Influencia de la selección de componentes principales en el aprendizaje computacional

Influence of Principal Component Selection on Computational Learning

Juan Pérez Salido

Grado en Ingeniería en Tecnologías Industriales
Trabajo Fin de Grado

Tutorizado por Dr. Miguel Alejandro Atencia Ruiz

Cotutorizado por Dr. Iván Gómez Gallego

UNIVERSIDAD DE MÁLAGA

Universidad de Málaga

octubre 2025

Publicado en octubre 2025 por
Juan Pérez Salido

A mis padres.

Resumen

El uso de técnicas de reducción de dimensionalidad, como el Análisis de Componentes Principales (PCA), supone una herramienta útil para optimizar el entrenamiento de modelos de aprendizaje automático, especialmente en entornos con limitaciones computacionales. Los conjuntos de datos de alta dimensión, como los de imágenes, contienen mucha información redundante, lo que puede afectar negativamente al rendimiento y eficiencia de los modelos de redes neuronales.

En este proyecto se ha desarrollado un estudio comparativo sobre la influencia del número de componentes seleccionados mediante PCA en el rendimiento de una red neuronal multicapa. Para ello, se han utilizado tres conjuntos de datos conocidos en el ámbito de la clasificación de imágenes: MNIST, Fashion MNIST y Digits. El entrenamiento del modelo se ha realizado en Google Colab utilizando Python y bibliotecas como TensorFlow, Keras y Scikit-learn.

Se han aplicado varios criterios automáticos para determinar el número óptimo de componentes principales: el criterio de Kaiser, el umbral de varianza explicada (95 por ciento) y el análisis paralelo de Horn. Además, se han evaluado configuraciones con la mitad de las características y con la totalidad de los datos originales. Cada configuración se ha entrenado y evaluado de forma sistemática para comparar su precisión en el conjunto de prueba.

Los resultados obtenidos muestran que es posible reducir de manera significativa el número de variables de entrada sin comprometer la precisión del modelo. En algunos casos, la reducción incluso mejora el rendimiento gracias a la eliminación de ruido. Esto demuestra que una buena selección del número de componentes puede aumentar la eficiencia del modelo sin afectar negativamente su capacidad predictiva.

Palabras clave: Aprendizaje Automático, PCA, Reducción de Dimensionalidad, Redes Neuronales, Componentes Principales, TensorFlow, Keras, Python, Google Colab.

Abstract

The use of dimensionality reduction techniques, such as Principal Component Analysis (PCA), is a valuable tool for optimizing the training of machine learning models, particularly in resource-constrained environments. High-dimensional datasets, such as image data, often contain redundant information, which can negatively impact the performance and efficiency of neural networks.

This project presents a comparative study on the influence of the number of principal components selected via PCA on the performance of a multilayer neural network. Three well-known image classification datasets were used: MNIST, Fashion MNIST, and Digits. The models were trained using Google Colab with Python and libraries such as TensorFlow, Keras, and Scikit-learn.

Several automatic criteria were applied to determine the optimal number of principal components: the Kaiser criterion, the explained variance threshold (95 per cent), and Horn's parallel analysis. In addition, configurations using half the number of original features and all original features were also evaluated. Each configuration was systematically trained and tested to compare its accuracy on the test set.

The results show that it is possible to significantly reduce the number of input variables without compromising model accuracy. In some cases, dimensionality reduction even improved performance by removing noise and redundancy. These findings demonstrate that selecting an appropriate number of components can lead to more efficient models without sacrificing predictive power.

Keywords: Machine Learning, PCA, Dimensionality Reduction, Neural Networks, Principal Components, TensorFlow, Keras, Python, Google Colab.

Índice general

Introducción

If a machine is expected to be infallible, it cannot also be intelligent
Si se espera que una máquina sea infalible, no puede también ser inteligente.

Alan Turing (1912–1954),
Computer Scientist

En este capítulo se introduce el contexto del aprendizaje computacional, resaltando la importancia de la reducción de dimensionalidad en la eficiencia de los modelos. Se presenta la problemática de la selección de componentes principales y su impacto en los modelos de deep learning.

1.1 Introducción

1.1.1 Motivación y contexto

En la actualidad, el aprendizaje automático y el análisis de datos de alta dimensión son pilares fundamentales en múltiples áreas de la ingeniería y la ciencia. En particular, el entrenamiento de redes neuronales profundas sobre conjuntos de datos complejos plantea desafíos computacionales significativos, tanto en tiempo de cómputo como en consumo de memoria. En este contexto, los métodos de reducción de dimensionalidad especialmente el Análisis de Componentes Principales (PCA, por sus siglas en inglés) se presentan como herramientas clave para simplificar los datos manteniendo la mayor cantidad posible de información relevante.

Este proyecto se centra en el estudio de cómo influye el número de componentes principales seleccionados mediante PCA en el rendimiento de una red neuronal multicapa. Se pretende evaluar si es posible reducir el número de variables de entrada sin comprometer significativamente la capacidad predictiva del modelo, contribuyendo así a un entrenamiento más eficiente.

1.1.2 Objetivos

Objetivo principal: Analizar el impacto de diferentes criterios de selección de componentes principales en PCA sobre el rendimiento y eficiencia computacional de una red neuronal multicapa para clasificación de imágenes.

Objetivos específicos:

- Comparar sistemáticamente cinco enfoques de selección (*Kaiser*, *Varianza Explicada 95%*, *Análisis Paralelo*, *Mitad de Características* y *Sin Reducción*) en tres datasets de referencia: **MNIST**, **Fashion MNIST** y **Digits**.
- Evaluar no solo la precisión predictiva sino también métricas de eficiencia como **tiempo de entrenamiento**, **coste computacional** y **rendimiento relativo**.
- Identificar el criterio óptimo para cada tipo de dataset y establecer recomendaciones prácticas para aplicaciones con restricciones computacionales.
- Determinar si la reducción dimensional puede, en algunos casos, mejorar el rendimiento mediante la eliminación de ruido y redundancia.

1.1.3 Metodología General

Este trabajo sigue un enfoque experimental comparativo que evalúa 15 configuraciones resultantes de aplicar 5 criterios de selección a 3 datasets de imágenes. El proceso sistemático incluye:

1. **Preprocesamiento estandarizado** de **MNIST** (70,000 imágenes 28×28), **Fashion MNIST** (70,000 imágenes 28×28) y **Digits** (1,797 imágenes 8×8).
2. **Aplicación de PCA** con determinación automática del número de componentes según cada criterio.
3. **Entrenamiento uniforme** de una red neuronal multicapa con arquitectura fija para todas las configuraciones.
4. **Evaluación comprehensiva** midiendo tanto precisión predictiva como métricas de eficiencia computacional.
5. **Análisis comparativo** de los datos evaluados entre compresión dimensional y rendimiento para cada combinación.

La metodología prioriza la **comparabilidad** y **reproducibilidad** sobre la optimización individual de cada modelo, permitiendo identificar patrones generales aplicables a problemas reales de aprendizaje automático con restricciones de recursos.

METODOLOGÍA

Premature optimization is the root of all evil (or at least most of it) in programming.
La optimización prematura es la raíz de todos los males (o al menos de la mayor parte) en la programación.

Donald Knuth,
Computer Scientist

En este capítulo se describe la metodología utilizada en el desarrollo del proyecto. Se detallan los fundamentos teóricos que sustentan el trabajo, los criterios de selección de componentes, las redes neuronales y las herramientas empleadas.

