

**SISTEMAS CONEXIONISTAS.**  
DESARROLLO DE UNA RED  
NEURONAL PARA LA CLASIFICACIÓN  
DE ESPECIES DEL DATASET IRIS

**COMPUTACIÓN NEUROBORROSA**  
**2024**

MARTA DE CASTRO LEIRA  
32718800N

## ÍNDICE:

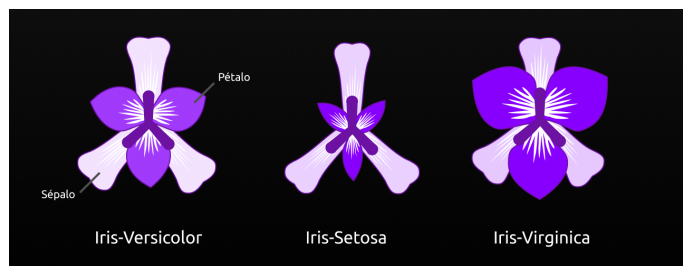
1. Introducción .....	3
2. Metodología .....	3
3. Herramientas .....	5
4. Resultados .....	5
4.1. Métricas de rendimiento en el conjunto de prueba .....	5
4.2. Evaluación de la precisión .....	9
4.3. Evaluación de la pérdida .....	11
4.4. Comparación de resultados .....	12
5. Conclusiones .....	13
6. Líneas Futuras .....	14
ANEXO: Código .....	15

## 1. INTRODUCCIÓN

Este trabajo explora la aplicación práctica de las redes neuronales, un enfoque basado en sistemas conexionistas, para abordar un problema de clasificación supervisada.

Los sistemas conexionistas son modelos computacionales inspirados en la estructura y funcionamiento del cerebro humano, donde el procesamiento de la información se lleva a cabo mediante la interacción de unidades simples, neuronas. Y una red neuronal es un tipo de sistema conexionista, en donde las neuronas están organizadas en capas (entrada, oculta y salida), conectadas mediante pesos sinápticos, permitiendo a la red neuronal reconocer patrones y realizar tareas como la predicción o clasificación.

Se emplea el dataset Iris, ampliamente reconocido en el ámbito del aprendizaje automático, para clasificar especies de flores (Setosa, Versicolor, Virginica) basándose en cuatro características: longitud y ancho de los sépalos, y longitud y ancho de los pétalos.



La clase setosa tiene los pétalos más cortos y anchos, y los sépalos más uniformes. La versicolor presenta pétalos de longitud intermedia y mayor variabilidad en los sépalos, mientras que la virginica tiene los pétalos más largos y estrechos, con sépalos ligeramente más grandes.

Este informe tiene como propósito:

- Clasificar correctamente las especies de flores del dataset Iris utilizando una red neuronal.
- Diseñar y entrenar un modelo de red neuronal.
- Experimentar con diferentes configuraciones en la capa oculta (4, 8, 16 y 32 neuronas) para evaluar su rendimiento.
- Analizar los resultados obtenidos mediante precisión, pérdida y matriz de confusión para identificar la configuración más eficiente.

## 2. METODOLOGÍA

Para abordar el problema de clasificación, el proceso comenzó con la selección del dataset Iris, ya mencionado en la introducción. Este conjunto de datos contiene 150 instancias divididas en tres clases: Versicolor, Setosa y Virginica, cada una descrita por cuatro características clave: longitud y ancho de sépalos, y longitud y ancho de pétalos.

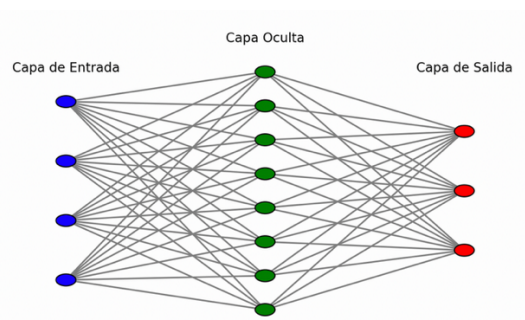
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa

Posteriormente se dividen los datos en dos conjuntos, un conjunto de entrenamiento (70%) y un conjunto de prueba (30%) garantizando que el modelo sea capaz de aprender patrones generales en los datos y no se limite únicamente a memorizar los ejemplos utilizados durante el entrenamiento.

Para preparar los datos, se realizó un preprocesamiento inicial en el que las etiquetas de clase (Setosa, Versicolor y Virginica) se transformaron mediante one-hot encoding, representando cada categoría como un vector binario en tres dimensiones. Este formato es necesario para que el modelo pueda procesar las clases como posibles salidas.

