Predicción del Ganador de una Partida de Catan utilizando Redes Neuronales

Max Gallardo

A01783128

Inteligencia Artificial Avanzada para la Ciencia de Datos I (TC3006C)

Profesor: Dr. Esteban Castillo Juarez

12.09.24

Introducción

El juego de mesa Catan, creado por Klaus Teuber en 1995, es uno de los juegos de estrategia más populares del mundo, donde los jugadores compiten para colonizar una isla ficticia obteniendo recursos y construyendo caminos, asentamientos y ciudades. Cada jugador lanza dados para obtener recursos de las tierras que controla, y utiliza estos recursos para obtener puntos de victoria de diversas maneras: construyendo asentamientos (1 punto cada uno), construyendo ciudades (2 puntos cada una), obteniendo cartas de desarrollo con puntos de victoria, o logrando ciertos objetivos, como tener la carretera más larga o el ejército más grande. El primer jugador en alcanzar 10 puntos de victoria gana la partida.

Catan es un juego en el que la estrategia y la planificación juegan un papel fundamental. Las decisiones iniciales del juego, como la posición de los primeros asentamientos y la distribución de los recursos, tienen un impacto significativo en el resultado de la partida. Estas decisiones iniciales representan un desafío complejo para los jugadores, quienes deben considerar múltiples factores, incluyendo la probabilidad de obtener ciertos recursos en función de los valores de los dados y la posición de los asentamientos en el tablero.

Dada la popularidad y la complejidad del juego, se presenta un desafío interesante: ¿podemos predecir quién ganará la partida de Catan basándonos únicamente en las posiciones iniciales de los asentamientos y los recursos que cada jugador obtiene al principio del juego? Para abordar este problema, utilizamos técnicas de ciencia de datos y aprendizaje profundo, específicamente una red neuronal profunda, con el objetivo de desarrollar un modelo que pueda hacer predicciones precisas sobre el resultado de una partida de Catan.

Este reporte documenta el desarrollo de dicho modelo, los procesos de preprocesamiento de los datos, la construcción de la red neuronal, la evaluación del modelo, y la implementación de una interfaz web utilizando Flask para permitir a los usuarios ingresar los datos iniciales de la partida y obtener una predicción sobre el ganador. El uso de técnicas avanzadas, como la regularización, el balanceo de clases mediante SMOTE, y el Early Stopping, resultaron fundamentales para mejorar la precisión del modelo y evitar el sobreajuste.

Metodología

Para desarrollar un modelo que pueda predecir si un jugador ganará o no una partida de Catan, se utilizó una red neuronal profunda. A continuación se detallan los principales pasos y técnicas empleadas en el desarrollo del modelo:

1. Preprocesamiento de los Datos

El dataset utilizado contiene información sobre las posiciones iniciales de los asentamientos de los jugadores y los recursos que obtienen al principio del juego. El preprocesamiento de los datos incluyó las siguientes tareas:

- **Selección de Características**: Se seleccionaron como características los valores de los dados de los dos primeros asentamientos de cada jugador, así como los recursos obtenidos (madera, trigo, arcilla, ovejas, piedra).
- **Balanceo de Clases**: El conjunto de datos original estaba desequilibrado, con más ejemplos de jugadores que no ganaron la partida. Para corregir este desbalance, se utilizó la técnica SMOTE (Synthetic Minority Over-sampling Technique), que genera ejemplos sintéticos de la clase minoritaria.
- **Estandarización**: Las características numéricas fueron escaladas utilizando la técnica de estandarización (media 0 y desviación estándar 1), lo que mejora el rendimiento de las redes neuronales.

2. Construcción de la Red Neuronal

Se implementó una red neuronal profunda con tres capas ocultas, cada una con regularización L2 para prevenir el sobreajuste. También se utilizó la técnica de Dropout para desactivar aleatoriamente un porcentaje de las neuronas durante el entrenamiento, lo que permite que el modelo generalice mejor a nuevos datos. El modelo fue entrenado utilizando la función de pérdida de entropía cruzada y el optimizador Adam.

3. Entrenamiento y Evaluación del Modelo

Para entrenar el modelo, se utilizaron 1000 épocas con Early Stopping, lo que detuvo el entrenamiento cuando no se observaron mejoras en el conjunto de validación durante 20 épocas consecutivas. Se evaluó el modelo utilizando las métricas de precisión, recall, F1-Score y la curva ROC.

A continuación se presenta el código correspondiente al preprocesamiento y construcción del modelo.