2.1 Fundamentos Teóricos

2.1.1 Análisis de Componentes Principales (PCA)

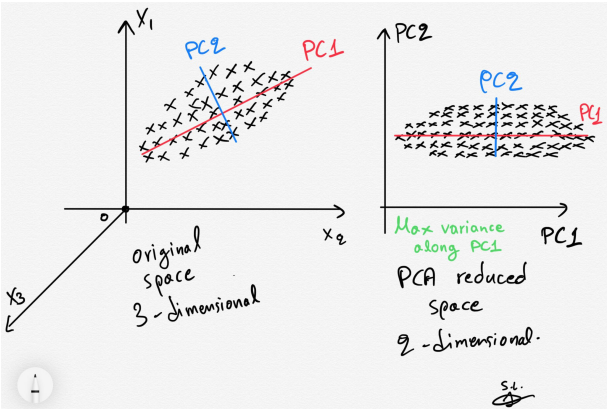

Figura 2.1: Reducción de dimensionalidad mediante PCA

El Análisis de Componentes Principales (PCA, por sus siglas en inglés) es una técnica de reducción de dimensionalidad que busca transformar los datos a un nuevo sistema de coordenadas. En este nuevo sistema, las dimensiones, llamadas **componentes principales**, están ordenadas de manera que la primera componente captura la mayor variabilidad posible, la segunda la siguiente mayor variabilidad, y así sucesivamente.

Detalles del Cálculo de PCA

El cálculo de PCA se fundamenta en encontrar los autovectores y autovalores de la matriz de covarianza de los datos. Estos elementos matemáticos son clave para entender cómo funciona la técnica.

1. Estandarización de los datos Antes de cualquier cálculo, es crucial estandarizar los datos. Este proceso consta de dos pasos: centrar y escalar.

- **Centrado:** Se ajustan los datos para que cada variable tenga una media de cero.
- **Escalado:** Se ajustan los datos para que cada variable tenga una desviación estándar de uno (también conocido como escalado *Z-score*).

Estandarización Matemática Dado un conjunto de datos con variables $X_1, X_2, \ldots, X_p$, cada variable X_j se transforma en una nueva variable estandarizada Z_j mediante la siguiente fórmula:

$$Z_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j}$$

donde:

- X_{ij} es el valor de la variable j en la observación i.
- $\mu_j = \frac{1}{n}\sum_{i=1}^{n} X_{ij}$ es la media de la variable j.
- $\sigma_j = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_{ij} - \mu_j)^2}$ es la desviación estándar de la variable j.

Así, cada variable estandarizada Z_j cumple que:

$$\mathbb{E}[Z_j] = 0, \quad \text{Var}(Z_j) = 1$$

¿Por qué es necesaria la estandarización? La estandarización es obligatoria en PCA por dos razones principales:

1. **Origen en Cero (Fundamental):** PCA es una operación vectorial y geométrica que consiste en una rotación de ejes alrededor del origen del espacio (el punto $[0,0,\ldots,0]$). Si los datos no están centrados (con media cero), esta rotación no se realizaría desde el centro de la nube de datos, lo que invalidaría el propósito de encontrar las direcciones de máxima varianza. El centrado asegura que el origen del PCA coincida con el centroide de los datos, permitiendo una rotación correcta.
2. **Equidad en las Escalas (Práctica):** La estandarización evita que las variables con magnitudes o unidades más grandes dominen los resultados. Por ejemplo, si una variable mide ingresos en euros y otra la edad en años, la primera, por tener valores numéricamente mayores, podría influir desproporcionadamente en el primer componente principal. El escalado asegura que todas las variables contribuyan de manera equitativa al análisis, basándose únicamente en su variabilidad relativa y no en su escala original.

2. Cálculo de la Matriz de Covarianza

Varianza y Covarianza Antes de construir la matriz de covarianza, es necesario definir cómo se calculan estos elementos:

- La **varianza** de una variable X_j mide la dispersión de sus valores respecto a su media μ_j:

$$\sigma_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_{ij} - \mu_j)^2$$

- La **covarianza** entre dos variables X_j y X_k mide cómo varían conjuntamente respecto a sus medias:

$$\text{cov}(X_j, X_k) = \frac{1}{n-1} \sum_{i=1}^{n} (X_{ij} - \mu_j)(X_{ik} - \mu_k)$$

Matriz de Covarianza Una vez calculadas, se organiza esta información en la **matriz de covarianza**:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \text{cov}(X_1, X_2) & \dots & \text{cov}(X_1, X_p) \\ \text{cov}(X_2, X_1) & \sigma_2^2 & \dots & \text{cov}(X_2, X_p) \\ \vdots & \vdots & \ddots & \vdots \\ \text{cov}(X_p, X_1) & \text{cov}(X_p, X_2) & \dots & \sigma_p^2 \end{pmatrix}$$

¿Para qué se calcula la matriz de covarianza? Esta matriz es fundamental porque mide la relación lineal entre cada par de variables.

- La **diagonal** contiene las varianzas de cada variable.
- Los **términos fuera de la diagonal** representan las covarianzas entre pares de variables.

La matriz de covarianza nos permite entender la estructura de la variabilidad de los datos. El objetivo de PCA es encontrar nuevas variables (las componentes principales) que sean combinaciones lineales de las originales y que estén **no correlacionadas entre sí**, es decir, que tengan una covarianza de cero.

3. Cálculo de Autovectores y Autovalores

El siguiente paso es encontrar los **autovectores** y **autovalores** de la matriz de covarianza Σ. Estos se definen mediante la ecuación característica:

$$\Sigma v = \lambda v$$

donde:

- v es un **autovector** (o vector propio),
- λ es el **autovalor** (o valor propio) asociado.

Interpretación de los autovalores Los **autovalores** indican cuánta varianza está contenida en la dirección del autovector correspondiente.

- Un **autovalor grande** implica que su autovector asociado es una dirección donde los datos tienen gran variabilidad.
- Un **autovalor pequeño** implica poca variabilidad en esa dirección.

De esta manera, los autovalores nos permiten ordenar los autovectores de mayor a menor importancia, seleccionando las direcciones que explican la mayor parte de la variabilidad de los datos.

Diagonalización de la matriz de covarianza Los autovectores de Σ forman una base ortogonal que **diagonaliza la matriz de covarianza**. Esto significa que, al proyectar los datos en la base definida por los autovectores, obtenemos nuevas variables (las **componentes principales**) cuya matriz de covarianza es diagonal:

$$\Sigma_Z = V^T \Sigma V = \Lambda$$

donde:

- V es la matriz formada por los autovectores de Σ,
- Λ es una matriz diagonal cuyos elementos son los autovalores.

El hecho de que Σ_Z sea diagonal implica que las componentes principales están **incorreladas**. En el caso particular de que los datos sigan una distribución normal multivariada, esta incorrelación es equivalente a **independencia estadística**.

4. Selección de Componentes y Proyección

Una vez obtenidos los autovalores y autovectores de la matriz de covarianza, se procede a ordenarlos en pares (v_j, λ_j) de mayor a menor según la magnitud de los autovalores λ_j.

Selección de componentes principales Los **autovectores** asociados a los mayores autovalores se convierten en las **componentes principales**, ya que representan las direcciones en las que los datos muestran mayor variabilidad. Si se desea reducir la dimensionalidad, se seleccionan los k autovectores con autovalores más grandes, descartando los que explican menor varianza (considerados menos relevantes).

Proyección de los datos Los k autovectores seleccionados forman la matriz V_k, que define la nueva base del espacio reducido. El conjunto de datos original X (ya estandarizado) se proyecta sobre esta base mediante:

$$Z = XV_k$$

donde:

- $X \in \mathbb{R}^{n \times p}$ es la matriz de datos original (con n observaciones y p variables).
- $V_k \in \mathbb{R}^{p \times k}$ contiene en sus columnas los k autovectores seleccionados.
- $Z \in \mathbb{R}^{n \times k}$ es la representación de los datos en el nuevo espacio de menor dimensión.

De esta manera, PCA logra la **reducción de dimensionalidad** conservando la mayor parte de la variabilidad de los datos, garantizando además que las nuevas variables (componentes principales) estén incorreladas entre sí gracias a la diagonalización previa.

2.1.2 Redes Neuronales

¿Cómo Funciona una Red Neuronal?

Una **red neuronal artificial (RNA)** es un modelo computacional inspirado en la estructura y funcionamiento del cerebro biológico, diseñado para aprender patrones complejos a partir de datos. Consiste en una serie de **neuronas interconectadas** dispuestas en capas, donde cada neurona es una unidad de procesamiento que realiza cálculos sobre la información de entrada.