El diseño de la red neuronal sigue esta arquitectura:

- Capa de entrada: con 4 neuronas que son las 4 características del dataset como entrada.
- Capa oculta: contiene un número variable de neuronas (4, 8, 16 o 32) para probar con diferentes configuraciones la respuesta.
- Capa de salida: está compuesta por 3 neuronas, una para cada clase del dataset (setosa, versicolor, virginica).
- Capa Dropout: esta capa adicional se añade para evitar el sobreajuste durante el entrenamiento. Se incluye con un valor de 0,2 lo que hace el 20 % de las neuronas se desactiven aleatoriamente.



Antes de que el modelo empiece a aprender se debe definir cómo se va a optimizar y como va a medir su rendimiento. Para lo que se usó el optimizador Adam, que ajusta los pesos de la red de manera eficiente durante el entrenamiento. También se empleó categorical crossentropy como función de pérdida, que mide que tan bien o mal hace las predicciones el modelo. Y como métrica se utilizó accuracy que indica el porcentaje de las predicciones que son correctas.

El modelo se entrena durante 100 épocas, lo que quiere decir que el modelo va a pasar por todo el conjunto de datos 100 veces para aprender de él. Durante este proceso, los datos se dividen en lotes de 10 muestras para mejorar la eficiencia, ya que el modelo no procesa todos los datos a la vez. Además, del conjunto de

entrenamiento, se reserva un 20% de los datos para validación, lo que permite evaluar el rendimiento del modelo mientras aprende y asegurarse de que no se sobreajuste a los datos de entrenamiento.

Para evaluar el rendimiento del modelo, se calcularon métricas como la precisión global, se analizaron las matrices de confusión para observar los errores entre clases, y se trazaron curvas de pérdida y precisión en entrenamiento y validación a lo largo de las épocas. Por otra parte, se llevaron a cabo experimentos con diferentes configuraciones del número de neuronas en la capa oculta, identificando la configuración que ofrecía los mejores resultados en términos de precisión.

Finalmente se ofrecieron visualizaciones que mostraron la evolución de las métricas clave, así como los ejemplos de predicciones y análisis de la matriz de confusión, permitiendo obtener una interpretación más clara del rendimiento del modelo.

### **3. HERRAMIENTAS**

Las herramientas empleadas para el desarrollo e implementación del proyecto fueron diversas y facilitaron el trabajo con redes neuronales y el análisis de datos. En primer lugar, el lenguaje de programación utilizado fue Python, debido a su versatilidad y robustez, además de ser una herramienta ampliamente adoptada en el campo del aprendizaje automático.

Para la construcción y entrenamiento de las redes neuronales se empleó la librería de Tensorflow y Keras. La biblioteca Scikit-learn se empleó para cargar el dataset de iris, dividir los conjuntos de entrenamiento y prueba, y evaluar diversas métricas de rendimiento como la precisión y la matriz de confusión.

Por último, para la visualización de resultados se usaron Matplotlib y Seaborn, herramientas gráficas que permitieron representar de manera clara y precisa las métricas de rendimiento y mostrar la matriz de confusión.

### **4. RESULTADOS**

En esta sección se presentan y analizan los resultados obtenidos al evaluar la red neuronal artificial (RNA) con configuraciones de 4, 8, 16 y 32 neuronas ocultas. Aunque se realizaron múltiples ejecuciones, los análisis mostrados corresponden a una ejecución específica seleccionada para garantizar coherencia entre los diferentes resultados reportados. Se incluyen métricas como precisión, recall y f1-score, junto con la matriz de confusión, para evaluar el desempeño del modelo en cada clase. Además, se analiza la evolución de la precisión y la pérdida durante el entrenamiento y la validación.

#### **4.1. MÉTRICAS DE RENDIMIENTO EN EL CONJUNTO DE PRUEBA**

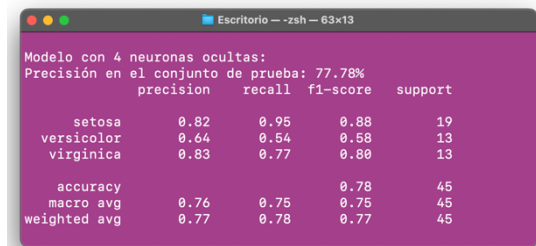
A continuación, se detallan las métricas de rendimiento obtenidas en el conjunto de prueba para cada configuración, destacando el comportamiento del modelo

en la clasificación de las instancias y su desempeño general. La matriz de confusión permite visualizar los aciertos y errores específicos para cada clase:

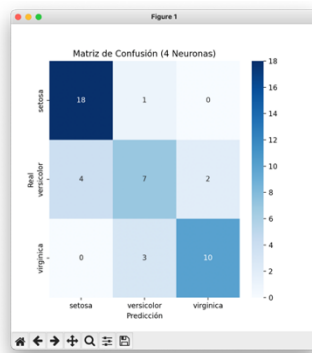
### Modelo con 4 neuronas ocultas:

Precisión global: 77.78%.