In [1]:
```
import pandas as pd
import numpy as np
import os  # Para obtener la ruta del escritorio
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers # type: ignore
from tensorflow.keras.optimizers.legacy import Adam # type: ignore
```

```
import seaborn as sns
import joblib

# Definir la ruta para guardar las imágenes en el escritorio
desktop_path = os.path.expanduser('~/Desktop')

# Cargar los datos
df = pd.read_csv('/Users/maxgallardo/Documents/TEC/Semestres/Semestre 7/TC30

# Preprocesamiento de datos
X = df[['settlement1_dice1', 'settlement1_dice2', 'settlement1_dice3',
        'settlement2_dice1', 'settlement2_dice2', 'settlement2_dice3',
        'num_lumber', 'num_wheat', 'num_clay', 'num_sheep', 'num_ore',
        'num_3G', 'num_2(X)', 'num_D']]
y = df['winner']

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Aplicar SMOTE para balancear las clases en los datos de entrenamiento
sm = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = sm.fit_resample(X_train, y_train)

# Estandarizar los datos
scaler = StandardScaler()
X_train_resampled = scaler.fit_transform(X_train_resampled)
X_test = scaler.transform(X_test)

# Ajuste de las clases ponderadas (class_weight) para manejar el desequilibr
class_weight = {0: 1.0, 1: 4}

# Construcción del modelo de red neuronal con 4 capas ocultas y regularizaci
model = models.Sequential()

# Capa de entrada
model.add(layers.Dropout(0.1, input_shape=(X_train_resampled.shape[1],)))

# Primera capa oculta con regularización L2 y Dropout
model.add(layers.Dense(128, activation='relu', kernel_regularizer=regularize
model.add(layers.Dropout(0.3))

# Segunda capa oculta con regularización L2 y Dropout
model.add(layers.Dense(64, activation='relu', kernel_regularizer=regularizer
model.add(layers.Dropout(0.3))

# Tercera capa oculta con regularización L2 y Dropout
model.add(layers.Dense(32, activation='relu', kernel_regularizer=regularizer
model.add(layers.Dropout(0.3))

# Capa de salida (clasificación binaria)
model.add(layers.Dense(1, activation='sigmoid'))

# Compilación del modelo con el optimizador legacy Adam
adam_optimizer = Adam(learning_rate=0.0005)
model.compile(optimizer=adam_optimizer, loss='binary_crossentropy', metrics=
```

```
# Early stopping para detener el entrenamiento cuando no haya mejoras
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patier

# Entrenamiento del modelo con class_weight
history = model.fit(X_train_resampled, y_train_resampled, epochs=1000, batch
                    validation_split=0.2, callbacks=[early_stopping], class_

# Evaluación del modelo
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy}')
```

```
Epoch 1/1000
19/19 [==============================] - 0s 5ms/step - loss: 1.3896 - accura
cy: 0.4335 - val_loss: 0.5171 - val_accuracy: 1.0000
Epoch 2/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.3510 - accura
cy: 0.3938 - val_loss: 0.4468 - val_accuracy: 1.0000
Epoch 3/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.3374 - accura
cy: 0.3851 - val_loss: 0.4275 - val_accuracy: 1.0000
Epoch 4/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.3252 - accura
cy: 0.4007 - val_loss: 0.4150 - val_accuracy: 1.0000
Epoch 5/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.2893 - accura
cy: 0.4162 - val_loss: 0.3885 - val_accuracy: 1.0000
Epoch 6/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.2458 - accura
cy: 0.4266 - val_loss: 0.3615 - val_accuracy: 1.0000
Epoch 7/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.2557 - accura
cy: 0.4387 - val_loss: 0.3324 - val_accuracy: 1.0000
Epoch 8/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.2154 - accura
cy: 0.4335 - val_loss: 0.3116 - val_accuracy: 1.0000
Epoch 9/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1887 - accura
cy: 0.4560 - val_loss: 0.3133 - val_accuracy: 1.0000
Epoch 10/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.2012 - accura
cy: 0.4698 - val_loss: 0.2881 - val_accuracy: 1.0000
Epoch 11/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1685 - accura
cy: 0.4611 - val_loss: 0.2739 - val_accuracy: 1.0000
Epoch 12/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1655 - accura
cy: 0.4836 - val_loss: 0.2729 - val_accuracy: 1.0000
Epoch 13/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1602 - accura
cy: 0.4784 - val_loss: 0.2463 - val_accuracy: 1.0000
Epoch 14/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1412 - accura
cy: 0.4784 - val_loss: 0.2489 - val_accuracy: 1.0000
Epoch 15/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1194 - accura