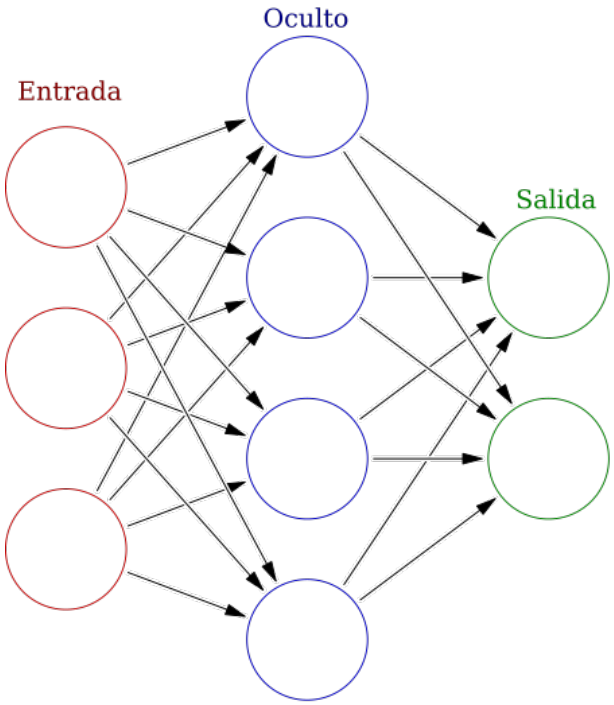

Figura 2.2: Red Neuronal Artificial
[1]

1. **Capa de Entrada:** Es la interfaz inicial de la red, donde los datos de entrada, representados como un vector $x = [x_1, x_2, ..., x_n]$, son recibidos. Cada x_i corresponde a una característica de la entrada y se asigna a una neurona de esta capa.

2. **Capas Ocultas:** Después de la capa de entrada, los datos se propagan a través de una o más **capas ocultas**. En estas capas, cada neurona j de una capa l recibe las salidas de las neuronas de la capa anterior $(l-1)$, las cuales son multiplicadas por unos **pesos** $w_{ij}^{(l)}$ y se les suma un **sesgo** $b_j^{(l)}$. El resultado de esta suma ponderada se conoce como la **entrada neta** $z_j^{(l)}$, la cual se calcula como:

$$z_j^{(l)} = \sum_{i=1}^{N_{l-1}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

donde N_{l-1} es el número de neuronas en la capa anterior y $a_i^{(l-1)}$ es la activación de la neurona i de la capa $l-1$. Posteriormente, $z_j^{(l)}$ se pasa a través de una **función de activación** $f(\cdot)$, produciendo la activación (salida) de la neurona j, $a_j^{(l)} = f(z_j^{(l)})$. Esta función introduce **no linealidad** en el modelo, permitiendo que la red aprenda relaciones complejas en los datos que no podrían ser capturadas por modelos lineales [3].

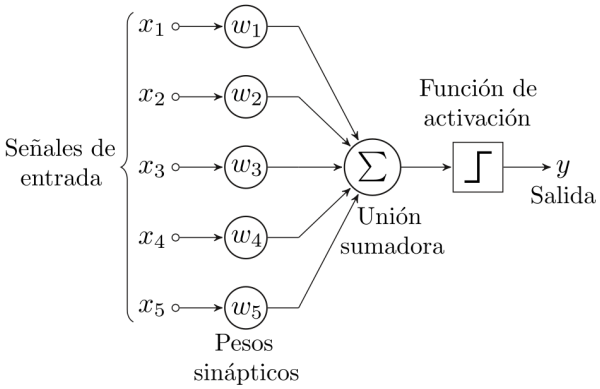

Figura 2.3: Diagrama de un perceptrón: Neurona Artificial Básica

3. **Capa de Salida:** Es la capa final que produce la predicción de la red. El número de neuronas en esta capa depende de la tarea; por ejemplo, para problemas de clasificación, puede tener una neurona por cada clase posible.

El objetivo fundamental del entrenamiento de una red neuronal es que **generalice** bien a datos no vistos, es decir, que no solo prediga correctamente en el conjunto de datos de **entrenamiento**, sino que mantenga un alto rendimiento en datos nuevos y desconocidos. Para lograr esto, la red ajusta iterativamente sus **pesos** y **sesgos** para **minimizar una función de pérdida** (o coste), que cuantifica la discrepancia entre las predicciones de la red y los valores reales. Este proceso de ajuste se realiza utilizando un **optimizador** y la técnica de **retropropagación** del error. El uso de conjuntos de datos de entrenamiento y **validación** es crucial para monitorizar la capacidad de generalización de la red y evitar el **sobreajuste**, un fenómeno en el que la red memoriza el ruido o detalles específicos de los datos de entrenamiento, lo que conduce a un pobre rendimiento en datos nuevos [4].

2.1.3 Nuestra Red Neuronal: Arquitectura y Entrenamiento Específicos

Hemos elegido la Red Neuronal Multicapa (MLP) por su óptimo equilibrio entre sencillez, facilidad de implementación y alta eficacia. Su estructura de alimentación hacia adelante la hace sencilla de comprender y configurar, requiriendo menos ajustes iniciales que arquitecturas más complejas, como las convolucionales. A pesar de su simplicidad, la MLP es potente y versátil, capaz de aprender patrones no lineales complejos, lo que se traduce en gran precisión. Además, la inclusión de Dropout mejora su robustez y capacidad de generalización.

Arquitectura

- **Capa de Entrada:** La dimensión de esta capa se establece igual al **número de componentes** seleccionados tras la aplicación de una técnica de reducción de dimensionalidad, como el Análisis de Componentes Principales (PCA). Esto asegura que la red reciba una representación densa y optimizada de las características originales.
- **Capa Oculta 1:** Contiene **256 neuronas**.
  - **Activación ReLU:** Se utiliza la función de activación Rectified Linear Unit (ReLU)[9], definida como $f(x) = \max(0, x)$. Se escoge la función de activación ReLU porque, a diferencia de la sigmoide o la tangente hiperbólica, no provoca el problema del gradiente que desaparece. Este problema ocurre cuando, durante la retropropagación, los gradientes se vuelven extremadamente pequeños en las capas iniciales, lo que impide que la red actualice sus pesos de manera efectiva y dificulta el aprendizaje en modelos profundos.
  - **Dropout (30 por ciento):** Durante cada paso de entrenamiento (batch), el 30 por ciento de las neuronas de esta capa se **desactivan aleatoriamente** (es decir, sus salidas se establecen a cero). Esta técnica de regularización previene el sobreajuste al reducir la co-adaptación compleja entre neuronas, forzando a la red a desarrollar representaciones más robustas y distribuidas al no depender excesivamente de ninguna neurona específica [12].
- **Capa Oculta 2:** Con **128 neuronas**.
  - **Activación ReLU + Dropout (30 por ciento):** Similar a la primera capa oculta, emplea la activación ReLU y aplica Dropout para la regularización.
- **Capa de Salida:** El número de neuronas en esta capa es igual al **número de clases**

presentes en el conjunto de datos. Por ejemplo, en el dataset Digits, que contiene imágenes de dígitos escritos a mano (del 0 al 9), la capa de salida de la red neuronal se configuraría con diez neuronas, una por cada dígito posible a predecir.

- **Activación Softmax:** Esta función de activación se emplea en la capa de salida para problemas de clasificación multiclase. Transforma un vector de valores reales $z = [z_1, z_2, ..., z_K]$ (las salidas de la capa anterior) en un vector de probabilidades $P = [p_1, p_2, ..., p_K]$, donde cada p_i representa la probabilidad de que la entrada pertenezca a la clase i. La función Softmax se define como:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

donde K es el número de clases. A diferencia de otras funciones de activación como ReLU o sigmoide, que no garantizan una interpretación probabilística ni que la suma de salidas sea 1, Softmax asegura que todas las probabilidades sean no negativas y que su suma sea igual a 1 ($\sum_{i=1}^{K} p_i = 1$). Esto permite interpretar las salidas como una distribución de probabilidad sobre las clases y facilita la toma de decisión al elegir la clase más probable. [2].

