\frac{C}{r} \times C}$, $W_2 \in \mathbb{R}^{C \times \frac{C}{r}}$, y δ representa la función ReLU. El resultado $\mathbf{s} \in \mathbb{R}^{1 \times 1 \times C}$ contiene los coeficientes de atención por canal.

3. **Scale (F_{scale}):** Finalmente, se realiza una reponderación canal por canal del tensor original $\mathbf{U}$ con los coeficientes aprendidos:

$$\tilde{\mathbf{X}}_c = s_c \cdot \mathbf{U}_c, \quad \text{para } c = 1, \ldots, C$$

   obteniendo la salida recalibrada $\tilde{\mathbf{X}} \in \mathbb{R}^{C \times H \times W}$.

Este mecanismo introduce una atención dinámica entre canales, permitiendo que la red enfoque su capacidad representacional en aquellas activaciones que son más relevantes en el contexto global, como se ilustra en la Figura 2.7.

2.4.2. Aplicaciones y Beneficios

Los bloques SE han sido incorporados exitosamente en arquitecturas profundas como ResNet, DenseNet o Inception, mejorando su precisión sin introducir un costo computacional significativo [18]. En tareas de visión por computadora, estos bloques han demostrado mejorar la capacidad del modelo para enfocar regiones relevantes del espacio de características y facilitar el aprendizaje profundo al mejorar el flujo de gradientes.

Además, su implementación modular los hace compatibles con tareas como clasificación, segmentación, y super-resolución, lo que explica su integración frecuente en arquitecturas residuales modernas [19].

2.4.3. Ejemplo

```python
import torch
import torch.nn as nn

class SEBlock(nn.Module):
    def __init__(self, channels, reduction=16):
        super(SEBlock, self).__init__()
        self.pool = nn.AdaptiveAvgPool2d(1)                # Global average pooling
        self.fc = nn.Sequential(
            nn.Linear(channels, channels // reduction),  # Bottleneck: reduce dim
            nn.ReLU(inplace=True),
            nn.Linear(channels // reduction, channels),  # Restore original dim
            nn.Sigmoid()                                 # Attention weights [0,1]
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.pool(x).view(b, c)                    # Compress spatial info
        y = self.fc(y).view(b, c, 1, 1)                # Expand back to conv shape
        return x * y.expand_as(x)                      # Scale input by attention
```

Listing 2.5: Bloque Squeeze-and-Excitation en PyTorch

3
Super Resolución

3.1. Súper Resolución de Imágenes

La **súper resolución** es el proceso de obtener imágenes de alta resolución (HR, por *high resolution*) a partir de una o más imágenes de baja resolución (LR, por *low resolution*) mediante una variedad de técnicas de visión computacional. Este problema inverso ha sido objeto de estudio durante décadas debido a su importancia en áreas como la industria del entretenimiento, microscopía, medicina, vigilancia y la astronomía moderna.

El reto principal radica en la naturaleza inherentemente ambigua del proceso: múltiples imágenes de alta resolución pueden corresponder a una misma imagen de baja resolución debido a la pérdida de información espacial durante la adquisición o muestreo, es por esto que en la actualidad la tarea de estudio central de la **súper resolución** es determinar el proceso de degradación que tuvo la imagen de alta a baja resolución y así invertirlo.

Figura 3.1: El espacio de todos los posibles resultados de super resolución

Es de esta forma que a lo largo de los años se han definido una variedad de técnicas que van desde enfoques deterministas (como los presentados con las interpolaciones) hasta los más contemporáneos basándose en el aprendizaje de los datos usando técnicas de aprendizaje automático, llevándonos a nuevas fronteras antes desconocidas para la mejora de los resultados en esta área.

Re-escalado o Sobre Muestreo Una pregunta natural que surge al hablar sobre *súper resolución* radica en entender la diferencia entre **re-escalar** o **sobresembrar** una imagen. Recordemos que de manera esencial el proceso de súper resolución busca revertir el proceso de degradado de una imagen, aumentando la información existente en la misma; en ocasiones esto se relaciona con aumentar la dimensión espacial de una imagen, sin embargo en la mayoría de las ocasiones se trata más de un proceso denominado en el área de procesamiento de señales como **sobre muestreo** ya que buscamos reconstruir los detalles faltantes.

3.2. Primeros métodos: aproximaciones clásicas

Antes de la era del aprendizaje profundo, las soluciones al problema de súper resolución se basaban principalmente en técnicas de interpolación donde el estimado de los valores espaciales faltantes se realizaba empleando procedimientos estadísticas aplicadas a los valores existentes. Estos métodos no requerían datos de entrenamiento y se basaban únicamente en la información contenida en la imagen de baja resolución.

3.2.1. Interpolación por vecino más cercano (Nearest Neighbor)

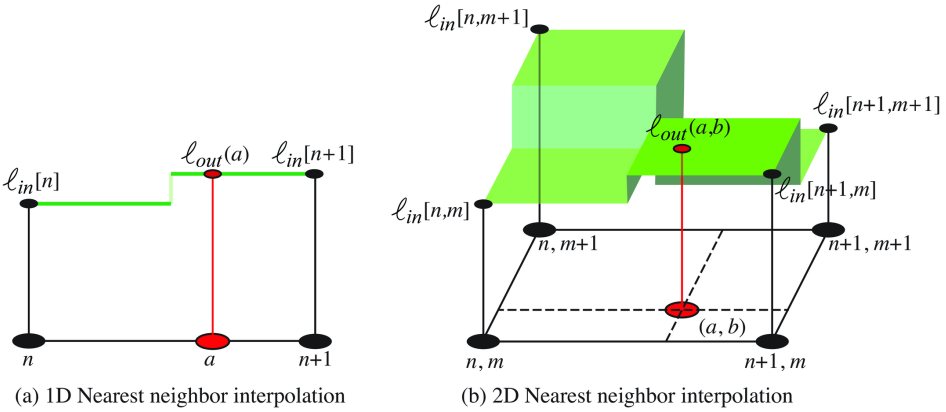
Este método asigna a cada píxel de la imagen escalada el valor del píxel más cercano en la imagen original. Es una forma de muestreo donde no se realiza interpolación real, solo se duplican los pixeles según el factor de escala.

Formalmente, si la imagen original está definida en una cuadrícula $I(i, j)$ y queremos obtener un nuevo valor en la posición flotante (x, y), el valor interpolado se define como:

$$I_{\text{NN}}(x, y) = I\left(\text{round}(x), \text{round}(y)\right)$$

Esta operación consiste en seleccionar el píxel cuya coordenada discreta se encuentra más próxima a (x, y) de acuerdo con la **métrica** ℓ_1, también conocida como *distancia del taxista*, la cual se define como:

$$d_{\ell_1}((x, y), (i, j)) = |x - i| + |y - j|$$



(a) 1D Nearest neighbor interpolation (b) 2D Nearest neighbor interpolation

Figura 3.2: Diagrama de funcionamiento de interpolación por vecino mas cercano

Esto equivale a aplicar un **kernel escalón**:

$$h(x) = \begin{cases} 1 & \text{si } |x| < 0.5 \\ 0 & \text{en otro caso} \end{cases}$$

Ejemplo: Consideremos una matriz 2×2:

$$I = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

Supongamos que queremos escalarla con un factor $s = 2$, es decir, a una imagen 4×4. Para ello, cada píxel original se replica en un bloque de 2×2.

Desarrollo de una columna: Veamos cómo se interpola la primera columna:

$$\text{Original: } \begin{bmatrix} 10 \\ 30 \end{bmatrix} \quad \longrightarrow \quad \text{Reescalada: } \begin{bmatrix} 10 \\ 10 \\ 30 \\ 30 \end{bmatrix}$$

Cada valor de la columna original se replica verticalmente según el factor de escala.

Resultado completo

$$I_{\text{NN}} = \begin{bmatrix} 10 & 10 & 20 & 20 \\ 10 & 10 & 20 & 20 \\ 30 & 30 & 40 & 40 \\ 30 & 30 & 40 & 40 \end{bmatrix}$$

Ventajas y desventajas Este método es extremadamente rápido y simple, sin embargo rápidamente produce bordes dentados y no reconstruye detalles suaves ni texturas, teniendo problemas en casos de re-escalado impar, donde se produce sobre representación de pixeles dando como resultado bloques asimétricos.

Código Esta técnica es fácilmente reproducible en Python empleando la biblioteca OpenCV, se presenta la implementación a continuación.

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Cargar la imagen desde el archivo
img = cv2.imread('example_image_150.jpg')

# Convertir de BGR (OpenCV) a RGB (matplotlib)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Escalar la imagen al doble usando interpolacion de vecino mas cercano
nn = cv2.resize(img_rgb, None, fx=2, fy=2, interpolation=cv2.INTER_NEAREST)

# Mostrar la imagen resultante
plt.imshow(nn)
plt.title("Interpolacion por vecino mas cercano")
plt.axis("off")
plt.show()
```

Listing 3.1: Nearest neighbor interpolation in OpenCV

Una vez ejecutado este código con la imagen deseada apodemos obtendremos los siguientes resultados.

(a) Imagen original 150x150 px

(b) Imagen resultante 300x300 px

Figura 3.3: Resultados de interpolación Nearest neighbor.

3.2.2. Interpolación bilineal

La **interpolación bilineal** es una técnica utilizada para estimar el valor de un píxel en una posición no entera (a, b) dentro de una imagen escalada asumiendo que el valor de la imagen cambia de forma *lineal* entre píxeles, es decir, interpola sobre un plano definido por los cuatro valores vecinos. Este valor se obtiene como una *combinación lineal ponderada* de los cuatro píxeles vecinos más cercanos en la imagen original.

Supongamos que (a, b) es una coordenada flotante, y que (n, m) representa la coordenada entera inferior (esquina superior izquierda) del bloque 2×2 que contiene a (a, b). Los píxeles vecinos relevantes son:

$$\ell_{\text{in}}[n, m], \quad \ell_{\text{in}}[n+1, m], \quad \ell_{\text{in}}[n, m+1], \quad \ell_{\text{in}}[n+1, m+1]$$

El valor interpolado en la posición (a, b) se calcula mediante la fórmula:

$$\ell_{\text{out}}(a, b) = (1-t_x)(1-t_y)\,\ell_{\text{in}}[n, m]+t_x(1-t_y)\,\ell_{\text{in}}[n+1, m]+(1-t_x)t_y\,\ell_{\text{in}}[n, m+1]+t_x t_y\,\ell_{\text{in}}[n+1, m+1]$$

donde:

$$t_x = a - n, \qquad t_y = b - m$$

Los términos t_x y t_y representan la distancia relativa del punto (a, b) respecto a la esquina superior izquierda del bloque en las direcciones x y y, respectivamente.

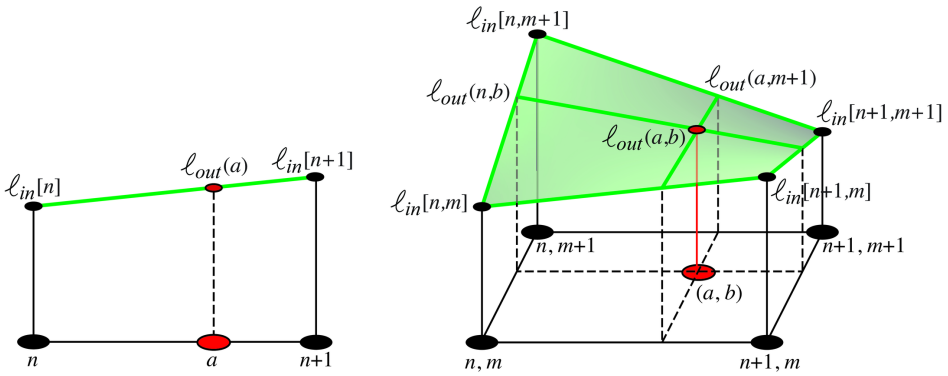


Figura 3.4: Diagrama del funcionamiento de la interpolación bilinear

Ejemplo: Consideremos la misma matriz 2×2:

$$I = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

Supongamos que queremos interpolar el valor en la posición $(0.5, 0.5)$, que se encuentra en el centro del cuadrado formado por los cuatro píxeles. Los cuatro valores vecinos son:

$$\begin{aligned} I(0,0) &= 10, & w_{00} &= (1-0.5)(1-0.5) = 0.25 \\ I(1,0) &= 30, & w_{10} &= 0.5 \cdot 0.5 = 0.25 \\ I(0,1) &= 20, & w_{01} &= 0.5 \cdot 0.5 = 0.25 \\ I(1,1) &= 40, & w_{11} &= 0.5 \cdot 0.5 = 0.25 \end{aligned}$$

Entonces, el valor interpolado es:

$$I_{\text{bil}}(0.5, 0.5) = 10 \cdot 0.25 + 30 \cdot 0.25 + 20 \cdot 0.25 + 40 \cdot 0.25 = \boxed{25}$$

Ventajas y desventajas Al aplicar este método podemos darnos cuenta como la calidad con respecto al método de vecino mas cercano es superior, presentando transiciones suaves con un bajo coste computacional, sin embargo su principal debilidad es esto mismo, al suavizar tanto los bordes tenemos un efecto de desenfoque muy pronunciado, borrando detalles finos y omitiendo texturas.

Resultado completo Notemos como a diferencia del método del método de **vecino mas cercano**, para los casos donde no tengamos al menos 4 valores para realizar la interpolación **no podemos realizar un cálculo directo**, es en este caso que empleamos técnicas de extrapolación, para este ejemplo concreto copiamos el valor del borde mas cercano, de tal forma que después de los cálculos correspondientes, obtenemos el siguiente resultado final

$$I = \begin{bmatrix} 10 & 20 & 30 & 30 \\ 15 & 25 & 35 & 35 \\ 20 & 30 & 40 & 40 \\ 20 & 30 & 40 & 40 \end{bmatrix}$$

Código Implementando en Python empleando la biblioteca OpenCV obtenemos el siguiente código:

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
# Cargar imagen desde archivo (OpenCV la carga en formato BGR por defecto)
img = cv2.imread("img1.png")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convertir de BGR a RGB
# Redimensionar la imagen usando interpolacion bilineal (duplicar dimension)
bilinear = cv2.resize(img, None, fx=2, fy=2, interpolation=cv2.INTER_LINEAR)
# Mostrar la imagen escalada
plt.imshow(bilinear)
plt.title("Bilinear interpolation")
plt.axis("off")
plt.show()
# Imprimir la matriz del canal rojo
print("Scaled matrix (Red channel):")
print(bilinear[:, :, 0])
```