cy: 0.5285 - val_loss: 0.2476 - val_accuracy: 1.0000
Epoch 16/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1073 - accura
cy: 0.5475 - val_loss: 0.2236 - val_accuracy: 1.0000
Epoch 17/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.1126 - accura
cy: 0.5009 - val_loss: 0.2153 - val_accuracy: 1.0000
Epoch 18/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0626 - accura
cy: 0.5389 - val_loss: 0.2133 - val_accuracy: 1.0000
Epoch 19/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0450 - accura
```

```
cy: 0.5717 - val_loss: 0.2069 - val_accuracy: 1.0000
Epoch 20/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0469 - accura
cy: 0.5527 - val_loss: 0.1828 - val_accuracy: 1.0000
Epoch 21/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0762 - accura
cy: 0.5717 - val_loss: 0.1983 - val_accuracy: 1.0000
Epoch 22/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0225 - accura
cy: 0.6114 - val_loss: 0.1860 - val_accuracy: 0.9862
Epoch 23/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0520 - accura
cy: 0.5509 - val_loss: 0.1827 - val_accuracy: 0.9862
Epoch 24/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0403 - accura
cy: 0.5855 - val_loss: 0.1780 - val_accuracy: 0.9862
Epoch 25/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0246 - accura
cy: 0.6045 - val_loss: 0.1800 - val_accuracy: 0.9862
Epoch 26/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0480 - accura
cy: 0.5993 - val_loss: 0.1708 - val_accuracy: 0.9862
Epoch 27/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0626 - accura
cy: 0.6097 - val_loss: 0.1641 - val_accuracy: 1.0000
Epoch 28/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0139 - accura
cy: 0.6321 - val_loss: 0.1738 - val_accuracy: 1.0000
Epoch 29/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0043 - accura
cy: 0.6079 - val_loss: 0.1696 - val_accuracy: 0.9862
Epoch 30/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.9756 - accura
cy: 0.6442 - val_loss: 0.1607 - val_accuracy: 0.9862
Epoch 31/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.9057 - accura
cy: 0.6563 - val_loss: 0.1583 - val_accuracy: 0.9862
Epoch 32/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.9761 - accura
cy: 0.6580 - val_loss: 0.1509 - val_accuracy: 1.0000
Epoch 33/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0173 - accura
cy: 0.6459 - val_loss: 0.1512 - val_accuracy: 1.0000
Epoch 34/1000
19/19 [==============================] - 0s 1ms/step - loss: 1.0041 - accura
cy: 0.6477 - val_loss: 0.1469 - val_accuracy: 1.0000
Epoch 35/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.9254 - accura
cy: 0.6753 - val_loss: 0.1590 - val_accuracy: 0.9862
Epoch 36/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9783 - accura
cy: 0.6494 - val_loss: 0.1468 - val_accuracy: 1.0000
Epoch 37/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8913 - accura
cy: 0.6926 - val_loss: 0.1370 - val_accuracy: 1.0000
Epoch 38/1000
```

```
19/19 [==============================] - 0s 2ms/step - loss: 0.9484 - accura
cy: 0.6684 - val_loss: 0.1316 - val_accuracy: 1.0000
Epoch 39/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9805 - accura
cy: 0.6459 - val_loss: 0.1463 - val_accuracy: 1.0000
Epoch 40/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9417 - accura
cy: 0.6857 - val_loss: 0.1396 - val_accuracy: 1.0000
Epoch 41/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9363 - accura
cy: 0.6580 - val_loss: 0.1237 - val_accuracy: 1.0000
Epoch 42/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9497 - accura
cy: 0.6563 - val_loss: 0.1335 - val_accuracy: 1.0000
Epoch 43/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9132 - accura
cy: 0.6978 - val_loss: 0.1299 - val_accuracy: 1.0000
Epoch 44/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9442 - accura
cy: 0.6373 - val_loss: 0.1194 - val_accuracy: 1.0000
Epoch 45/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9062 - accura
cy: 0.6978 - val_loss: 0.1202 - val_accuracy: 1.0000
Epoch 46/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.9248 - accura
cy: 0.6667 - val_loss: 0.1251 - val_accuracy: 1.0000
Epoch 47/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8838 - accura
cy: 0.6926 - val_loss: 0.1448 - val_accuracy: 1.0000
Epoch 48/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8951 - accura
cy: 0.7098 - val_loss: 0.1210 - val_accuracy: 1.0000
Epoch 49/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8750 - accura
cy: 0.6960 - val_loss: 0.1256 - val_accuracy: 1.0000
Epoch 50/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8874 - accura