- **Setosa:** 18 de 19 instancias clasificadas correctamente, con una precisión del 82% y un f1-score de 88%, lo que refleja que esta clase es la más fácil de identificar.
- **Versicolor:** Solo 7 instancias clasificadas correctamente; 4 fueron mal clasificadas como setosa y 2 como virginica, evidenciando una alta confusión, principalmente con la clase setosa.
- **Virginica:** 10 instancias clasificadas correctamente, pero 3 fueron confundidas con versicolor.



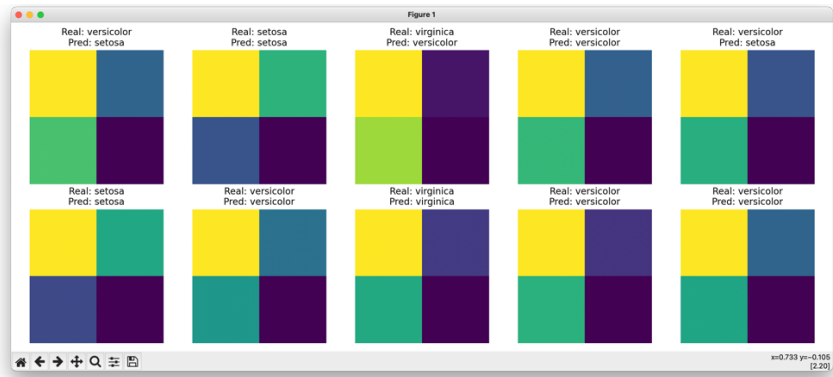
```
Modelo con 4 neuronas ocultas:
Precisión en el conjunto de prueba: 77.78%
precision recall f1-score support
setosa      0.82    0.95    0.88    19
versicolor  0.64    0.54    0.58    13
virginica    0.83    0.77    0.80    13
accuracy    0.76    0.75    0.75    45
macro avg   0.76    0.75    0.75    45
weighted avg 0.77    0.78    0.77    45
```



La clase setosa obtuvo el mejor desempeño, indicando que el modelo es más preciso al clasificarla. Sin embargo, las clases versicolor y virginica presentaron un desempeño más bajo. En particular, la clase versicolor mostró un recall de 54%, lo que indica que muchas instancias de esta clase no fueron correctamente identificadas.

A continuación, se muestran ejemplos de las predicciones realizadas por el modelo. Cada bloque representa una muestra del conjunto de prueba. En cada matriz, se indica la clase real y la clase predicha por el modelo. Las matrices de color tienen un formato de 2x2, donde cada celda representa una de las cuatro características del dataset Iris. Los colores en las celdas reflejan los valores de estas características, donde tonos más claros indican valores más altos y tonos más oscuros indican valores más bajos.

La coincidencia entre la clase real y la predicha significa una predicción correctamente, mientras que una diferencia representa un error. Este análisis permite observar errores comunes en la clasificación de las clases versicolor y setosa, como se indicó en el análisis anterior, lo que refleja una alta confusión entre estas dos clases. En cambio, la clase setosa presenta un desempeño superior, con predicciones mayormente correctas.



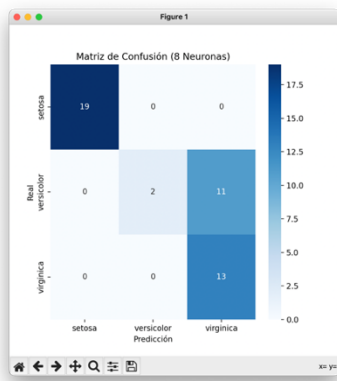
## Modelo con 8 neuronas ocultas:

Precisión global: 75.56%.

- **Setosa:** El modelo logra 100% de precisión y recall en esta clase, con las 19 instancias correctamente clasificadas.
- **Versicolor:** La mayoría de las instancias de versicolor son clasificadas incorrectamente como virginica (11 de 13 instancias). Solo 2 fueron correctamente identificadas.
- **Virginica:** La clase virginica alcanza 100% de precisión, con 13 instancias correctamente clasificadas.

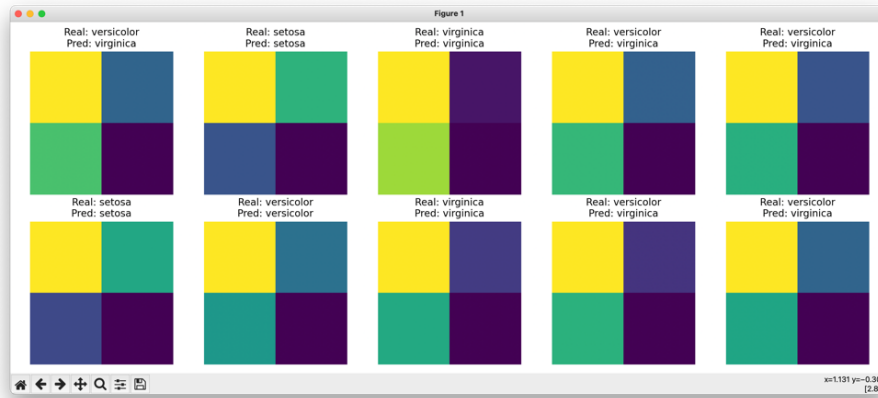
Modelo con 8 neuronas ocultas:  
Precisión en el conjunto de prueba: 75.56%