Listing 3.2: Bilinear interpolation in OpenCV with 2x2 matrix

Una vez ejecutado este código con la imagen deseada apodemos obtendremos los siguientes resultados.

(a) Imagen original 150x150 px

(b) Imagen resultante 300x300 px

Figura 3.5: Resultados de interpolación Bilineal.

3.2.3. Interpolación bicúbica

La **interpolación bicúbica** es un método que estima el valor de un nuevo píxel como una *combinación ponderada de 16 píxeles vecinos*, organizados en una vecindad de 4×4. Esto permite obtener imágenes más suaves y agradables visualmente que con métodos más simples como el vecino más cercano o la interpolación bilineal.

Formalmente para una coordenada flotante (x, y), se identifica el píxel entero inferior más cercano:

$$(i, j) = (\lfloor x \rfloor, \lfloor y \rfloor)$$

El valor interpolado se calcula como:

$$I_{\text{bic}}(x, y) = \sum_{m=-1}^{2} \sum_{n=-1}^{2} I(i+m, j+n)\, h(x - (i+m))\, h(y - (j+n))$$

donde $h(t)$ es una **función polinómica de grado 3 definida por tramos**, que asigna pesos según la distancia al punto (x, y).

La función $h(t)$ depende de un parámetro $a \in [-1, 0]$, siendo comúnmente $a = -0.5$. Esta función se define como:

$$h(t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1 & \text{si } |t| < 1 \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a & \text{si } 1 \leq |t| < 2 \\ 0 & \text{si } |t| \geq 2 \end{cases}$$

Ejemplo Sea la siguiente matriz 4×4:

$$I = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 20 & 30 & 40 & 50 \\ 30 & 40 & 50 & 60 \\ 40 & 50 & 60 & 70 \end{bmatrix}$$

Queremos interpolar el valor en $(1.5, 1.5)$, que se encuentra en el centro de la submatriz central. Los elementos relevantes para el cálculo son:

- Centro entero inferior: $i = 1,\ j = 1$
- Se consideran los píxeles vecinos $I(i + m,\ j + n)$ con $m, n \in \{-1,\ 0,\ 1,\ 2\}$
- Evaluamos el kernel cúbico en la forma $h(1.5 - (i+m)) = h(0.5 - m)$ para cada m, y de forma análoga para n

Cálculo parcial del kernel: Usamos $a = -0.5$. Para ejemplo:

$$h(0.5) = (1.5)(0.5)^3 - (2.5)(0.5)^2 + 1 = 0.1875 - 0.625 + 1 = \boxed{0.5625}$$

Igualmente:

$$h(-0.5) = h(0.5) = 0.5625, \quad h(1.5) = -0.0625, \quad h(-1.5) = h(1.5) = -0.0625$$

Interpolación final El valor interpolado es:

$$I_{\text{bic}}(1.5, 1.5) = \sum_{m=-1}^{2} \sum_{n=-1}^{2} I(1+m, 1+n) \cdot h(0.5 - m) \cdot h(0.5 - n)$$

Este cálculo involucra un total de 16 valores de la imagen (provenientes de una vecindad 4×4), junto con 16 evaluaciones del kernel cúbico h, que actúan como pesos. Luego, se realizan 16 multiplicaciones entre los valores y sus respectivos pesos, seguidas de una suma total para obtener el valor interpolado.

Ventajas y desventajas Una de las principales ventajas de la interpolación bicúbica es que produce bordes más suaves y una mejor continuidad visual en la imagen. Sin embargo, su principal desventaja es que resulta más costosa computacionalmente que la interpolación bilineal, y puede generar artefactos de sobreoscilación cuando se aplica a imágenes con alto contraste.