Configuración del Entrenamiento

- **Batch Size (Tamaño de Lote):**

El **Batch Size** (tamaño de lote) es un hiperparámetro fundamental en el entrenamiento de modelos de **Machine Learning**, especialmente en redes neuronales. Se refiere al número de ejemplos de entrenamiento que se procesan antes de que los parámetros internos del modelo (pesos y sesgos) sean actualizados. Durante el entrenamiento, el conjunto de datos se divide en lotes. En cada iteración, un lote de datos se alimenta a la red neuronal, se calcula la pérdida (error) y, basándose en esta pérdida, se actualizan los pesos del modelo. Esta actualización se realiza mediante un algoritmo de optimización, como el **Descenso de Gradiente Estocástico (SGD)** o sus variantes (**Adam**, RMSprop, etc.).

- **Batch Size pequeño:** Implica actualizaciones de pesos más frecuentes y ruidosas. Esto puede llevar a una convergencia más lenta pero potencialmente a un mejor rendimiento en la generalización, evitando mínimos locales "malos"[8].

  - **Batch Size grande:** Conduce a actualizaciones de pesos menos frecuentes y más estables. Puede acelerar la convergencia y aprovechar mejor la paralelización computacional, pero existe el riesgo de caer en mínimos locales no óptimos y una menor capacidad de generalización [5].

- **Optimizador Adam:** Se utiliza el algoritmo de optimización Adam (Adaptive Moment Estimation) para ajustar los pesos de la red. Adam es un método de descenso de gradiente estocástico que adapta las tasas de aprendizaje para cada parámetro de forma individual, basándose en los momentos de primer y segundo orden de los gradientes, lo que lo hace computacionalmente eficiente y robusto para una amplia gama de problemas de aprendizaje profundo [6].

- **Función de pérdida:** `sparse categorical crossentropy`**:** Esta función de pérdida es adecuada para problemas de clasificación multiclase donde las etiquetas de las clases se proporcionan como enteros (categorías dispersas), en lugar de vectores one-hot codificados. Mide la diferencia entre la distribución de probabilidad predicha por la red (producida por la activación Softmax) y la distribución de probabilidad verdadera de la clase. El objetivo del proceso de optimización es **minimizar** esta función de pérdida.

- **Validación interna con** `validation split` **de 0.15:** Durante cada **época de entrenamiento** (un ciclo completo sobre todo el conjunto de datos de entrenamiento), el 15 por ciento de los datos de entrenamiento se separa automáticamente para formar un conjunto de validación. Esto permite **monitorizar el rendimiento** de la red en datos no vistos durante el ajuste de pesos, lo cual es crucial para detectar el sobreajuste y evaluar la capacidad de generalización del modelo en cada iteración del entrenamiento.

- **Técnica de** `early stopping` **para evitar sobreajuste:** El entrenamiento es un proceso **iterativo** donde los pesos de la red se ajustan a lo largo de múltiples épocas. La técnica de **early stopping** es una estrategia de regularización que **detiene el proceso de entrenamiento** cuando el rendimiento de la red en el conjunto de validación deja de mejorar durante un número predefinido de épocas (**paciencia**). Esto previene que la red continúe ajustándose excesivamente a los datos de entrenamiento y empiece a memorizar el ruido, lo que se traduciría en un rendimiento deficiente en datos nuevos y no vistos [10].

2.2 Diseño Experimental

2.2.1 Objetivo del Estudio

El objetivo principal de este trabajo es realizar un análisis comparativo sistemático de diferentes criterios para la selección del número de componentes principales en el Análisis de Componentes Principales (PCA), evaluando su influencia tanto en el rendimiento predictivo como en la eficiencia computacional de una red neuronal multicapa aplicada a problemas de clasificación de imágenes.

Para ello, se plantean los siguientes objetivos específicos:

1. Evaluar el impacto de la reducción dimensional mediante PCA en la precisión de clasificación, comparando cinco enfoques de selección de componentes frente al uso del conjunto de datos original sin reducción.
2. Analizar la eficiencia computacional de cada criterio, considerando métricas como el tiempo de entrenamiento, el coste computacional y el rendimiento relativo.
3. Identificar el criterio de selección óptimo que ofrezca el mejor equilibrio entre compresión de datos y capacidad predictiva para cada tipo de dataset.
4. Determinar si la reducción de dimensionalidad puede, en algunos casos, mejorar el rendimiento del modelo mediante la eliminación de ruido y redundancia en los datos.

Este enfoque experimental permitirá establecer recomendaciones prácticas sobre la selección de componentes principales en aplicaciones de aprendizaje automático con restricciones computacionales.

2.2.2 Criterios de Selección Evaluados

En este estudio se han evaluado cinco enfoques diferentes para determinar el número óptimo de componentes principales en PCA, con el objetivo de comparar su efectividad en aplicaciones de aprendizaje automático.

Criterio de Kaiser Este criterio retiene únicamente los componentes principales cuyos autovalores superan el valor 1. La justificación subyacente es que un autovalor

menor que 1 indica que el componente explica menos varianza que una sola variable original. En la práctica, este método ofrece una reducción dimensional agresiva que puede ser beneficiosa para eliminar ruido en datasets con alta dimensionalidad.

Varianza Explicada (95 %) Este enfoque conserva el número mínimo de componentes necesarios para capturar al menos el 95 % de la varianza total de los datos. A diferencia del criterio de Kaiser, este método prioriza la conservación de información sobre la compresión máxima, manteniendo componentes adicionales que contribuyen significativamente a la varianza acumulada.

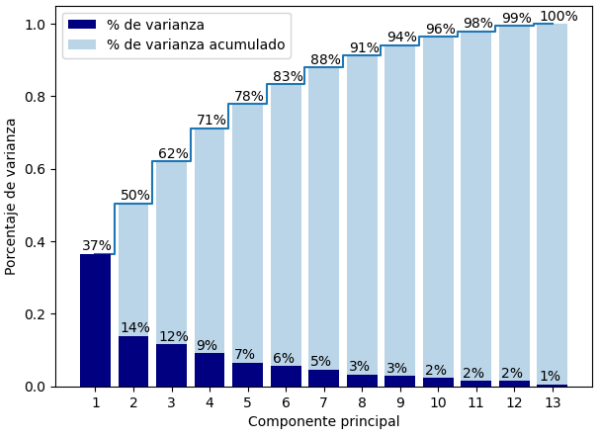

Figura 2.4: Varianza Explicada

Análisis Paralelo de Horn Técnica más sofisticada que compara los autovalores reales con los obtenidos de datasets aleatorios con la misma dimensionalidad. Solo se retienen aquellos componentes cuyos autovalores reales superan los correspondientes a datos aleatorios. Este enfoque proporciona una base empírica para distinguir entre componentes que representan estructura real de los datos y aquellos que podrían atribuirse al azar.

Enfoques de Referencia Para establecer comparaciones significativas, se incluyeron dos configuraciones adicionales:

- **Mitad de características**: Reducción arbitraria al 50 % del número original de dimensiones, sin criterio estadístico.
- **Sin reducción**: Utilización de todas las características originales, sirviendo como línea base para evaluar el impacto de la reducción dimensional.

La selección de estos cinco criterios permite abarcar un espectro amplio de estrategias, desde métodos simples y ampliamente utilizados (Kaiser, Varianza) hasta técnicas más robustas basadas en simulaciones (Análisis Paralelo), proporcionando una comparación comprehensiva de su efectividad en el contexto de redes neuronales.

2.2.3 Métricas de Evaluación

Para realizar una comparación comprehensiva de los diferentes criterios de selección, se han definido métricas que evalúan tanto el rendimiento predictivo como la eficiencia computacional.

Configuración del Entrenamiento Todos los experimentos siguieron la misma configuración para garantizar la comparabilidad:

- **Máximo de épocas**: 100, con parada temprana (Early Stopping) que monitoriza la precisión de validación.
- **Paciencia**: 5 épocas (el entrenamiento se detiene si no hay mejora en validación).
- **Tamaño de lote**: 64 muestras.
- **Validación**: 15% de los datos de entrenamiento reservados para validación.
- **Restauración de mejores pesos**: Se conservan los pesos que lograron la mejor precisión de validación.