cy: 0.6839 - val_loss: 0.1145 - val_accuracy: 1.0000
Epoch 51/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8499 - accura
cy: 0.7185 - val_loss: 0.1136 - val_accuracy: 1.0000
Epoch 52/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8694 - accura
cy: 0.7012 - val_loss: 0.1182 - val_accuracy: 1.0000
Epoch 53/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8845 - accura
cy: 0.7081 - val_loss: 0.1115 - val_accuracy: 1.0000
Epoch 54/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8786 - accura
cy: 0.6978 - val_loss: 0.1149 - val_accuracy: 1.0000
Epoch 55/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8735 - accura
cy: 0.6908 - val_loss: 0.1031 - val_accuracy: 1.0000
Epoch 56/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8368 - accura
cy: 0.7081 - val_loss: 0.1009 - val_accuracy: 1.0000
```

```
Epoch 57/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.8937 - accura
cy: 0.6805 - val_loss: 0.1050 - val_accuracy: 1.0000
Epoch 58/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8577 - accura
cy: 0.7081 - val_loss: 0.1005 - val_accuracy: 1.0000
Epoch 59/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8718 - accura
cy: 0.7133 - val_loss: 0.1221 - val_accuracy: 1.0000
Epoch 60/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8101 - accura
cy: 0.7651 - val_loss: 0.1204 - val_accuracy: 1.0000
Epoch 61/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7923 - accura
cy: 0.7288 - val_loss: 0.1077 - val_accuracy: 1.0000
Epoch 62/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8434 - accura
cy: 0.7306 - val_loss: 0.1060 - val_accuracy: 1.0000
Epoch 63/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8163 - accura
cy: 0.7116 - val_loss: 0.1021 - val_accuracy: 1.0000
Epoch 64/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8099 - accura
cy: 0.7271 - val_loss: 0.0992 - val_accuracy: 1.0000
Epoch 65/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7535 - accura
cy: 0.7617 - val_loss: 0.0995 - val_accuracy: 1.0000
Epoch 66/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8187 - accura
cy: 0.7323 - val_loss: 0.1087 - val_accuracy: 1.0000
Epoch 67/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7631 - accura
cy: 0.7392 - val_loss: 0.0996 - val_accuracy: 1.0000
Epoch 68/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7678 - accura
cy: 0.7651 - val_loss: 0.0982 - val_accuracy: 1.0000
Epoch 69/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7677 - accura
cy: 0.7737 - val_loss: 0.0998 - val_accuracy: 1.0000
Epoch 70/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7937 - accura
cy: 0.7582 - val_loss: 0.0984 - val_accuracy: 1.0000
Epoch 71/1000
19/19 [==============================] - 0s 3ms/step - loss: 0.7719 - accura
cy: 0.7513 - val_loss: 0.0949 - val_accuracy: 1.0000
Epoch 72/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8413 - accura
cy: 0.7392 - val_loss: 0.0941 - val_accuracy: 1.0000
Epoch 73/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7509 - accura
cy: 0.7617 - val_loss: 0.0915 - val_accuracy: 1.0000
Epoch 74/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8134 - accura
cy: 0.7530 - val_loss: 0.0932 - val_accuracy: 1.0000
Epoch 75/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7797 - accura
```

```
cy: 0.7634 - val_loss: 0.0910 - val_accuracy: 1.0000
Epoch 76/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7651 - accura
cy: 0.7409 - val_loss: 0.0931 - val_accuracy: 1.0000
Epoch 77/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7931 - accura
cy: 0.7323 - val_loss: 0.0895 - val_accuracy: 1.0000
Epoch 78/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7590 - accura
cy: 0.7150 - val_loss: 0.0873 - val_accuracy: 1.0000
Epoch 79/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.8016 - accura
cy: 0.7444 - val_loss: 0.0903 - val_accuracy: 1.0000
Epoch 80/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7113 - accura
cy: 0.7789 - val_loss: 0.0954 - val_accuracy: 1.0000
Epoch 81/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7422 - accura
cy: 0.7703 - val_loss: 0.0901 - val_accuracy: 1.0000
Epoch 82/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7497 - accura
cy: 0.7547 - val_loss: 0.0783 - val_accuracy: 1.0000
Epoch 83/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7794 - accura
cy: 0.7323 - val_loss: 0.0808 - val_accuracy: 1.0000
Epoch 84/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7572 - accura
cy: 0.7530 - val_loss: 0.0841 - val_accuracy: 1.0000