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Cargar imagen desde archivo
img = cv2.imread('example_image_150.jpg')
# Convertir de BGR (formato de OpenCV) a RGB (para matplotlib)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Escalar la imagen usando interpolacion bicubica (2x)
bicubic = cv2.resize(img_rgb, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
# Mostrar la imagen escalada
plt.imshow(bicubic)
plt.title("Bicubic interpolation")
plt.axis("off")
plt.show()
# Guardar imagen escalada
plt.imsave("img_bicubic.jpg", bicubic)
```

Listing 3.3: Bicubic interpolation in OpenCV

Obteniendo Así el siguiente resultado:

(a) Imagen original 150x150 px **(b)** Imagen resultante 300x300 px

Figura 3.6: Resultados de interpolación Bilineal.

3.2.4. Limitaciones de los métodos clásicos

Estos métodos clásicos, aunque útiles en aplicaciones simples, comparten limitaciones fundamentales:

- No recuperan detalles perdidos durante el muestreo original.
- No aprovechan información contextual ni patrones semánticos.
- Generan resultados que no se alinean con la percepción humana de calidad.

Esto motivó la exploración de métodos de aprendizaje automático capaces de aprender transformaciones más complejas a partir de grandes cantidades de datos.

3.3. Súper resolución basada en aprendizaje profundo

El desarrollo del *aprendizaje profundo* ha revolucionado el campo de la súper resolución (SR), los enfoques que emplean redes neuronales profundas han adquirido gran relevancia en el área. Esta popularidad se debe principalmente a su habilidad para aprender de manera automática transformaciones complejas y no lineales que permiten reconstruir imágenes de alta resolución (HR) a partir de versiones de baja resolución (LR) capturando detalles de alta frecuencia que antes parecían imposibles, recuperando texturas, detalles estructurales y patrones de manera eficiente.

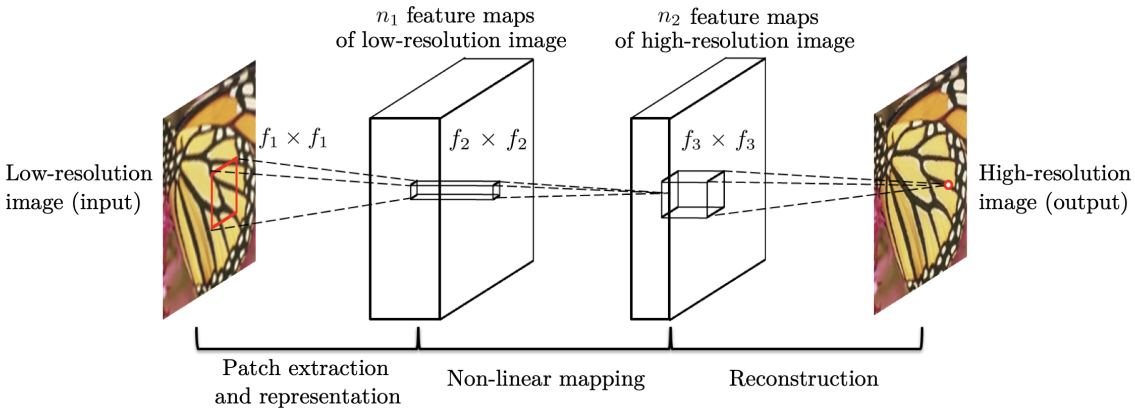


Figura 3.7: Diagrama del funcionamiento de la red convolucional SRCNN

Esta metodología destaca por su fuerte enfoque en el uso de **grandes conjuntos de datos**, mediante los cuales las diferentes arquitecturas de redes neurales aprenden a reconstruir los píxeles faltantes usando el conocimiento adquirido de imágenes similares; este proceso se puede dar en diferentes etapas de la red neuronal, lo que nos permite obtener una gran variedad de arquitecturas con diferentes propositos.

3.3.1. Principales arquitecturas en SR mediante aprendizaje profundo

Podemos agrupar los enfoques de redes neuronales mas usados dentro de las siguientes cuatro clases de arquitecturas:

1. **Redes neuronales convolucionales (CNNs):** Representan la base de la mayoría de los primeros avances. Modelos como SRCNN [20] y sus sucesores (VDSR, EDSR, RCAN) demuestran que el aumento de profundidad y el uso de bloques residuales y atención permiten reconstrucciones cada vez más precisas.

2. **Redes generativas adversarias (GANs):** Introducidas en SR por SRGAN [21], estas redes utilizan una competencia entre un generador y un discriminador para mejorar la calidad perceptual de las imágenes, priorizando la plausibilidad visual por encima de métricas tradicionales como PSNR.

3. **Autoencoders variacionales (VAEs):** Menos frecuentes, pero útiles para aprender representaciones latentes suaves. Se aplican a veces como regularización o como parte de arquitecturas híbridas.

4. **Modelos de difusión:** Recientemente se han convertido en el estado del arte perceptual. Métodos como D3SR [22], SeeSR [23] y variantes de Stable Diffusion adaptadas para SR [24] permiten reconstrucciones realistas de alta fidelidad mediante un proceso inverso de refinamiento estocástico.

Además, existen arquitecturas mixtas o especializadas:

- **Transformers:** redes como SwinIR[?] y sus variantes (Swin2SR, SwinFIR) combinan autoatención con convoluciones para capturar dependencias de largo alcance.
- **Modelos ligeros:** como FSRCNN o MobileSR diseñados para dispositivos embebidos.
- **Métodos ciegos (blind SR):** donde no se asume conocimiento de la degradación (p.ej., Real-ESRGAN, StarSRGAN).

Métodos de aumento de resolución

La estrategia del aumento de resolución (también llamada **sobre muestreo** o como *upsampling* en inglés) dentro de una red neuronal es una decisión arquitectónica crucial en tareas de súper resolución (SR), ya que impacta directamente la calidad de reconstrucción y la eficiencia computacional. A lo largo del tiempo se han propuesto diversos enfoques, los cuales pueden agruparse en las siguientes categorías:

- **Aumento antes de la entrada de la red**
  Este enfoque, empleado en modelos pioneros como SRCNN, consiste en aplicar primero un método de interpolación tradicional (e.g., bicúbica) para escalar la imagen de baja resolución (LR) a una resolución mayor aproximada. Posteriormente, una red convolucional (CNN) refina esta imagen para reconstruir detalles más precisos.

  - **Ventajas:** Simplifica la tarea de la red, ya que se enfoca en el refinamiento. Admite factores de escala arbitrarios.
  - **Limitaciones:** La interpolación puede introducir desenfoques y ruido. Además, operar en el espacio de alta resolución desde el inicio incrementa el costo computacional.

- **Aumento al final de la red**
  Este marco realiza las operaciones principales en el espacio de baja resolución, aplicando el upsampling al final mediante capas aprendibles. Un ejemplo destacado es ESPCN, que introduce la *convolución sub-píxel*.

  - **Ventajas:** Alta eficiencia computacional y menor consumo de memoria. Adecuado para aplicaciones en tiempo real.
  - **Contribuciones:** ESPCN es considerablemente más rápido que los métodos anteriores basados en CNNs, gracias al uso de capas de upsampling de extremo a extremo.

- **Aumento progresivo**
  Modelos como LapSRN escalan la imagen en múltiples etapas. En cada paso se aplica un pequeño aumento de resolución seguido de un refinamiento, descomponiendo el problema en subproblemas más manejables.

  - **Ventajas:** Facilita el aprendizaje para factores de escala grandes y permite SR multiescala en una sola arquitectura.
  - **Limitaciones:** Aumenta la complejidad del diseño del modelo y puede presentar inestabilidad durante el entrenamiento.

Capas específicas de Upsampling

Además de los marcos conceptuales, se utilizan diferentes tipos de capas para realizar el upsampling:

- **Convolución Transpuesta (Deconvolution):** Inserta ceros en la imagen para expandirla antes de aplicar convolución. Puede generar artefactos como patrones de tablero de ajedrez por solapamiento irregular.

- **Capa Sub-píxel (Pixel Shuffle):** Genera múltiples canales a través de convolución y los reorganiza espacialmente. Es más eficiente que la convolución transpuesta y ofrece mayor campo receptivo.
- **Módulo Meta-Upscale:** Permite upsampling para factores de escala arbitrarios mediante meta-aprendizaje. Aprueba zoom continuo en un solo modelo sin necesidad de reentrenar.

3.3.2. Métricas en Súper Resolución

La evaluación objetiva de modelos de súper resolución es un desafío abierto, especialmente cuando las imágenes objetivo tienen características perceptuales o estadísticas alejadas de lo natural, como en el caso de datos cosmológicos. A continuación, se presenta una revisión estructurada de las métricas clásicas, perceptuales y basadas en aprendizaje profundo, junto con sus limitaciones y dominios de aplicación.

Métricas Clásicas: Simples pero Limitadas

Las métricas tradicionales como el Error Cuadrático Medio (MSE) y la Relación Pico Señal-Ruido (PSNR) se han utilizado ampliamente por su simplicidad computacional. Miden diferencias de intensidad entre imágenes a nivel de píxel:

- **MSE (Mean Squared Error):** calcula el promedio del error cuadrático entre píxeles correspondientes.
- **PSNR (Peak Signal-to-Noise Ratio):** expresa el error MSE en dB, favoreciendo reconstrucciones más suaves.

Si bien estas métricas son interpretables y fáciles de calcular, no correlacionan adecuadamente con la percepción visual humana [?]. Por ejemplo, una imagen con bajo MSE puede contener artefactos estructurales evidentes para un observador humano.

Similitud Estructural: SSIM y Derivados

El Índice de Similitud Estructural (SSIM) [?] fue propuesto como una mejora perceptual frente al MSE. Evalúa imágenes en términos de luminancia, contraste y estructura local, reflejando mejor cómo percibe el ojo humano los detalles organizados. SSIM ha sido ampliamente adoptado como métrica base en tareas de visión por computadora.

Otros métodos relacionados incluyen:

- **MS-SSIM:** Extensión multiescala que mejora la robustez ante degradaciones globales [?].

Métricas Perceptuales y Basadas en Información Visual

En respuesta a las limitaciones de SSIM y PSNR, se han desarrollado métricas más avanzadas, basadas en teoría de la información, propiedades estadísticas o redes neuronales entrenadas.

- **FSIM (Feature Similarity Index)** [?]: Utiliza la *phase congruency* y el gradiente para capturar bordes y estructuras salientes, correlacionando bien con la percepción humana.
- **VIF (Visual Information Fidelity)** [?]: Modela la imagen como un proceso estocástico multiescala y estima cuánta información visual se retiene en la reconstrucción, considerando el sistema visual humano como canal de transmisión.
- **MAD (Most Apparent Distortion)** [?]: Ajusta su comportamiento según la visibilidad de las distorsiones, distinguiendo entre artefactos sutiles y evidentes.
- **LPIPS (Learned Perceptual Image Patch Similarity)** [?]: Entrena redes neuronales profundas para medir similitud perceptual a nivel de parches de imagen. Se ha validado ampliamente contra juicios humanos.

- **DISTS (Deep Image Structure and Texture Similarity)** [?]: Combina activaciones neuronales relacionadas con textura y estructura, ofreciendo una mejor correlación con percepciones humanas complejas.

3.3.3. Evaluación en Imágenes No Naturales: Caso Cosmológico

En dominios como la cosmología, donde las imágenes no contienen objetos definidos y presentan características estadísticas gaussianas o multifractales, las métricas perceptuales basadas en bordes o estructuras naturales pueden no ser adecuadas. En estos casos:

- **VIF** es preferible por su enfoque informacional multiescala, capaz de capturar la preservación de propiedades estadísticas relevantes.
- **DISTS y LPIPS** también han mostrado utilidad al correlacionar con detalles espaciales o texturales no evidentes para métricas clásicas [?].

3.3.4. Implementación Práctica

El cálculo de estas métricas puede realizarse mediante diversas herramientas de código abierto:

- `scikit-image` para PSNR, SSIM y MSE.
- `piq` (Perceptual Image Quality) y `TorchMetrics` para LPIPS, DISTS, VIF y MS-SSIM.
- `pywt` y modelos Gaussian Scale Mixture para implementar variantes personalizadas de VIF o MAD.

Estas métricas pueden integrarse en flujos de evaluación de modelos de súper resolución, como métricas de validación, scoring o pérdida perceptual durante entrenamiento.

3.4. Wavelets

Las wavelets son funciones base utilizadas para descomponer señales e imágenes en múltiples niveles de resolución y frecuencia. Esta descomposición multiescala resulta útil para métricas como VIF, que necesitan analizar la información en diferentes escalas, desde la estructura global hasta los detalles más finos.

4
Wavelets