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	19
versicolor	1.00	0.15	0.27	13
virginica	0.54	1.00	0.70	13
accuracy			0.76	45
macro avg	0.85	0.72	0.66	45
weighted avg	0.87	0.76	0.70	45



Aunque la precisión global disminuye ligeramente en comparación con el modelo con 4 neuronas ocultas, la clase setosa mantiene un desempeño perfecto. Sin embargo, versicolor presenta una caída significativa en el rendimiento, con un recall de solo 15%, lo que sugiere un fallo importante al intentar clasificar correctamente esta clase. En cambio, la clase virginica es clasificada con precisión perfecta.

En los ejemplos de esta red, se observa que varias instancias de la clase versicolor son clasificadas incorrectamente como virginica, mientras que las clases setosa y virginica mantienen una clasificación precisa, como nos indicaban los análisis.



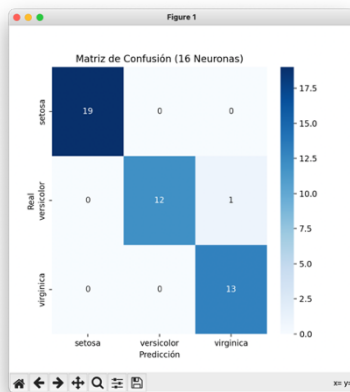
## Modelo con 16 neuronas ocultas:

Precisión global: 97.78%.

- **Setosa:** Mantiene 100% de precisión y recall, con todas las instancias correctamente clasificadas.
- **Versicolor:** 12 instancias clasificadas correctamente y solo 1 mal clasificada como virginica.
- **Virginica:** 100% de precisión, con 13 instancias clasificadas correctamente.

Modelo con 16 neuronas ocultas:  
Precisión en el conjunto de prueba: 97.78%

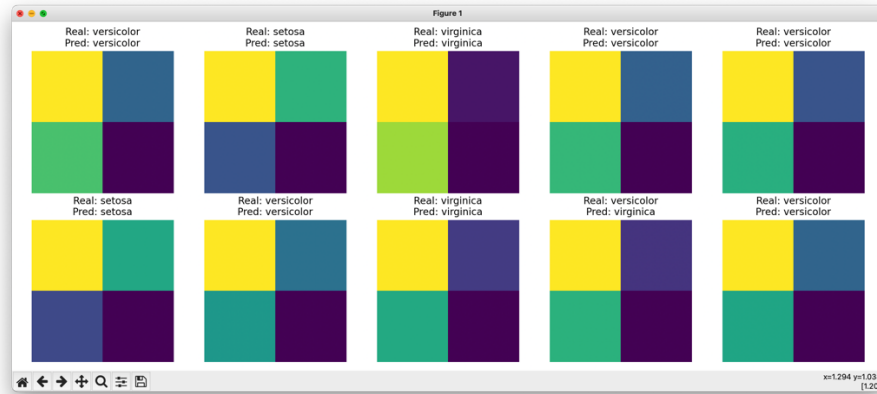
	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	19
versicolor	1.00	0.92	0.96	13
virginica	0.93	1.00	0.96	13
accuracy			0.98	45
macro avg	0.98	0.97	0.97	45
weighted avg	0.98	0.98	0.98	45



Se observa una mejora significativa en todas las métricas con respecto a las configuraciones anteriores. La clase setosa mantiene un desempeño perfecto, mientras que las clases versicolor y virginica también logran valores altos en precisión y recall, superando el 90% en ambas métricas. Esto indica que el modelo está comenzando a captar de manera más efectiva la complejidad de los datos.

En los ejemplos de este modelo, la clase versicolor se clasifica correctamente con mayor frecuencia, solo falla 1, mientras que las clases setosa y virginica mantienen una predicción perfecta. Este comportamiento refleja una mejora notable en la capacidad del modelo para distinguir entre las tres clases.





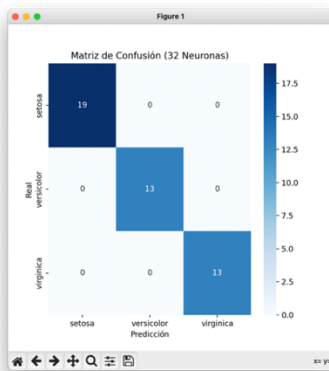
## Modelo con 32 neuronas ocultas:

Precisión global: 100%.