Métricas Principales

- **Precisión de clasificación**: Porcentaje de predicciones correctas en el conjunto de prueba.
- **Tiempo total de entrenamiento**: Tiempo requerido para completar el entrenamiento.
- **Épocas efectivas**: Número real de épocas hasta convergencia.

Métricas Derivadas Para analizar la eficiencia computacional, se calcularon:

- **Tiempo por época**:

$$\text{Tiempo por época} = \frac{\text{Tiempo total}}{\text{Épocas efectivas}}$$

- **Coste computacional**:

$$\text{Coste} = \text{Tiempo por época} \times \text{Número de componentes}$$

- **Rendimiento relativo**:

$$\text{Rendimiento relativo} = \frac{\text{Precisión}}{\text{Tiempo por época}}$$

Estas métricas permiten evaluar no solo la capacidad predictiva de cada configuración, sino también su eficiencia en términos de recursos computacionales, proporcionando una visión completa del equilibrio entre rendimiento y coste.

2.2.4 Proceso Experimental

El diseño experimental se estructuró como un estudio comparativo sistemático que evaluó las 15 combinaciones resultantes de aplicar los 5 criterios de selección de componentes a los 3 conjuntos de datos. El proceso se ejecutó de forma idéntica para cada configuración, garantizando la reproducibilidad y comparabilidad de los resultados.

Matriz de Experimentos

$$3 \text{ datasets} \times 5 \text{ criterios} = 15 \text{ configuraciones experimentales}$$

- **Datasets**: MNIST (784 características), Fashion MNIST (784 características), Digits (64 características).

- **Criterios**: Kaiser, Varianza Explicada 95%, Análisis Paralelo, Mitad de Características, Sin Reducción.

Flujo Experimental por Configuración

1. **Preprocesamiento Inicial**
   - División del dataset: 80 % entrenamiento, 20 % prueba.
   - Estandarización de características (media=0, varianza=1).
2. **Aplicación de PCA y Selección de Componentes**
   - Cálculo de componentes principales para el conjunto de entrenamiento.
   - Determinación del número de componentes según el criterio específico.
   - Proyección de datos de entrenamiento y prueba al nuevo espacio dimensional.
3. **Entrenamiento de la Red Neuronal**
   - Configuración idéntica de arquitectura e hiperparámetros para todos los experimentos.
   - Entrenamiento con validación interna (15 % de datos de entrenamiento).
   - Aplicación de *early stopping* para evitar sobreajuste.
4. **Evaluación y Registro de Métricas**
   - Cálculo de precisión en conjunto de prueba.
   - Medición de tiempo total de entrenamiento.
   - Cálculo de métricas derivadas (tiempo por época, coste computacional, rendimiento relativo).

Estrategia de Validación Cada experimento se ejecutó en las mismas condiciones de hardware y software, utilizando Google Colab como entorno de ejecución estandarizado. La aleatoriedad se controló mediante semillas fijas en los procesos estocásticos, asegurando que las diferencias observadas se atribuyan únicamente a los criterios de selección evaluados y no a variaciones aleatorias en el entrenamiento.

Este diseño permite una comparación directa y equitativa entre los diferentes enfoques de selección de componentes, proporcionando evidencia empírica sólida sobre su efectividad en aplicaciones prácticas de aprendizaje automático.

2.3 Conjuntos de datos utilizados

Se han utilizado tres conjuntos de datos de acceso público, todos ellos compuestos por imágenes en escala de grises de baja resolución, lo que permite un tratamiento computacional viable en Google Colab:

- **MNIST**[7]: conjunto de 70.000 imágenes en escala de grises (28x28 px) de dígitos manuscritos.
- **Fashion MNIST** [13]: 70.000 imágenes (28x28 px) de prendas de ropa.
- **Digits** [11]: 1.797 imágenes pequeñas (8x8 px) de dígitos escritos a mano.

Se descartaron datasets más complejos como CIFAR-10 o SVHN debido a su mayor número de características y necesidades computacionales incompatibles con los límites de Google Colab.

A continuación, se muestran imágenes representativas de cada conjunto:

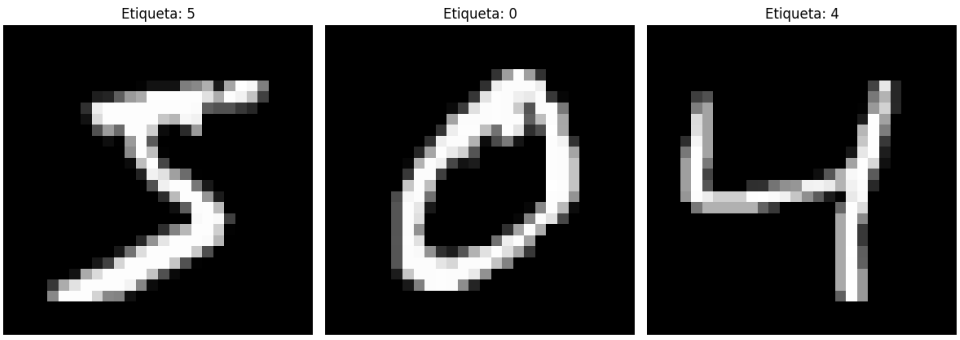

Figura 2.5: Muestras aleatorias de MNIST

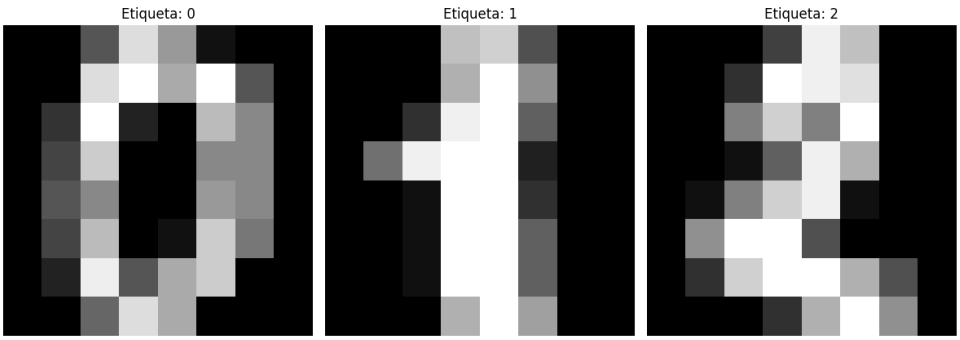

Figura 2.6: Muestras aleatorias de DIGITS

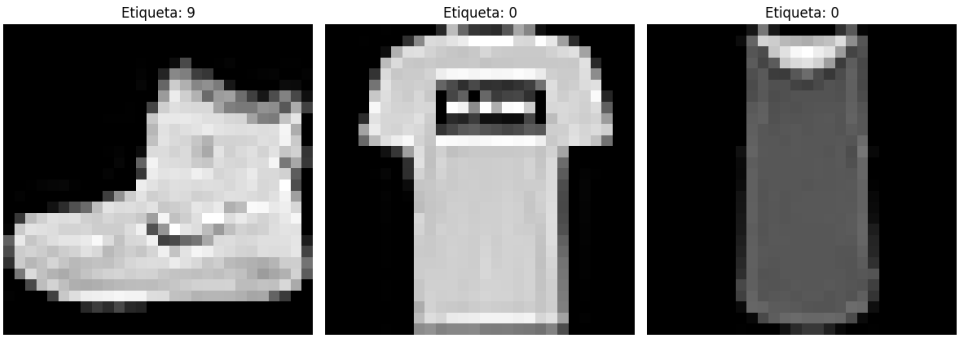

Figura 2.7: Muestras aleatorias de FASHION MNIST

2.4 Herramientas y entorno de desarrollo

Para la implementación de este proyecto se ha empleado el entorno en la nube Google Colab, que proporciona acceso gratuito a recursos computacionales limitados pero suficientes para redes neuronales simples. El lenguaje de programación utilizado ha sido **Python**, junto con las siguientes bibliotecas:

- `NumPy`: para manipulación de datos y estructuras numéricas.
- `Matplotlib`: para la generación de gráficos.
- `Scikit-learn`: para la carga de datasets, escalado de datos y aplicación de PCA.
- `TensorFlow` y `Keras`: para el diseño, entrenamiento y evaluación de redes neuronales multicapa.