Epoch 85/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7245 - accura
cy: 0.7444 - val_loss: 0.0899 - val_accuracy: 1.0000
Epoch 86/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7212 - accura
cy: 0.7634 - val_loss: 0.0857 - val_accuracy: 1.0000
Epoch 87/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7388 - accura
cy: 0.7686 - val_loss: 0.0880 - val_accuracy: 1.0000
Epoch 88/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7122 - accura
cy: 0.7997 - val_loss: 0.0840 - val_accuracy: 1.0000
Epoch 89/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7513 - accura
cy: 0.7668 - val_loss: 0.0786 - val_accuracy: 1.0000
Epoch 90/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6956 - accura
cy: 0.7824 - val_loss: 0.0771 - val_accuracy: 1.0000
Epoch 91/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7460 - accura
cy: 0.7530 - val_loss: 0.0700 - val_accuracy: 1.0000
Epoch 92/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7048 - accura
cy: 0.7703 - val_loss: 0.0820 - val_accuracy: 1.0000
Epoch 93/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7488 - accura
cy: 0.7686 - val_loss: 0.0802 - val_accuracy: 1.0000
Epoch 94/1000
```

```
19/19 [==============================] - 0s 1ms/step - loss: 0.6620 - accura
cy: 0.7927 - val_loss: 0.0742 - val_accuracy: 1.0000
Epoch 95/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7078 - accura
cy: 0.7945 - val_loss: 0.0733 - val_accuracy: 1.0000
Epoch 96/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7707 - accura
cy: 0.7737 - val_loss: 0.0753 - val_accuracy: 1.0000
Epoch 97/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7233 - accura
cy: 0.7703 - val_loss: 0.0796 - val_accuracy: 1.0000
Epoch 98/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6521 - accura
cy: 0.7876 - val_loss: 0.0784 - val_accuracy: 1.0000
Epoch 99/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6769 - accura
cy: 0.7979 - val_loss: 0.0772 - val_accuracy: 1.0000
Epoch 100/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6697 - accura
cy: 0.8048 - val_loss: 0.0711 - val_accuracy: 1.0000
Epoch 101/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6574 - accura
cy: 0.8048 - val_loss: 0.0677 - val_accuracy: 1.0000
Epoch 102/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6791 - accura
cy: 0.7997 - val_loss: 0.0643 - val_accuracy: 1.0000
Epoch 103/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7490 - accura
cy: 0.7807 - val_loss: 0.0696 - val_accuracy: 1.0000
Epoch 104/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7333 - accura
cy: 0.7893 - val_loss: 0.0721 - val_accuracy: 1.0000
Epoch 105/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6832 - accura
cy: 0.7979 - val_loss: 0.0683 - val_accuracy: 1.0000
Epoch 106/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6619 - accura
cy: 0.8169 - val_loss: 0.0709 - val_accuracy: 1.0000
Epoch 107/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6561 - accura
cy: 0.7997 - val_loss: 0.0667 - val_accuracy: 1.0000
Epoch 108/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6976 - accura
cy: 0.7651 - val_loss: 0.0712 - val_accuracy: 1.0000
Epoch 109/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7303 - accura
cy: 0.7772 - val_loss: 0.0755 - val_accuracy: 1.0000
Epoch 110/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6604 - accura
cy: 0.7945 - val_loss: 0.0848 - val_accuracy: 1.0000
Epoch 111/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6920 - accura
cy: 0.8117 - val_loss: 0.0756 - val_accuracy: 1.0000
Epoch 112/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6391 - accura
cy: 0.8014 - val_loss: 0.0690 - val_accuracy: 1.0000
```

```
Epoch 113/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6343 - accura
cy: 0.8117 - val_loss: 0.0685 - val_accuracy: 1.0000
Epoch 114/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6539 - accura
cy: 0.7979 - val_loss: 0.0684 - val_accuracy: 1.0000
Epoch 115/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6263 - accura
cy: 0.8117 - val_loss: 0.0710 - val_accuracy: 1.0000
Epoch 116/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6675 - accura
cy: 0.8117 - val_loss: 0.0650 - val_accuracy: 1.0000
Epoch 117/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6256 - accura
cy: 0.8135 - val_loss: 0.0608 - val_accuracy: 1.0000
Epoch 118/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6365 - accura
cy: 0.8117 - val_loss: 0.0693 - val_accuracy: 1.0000
Epoch 119/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6888 - accura
cy: 0.8031 - val_loss: 0.0710 - val_accuracy: 1.0000