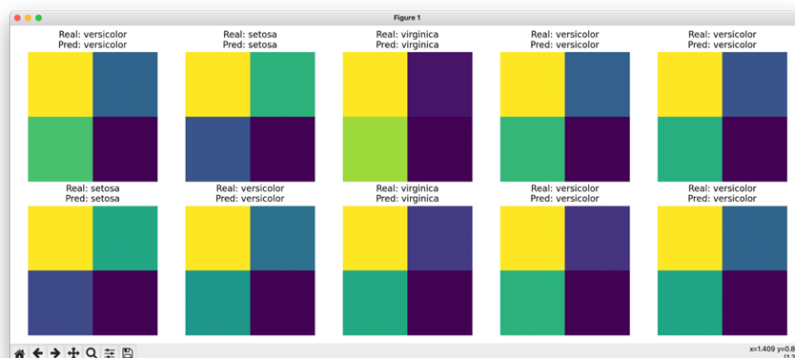
- **Setosa:** Mantiene 100% de precisión, con 19 instancias correctamente clasificadas.
- **Versicolor:** La precisión mejora completamente, alcanzando 13 de 13 instancias correctamente clasificadas, sin confusión con otras clases.
- **Virginica:** También alcanza 100% de precisión, con 13 instancias correctamente clasificadas.

Modelo con 32 neuronas ocultas:  
Precisión en el conjunto de prueba: 100.00%

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	19
versicolor	1.00	1.00	1.00	13
virginica	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45



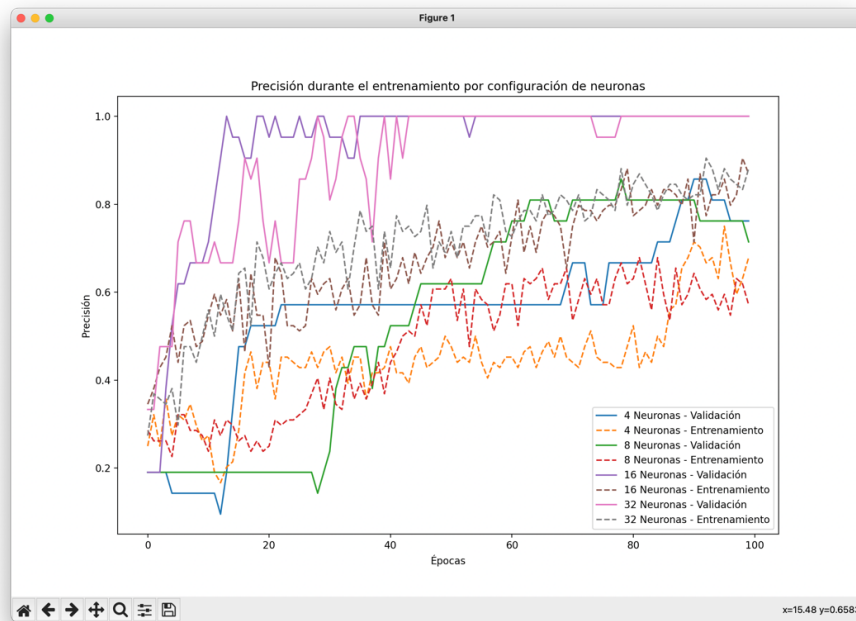
Este modelo alcanza un desempeño perfecto en todas las métricas, con 100% de precisión, recall y f1-score para todas las clases. En los ejemplos de este modelo, las tres clases (setosa, versicolor y virginica) son clasificadas correctamente en su totalidad, reflejando un desempeño perfecto del modelo sin errores de predicción.



## 4.2. EVALUACIÓN DE LA PRECISIÓN

La precisión es una métrica que mide el porcentaje de predicciones correctas realizadas por el modelo, comparando sus resultados con los valores reales. La evolución de la precisión a lo largo de las 100 épocas muestra cómo el modelo mejora en función del número de neuronas ocultas:

- 4 neuronas ocultas:
  - **Validación** (color azul): La precisión de validación comienza con valores bajos y aumenta de forma gradual y lenta a lo largo de las épocas, alcanzando un punto de estabilización alrededor del 60% - 70%.
  - **Entrenamiento** (color naranja): La precisión en entrenamiento también inicia con valores bajos, pero crece de manera continua y más rápida que en validación, aunque con oscilaciones visibles.
- 8 neuronas ocultas:
  - **Validación** (color verde): La precisión de validación comienza a mejorar sobre la época 27. A medida que avanzan las épocas, la línea tiene leves ascensos, pero no logra un progreso estable ni constante.
  - **Entrenamiento** (color rojo): La línea de entrenamiento presenta una tendencia ascendente inicial, pero no alcanza niveles elevados de precisión, se mantiene entorno al 60%.
- 16 neuronas ocultas:
  - **Validación** (color morado): mejora de forma notable en las primeras épocas y mantiene un comportamiento más estable. La línea asciende progresivamente hasta acercarse a valores cercanos al 90% - 95%, reflejando un aprendizaje efectivo.
  - **Entrenamiento** (color marrón): comienza con valores alrededor de 40% y muestra un crecimiento progresivo y oscilaciones a lo largo de las épocas. Al final, se estabiliza alrededor del 85% - 90%.
- 32 neuronas ocultas:
  - **Validación** (color rosa): La precisión de validación asciende en las primeras épocas, alcanzando rápidamente valores altos y finalmente alcanzando el **100%**, lo que refleja un ajuste perfecto a los datos de validación.
  - **Entrenamiento** (color gris): no alcanza el 100%. Se estabiliza alrededor del 85% - 90%, lo que indica un buen rendimiento, aunque con margen de mejora en el aprendizaje del modelo.