Dentro del análisis de señales, existen diversas técnicas que permiten extraer distintas características de la señal que buscamos estudiar. En este contexto, dos técnicas son ampliamente utilizadas: el **análisis de amplitud** y el **análisis de Fourier**.

Ambas técnicas nos permiten obtener información certera de dos componentes esenciales de la señal por separado: **la componente temporal y laa componente frecuencial**. Sin embargo, en una gran variedad de aplicaciones es de gran utilidad conocer ambas componentes de manera simultánea. Para responder a esta necesidad surgen los **Wavelets**, herramientas matemáticas que permiten identificar la presencia de distintas frecuencias en una señal dentro de intervalos de tiempo específicos, exploraremos su funcionamiento y aplicaciones.

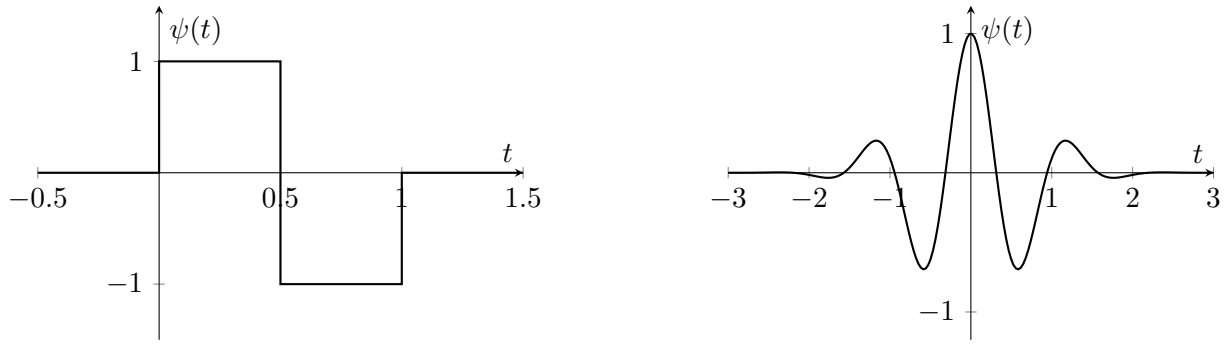
4.1. Definición

La palabra *wavelet* proviene del inglés donde puede ser traducida como "onda pequeña", formalmente los wavelets son un conjunto de funciones oscilatorias $\psi(t)$ (reales o complejas) continuas, cuadrado integrables, de soporte compacto (es decir, decaen rápidamente) y que pueden fungir como base para construir otras funciones.

Bajo la interpretación original de Morlet, matemáticamente este comportamiento se describe mediante la siguiente expresión:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right).$$

donde a es el factor de escala del wavelet, mientras que b representa un factor de traslación a lo largo del tiempo.



(a) Wavelet Haar. **(b)** Wavelet Morlet.

Figura 4.1: Ejemplos de Wavelets usados ampliamente.

Así, al descomponer la señal en los wavelets que la componen podemos **conocer de manera simultanea** la información temporal y secuencial que la compone en el mismo análisis.

4.2. Transformada continua de Wavelets

Para extraer las componentes wavelets que componen una señal empleamos una técnica conocida como **transformada continua de wavelets**, esta herramienta toma cono base un **wavelet madre** (que es la manera en que nos referimos a la función generadora) que se escalará y trasladará de manera continua obteniendo diferentes wavelets "hijos" los cuales de manera análoga a como sucede en **la transformada de Fourier** al ser una base nos entregan una representación de nuestra señal en la escala de tiempo y frecuencia [25].

Matemáticamente esta operación se define de la siguiente manera:

$$X_w(a,b) = \frac{1}{|a|^{1/2}} \int_{-\infty}^{\infty} x(t)\overline{\psi}\left(\frac{t-b}{a}\right) \mathrm{d}t$$

Donde a es el factor de escala del wavelet, b representa una traslación a lo largo del tiempo y $\overline{\psi}$ representa el dual de nuestra función. Mientras que **para recuperar nuestra señal original** la transformada correspondiente es:

$$x(t) = C_\psi^{-1} \int_0^{\infty} \int_{-\infty}^{\infty} X_w(a,b) \frac{1}{|a|^{1/2}} \tilde{\psi}\left(\frac{t-b}{a}\right) \mathrm{d}b \, \frac{\mathrm{d}a}{a^2}$$

donde C_ψ representa una constante que se calcula de la forma:

$$C_\psi = \int_{-\infty}^{\infty} \frac{\overline{\hat{\psi}}(\omega)\hat{\tilde{\psi}}(\omega)}{|\omega|} \mathrm{d}\omega$$

Tomemos como ejemplo la señal mostrada en la figura 4.2, que corresponde al producto de 2 funciones seno individuales donde su frecuencia y amplitud varían en el tiempo.

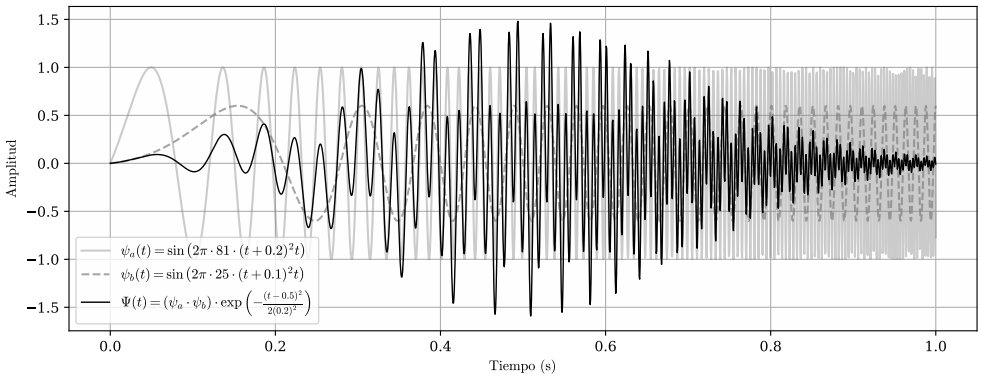


Figura 4.2: Señal compuesta de 2 señales con frecuencia creciente

Esperaríamos que al realizar nuestra transformada obtengamos de manera exacta los componentes temporales y frecuenciales de la señal como se muestra en la figura 4.3,

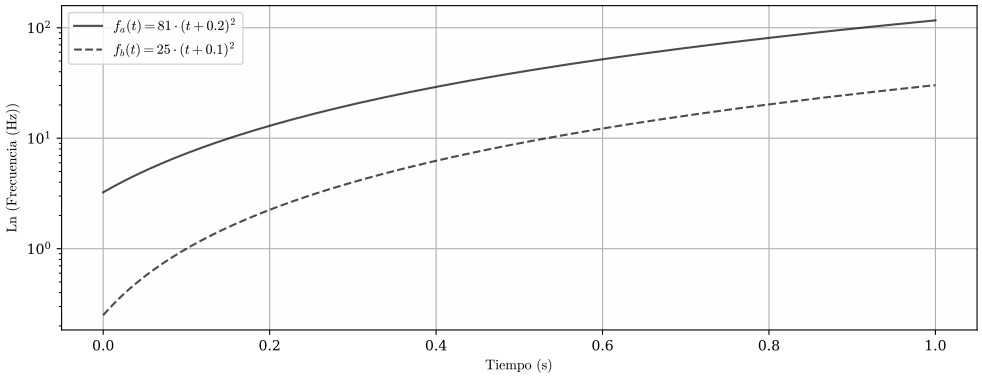


Figura 4.3: Gráfico de frecuencia-tiempo de la señal

Sin embargo en la práctica esto es muy dificl de obtener, principalmente por la relación que tiene este análisis con principio de incertidumbre de Heisenberg. Una escala grande en los wavelets ofrece buena resolución en frecuencia pero baja precisión en tiempo, mientras que una pequeña permite

una excelente localización temporal a expensas de una menor resolución en frecuencia, de esta forma en un análisis balanceado obtendremos una gráfica como la que podemos observar en la figura 4.4. Esta representación de los coeficientes resultantes de la transformada de wavelets es conocida

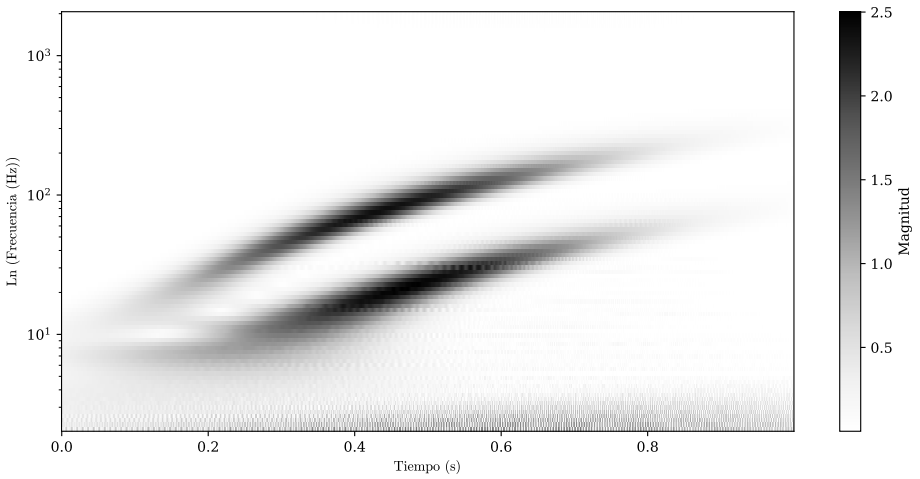


Figura 4.4: Escalograma de coeficientes de wavelet Morlet complejo aplicados a la señal de la figura 4.2, donde la escala representa la magnitud de las compponentes en el tiempo

como **escalograma** y contiene información acerca de las frecuencias en la señal, su ubicación y la magnitud con las que estas aparecen, notemos como en la figura 4.4 frecuencias mas bajas conllevan a una menor certeza temporal, mientras que entre mas aumenta la frecuencia esta distorsión va disminuyendo.

4.3. Transformada Discreta de Wavelets

Un aspecto importante a tratar en cualquier análisis de señales es el costo computacional que requiere realizarlo, dado que el costo computacional de realizar una transformada conitnua de wavelets es computacionalmente alto, la **transformada discreta de wavelets** (DWT por sus siglas en inglés) realiza el análisis en escalas y posiciones discretas, permitiendo una representación más compacta y eficiente de las señales.

Su principio fundamental consiste en aplicar de manera sucesiva filtros de baja y alta frecuencia a la señal original, dividiéndola en dos partes:

- **Coeficientes de aproximación** (L): capturan las componentes de baja frecuencia de la señal, representando su estructura global.
- **Coeficientes de detalle** (H): capturan las componentes de alta frecuencia, representando cambios rápidos o detalles finos.

En cada nivel de descomposición:

1. Se aplica una **transformada** a la señal, generando dos conjuntos: L_1 (baja frecuencia) y H_1 (alta frecuencia).
2. Se **copia** el conjunto de baja frecuencia L_1, que contiene la mayor parte de la información importante.
3. El proceso se repite recursivamente sobre L_1, generando nuevos conjuntos L_2, H_2, y así sucesivamente.

Este procedimiento jerárquico permite descomponer la señal en múltiples niveles de resolución, como se ilustra en la Figura 4.5.

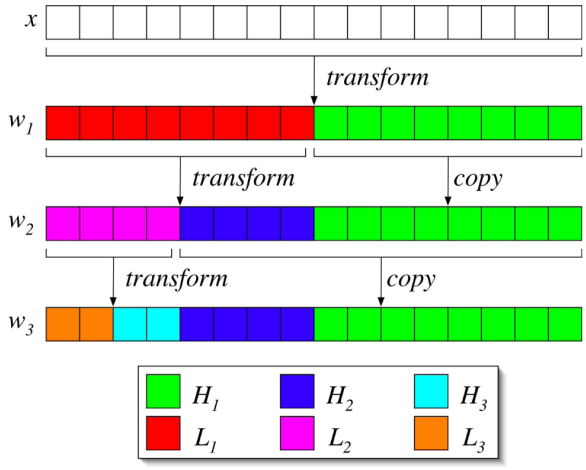


Figura 4.5: Esquema de la Transformada Discreta de Wavelets. En cada nivel, se aplica una transformación y se copia la parte de baja frecuencia para el siguiente nivel de análisis.

En términos matemáticos, para una señal discreta $x[n]$, los coeficientes de detalle $d[n]$ y los coeficientes de aproximación $a[n]$ se calculan mediante convolución y submuestreo:

$$d[n] = \sum_k x[k]g[2n-k] \quad \text{(detalle)} \qquad a[n] = \sum_k x[k]h[2n-k] \quad \text{(aproximación)}$$

donde $h[k]$ y $g[k]$ son los filtros de paso bajo y paso alto, respectivamente, y el submuestreo por un factor de 2 reduce la cantidad de datos a cada nivel.

Una vez más usemos la señal de la figura 4.2, al aplicar nuestra transformada discreta usando el wavelet discreto Daubechies con un nivel 4, obtenemos los coeficientes mostrados en la figura 4.6

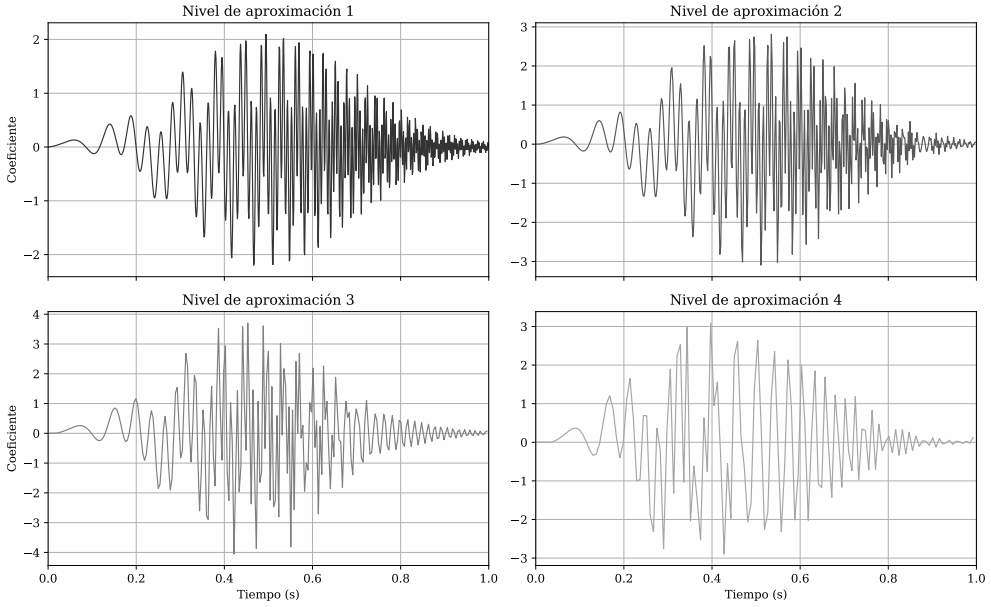
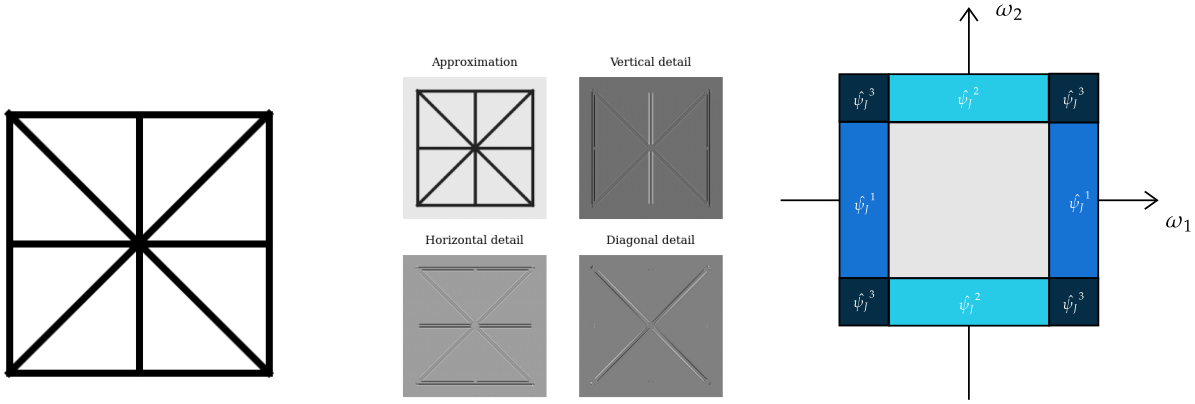


Figura 4.6: Coeficientes a cuarto nivel de transformada discreta de wavelets

4.3.1. Su aplicación en imágenes

Para una imagen bidimensional el proceso iterativo es similar al mostrado en la figura 4.5, en este caso la DWT implica un proceso en dos etapas: primero se aplica esta transformación en la dirección horizontal y luego en la dirección vertical. Como señala Mallat [26], si las funciones pasa bajos y pasa altos $\{\phi(t), \psi(t)\}$ generan una base ortonormal en $L^2(\mathbb{R})$, entonces se puede formar una base ortonormal en $L^2(\mathbb{R}^2)$. El resultado es una jerarquía de subimágenes que representan distintos niveles de detalle y aproximación. Esta descomposición produce cuatro tipos de coeficientes:



(a) Imagen a procesar **(b)** Resultado de la DWT. **(c)** Soporte en el espacio de Fourier.

Figura 4.7: Discrete Wavelet Transform en una imagen 2D

- **Coeficientes de Aproximación (LL):** Contienen la información de baja frecuencia de la imagen, representando la mayor parte de la energía visual (área gris central en la figura 4.7c).

$$\phi(x, y) = \phi(x)\phi(y)$$

- **Coeficientes de Detalle Horizontal (HL):** Capturan las características de alta frecuencia en la dirección horizontal (coeficientes $\hat{\psi}_J^2$ de la figura 4.7c).

$$\psi^{(1)}(x, y) = \psi(x)\phi(y)$$

- **Coeficientes de Detalle Vertical (LH):** Capturan las características de alta frecuencia en la dirección vertical (coeficientes $\hat{\psi}_J^1$ de la figura 4.7c).

$$\psi^{(2)}(x, y) = \phi(x)\psi(y)$$

- **Coeficientes de Detalle Diagonal (HH):** Representan las altas frecuencias que son combinaciones de direcciones horizontales y verticales (coeficientes $\hat{\psi}_J^3$ de la figura 4.7c).

$$\psi^{(3)}(x, y) = \psi(x)\psi(y)$$

De manera análoga al caso unidimensional, este proceso puede continuar realizando las mismas operaciones sobre las frecuencias bajas restantes de manera iterativa donde a cada nivel de resolución $j \in \mathbb{Z}$, estas funciones se dilatan y trasladan para construir las familias:

$$\begin{aligned} \phi_{j,n}(x, y) &= 2^j \phi(2^j x - n_1)\phi(2^j y - n_2), \\ \psi_{j,n}^{(i)}(x, y) &= 2^j \psi^{(i)}(2^j x - n_1, 2^j y - n_2), \quad i = 1, 2, 3, \end{aligned} \tag{4.1}$$

con $n = (n_1, n_2) \in \mathbb{Z}^2$. Bajo esta construcción, el conjunto completo de funciones $\{\phi_{j,n}, \psi^{(i)}_{j,n}\}$ forma una **base ortonormal** de $L^2(\mathbb{R}^2)$, es decir:

$$\langle \psi^{(i)}_{j,n}, \psi^{(i')}_{j',n'} \rangle = \delta_{j,j'}\delta_{n,n'}\delta_{i,i'}, \tag{4.2}$$

donde δ denota la delta de Kronecker. Esta ortogonalidad garantiza que la energía total de la imagen está preservada en los coeficientes (teorema de Parseval), que no existe redundancia, y que la transformada es perfectamente invertible.

En consecuencia, esta transformada ofrece una representación **esparsa**[1]**, eficiente y estructuralmente interpretable de imágenes** que contienen bordes, transiciones abruptas o texturas localizadas. Dicha representación permite no solo la compresión efectiva de datos visuales, sino también su análisis multiescala, extracción de características visuales y la supresión eficiente de ruido.

Por ejemplo, al aplicar algoritmos de cuantización o umbralización a los coeficientes de detalle, es posible obtener altos niveles de compresión sin degradar perceptiblemente la calidad de la imagen reconstruida [27, 28]. Además, diversos estudios han demostrado que el comportamiento estadístico de los coeficientes puede verse alterado por la presencia de ruido, lo cual debe considerarse especialmente en contextos de transmisión de datos o imágenes adquiridas en condiciones no ideales [29].

[1] La propiedad de *esparsidad* implica que la mayoría de los coeficientes generados por la transformada son insignificantes (cercanos a cero), mientras que solo una fracción reducida concentra la información más relevante de la señal visual

4.4. Transformada escalonada de Wavelets

Conocidas las técnicas anteriores, es natural preguntarse si existe alguna forma de lograr representaciones que sean estables ante pequeñas deformaciones, con bajo costo computacional y, al mismo tiempo conserven la estructura multiescala de la señal original. La **transformada escalonada de wavelets** o como nos referiremos de ahora en adelante **Wavelet Scattering Transform** (WST por su nombre en ingés) surge como una técnica que permite extraer representaciones invariantes y estables utilizando una cascada estructurada de convoluciones y no linealidades, manteniendo la información esencial de la señal.