Epoch 120/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6697 - accura
cy: 0.7997 - val_loss: 0.0673 - val_accuracy: 1.0000
Epoch 121/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5772 - accura
cy: 0.8290 - val_loss: 0.0663 - val_accuracy: 1.0000
Epoch 122/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6154 - accura
cy: 0.8066 - val_loss: 0.0627 - val_accuracy: 1.0000
Epoch 123/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6269 - accura
cy: 0.8031 - val_loss: 0.0596 - val_accuracy: 1.0000
Epoch 124/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6935 - accura
cy: 0.8031 - val_loss: 0.0677 - val_accuracy: 1.0000
Epoch 125/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5704 - accura
cy: 0.8273 - val_loss: 0.0690 - val_accuracy: 1.0000
Epoch 126/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6866 - accura
cy: 0.8083 - val_loss: 0.0634 - val_accuracy: 1.0000
Epoch 127/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6246 - accura
cy: 0.8117 - val_loss: 0.0621 - val_accuracy: 1.0000
Epoch 128/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5799 - accura
cy: 0.8152 - val_loss: 0.0648 - val_accuracy: 1.0000
Epoch 129/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6050 - accura
cy: 0.8342 - val_loss: 0.0615 - val_accuracy: 1.0000
Epoch 130/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6109 - accura
cy: 0.8152 - val_loss: 0.0611 - val_accuracy: 1.0000
Epoch 131/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5948 - accura
```

```
cy: 0.8394 - val_loss: 0.0697 - val_accuracy: 1.0000
Epoch 132/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5733 - accura
cy: 0.8463 - val_loss: 0.0609 - val_accuracy: 1.0000
Epoch 133/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6206 - accura
cy: 0.8238 - val_loss: 0.0572 - val_accuracy: 1.0000
Epoch 134/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5927 - accura
cy: 0.8256 - val_loss: 0.0590 - val_accuracy: 1.0000
Epoch 135/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6603 - accura
cy: 0.8290 - val_loss: 0.0570 - val_accuracy: 1.0000
Epoch 136/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6010 - accura
cy: 0.8256 - val_loss: 0.0552 - val_accuracy: 1.0000
Epoch 137/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.7046 - accura
cy: 0.7858 - val_loss: 0.0581 - val_accuracy: 1.0000
Epoch 138/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5647 - accura
cy: 0.8342 - val_loss: 0.0581 - val_accuracy: 1.0000
Epoch 139/1000
19/19 [==============================] - 0s 3ms/step - loss: 0.5766 - accura
cy: 0.8359 - val_loss: 0.0595 - val_accuracy: 1.0000
Epoch 140/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5817 - accura
cy: 0.8359 - val_loss: 0.0544 - val_accuracy: 1.0000
Epoch 141/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6055 - accura
cy: 0.8204 - val_loss: 0.0525 - val_accuracy: 1.0000
Epoch 142/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5853 - accura
cy: 0.8325 - val_loss: 0.0550 - val_accuracy: 1.0000
Epoch 143/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5711 - accura
cy: 0.8238 - val_loss: 0.0575 - val_accuracy: 1.0000
Epoch 144/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5732 - accura
cy: 0.8307 - val_loss: 0.0572 - val_accuracy: 1.0000
Epoch 145/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5671 - accura
cy: 0.8411 - val_loss: 0.0609 - val_accuracy: 1.0000
Epoch 146/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6314 - accura
cy: 0.8325 - val_loss: 0.0637 - val_accuracy: 1.0000
Epoch 147/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6423 - accura
cy: 0.8169 - val_loss: 0.0629 - val_accuracy: 1.0000
Epoch 148/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5574 - accura
cy: 0.8480 - val_loss: 0.0572 - val_accuracy: 1.0000
Epoch 149/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5323 - accura
cy: 0.8377 - val_loss: 0.0593 - val_accuracy: 1.0000
Epoch 150/1000
```

```
19/19 [==============================] - 0s 1ms/step - loss: 0.5975 - accura
cy: 0.8359 - val_loss: 0.0554 - val_accuracy: 1.0000
Epoch 151/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6407 - accura
cy: 0.8066 - val_loss: 0.0550 - val_accuracy: 1.0000
Epoch 152/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6148 - accura
cy: 0.8169 - val_loss: 0.0569 - val_accuracy: 1.0000
Epoch 153/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.6019 - accura
cy: 0.8359 - val_loss: 0.0596 - val_accuracy: 1.0000
Epoch 154/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5740 - accura