En general, se observa que el aumento en el número de neuronas ocultas mejora significativamente la precisión del modelo, especialmente en la validación. Las configuraciones con 16 y 32 neuronas destacan por su estabilidad y alto rendimiento, evidenciando una mayor capacidad de generalización del modelo.

### 4.3. EVALUACIÓN DE LA PÉRDIDA

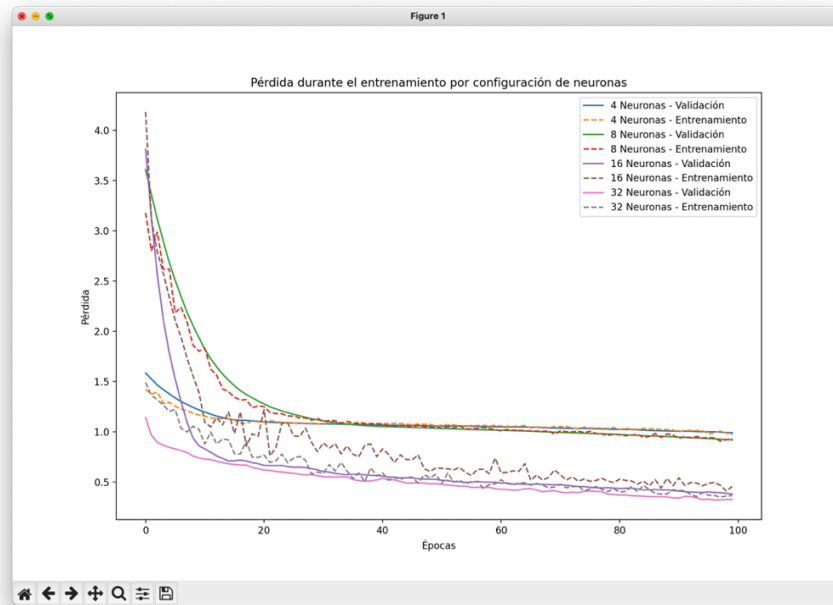
La pérdida, en el contexto del entrenamiento de redes neuronales, es una métrica que mide el error entre las predicciones del modelo y los valores reales. Una alta pérdida indica que el modelo está lejos de hacer predicciones precisas, mientras que una pérdida baja refleja que el modelo está aprendiendo correctamente y ajustándose bien a los datos.

La evolución de la pérdida en las 100 épocas refleja cómo el modelo reduce el error en función del número de neuronas ocultas:

- 4 neuronas ocultas:
  - **Validación** (color azul) y **Entrenamiento** (color naranja): la pérdida comienza en valores alrededor al 1.5 y desciende gradualmente a lo largo de las épocas, pero no logra estabilizarse por debajo de 1.0.
- 8 neuronas ocultas:
  - **Validación** (color verde) y **Entrenamiento** (color rojo): la pérdida en comienza en valores entre 3 y 3.5 y disminuye rápidamente en las primeras épocas. Sin embargo, su descenso se ralentiza y se estabiliza cerca de 1.0.
- 16 neuronas ocultas:
  - **Validación** (color morado) y **Entrenamiento** (color marrón): muestran un descenso notable desde valores altos en las primeras épocas, alcanzando valores cercanos a 0.5 y manteniéndose estable al final. El entrenamiento se comporta de

manera más eficiente en comparación con configuraciones con menos neuronas.

- 32 neuronas ocultas:
  - **Validación** (color rosa) y **Entrenamiento** (color gris): inicia con valores bajos y desciende rápidamente en las primeras épocas, estabilizándose en torno a 0.3 y 0.4, lo que indica que el modelo ha logrado aprender de manera eficiente y sin sobreajuste evidente.



A medida que aumenta el número de neuronas ocultas, la pérdida disminuye más rápidamente y alcanza valores más bajos. Las configuraciones con 16 y 32 neuronas muestran un comportamiento superior, con pérdidas estabilizadas alrededor de 0.5 y 0.3, respectivamente. En contraste, las configuraciones de 4 y 8 neuronas mantienen pérdidas más altas (~1.0), indicando un rendimiento limitado y menor capacidad de ajuste del modelo.