Tiene una construcción jerárquica basada en la aplicación sucesiva de:

- **Convoluciones** con un banco de filtros wavelet.
- **No linealidades** (modulo) después de cada convolución.
- **Promedios** mediante un filtro de baja frecuencia.

De esta manera, captura información en distintas escalas, proporcionando una representación robusta a variaciones locales y deformaciones pequeñas. El procedimiento para realizar la WST se ilustra en la figura 4.8 y se organiza de manera jerárquica en niveles:

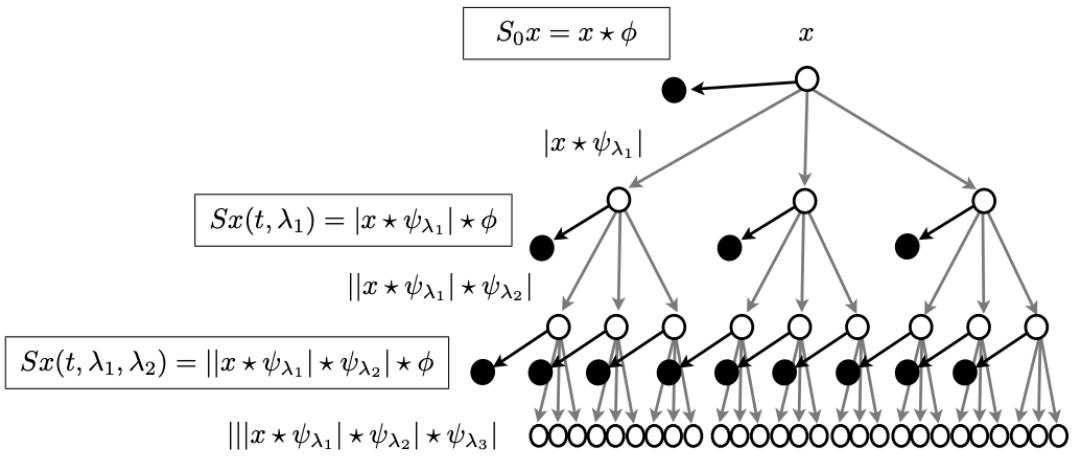


Figura 4.8: Esquema de la Wavelet Scattering Transform. Cada nivel consiste en una convolución, aplicación de módulo y promediado.

1. **Primer orden:** La señal original x se convolve (denotado con $\star$) con un conjunto de wavelets ψ_{λ_1}, se toma el módulo estabilizando los coeficientes ante deformaciones pequeñas, y posteriormente se realiza un promedio con un filtro de baja frecuencia gaussiano ϕ:

$$Sx(t, \lambda_1) = |x \star \psi_{\lambda_1}| \star \phi$$

2. **Segundo orden:** Se toman los coeficientes de primer orden $|x \star \psi_{\lambda_1}|$, se vuelven a convolver con nuevas wavelets ψ_{λ_2}, aplicando nuevamente el módulo y el promedio:

$$Sx(t, \lambda_1, \lambda_2) = ||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \star \phi$$

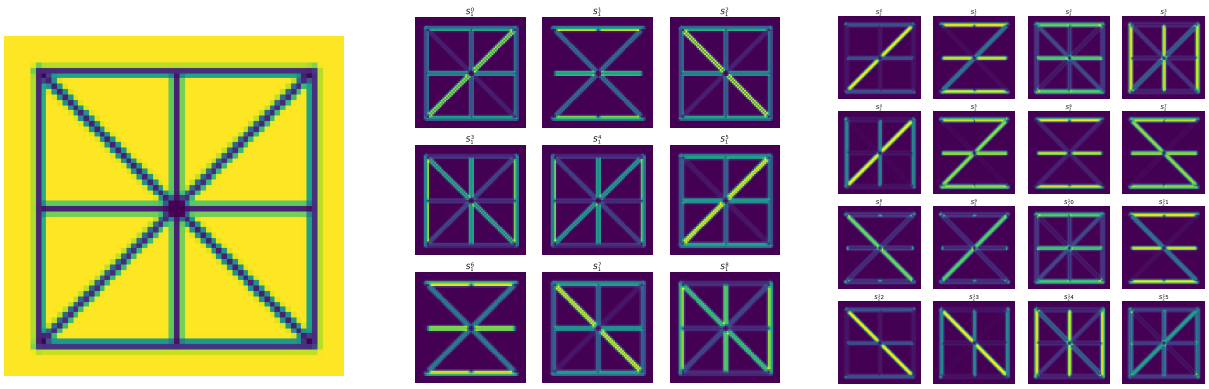
3. **Tercer orden y superiores:** El proceso puede repetirse sucesivamente, permitiendo capturar interacciones de frecuencias más complejas.

La estructura resultante se asemeja a la estructura de capas de una **red neuronal convolucional** con la diferencia de que en esta estructura los kernels de cada convolución están definidos y no necesita entrenamiento, dotando la arquitectura de una mayor interpretabilidad y control de los coeficientes resultantes.

4.4.1. Implementación y visualización

Para ilustrar el funcionamiento práctico de la **Wavelet Scattering Transform (WST)**, aplicamos el algoritmo de Mallat et al.[30] a una imagen de prueba. Utilizamos el paquete `Kymatio`, que implementa eficientemente la WST para señales 2D.

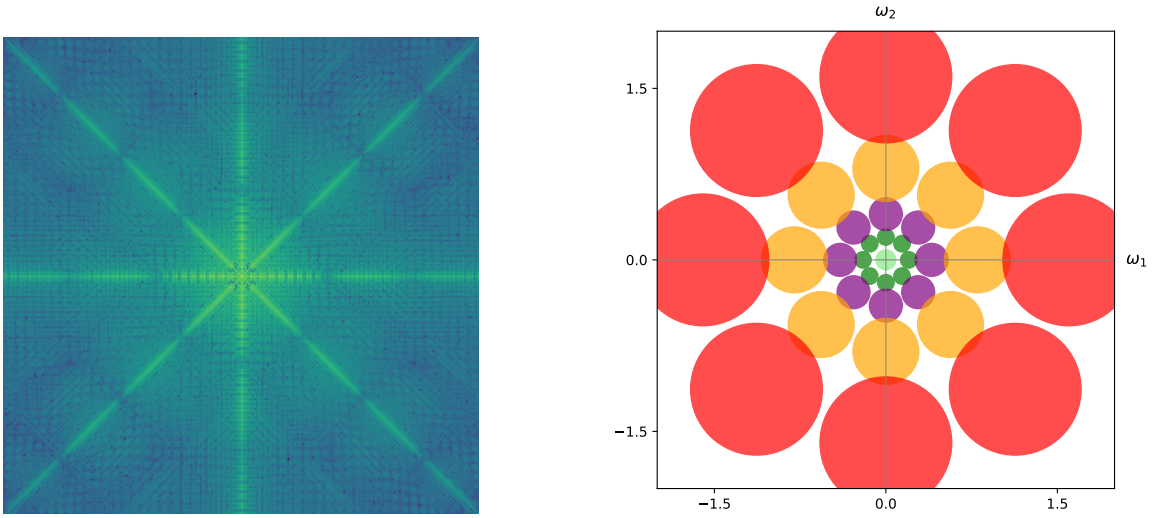
Partimos de la misma imagen mostrada en la figura 4.7a, a la que se le aplica una WST de orden 2 con $J = 2$ niveles de escala y $L = 5$ orientaciones por escala. Esto genera una colección de coeficientes $Sx(t, \lambda)$, donde cada $\lambda = (j, \theta)$ representa una escala y orientación.



(a) Coeficiente S^0 **(b)** Primeros coeficientes S^1 **(c)** Primeros coeficientes S^2

Figura 4.9: Resultados de aplicar la WST

En la Figura 4.10b, visualizamos algunos filtros $\psi_{j,\theta}$ y el filtro de paso bajo ϕ en el dominio de Fourier. Cada filtro ψ está localizado en una corona espectral y orientado angularmente, permitiendo una descomposición selectiva en escala y orientación.



(a) Transformada de Fourier (FFT) de imagen original **(b)** Primeros coeficientes S^1

Figura 4.10: K-Space resultante de la Transformada rápida de Fourier (FFT) de la figura 4.7a, junto con el soporte usado por la WST realizada en la figura 4.9

Es importante notar como en la figura 4.9 se observa que cada canal responde a características distintivas de la imagen: bordes, patrones periódicos u orientaciones dominantes. Estas características son invariantes por traslación y estables frente a deformaciones[30], lo que las hace ideales para tareas de clasificación, compresión o modelado estadístico.

4.4.2. Propiedades estadísticas de la Transformada Wavelet Scattering (WST)

La Transformada Wavelet Scattering (WST) es un marco robusto para crear representaciones de señales que son invariantes a traslaciones, estables ante deformaciones y capaces de preservar información de momentos estadísticos de orden superior.

Marco matemático

Dada una señal $x \in L^2(\mathbb{R}^d)$, la WST aplica iterativamente convoluciones con filtros pasabanda direccionales ψ_λ, seguidas por el módulo y un operador de promediado con filtro pasa bajas. Para un camino $p = (\lambda_1, \ldots, \lambda_m)$, el propagador de scattering se define como:

$$U[p]x(u) = |||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \cdots \star \psi_{\lambda_m}|$$

Para lograr invariancia por traslación, se aplica un filtro pasa bajas ϕ_{2^J} de escala 2^J:

$$S[p]x(u) = U[p]x \star \phi_{2^J}(u)$$

La representación completa de scattering es:

$$\mathcal{S}_J x = \{S[p]x(u)\}_{p \in \mathcal{P}_m,\, m \leq M}$$

Propiedades estadísticas

- **Invariancia por traslación:** Cada coeficiente $S[p]x(u)$ es invariante a traslaciones menores a la escala 2^J:
$$x'(u) = x(u - c) \Rightarrow S[p]x'(u) \approx S[p]x(u)$$
- **Estabilidad ante deformaciones:** La WST es Lipschitz continua respecto a difeomorfismos suaves [31]:
$$\|\mathcal{S}_J(L_\tau x) - \mathcal{S}_J(x)\| \leq C\|x\| \cdot \sup_u |\nabla \tau(u)|$$
- **Preservación de estadísticas de orden superior:** A diferencia del espectro de potencia, que solo captura correlaciones de segundo orden, la WST preserva interacciones complejas entre componentes frecuenciales [31]. Los coeficientes de primer orden actúan como detectores de energía:
$$S[\lambda_1](x) = \int |x \star \psi_{\lambda_1}(u)| \star \phi_J(u)\, du$$
Los coeficientes de segundo orden miden co-ocurrencias entre bandas, reflejando momentos de cuarto orden:
$$S[\lambda_1, \lambda_2](x) = \int ||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \star \phi_J(u)\, du$$
Estos permiten distinguir señales no gaussianas que serían indistinguibles con solo el espectro de potencia. Resultados empíricos confirman que la WST captura suficiente información para reconstruir distribuciones no gaussianas y estimar parámetros cosmológicos [32], ofreciendo una representación robusta y adaptable a las propiedades de datos reales.
- **Esperanza sobre procesos estacionarios:** Para un proceso estacionario X, el valor esperado de un coeficiente de scattering
$$\mathbb{E}[S[p]X(u)]$$
depende de los momentos estadísticos de orden superior de la distribución de X, permitiendo estimar estructuras no gaussianas de forma robusta.
- **Invertibilidad y retención de información:** Bajo condiciones suaves, la WST conserva suficiente información para reconstruir la señal (hasta una traslación global), y la energía de los coeficientes disminuye con el aumento del orden:
$$\text{energía}(S[p]x) \to 0 \quad \text{cuando } |p| \to \infty$$

Estas características hacen de la WST una representación estadística de **baja dimensión pero expresiva**, adecuada para clasificación, modelado generativo e inferencia física, como se demostró en [33] y [?].

4.4.3. Correlaciones a larga distancia y estructuras jerárquicas

En muchas señales físicas —como los campos de densidad cosmológicos— existen correlaciones que se extienden más allá de las escalas locales. Estas **correlaciones a larga distancia** no son explicables únicamente a través de estadísticas locales o de segundo orden (como el espectro de potencia), sino que implican interacciones entre estructuras multiescala: cúmulos, filamentos y vacíos que coexisten y se correlacionan espacialmente.

El trabajo de Cheng et al.[32] y la tesis doctoral de Bruna[31] demuestran que representaciones jerárquicas como la **Wavelet Scattering Transform** (WST) permiten capturar estas correlaciones a gran escala gracias a su estructura multirresolutiva y su estabilidad frente a deformaciones. En particular, Cheng et al. muestran que los coeficientes de scattering retienen información suficiente para modelar correlaciones espaciales de largo alcance, sin necesidad de reconstruir fases explícitas.

Justificación matemática

Sea $x(\mathbf{u})$ una realización de un campo físico (por ejemplo, la densidad de materia oscura). La correlación de segundo orden está dada por:

$$C_2(\mathbf{r}) = \mathbb{E}[x(\mathbf{u})x(\mathbf{u}+\mathbf{r})]$$

Sin embargo, para campos no gaussianos, es necesario incorporar correlaciones de orden superior:

$$C_n(\mathbf{r}_1, \ldots, \mathbf{r}_{n-1}) = \mathbb{E}[x(\mathbf{u})x(\mathbf{u}+\mathbf{r}_1)\cdots x(\mathbf{u}+\mathbf{r}_{n-1})]$$

Estas correlaciones de orden alto pueden mantenerse no nulas a distancias largas si el campo presenta una **estructura jerárquica persistente**. Como muestran Bruna y Mallat, los coeficientes de la WST actúan como estadísticos invariables que retienen dependencias multiescala entre bandas espaciales y frecuenciales, lo cual es clave para caracterizar procesos físicos no gaussianos.

Arquitecturas convolucionales jerárquicas

Las redes convolucionales profundas (CNN), especialmente aquellas con bloques residuales y estructuras jerárquicas multiescala, son herramientas poderosas para capturar dependencias espaciales globales. Esta capacidad proviene del crecimiento progresivo del *receptive field* con la profundidad de la red, lo cual permite que capas superiores respondan a patrones estructurales a escalas cada vez mayores.

Una capa convolucional típica puede expresarse como:

$$f^{(l)} = \sigma\left(W^{(l)} * f^{(l-1)} + b^{(l)}\right),$$

donde:

- $f^{(l)} \in \mathbb{R}^{C\times H\times W}$ es la activación de la capa l,
- $W^{(l)}$ representa los filtros convolucionales,
- $b^{(l)}$ son los sesgos,
- σ es una función de activación no lineal.

A medida que la señal atraviesa capas sucesivas, el campo receptivo crece de manera exponencial, facilitando la integración de información local y global. Este principio es fundamental en arquitecturas profundas como VGGNet[34], donde la composición de convoluciones pequeñas y profundas da lugar a filtros jerárquicos altamente especializados, o en redes como ResNet[35], donde los bloques residuales permiten entrenar redes muy profundas sin degradación de la señal.

Además, se ha demostrado que unidades individuales en redes profundas aprenden representaciones semánticas complejas que son consistentes a través de inicializaciones distintas[36]. Este hallazgo

indica que las CNNs pueden aprender funciones $\hat{C}_n$ que aproximan correlaciones estadísticas de orden alto, haciendo posible modelar interacciones no lineales de largo alcance, similares a las capturadas por funciones de correlación de orden superior en física estadística.

Este tipo de representación distribuida y jerárquica es especialmente útil en problemas como la reconstrucción de estructuras cosmológicas no gaussianas o la caracterización de patrones complejos en imágenes médicas, donde las correlaciones simples no bastan para capturar la información relevante del sistema.

4.5. Modelado probabilístico mediante la WST y resultados recientes

Una de las aplicaciones más destacadas de la Wavelet Scattering Transform (WST) es su papel como **base estadística** para construir modelos probabilísticos de campos físicos altamente no gausianos. En particular, Cheng et al. [37] introducen el concepto de *scattering spectra* como una forma eficiente y robusta de estimar el espacio de probabilidad de campos estacionarios, usando una versión reducida y regularizada de los coeficientes de la WST.

4.5.1. Modelo de Gibbs con estadística WST

El marco teórico se basa en la construcción de modelos de Gibbs definidos por:

$$p_\theta(x) = \frac{1}{Z_\theta} \exp(-\langle \theta, \Phi(x) \rangle) \tag{4.3}$$