cy: 0.8497 - val_loss: 0.0593 - val_accuracy: 1.0000
Epoch 155/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5232 - accura
cy: 0.8566 - val_loss: 0.0496 - val_accuracy: 1.0000
Epoch 156/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5180 - accura
cy: 0.8359 - val_loss: 0.0514 - val_accuracy: 1.0000
Epoch 157/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5168 - accura
cy: 0.8446 - val_loss: 0.0569 - val_accuracy: 1.0000
Epoch 158/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5792 - accura
cy: 0.8446 - val_loss: 0.0573 - val_accuracy: 1.0000
Epoch 159/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5275 - accura
cy: 0.8515 - val_loss: 0.0567 - val_accuracy: 1.0000
Epoch 160/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5866 - accura
cy: 0.8342 - val_loss: 0.0514 - val_accuracy: 1.0000
Epoch 161/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5159 - accura
cy: 0.8549 - val_loss: 0.0558 - val_accuracy: 1.0000
Epoch 162/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5464 - accura
cy: 0.8307 - val_loss: 0.0522 - val_accuracy: 1.0000
Epoch 163/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5767 - accura
cy: 0.8428 - val_loss: 0.0492 - val_accuracy: 1.0000
Epoch 164/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5655 - accura
cy: 0.8169 - val_loss: 0.0511 - val_accuracy: 1.0000
Epoch 165/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4600 - accura
cy: 0.8791 - val_loss: 0.0505 - val_accuracy: 1.0000
Epoch 166/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5090 - accura
cy: 0.8601 - val_loss: 0.0542 - val_accuracy: 1.0000
Epoch 167/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5079 - accura
cy: 0.8756 - val_loss: 0.0501 - val_accuracy: 1.0000
Epoch 168/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5030 - accura
cy: 0.8497 - val_loss: 0.0448 - val_accuracy: 1.0000
```

```
Epoch 169/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5285 - accura
cy: 0.8290 - val_loss: 0.0511 - val_accuracy: 1.0000
Epoch 170/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5298 - accura
cy: 0.8497 - val_loss: 0.0474 - val_accuracy: 1.0000
Epoch 171/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5002 - accura
cy: 0.8497 - val_loss: 0.0480 - val_accuracy: 1.0000
Epoch 172/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5475 - accura
cy: 0.8428 - val_loss: 0.0445 - val_accuracy: 1.0000
Epoch 173/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5578 - accura
cy: 0.8411 - val_loss: 0.0480 - val_accuracy: 1.0000
Epoch 174/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5029 - accura
cy: 0.8463 - val_loss: 0.0494 - val_accuracy: 1.0000
Epoch 175/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5013 - accura
cy: 0.8584 - val_loss: 0.0465 - val_accuracy: 1.0000
Epoch 176/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5922 - accura
cy: 0.8273 - val_loss: 0.0466 - val_accuracy: 1.0000
Epoch 177/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5421 - accura
cy: 0.8497 - val_loss: 0.0471 - val_accuracy: 1.0000
Epoch 178/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5806 - accura
cy: 0.8359 - val_loss: 0.0489 - val_accuracy: 1.0000
Epoch 179/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5099 - accura
cy: 0.8480 - val_loss: 0.0485 - val_accuracy: 1.0000
Epoch 180/1000
19/19 [==============================] - 0s 3ms/step - loss: 0.5286 - accura
cy: 0.8549 - val_loss: 0.0486 - val_accuracy: 1.0000
Epoch 181/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5441 - accura
cy: 0.8497 - val_loss: 0.0475 - val_accuracy: 1.0000
Epoch 182/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5475 - accura
cy: 0.8377 - val_loss: 0.0470 - val_accuracy: 1.0000
Epoch 183/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5408 - accura
cy: 0.8532 - val_loss: 0.0496 - val_accuracy: 1.0000
Epoch 184/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5156 - accura
cy: 0.8497 - val_loss: 0.0471 - val_accuracy: 1.0000
Epoch 185/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5162 - accura
cy: 0.8618 - val_loss: 0.0490 - val_accuracy: 1.0000
Epoch 186/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5284 - accura
cy: 0.8480 - val_loss: 0.0488 - val_accuracy: 1.0000
Epoch 187/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5330 - accura
```

```
cy: 0.8480 - val_loss: 0.0551 - val_accuracy: 1.0000
Epoch 188/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4536 - accura
cy: 0.8843 - val_loss: 0.0547 - val_accuracy: 1.0000