#### 4.4. COMPARACIÓN DE RESULTADOS

El desempeño del modelo mejora a medida que se incrementa el número de neuronas ocultas. Con 4 neuronas, la precisión global es limitada (77.78%) debido a una alta confusión entre las clases versicolor y virginica, mientras que la clase setosa destaca con buenos resultados. Con 8 neuronas, aunque setosa mantiene un 100% de precisión, la clasificación de versicolor empeora significativamente. Al aumentar a 16 neuronas, la precisión global sube a 97.78%, con un desempeño casi perfecto en todas las clases, destacando una notable reducción en las confusiones. Finalmente, con 32 neuronas, el modelo alcanza una precisión perfecta del 100%, clasificando correctamente todas las instancias de cada clase, lo que refleja la capacidad del modelo para generalizar y aprender completamente las características de los datos.

```

Modelo con 4 neuronas ocultas:
Precisión en el conjunto de prueba: 77.78%
precision    recall  f1-score   support

 setosa      0.82    0.95    0.88     19
 versicolor  0.64    0.54    0.58     13
 virginica   0.83    0.77    0.80     13

 accuracy    0.76
 macro avg   0.75
 weighted avg 0.77

2/2 [=====] - 0s 1ms/step

Modelo con 8 neuronas ocultas:
Precisión en el conjunto de prueba: 75.56%
precision    recall  f1-score   support

 setosa      1.00    1.00    1.00     19
 versicolor  1.00    0.15    0.27     13
 virginica   0.54    1.00    0.70     13

 accuracy    0.76
 macro avg   0.85
 weighted avg 0.87

2/2 [=====] - 0s 1ms/step

Modelo con 16 neuronas ocultas:
Precisión en el conjunto de prueba: 97.78%
precision    recall  f1-score   support

 setosa      1.00    1.00    1.00     19
 versicolor  1.00    0.92    0.96     13
 virginica   0.93    1.00    0.96     13

 accuracy    0.98
 macro avg   0.97
 weighted avg 0.98

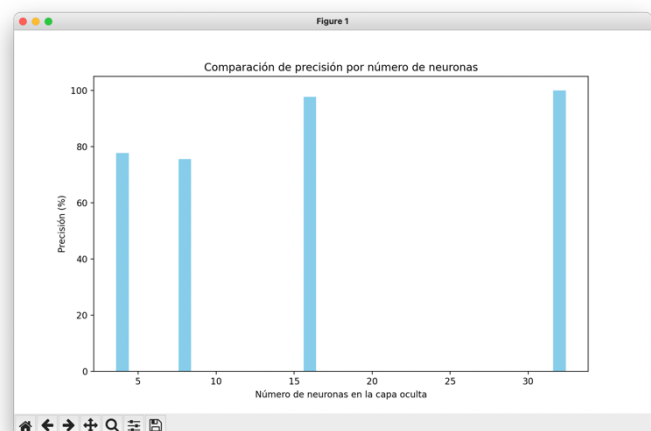
2/2 [=====] - 0s 1ms/step

Modelo con 32 neuronas ocultas:
Precisión en el conjunto de prueba: 100.00%
precision    recall  f1-score   support

 setosa      1.00    1.00    1.00     19
 versicolor  1.00    1.00    1.00     13
 virginica   1.00    1.00    1.00     13

 accuracy    1.00
 macro avg   1.00
 weighted avg 1.00

```



## 5. CONCLUSIONES

Los modelos con 4 y 8 neuronas ocultas presentaron limitaciones significativas, en la ejecución analizada en la clasificación de las clases versicolor y virginica. Por otra parte, los modelos con 16 y 32 neuronas ocultas demostraron un rendimiento superior. Aunque en algunas ejecuciones el modelo con 32 neuronas alcanzó una precisión del 100% (como la analizada en el apartado de resultados), otras ejecuciones revelaron que el modelo con 16 neuronas puede tender a ser más consistente en términos de desempeño global. Ambos modelos destacan por lograr altos valores en precisión, recall y f1-score para todas las clases. La pérdida disminuye más rápidamente y alcanza valores más bajos a medida que aumenta el número de neuronas ocultas. Los modelos con 16 y 32 neuronas mostraron una estabilización más eficiente en comparación con las configuraciones más simples (4 y 8 neuronas). En conclusión, se determina que las configuraciones con 16 y 32 neuronas ocultas son las más adecuadas para el problema de clasificación analizado, ya que logran capturar las características del conjunto de datos de manera más efectiva.

Este informe tenía como propósito:

- Diseñar y entrenar un modelo de red neuronal: Este objetivo se cumplió al implementar redes neuronales artificiales con diferentes configuraciones de neuronas ocultas y entrenar cada modelo utilizando el dataset Iris.
- Experimentar con diferentes configuraciones en el modelo para evaluar su rendimiento: Se exploraron múltiples configuraciones, demostrando que las redes con 16 y 32 neuronas ocultas ofrecieron el mejor rendimiento en términos de estabilidad y precisión.