donde:

- $x \in \mathbb{R}^{L^d}$ es un campo aleatorio donde el índice de sitio u pertenece a una red cúbica d-dimensional de tamaño L.
- $\Phi(x) \in \mathbb{R}^M$ es un vector de estadísticas potenciales, de dimensión M.
- $\theta = \{\theta_m\}_{m \leq M}$ son los parámetros conjugados.
- Z_θ es la función de partición que asegura la normalización.

El elemento central del enfoque propuesto por Cheng et al. [?] radica en la elección del funcional estadístico $\Phi(x)$. En lugar de utilizar momentos tradicionales del campo x, los autores proponen emplear el **vector de espectros de scattering**, definido como:

$$\Phi(x) = S(x) = (S_1(x), S_2(x), S_3(x), S_4(x)), \tag{4.4}$$

donde cada componente corresponde a una familia de momentos de distinto orden construidos a partir de la transformada wavelet compleja Wx. Específicamente:

- $S_1(x)[\lambda] = \text{Ave}_u \left[|Wx[u, \lambda]| \right]$: estimador del valor esperado del módulo de los coeficientes wavelet, asociado a estadísticas de primer orden.
- $S_2(x)[\lambda] = \text{Ave}_u \left[|Wx[u, \lambda]|^2 \right]$: estimador del espectro de potencias de segundo orden.
- $S_3(x)[\lambda, \lambda'] = \text{Ave}_u \left[|Wx[u, \lambda]| \cdot Wx^*[u, \lambda'] \right]$: correlaciones cruzadas de tercer orden entre escalas y orientaciones distintas.
- $S_4(x)[\lambda, \lambda', \gamma] = \text{Ave}_u \left[W|Wx|^2[u, \lambda, \gamma] \cdot W|Wx|^2 * [u, \lambda', \gamma] \right]$: estimador de correlaciones de cuarto orden de la energía entre pares de bandas.

Este vector $S(x)$ forma una representación jerárquica y multiescala del campo x, que preserva estadísticas de alto orden relevantes para modelar estructuras no gaussianas. Notablemente, la dimensión total del vector crece de forma logarítmica con la resolución del campo:

$$\dim S(x) = \mathcal{O}(\log^4 L),$$

lo que permite una compresión efectiva sin pérdida significativa de información estadística.

4.5.2. Justificación teórica y conexión con entropía máxima

Este enfoque es consistente con el principio de **entropía máxima** de Jaynes, que postula que la mejor estimación para una distribución $p(x)$ bajo restricciones $\mathbb{E}[\Phi(x)]$ es aquella que maximiza la entropía sujeta a esas restricciones. La WST proporciona una base natural para este tipo de modelado, pues:

- Captura información multiescala y no gaussiana.
- Es invariante a traslaciones y estable frente a deformaciones pequeñas.
- Proporciona estadísticas de baja varianza y alta interpretabilidad.

Esto permite construir modelos eficientes incluso con pocos ejemplos, mediante versiones *microcanónicas* que restringen las muestras a cumplir aproximadamente $\|\Phi(x) - \Phi(\bar{x})\|^2 \leq \varepsilon$. Formalmente, el conjunto microcanónico se define como:

$$\Omega_\varepsilon = \left\{ x \in \mathbb{R}^{L^d} : \|\Phi(x) - \Phi(\tilde{x}_1)\|^2 \leq \varepsilon \right\} \tag{4.5}$$

Para $n > 1$ muestras $\tilde{x}_i$, la definición de Ω_ε se extiende reemplazando $\Phi(\tilde{x}_1)$ por $Ave_i \Phi(\tilde{x}_i)$. Si $\Phi(x)$ se concentra alrededor de $\mathbb{E}\{\Phi(x)\}$, el modelo microcanónico converge al modelo macrocanónico cuando la longitud del sistema L tiende a infinito y ε tiende a 0.

Síntesis con la WST

El uso combinado de WST y CNN explota lo mejor de ambos mundos:

- La WST proporciona un conjunto de características multiescala interpretables y estables, que actúan como *descriptores globales*.
- La CNN residual captura relaciones espaciales jerárquicas y permite predecir *correcciones locales* basadas en contexto multiescala.

Este enfoque integrado permite modelar el campo x como una muestra generada a partir de una distribución no gaussiana definida tanto por características locales como globales:

$$x \sim p(x) \propto \exp\left(-\langle \theta, \Phi(x) \rangle - \mathcal{E}_\psi(x)\right)$$

donde:

- $\Phi(x)$ es el vector de coeficientes de WST (macroestructura),
- $\mathcal{E}_\psi(x)$ es la energía aprendida mediante CNN (estructura jerárquica local),
- y el modelo actúa como una forma de **modelo energético híbrido** entre estadística explícita e inferencia neuronal.

4.5.3. Aplicación al modelo generador

Este marco teórico respalda el uso de la WST como entrada estadística en modelos generativos como el propuesto en esta memoria. Al entrenar una red residual para predecir $\hat{x}_{\text{HR}} = x_{\text{UP}} + \Delta x$, se puede imponer que los coeficientes $\Phi(\hat{x})$ sean similares a $\Phi(x)$, integrando así un conocimiento estadístico y físico en la tarea de super-resolución, alineado con la distribución de Gibbs:

$$\min_\psi \left\{ \|\hat{x}_\psi - x\|^2 + \lambda \cdot \|\Phi(\hat{x}_\psi) - \Phi(x)\|^2 \right\}$$

Este término regulariza el espacio de soluciones para que las correcciones aprendidas sean estadísticamente coherentes con la física del campo subyacente.

5 Principal Component Analysis

En el mundo contemporáneo, el análisis de los datos de entrada que damos a nuestros modelos de inferencia estadística es crucial no solo para un mejor entrenamiento, sino que es imprescindible para dotarlo de una adecuada interpretación; a menudo esta tarea se vuelve titánica dada la dimension del espacio de parámetros. Para resolver este problema surgieron los algoritmos de reducción de dimensionalidad; estos buscan proyectar nuestros datos en un espacio dimensional más compacto preservando la mayor cantidad de información posible, permitiéndonos buscar patrones o inferir correlaciones [38]. Es así que el Análisis de Componentes Principales (PCA, por sus siglas en inglés) surge como técnica estadística para la reducción de dimensionalidad, la cual a diferencia de otros métodos de reducción, busca las direcciones de máxima varianza, permitiendo representar mediante una proyección los datos en un nuevo sistema de coordenadas más compacto [39].

5.1. Definición

Formalmente, PCA transforma un conjunto de datos correlacionados en un nuevo conjunto de variables no correlacionadas llamadas **componentes principales**. Cada componente principal es una combinación lineal de las variables originales y representa una dirección de máxima varianza dado que estas preservan la mayor cantidad de información en los datos [38].

Matemáticamente, dado un conjunto de datos $X \in \mathbb{R}^{n \times p}$, donde n es el número de muestras y p el número de variables, el procedimiento para aplicar PCA es el siguiente:

1. **Estandarización de los datos:** Centrar los datos restando la media de cada variable.
2. **Cálculo de la matriz de covarianza:**
$$\mathbf{C} = \frac{1}{n-1}\mathbf{X}^\top \mathbf{X}$$
3. **Obtención de eigenvectores y eigenvalores:** Resolver el problema de autovalores
$$\mathbf{C}\mathbf{v} = \lambda \mathbf{v}$$
donde λ son los eigenvalores (varianzas explicadas) y $\mathbf{v}$ los eigenvectores (direcciones principales).
4. **Ordenamiento:** Clasificar los eigenvectores en orden descendente según sus eigenvalores.
5. **Proyección:** Transformar los datos originales sobre el subespacio generado por los primeros k componentes:
$$\mathbf{Z} = \mathbf{X}\mathbf{V}_k$$
donde $\mathbf{V}_k$ contiene los primeros k eigenvectores de manera ordenada.

Cada componente principal puede interpretarse como una nueva dimensión que captura una parte de la variabilidad total de los datos. El primer componente explica la mayor cantidad de varianza posible, el segundo explica la mayor cantidad de varianza restante, y así sucesivamente.

Es importante recalcar que este método al emplear principalmente **operaciones matriciales** tiene un costo computacional sustancialmente menor con respecto a otros métodos de reducción de dimensionalidad, haciendo PCA la opción mas popular para este propósito, sin embargo esto viene con una limitación importante: **PCA es un método que asume relaciones lineales**, lo que significa que para relaciones no lineales la interpretación de la conexión entre los componentes puede ser compleja; para solucionar esto existen extensiones como **Kernel PCA** o técnicas contemporáneas como **Uniform Manifold Approximation (UMAP)**.

5.2. Ejemplo de uso

Analicemos un ejemplo sencillo para ilustrar el funcionamiento de esta técnica. Supongamos que en un experimento de mecánica queremos reconstruir la trayectoria de una partícula que se desplaza a velocidades tan elevadas que a simple vista resulta imposible seguir su recorrido como se muestra en la figura 5.1. Para ello disponemos de tres cámaras que, en distintos instantes de tiempo capturan la posición de la partícula.

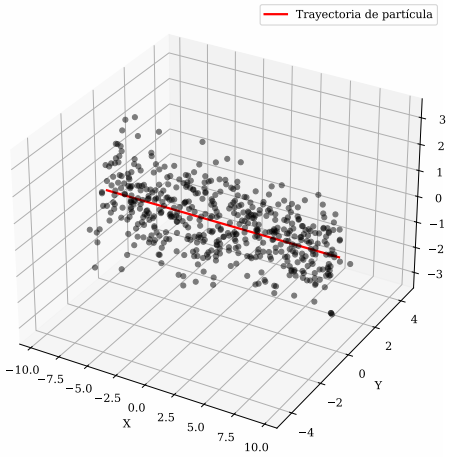
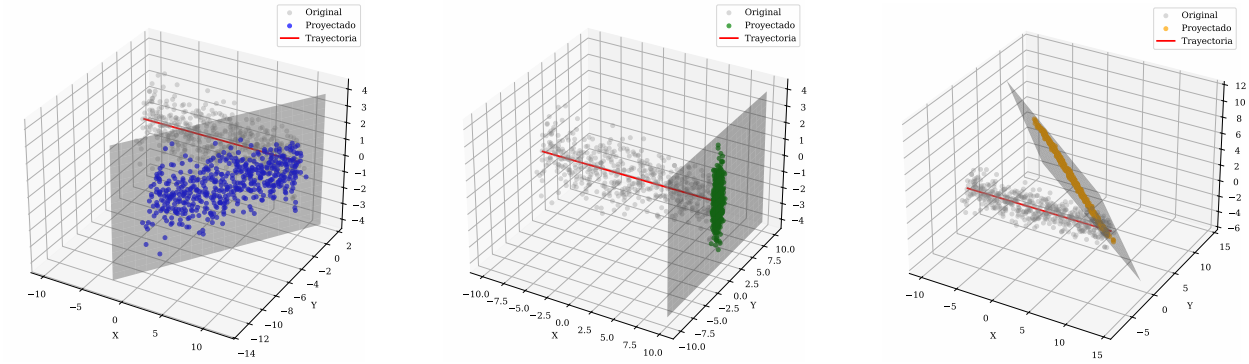


Figura 5.1: Diferentes posiciones de movimiento de la partícula con incertidumbre asociada (500 muestras)

Dado que desconocemos por completo la trayectoria real, tampoco podemos fijar de manera inicial un sistema de ejes coordenados común (notemos como la orientación y ubicación de las cámaras es irrelevante en este análisis). Cada cámara, entonces, nos proporciona un conjunto de datos correspondiente a las proyecciones de la posición de la partícula sobre su plano de visión. Este proceso de captura y proyección se ilustra en la figura 5.2.

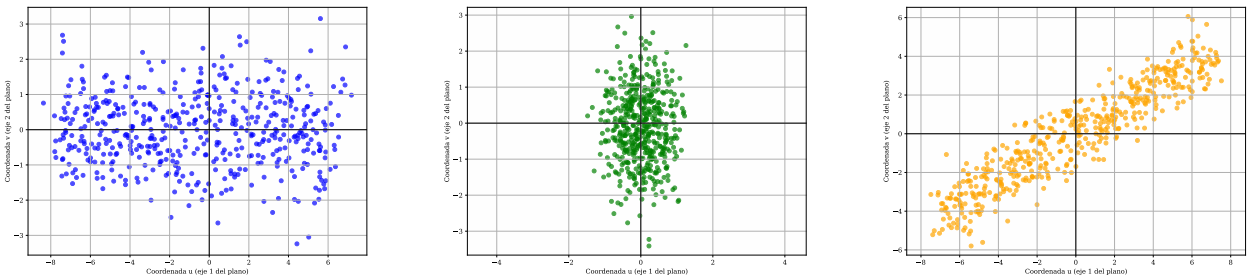
De este modo, a primera vista podría parecer que, al registrar la trayectoria de la partícula con tres cámaras, estamos obligados a trabajar en un espacio de datos de seis dimensiones (dos coordenadas por cámara). Sin embargo, esta situación es una oportunidad perfecta para aplicar el Análisis de Componentes Principales (PCA). Al reducir las dimensiones de nuestro conjunto de datos, podremos llevar a cabo un análisis más claro, eficiente y robusto.



(a) Cámara A **(b)** Cámara B **(c)** Cámara C

Figura 5.2: Diferentes planos de captura para cada cámara.

Comencemos por analizar las proyecciones individuales de cada cámara mostradas en la figura 5.3



(a) Plano cámara A **(b)** Plano cámara B **(c)** Plano cámara C

Figura 5.3: Proyecciones de los datos de posiciones para cada cámara.

Notemos como las distribuciones de las posiciones en los distintos planes focales es completamente distinta para cada caso, lo que a primera vista nos haría pensar que no están relacionadas, obteniendo así para cada posición un vector de datos $\vec{X_n}$ con 6 entradas (las coordenadas de cada cámara) en un espacio de 6 dimensiones.

$$\vec{X_n}^{\mathrm{T}} = \begin{bmatrix} x_A & y_A & x_B & y_B & x_C & y_C \end{bmatrix}$$

De nuestros conocimientos de álgebra lineal sabemos que cualquier punto de nuestro conjunto de datos es una combinación lineal de una base ortonormal, lo que nos lleva a preguntarnos ¿Existe alguna base que nos facilite el análisis de nuestros datos?, precisamente **esta pregunta es la que responde PCA**.

Para nuestro análisis usaremos el lenguaje de programación *python*, en conjunto con las biblioteca *Scikit-learn* [40]; el notebook con el código completo se puede encontrar en esta dirección.

Almacenamos los 500 vectores de posición en un arreglo de `NumPy`, el cual será utilizado como entrada de datos. Por defecto `Scikit-learn`, estandariza los datos antes de aplicar el análisis. Al utilizar `PCA()` ingresamos el número de dimensiones en nuestro espacio, donde obtenemos una nueva base ortogonal sobre la cual podemos representar los datos, lo que permite llevar a cabo una reducción de dimensionalidad que busacmos.