2.4.1 Desarrollo en Python

El código ha sido desarrollado completamente en Python, siguiendo un enfoque detallado y didáctico que prescinde de automatismos innecesarios. Todas las etapas críticas desde la reducción dimensional hasta el entrenamiento y evaluación del modelo han sido construidas manualmente con una lógica clara y explícita. Se implementaron desde cero los tres criterios de selección de componentes principales (Kaiser, Varianza Explicada y Análisis Paralelo de Horn), utilizando únicamente operaciones fundamentales sobre valores propios y transformaciones PCA.

La estructura del código es modular y reproducible. Se parte de la carga y normalización de distintos datasets, se calculan y aplican las transformaciones PCA según los criterios definidos, y se entrena una red neuronal profunda diseñada con Keras. Esta red se ajusta automáticamente mediante técnicas como `Dropout` y `EarlyStopping`, evaluando el rendimiento para cada escenario de reducción. Las métricas se visualizan y comparan gráficamente para analizar el impacto de cada criterio sobre el desempeño del modelo. Se anexa el código fuente completo en el apéndice §A.2.

2.4.2 Flujo de Ejecución del Programa: Paso a Paso

Figura 2.8: Diagrama de Flujo de Ejecución

- **Configuración Inicial: Carga y Preprocesamiento de Datos**
  Comando:

  ```
  from sklearn.datasets import load_digits, fetch_openml
  ```

  Librería: `scikit-learn`

- **Escalado de Características**
  Propósito: Normalizar las características para que tengan media 0 y varianza 1, esencial para PCA y redes neuronales.
  Comando:

  ```
  StandardScaler().fit_transform(X)
  ```

  Librería: `scikit-learn`

- **Determinación de Componentes Óptimos y Aplicación de PCA**

  - **Criterio de Kaiser**
    Comando: `explained_variance_`
    Librería: `scikit-learn`
  - **Criterio de Varianza Explicada**
    Comando: `explained_variance_ratio_`
    Librería: `scikit-learn`
  - **Análisis Paralelo de Horn**
    Comando:

    ```
    random_eigenvalues = []
      for _ in range(n_replications):
          random_data = np.random.normal(loc=0, scale=1,
          size=(n_samples, n_features))
          random_pca = PCA().fit(random_data)
          random_eigenvalues.append(random_pca.explained_variance_)

      n_components = np.sum(eigenvalues_real[:max_components]
      > mean_random_eigenvalues[:max_components])
    ```

    Librería: `scikit-learn` y `NumPy`
  - **Reducción de Dimensionalidad con PCA**
    Propósito: Proyectar datos en espacio de componentes principales.
    Comando:

    ```
    PCA(n_components=k).fit_transform(X_scaled)
    ```

Librería: `scikit-learn`

- **Entrenamiento y Evaluación del Modelo**

  - **Definición de la Arquitectura de la Red**
    Propósito: Construir red neuronal MLP.
    Comando:

```
model_dense = models.Sequential([
    layers.Input(shape=(n_components,)),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(num_classes, activation='softmax')
])
```

    Librería: `tensorflow.keras`

  - **Configuración del Compilador**
    Propósito: Definir optimizador, función de pérdida y métricas.
    Comando:

```
model_dense.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])
```

    Librería: `tensorflow.keras`

  - **Configuración de entrenamiento**
    Propósito: Entrenar la red con validación y detener si no mejora.
    Comando:

```
history = model_dense.fit(X_train_pca, y_train,
                          epochs=100,
                          batch_size=64,
                          validation_split=0.15,
                          verbose=0,
                          callbacks=[early_stopping])
```

Librería: `tensorflow.keras`

- **Generación de Visualizaciones**
  Propósito: Crear gráficos de métricas de rendimiento usando los datos del historial de entrenamiento.
  Comando:

  ```
  matplotlib.pyplot.plot()
  ```

  Librería: `matplotlib`

Resultados

"Los datos no son la respuesta, son la materia prima para obtenerla. El verdadero valor está en cómo los transformamos en conocimiento accionable."

Hilary Mason,
Científico de Datos

Este capítulo presenta los hallazgos obtenidos tras aplicar diferentes metodologías de selección de componentes principales en conjuntos de datos de referencia, analizando su impacto en el rendimiento de modelos de aprendizaje profundo.

3.1 MNIST

Los resultados de precisión fueron:

- **Todas las características (784 comp.)**: **96.36 %**
- **Mitad de características (392 comp.)**: 97.14 %
- **Criterio de Kaiser (179 comp.)**: 97.06 %
- **Varianza explicada (332 comp.)**: 97.21 %
- **Análisis Paralelo (150 comp.)**: 97.31 %

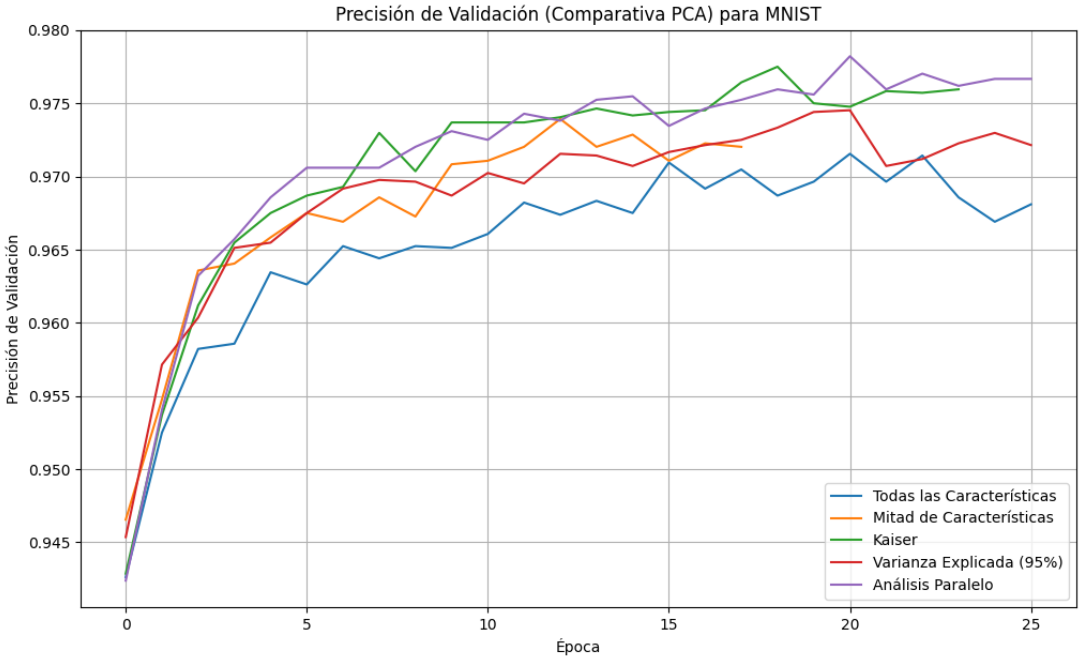

Figura 3.1: Curvas de precisión de validación para MNIST.

Se observa que los métodos de reducción mejoran ligeramente la precisión respecto a usar todas las características.
El Análisis Paralelo obtuvo 97.31 % con 150 componentes.