- Analizar los resultados obtenidos en términos de precisión, pérdida y matriz de confusión, para identificar la mejor configuración: Se realizó un análisis detallado de los resultados, destacando métricas como precisión, *recall*, *f1-score* y pérdida, concluyendo que las configuraciones con 16 y 32 neuronas son las más adecuadas.

Por tanto, los objetivos planteados al inicio del trabajo se han cumplido satisfactoriamente, permitiendo validar la eficacia de las redes neuronales para la clasificación supervisada en el dataset Iris.

## **6. LINEAS FUTURAS**

A partir de los resultados obtenidos en este estudio sobre la clasificación de especies del dataset Iris mediante redes neuronales, se identifican diversas oportunidades de mejora y expansión del trabajo. Las líneas futuras propuestas son las siguientes: comparar el rendimiento de las redes neuronales con otros algoritmos clásicos de clasificación, como Máquinas de Vectores de Soporte (SVM), eficaces para datos de baja dimensión; Árboles de Decisión y métodos ensamblados como Random Forest y Gradient Boosting; y k-Nearest Neighbors (k-NN) como modelo base de comparación. Asimismo, se propone evaluar cómo diferentes técnicas de preprocesamiento afectan el rendimiento del modelo, considerando la normalización y estandarización de características, así como la reducción de dimensionalidad mediante algoritmos como PCA (Análisis de Componentes Principales).

## Anexo: Código Fuente

```
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Cargar el dataset Iris
iris = load_iris()
X = iris.data # Características (4 características por muestra)
y = iris.target # Etiquetas (3 clases: setosa, versicolor, virginica)

# 2. Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# 3. Codificar las etiquetas en formato one-hot
y_train = tf.keras.utils.to_categorical(y_train, num_classes=3)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=3)

# Función para crear, entrenar y evaluar modelos
def build_and_evaluate_model(hidden_neurons):
    model = Sequential([
        Dense(hidden_neurons, input_dim=4, activation='relu'),
        Dropout(0.2),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=100, batch_size=10,
validation_split=0.2, verbose=0)
    y_pred = model.predict(X_test)
    y_pred_classes = tf.argmax(y_pred, axis=1).numpy()
    y_test_classes = tf.argmax(y_test, axis=1).numpy()

    accuracy = accuracy_score(y_test_classes, y_pred_classes)

    print(f"\nModelo con {hidden_neurons} neuronas ocultas:")
    print(f"Precisión en el conjunto de prueba: {accuracy * 100:.2f}%")
    print(classification_report(y_test_classes, y_pred_classes,
target_names=iris.target_names, zero_division=1))
```

```

#matriz de confusión
cm = confusion_matrix(y_test_classes, y_pred_classes)
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title(f'Matriz de Confusión ({hidden_neurons} Neuronas)')
plt.show()

#10 predicciones con imshow
fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i].reshape(2, 2), cmap='viridis')
    pred_label = iris.target_names[y_pred_classes[i]]
    true_label = iris.target_names[y_test_classes[i]]
    ax.set_title(f'Real: {true_label}\nPred: {pred_label}')
    ax.axis('off')
plt.tight_layout()
plt.show()

return model, history, accuracy

# Configuraciones de neuronas a probar
neuron_configs = [4, 8, 16, 32]
histories = {}
accuracies = []

# Entrenar y evaluar modelos
for neurons in neuron_configs:
    model, history, accuracy = build_and_evaluate_model(neurons)
    histories[neurons] = history
    accuracies.append(accuracy)

# Visualizar precisión de las diferentes configuraciones
plt.figure(figsize=(10, 6))
plt.bar(neuron_configs, [acc * 100 for acc in accuracies], color='skyblue')
plt.xlabel('Número de neuronas en la capa oculta')
plt.ylabel('Precisión (%)')
plt.title('Comparación de precisión por número de neuronas')
plt.show()

# Visualización de la precisión durante el entrenamiento
plt.figure(figsize=(12, 8))
for neurons, history in histories.items():
    plt.plot(history.history['val_accuracy'], label=f'{neurons} Neuronas - Validación')

```



```

plt.plot(history.history['accuracy'], linestyle='dashed', label=f'{neurons}
Neuronas - Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.title('Precisión durante el entrenamiento por configuración de neuronas')
plt.legend()
plt.show()

# Visualización de la pérdida durante el entrenamiento
plt.figure(figsize=(12, 8))
for neurons, history in histories.items():
    plt.plot(history.history['val_loss'], label=f'{neurons} Neuronas - Validación')
    plt.plot(history.history['loss'], linestyle='dashed', label=f'{neurons} Neuronas -
Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.title('Pérdida durante el entrenamiento por configuración de neuronas')
plt.legend()
plt.show()

# Guardar el modelo con la mejor configuración
# best_neurons = neuron_configs[accuracies.index(max(accuracies))]
# best_model, _, _ = build_and_evaluate_model(best_neurons)
# best_model.save('modelo_iris_mejor.keras')

```