```
data = positions.T            # Carga de datos
pca = PCA(n_components=6)     # Definimos PCA con 6 componentes
pca_result = pca.fit_transform(data)            # Ejecutamos PCA
explained_var = pca.explained_variance_ratio_   # Rendimiento de componentes
```

De esta forma obtenemos que para cada nuevo vector ortonormal en nuestro ejemplo, la varianza (de mayor a menor) está dada por:

$$Var(\vec{X}) = [\quad 0.935 \quad 0.0573 \quad 0.007 \quad 0. \quad 0. \quad 0. \quad]$$

Podemos observar de mejor manera este resultado en la figura 5.4 donde podemos comprobar cómo PCA no solo funge como técnica de reducción de dimensionalidad, sino que al mismo tiempo nos permite identificar el número de componentes relevantes para representar un conjunto de datos.

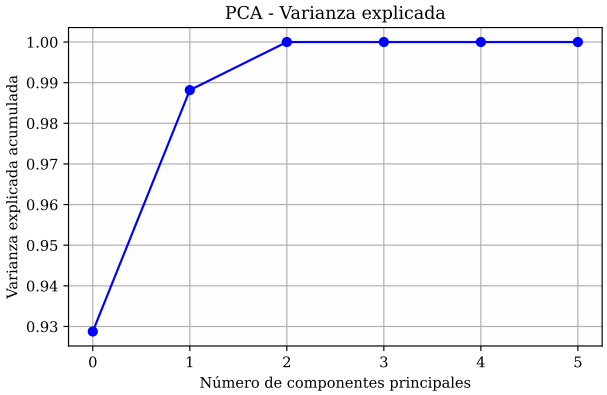
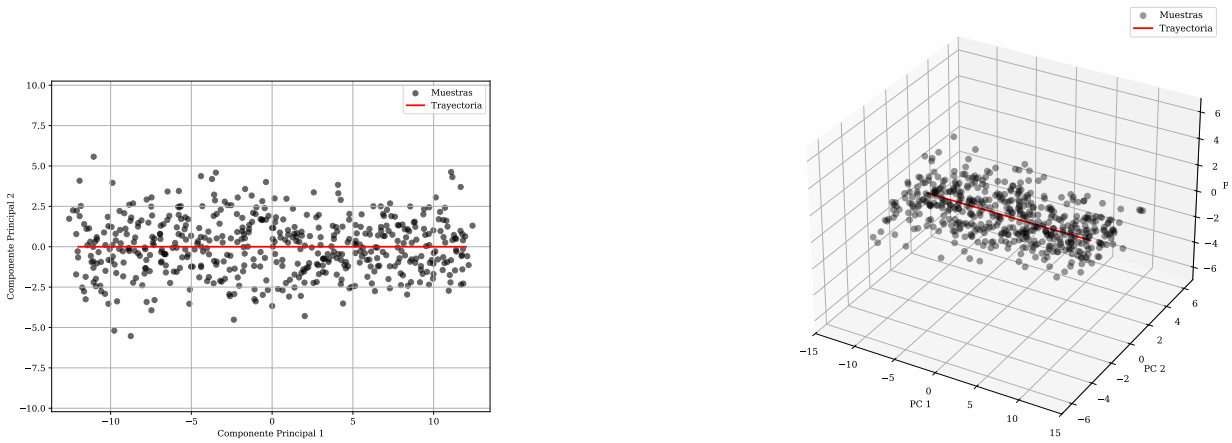


Figura 5.4: Evolución de Varianza por número de componentes

Así solo resta proyectar nuestros datos en el numero de dimensiones que deseemos, donde podemos extraer esta proyección y graficarlos en un diagrama de dispersión (figura 5.5).