3.2 Digits

Los resultados de precisión fueron:

- **Todas las características (64 comp.)**: 97.5 %
- **Mitad de características (32 comp.)**: 96.39 %
- **Criterio de Kaiser (19 comp.)**: 95.83 %
- **Varianza explicada (40 comp.)**: **97.78 %**
- **Análisis Paralelo (16 comp.)**: 96.11 %

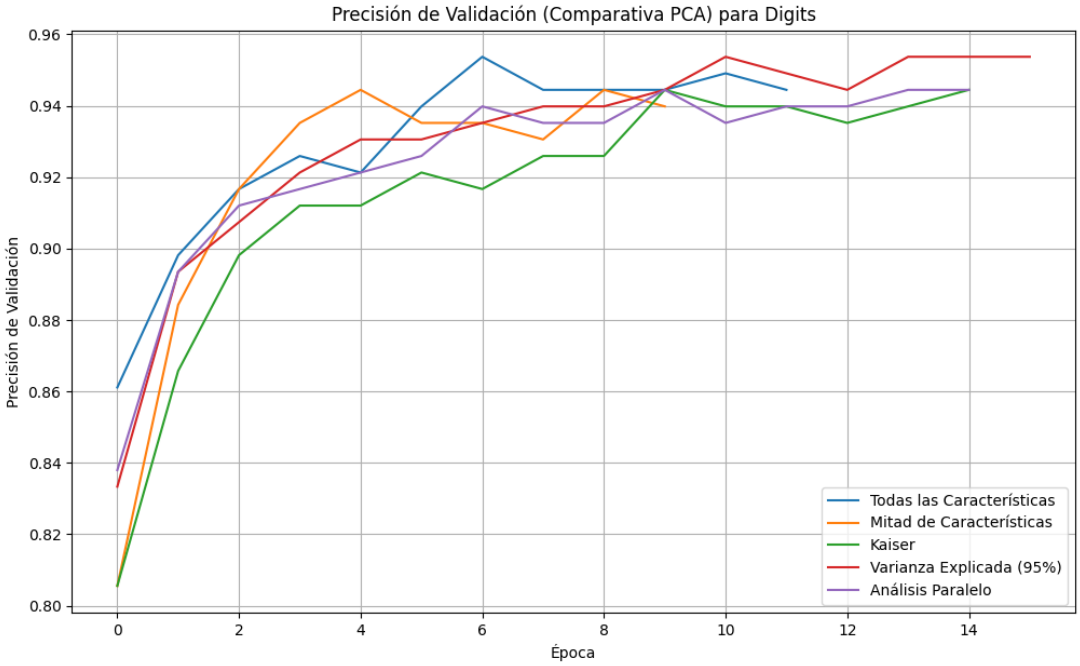

Figura 3.2: Curvas de precisión de validación para Digits.

Se observan descensos sutiles en la precisión con la mayoría de las reducciones, excepto con el criterio de Varianza Explicada, que supera ligeramente la precisión de los datos originales. Cabe destacar que este criterio aplica la reducción más conservadora, manteniendo el 62.5 % de las características iniciales, mientras que enfoques más agresivos como el Análisis Paralelo (25 % de características) podrían estar eliminando información relevante

3.3 Fashion MNIST

Los resultados de precisión fueron:

- **Todas las características (784 comp.)**: **89.16 %**
- **Mitad de características (392 comp.)**: 89.34 %
- **Criterio de Kaiser (79 comp.)**: 88.17 %
- **Varianza explicada (256 comp.)**: **89.80 %**
- **Análisis Paralelo (69 comp.)**: 88.31 %

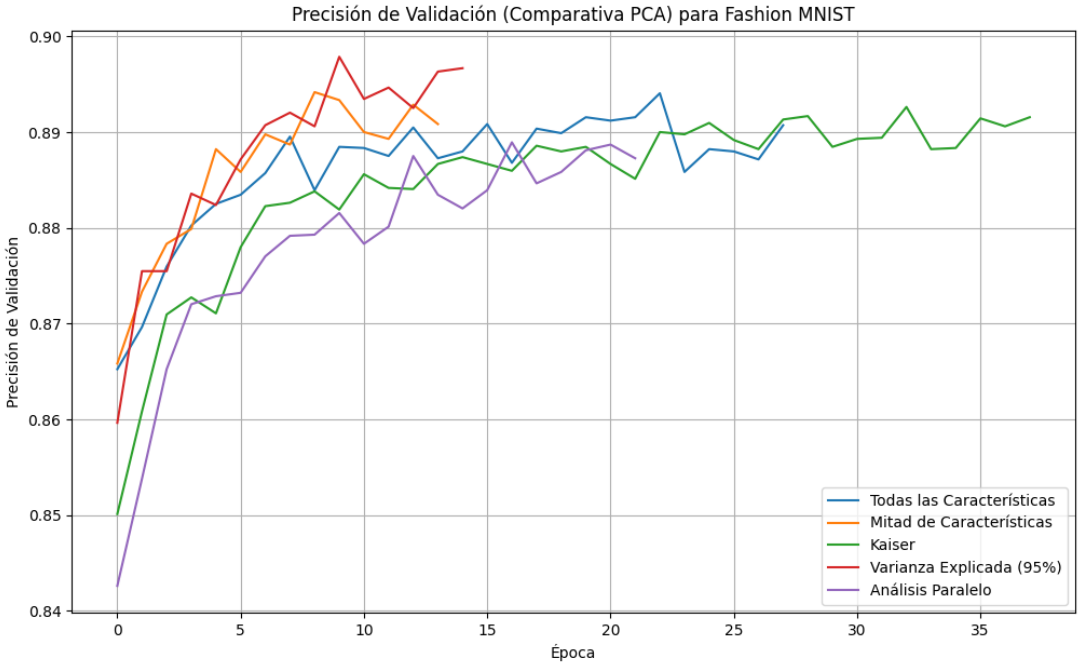

Figura 3.3: Curvas de precisión de validación para Fashion MNIST.

A diferencia de los comportamientos observados en MNIST y Digits, este conjunto de datos presenta resultados mixtos. El mejor rendimiento se logra con el criterio de Varianza Explicada, mientras que tanto este enfoque como la reducción a la mitad de características -ambos con niveles de compresión moderados- consiguen mejorar la precisión respecto a los datos originales. Los criterios más agresivos (Kaiser y Análisis Paralelo), que retienen menos del 10% de las características originales, muestran un rendimiento inferior.

3.4 Métricas Derivadas

Dataset	Reducción	Comp.	Tiempo (s)	Épocas	Accuracy	Tpo/Ép. (s)	Rend.	Coste
MNIST	Todas	784	69.06	26	0.967	2.66	0.364	2082.43
MNIST	Mitad	392	42.45	18	0.970	**2.36**	**0.411**	924.47
MNIST	Kaiser	179	59.22	24	0.974	2.47	0.395	441.68
MNIST	Var. 95%	332	70.56	26	0.973	2.71	0.359	901.00
MNIST	Paralelo	150	70.16	26	**0.976**	2.70	0.362	**404.77**
Digits	Todas	64	7.60	12	0.975	0.63	1.539	40.53
Digits	Mitad	32	5.53	10	0.964	0.55	1.743	17.70
Digits	Kaiser	19	6.82	15	0.958	**0.46**	**2.108**	**8.64**
Digits	Var. 95%	40	8.79	16	**0.978**	0.55	1.780	21.98
Digits	Paralelo	16	8.29	15	0.961	0.55	1.739	8.84
FMNIST	Todas	784	70.31	28	0.889	2.51	0.354	1968.68
FMNIST	Mitad	392	35.55	14	0.893	2.54	0.351	995.40
FMNIST	Kaiser	79	87.46	38	0.892	**2.30**	**0.387**	**181.83**
FMNIST	Var. 95%	256	43.95	15	**0.896**	2.93	0.306	750.08
FMNIST	Paralelo	69	60.10	22	0.888	2.73	0.325	188.50

Cuadro 3.1: Tabla completa de resultados por dataset y criterio

3.4.1 Análisis Comparativo de Métricas Derivadas

El Cuadro 3.1 presenta el resumen completo de resultados para todas las configuraciones evaluadas, incluyendo las métricas de eficiencia computacional calculadas.

Principales observaciones numéricas

- En **MNIST**, la configuración con Análisis Paralelo (150 componentes) alcanza la máxima precisión (0.976) con el menor coste computacional (404.77).
- En **Digits**, el criterio de Varianza Explicada logra la mayor precisión (0.978) manteniendo un coste computacional moderado (21.98).
- En **Fashion MNIST**, la reducción por Varianza Explicada consigue la mejor precisión (0.896).
- Las configuraciones con todos los componentes originales muestran consistentemente los costes computacionales más elevados en todos los datasets.