```
# Dos dimensiones
plt.scatter(pca_result[:, 0],pca_result[:, 1],alpha=0.6,c='black',label='Muestras')
# Tres dimensiones
ax.scatter(pca_result[:, 0], pca_result[:, 1], pca_result[:, 2],
           alpha=0.4, c='black', s=40, label='Muestras')
```



(a) Proyección 2 componentes **(b)** Proyección 3 componentes

Figura 5.5: Diagramas de dispersión de diferentes proyecciones sobre componentes principales

Con esto podemos observar como reconstruimos la trayectoria original de nuestra partícula como la mostrada en la figura 5.1, es así como de un muestro de 6 variables sin correlación aparente podemos reconstruir y analizar los datos originales.

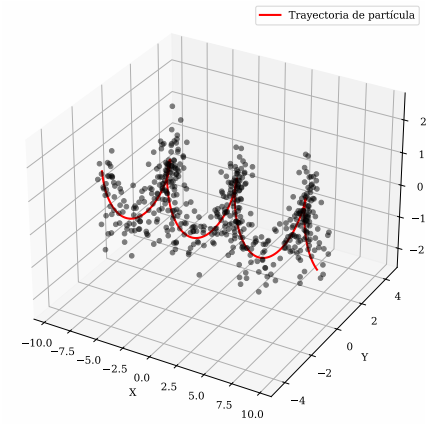
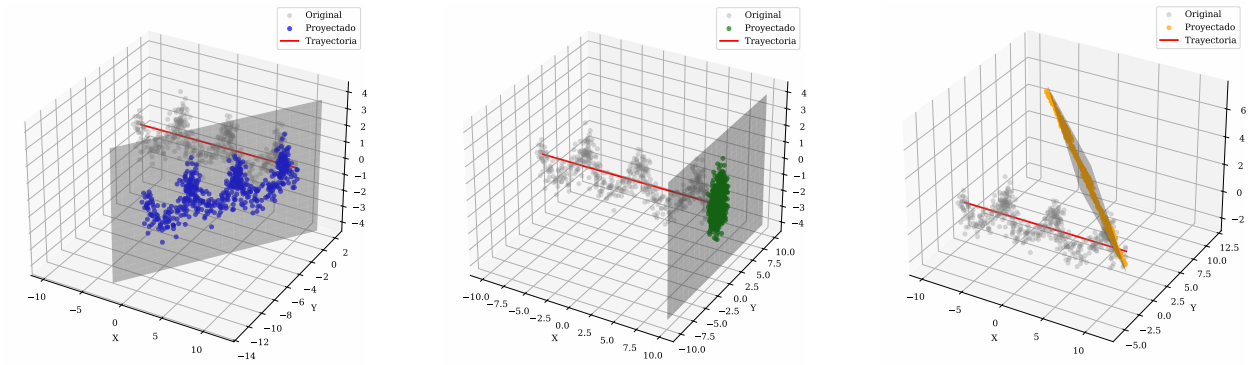


Figura 5.6: Trayectoria en espiral

Es importante notar que para datos que no presentan relaciones perfectamente lineales como la trayectoria mostrada en la figura 5.6, aún podemos implementar PCA para darnos una idea general de los datos que estamos analizando, permitiéndonos implementar otras técnicas si es necesario, eliminar datos irrelevantes o simplemente compactar nuestra representación en un numero de componentes con una varianza que nos resulte útil, tal es el caso del area de **Machine Learning** donde un menor numero de parámetros de entrada conducen a menores tiempos de entrenamiento y una mejor interpretabilidad del modelo.

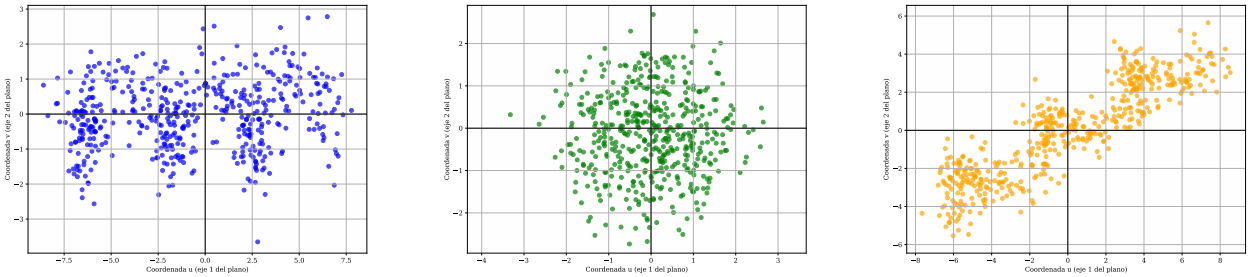
Podemos observar en la figura 5.8 como las proyecciones capturadas por cada cámara tienen un comportamiento muy diferente dependiendo de la orientación de la camara.



(a) Cámara A **(b)** Cámara B **(c)** Cámara C

Figura 5.7: Diferentes planos de captura para cada cámara.

Al analizar las proyecciones mostradas en la figura 5.8, de manera directa podemos notar cómo el ruido y la orientación toman un papel crucial en las proyecciones.



(a) Cámara A **(b)** Cámara B **(c)** Cámara C

Figura 5.8: Diferentes planos de captura para cada cámara.

Aplicando de nuevo PCA como en el ejemplo anterior, obtenemos una gráfica de desempeño por componentes.

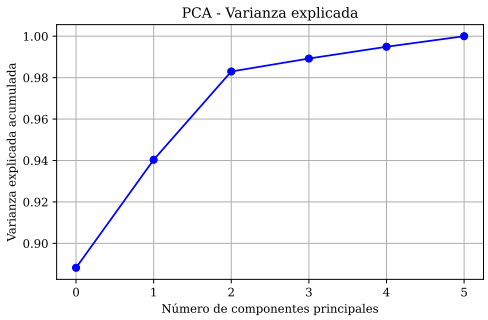
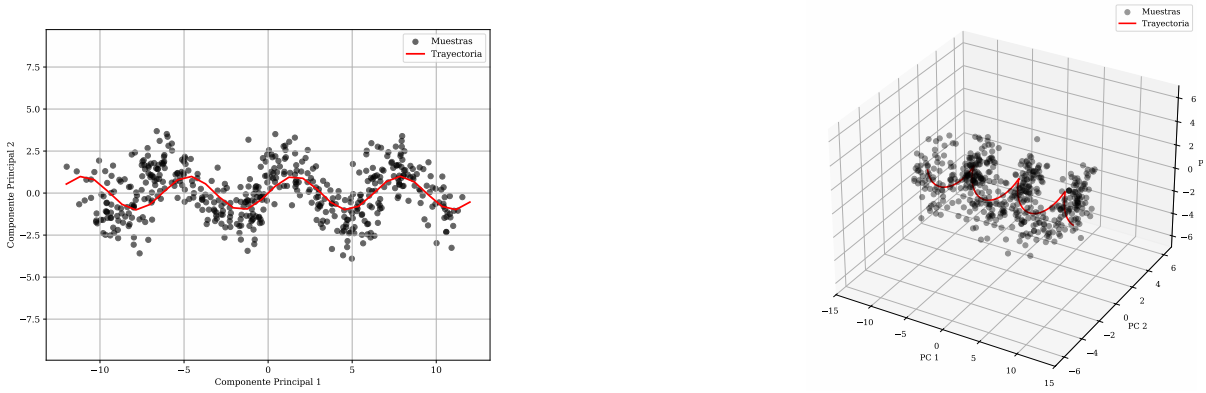


Figura 5.9: Evolución de Varianza por número de componentes

Donde podemos observar cómo, a diferencia de la figura 5.4, la varianza por número de componentes no se maximiza en 3 dimensiones, sino que está crece de manera continua hasta alcanzar el total de los datos.



(a) Proyección 2 componentes **(b)** Proyección 3 componentes

Figura 5.10: Diagramas de dispersión de diferentes proyecciones sobre componentes principales

Así, podemos resaltar que una de las partes más cruciales de la reducción de la dimensionalidad en general recae en la interpretación de los componentes necesarios para una adecuada interpretación.

Referencias

[1] Anagoria. Sternenliste aus uruk (star list from uruk). `https://commons.wikimedia.org/wiki/File:-200_Sternenliste_aus_Uruk_star_list_anagoria.JPG`, 2009. Licensed under CC BY 3.0 via Wikimedia Commons.

[2] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G. R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *arXiv preprint arXiv:1803.08823*, 2018.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2nd edition, 2022.

[5] F. Chollet and the Asimov Institute. The neural network zoo. `https://www.asimovinstitute.org/neural-network-zoo/`, 2017. Accessed: 2024-05-27.

[6] DeepMind and EMBL-EBI. Alphafold protein structure database. `https://alphafold.ebi.ac.uk/`, 2021. Accessed: 2024-05-27.

[7] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[8] Aliki D. Mouratidou, Georgios A. Drosopoulos, and Georgios E. Stavroulakis. Ensemble of physics-informed neural networks for solving plane elasticity problems with examples. *Acta Mechanica*, 235:6703–6722, 2024.

[9] Majid Rasht-Behesht, Christian Huber, Khemraj Shukla, and George Em Karniadakis. Physics-informed neural networks (pinns) for wave propagation and full waveform inversions. *arXiv preprint arXiv:2108.12035*, 2021.

[10] Thi Nguyen Khoa Nguyen, Thibault Dairay, Raphaël Meunier, and Mathilde Mougeot. Physics-informed neural networks for non-newtonian fluid thermo-mechanical problems: an application to rubber calendering process. *arXiv preprint arXiv:2201.13389*, 2022.

[11] Enrico Schiassi, Roberto Furfaro, Mario De Florio, and Barry D. Ganapol. Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics. *Annals of Nuclear Energy*, 170:108743, 2022.

[12] Hideki Tanimura, Jia Liu, Albert Bonnefous, and Sanmay Ganguly. Velocity reconstruction with graph neural networks. *arXiv preprint arXiv:2402.14239*, 2024.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Saturn Cloud. A cnn sequence to classify handwritten digits. `https://saturncloud.io/images/blog/a-cnn-sequence-to-classify-handwritten-digits.webp`, 2022. Accessed: 2024-05-27.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[17] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141, 2018.

[19] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[20] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[21] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4681–4690, 2017.

[22] Jianze Li, Jiezhang Cao, Zichen Zou, Xiongfei Su, Xin Yuan, Yulun Zhang, Yong Guo, and Xiaokang Yang. Unleashing the power of one-step diffusion based image super-resolution via a large-scale diffusion discriminator (d3 sr). *NeurIPS*, 2025.

[23] Rongyuan Wu, Lingchen Sun, Zhiyuan Ma, and Lei Zhang. One-step effective diffusion network for real-world image super-resolution (seesr). In *CVPR*, 2024.

[24] Jianyi Wang, Zongsheng Yue, Shangchen Zhou, Kelvin C. K. Chan, and Chen Change Loy. Exploiting diffusion prior for real-world image super-resolution. *IJCV*, 2024.

[25] Christopher Torrence and Gilbert P. Compo. A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, 79(1):61–78, 1998.

[26] Stéphane Mallat. *A Wavelet Tour of Signal Processing: The Sparse Way*. Academic Press, Burlington, MA, 3rd edition, 2009.

[27] M. Rajmohan and C. Chandrasekar. Performance analysis of image compression using spiht and wdr algorithms. In *2015 International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–5. IEEE, 2015.

[28] Ali M. Reza. An improved wavelet-based image compression algorithm using threshold optimization. *Engineering and Technology Journal*, 28(7):2, 2010.

[29] Fajar Pradana, Bambang Irawan, and Retantyo Wardoyo. Sensitivity analysis of wavelet transform in image transmission over noisy channels. *TELKOMNIKA Telecommunication Computing Electronics and Control*, 14(2):482–490, 2016.

[30] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *arXiv preprint arXiv:1203.1513*, 2012.

[31] Joan Bruna. *Scattering Representations for Recognition*. PhD thesis, École Normale Supérieure, 2013.

[32] Sihao Cheng, Jeffrey Regier, Stéphane Mallat, and Cora Dvorkin. Multiscale statistical analysis of non-gaussian fields using wavelets. *arXiv preprint arXiv:2306.17210*, 2023.

[33] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *Ieee Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[36] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 2009. Technical Report 1341.

[37] Sihao Cheng, Jeffrey Regier, Cora Dvorkin, and Stéphane Mallat. Scattering spectra models for physics. *arXiv preprint arXiv:2306.17210*, 2023.

[38] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Sebastopol, CA, 2 edition, 2019.

[39] Jonathon Shlens. A tutorial on principal component analysis. `https://arxiv.org/abs/1404.1100`, 2005. Version 2.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.