Tendencias generales identificadas

- Existe una correlación inversa entre el número de componentes y el coste computacional.
- Las reducciones más agresivas (menos componentes) suelen requerir menos tiempo por época.
- No se observa una relación directa entre el número de componentes y la precisión final.

Conclusiones

"Todos los modelos están equivocados, pero algunos son útiles. El reto está en encontrar cuán equivocados deben ser para dejar de ser útiles."

George E.P. Box (1919–2013),
Estadístico

Este capítulo sintetiza los principales hallazgos del estudio, evalúa las limitaciones metodológicas y propone líneas futuras para extender la investigación sobre selección de componentes principales en modelos de deep learning.

4.1 Conclusiones

Este estudio ha demostrado que la selección adecuada del número de componentes en PCA puede optimizar significativamente el entrenamiento de redes neuronales, logrando en muchos casos mejorar tanto la eficiencia computacional como el rendimiento predictivo. Los resultados obtenidos permiten extraer las siguientes conclusiones:

- **Reducción dimensional sin pérdida de precisión**: La reducción de dimensionalidad no solo no perjudica el rendimiento, sino que en muchos casos lo mejora. En **MNIST**, reducir de 784 a 150 componentes mediante Análisis Paralelo aumentó la precisión del 96.7% al **97.6%**, reduciendo simultáneamente el coste computacional en un **80%**. Este fenómeno sugiere que PCA elimina eficazmente ruido y redundancia en los datos.

- **No existe un criterio universal óptimo**: La efectividad de cada método depende de las características específicas del dataset. Mientras que en **MNIST** el Análisis Paralelo (reducción del 80%) ofrece el mejor rendimiento, en **Fashion MNIST** la Varianza Explicada (reducción del 67%) resulta superior, y en **Digits** ambos criterios muestran ventajas complementarias. Esto refleja que la complejidad intrínseca de cada dataset determina el nivel óptimo de compresión.

- **La Varianza Explicada como criterio más robusto**: Este método demostró consistentemente mantener o superar la precisión de los datos originales en todos los datasets, mostrando especial resistencia a reducciones excesivamente agresivas. Su enfoque conservador (manteniendo 30–60% de características) lo convierte en la opción más fiable para aplicaciones donde la estabilidad es prioritaria.

- **Eficiencia computacional significativa**: Las configuraciones reducidas mostraron sistemáticamente menores tiempos por época y costes computacionales. En **Digits**, el criterio de Kaiser logró el mejor rendimiento relativo (2.108) con el menor coste absoluto (8.64), demostrando que reducciones agresivas pueden ser óptimas en contextos con restricciones computacionales severas.

- **El uso completo raramente se justifica**: Emplear todas las características originales resultó consistentemente en los mayores costes computacionales sin ofrecer ventajas predictivas proporcionales. En todos los casos evaluados, al menos un criterio de reducción igualó o superó la precisión del dataset completo con una fracción del coste computacional.

4.2 Limitaciones del Estudio

- **Alcance de datasets limitado**: La utilización exclusiva de imágenes de baja resolución (MNIST, Fashion MNIST, Digits) restringe la generalización de resultados a problemas de visión computerizada más complejos.
- **Arquitectura de red básica**: El uso de una MLP simple, aunque adecuado para fines comparativos, no explora cómo se comportaría la reducción dimensional en arquitecturas más avanzadas como redes convolucionales.
- **Entorno computacional restrictivo**: Las limitaciones de Google Colab impidieron la experimentación con datasets más grandes y modelos más complejos.
- **Hiperparámetros fijos**: La decisión de mantener constantes todos los hiperparámetros, aunque necesaria para garantizar comparabilidad, puede haber suboptimizado el rendimiento individual de cada configuración.

4.3 Trabajos Futuros

- **Extensión a datasets complejos**: Aplicar la metodología a **CIFAR-10, SVHN** u otros datasets de mayor resolución y complejidad semántica.
- **Evaluación en arquitecturas avanzadas**: Estudiar el comportamiento de estos criterios en **redes convolucionales, transformers** u otras arquitecturas *state-of-the-art*.
- **Comparativa con técnicas modernas**: Contrastar PCA con métodos de reducción dimensional no lineales como **UMAP, t-SNE** o **autoencoders**.
- **Análisis multivariable de eficiencia**: Incorporar métricas adicionales como **consumo energético, uso de memoria** y **tiempos de inferencia**.
- **Desarrollo de criterios adaptativos**: Investigar la creación de criterios híbridos que se adapten automáticamente a las características específicas de cada dataset.

A

Apéndices

"La combinación adecuada de datos y curiosidad puede llevar a experimentación y análisis que cambien el mundo."

John Tukey (1915–2000),
Matemático

Este capítulo complementario contiene material técnico adicional que apoya la investigación principal, incluyendo detalles implementación, especificaciones técnicas y resultados extendidos.

A.1 Detalles técnicos adicionales

En cada cálculo del número de componentes óptimos se ha verificado no exceda el número original de características y que sea mayor o igual a 1, para evitar errores en la transformación.

A.2 Código fuente

El código fuente completo utilizado para este proyecto ha sido incluido como enlace a GitHub:
`https://github.com/JuanPesa/PCA-NCOMP-SELECTION.git`
Se ha documentado el código con comentarios para facilitar su comprensión y replicabilidad.

A.3 Asistencia de Inteligencia Artificial en el Desarrollo

Durante el desarrollo de este Trabajo Fin de Grado, se ha integrado la inteligencia artificial (IA) como una herramienta de apoyo fundamental para optimizar diversas fases del proceso de programación y comprensión de conceptos. Específicamente, se ha utilizado la IA de Google Colaboratory (Google Colab) para los siguientes propósitos:

- **Programación Asistida y Generación de Código:** La IA de Google Colab ha sido empleada para asistir en la escritura de fragmentos de código, sugerir implementaciones eficientes y generar estructuras de funciones o clases. Esta capacidad ha permitido acelerar el proceso de codificación y explorar diferentes enfoques algorítmicos.

- **Depuración (Debugging) y Resolución de Errores:** La inteligencia artificial ha facilitado la identificación y corrección de errores en el código. A través de sugerencias y explicaciones, ha contribuido a la depuración de scripts y al entendimiento de las causas subyacentes de los fallos, mejorando la robustez del programa.

- **Explicación y Comprensión de Conceptos Técnicos:** Para una mejor comprensión de los fundamentos teóricos y las complejidades de las bibliotecas utilizadas (como TensorFlow, Keras o Scikit-learn), la IA ha proporcionado explicaciones

detalladas sobre conceptos, funcionalidades de API y buenas prácticas de programación. Esto ha sido especialmente útil para reforzar el conocimiento sobre el Análisis de Componentes Principales (PCA) y el funcionamiento de las redes neuronales.

- **Optimización del Rendimiento:** La IA también ha colaborado en la sugerencia de mejoras para la eficiencia del código y la optimización del rendimiento de los modelos, proponiendo ajustes en la arquitectura de la red neuronal o en el preprocesamiento de los datos.

Es importante destacar que, si bien la inteligencia artificial ha sido una herramienta de apoyo valiosa, **todas las decisiones críticas de diseño, implementación y análisis de resultados han sido realizadas y validadas por el autor**, asegurando la originalidad y la comprensión profunda del trabajo desarrollado.

Bibliografía

[1] Red neuronal artificial. https://es.wikipedia.org/wiki/Red_neuronal_artificial, 2024. Última modificación: 20 de junio de 2024. (page 23).

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. (page 26).

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. (page 24).

[4] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2 edition, 1999. (page 24).

[5] N. S. Keskar, N. Mudigonda, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1706.05929*, 2016. (page 27).

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (page 27).

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. URL https://www.openml.org/d/554. (page 33).

[8] D. Masters and P. Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018. (page 26).

[9] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010. (page 25).

[10] L. Prechelt. Early stopping—but when? In G. B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 55–69. Springer, 1998. (page 27).

[11] scikit-learn developers. Digits dataset. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html, 2024. URL

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html. Accessed: 22 June 2025. (page 33).

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent overfitting. *Journal of Machine Learning Research*, 15 (1):1929–1958, 2014. (page 25).

[13] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017. URL https://www.openml.org/d/40996. (page 33).