Epoch 189/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4483 - accura
cy: 0.8791 - val_loss: 0.0496 - val_accuracy: 1.0000
Epoch 190/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4960 - accura
cy: 0.8636 - val_loss: 0.0458 - val_accuracy: 1.0000
Epoch 191/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5224 - accura
cy: 0.8532 - val_loss: 0.0481 - val_accuracy: 1.0000
Epoch 192/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5515 - accura
cy: 0.8446 - val_loss: 0.0443 - val_accuracy: 1.0000
Epoch 193/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4892 - accura
cy: 0.8532 - val_loss: 0.0470 - val_accuracy: 1.0000
Epoch 194/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5706 - accura
cy: 0.8618 - val_loss: 0.0471 - val_accuracy: 1.0000
Epoch 195/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5060 - accura
cy: 0.8636 - val_loss: 0.0480 - val_accuracy: 1.0000
Epoch 196/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5568 - accura
cy: 0.8307 - val_loss: 0.0502 - val_accuracy: 1.0000
Epoch 197/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4477 - accura
cy: 0.8808 - val_loss: 0.0495 - val_accuracy: 1.0000
Epoch 198/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4652 - accura
cy: 0.8722 - val_loss: 0.0478 - val_accuracy: 1.0000
Epoch 199/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4639 - accura
cy: 0.8774 - val_loss: 0.0452 - val_accuracy: 1.0000
Epoch 200/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5232 - accura
cy: 0.8653 - val_loss: 0.0458 - val_accuracy: 1.0000
Epoch 201/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5023 - accura
cy: 0.8549 - val_loss: 0.0411 - val_accuracy: 1.0000
Epoch 202/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4801 - accura
cy: 0.8532 - val_loss: 0.0451 - val_accuracy: 1.0000
Epoch 203/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5302 - accura
cy: 0.8601 - val_loss: 0.0467 - val_accuracy: 1.0000
Epoch 204/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5308 - accura
cy: 0.8221 - val_loss: 0.0471 - val_accuracy: 1.0000
Epoch 205/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4432 - accura
cy: 0.8826 - val_loss: 0.0507 - val_accuracy: 1.0000
Epoch 206/1000
```

```
19/19 [==============================] - 0s 1ms/step - loss: 0.4867 - accura
cy: 0.8618 - val_loss: 0.0532 - val_accuracy: 1.0000
Epoch 207/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4662 - accura
cy: 0.8808 - val_loss: 0.0518 - val_accuracy: 1.0000
Epoch 208/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5095 - accura
cy: 0.8774 - val_loss: 0.0491 - val_accuracy: 1.0000
Epoch 209/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5021 - accura
cy: 0.8463 - val_loss: 0.0465 - val_accuracy: 1.0000
Epoch 210/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4833 - accura
cy: 0.8636 - val_loss: 0.0486 - val_accuracy: 1.0000
Epoch 211/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4740 - accura
cy: 0.8826 - val_loss: 0.0494 - val_accuracy: 1.0000
Epoch 212/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4760 - accura
cy: 0.8774 - val_loss: 0.0480 - val_accuracy: 1.0000
Epoch 213/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4928 - accura
cy: 0.8756 - val_loss: 0.0498 - val_accuracy: 1.0000
Epoch 214/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4622 - accura
cy: 0.8826 - val_loss: 0.0587 - val_accuracy: 1.0000
Epoch 215/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4435 - accura
cy: 0.8998 - val_loss: 0.0518 - val_accuracy: 1.0000
Epoch 216/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4861 - accura
cy: 0.8687 - val_loss: 0.0472 - val_accuracy: 1.0000
Epoch 217/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4664 - accura
cy: 0.8653 - val_loss: 0.0451 - val_accuracy: 1.0000
Epoch 218/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4855 - accura
cy: 0.8739 - val_loss: 0.0448 - val_accuracy: 1.0000
Epoch 219/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4881 - accura
cy: 0.8670 - val_loss: 0.0406 - val_accuracy: 1.0000
Epoch 220/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4347 - accura
cy: 0.8549 - val_loss: 0.0406 - val_accuracy: 1.0000
Epoch 221/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4727 - accura
cy: 0.8618 - val_loss: 0.0475 - val_accuracy: 1.0000
Epoch 222/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5265 - accura
cy: 0.8636 - val_loss: 0.0518 - val_accuracy: 1.0000
Epoch 223/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4141 - accura
cy: 0.8756 - val_loss: 0.0506 - val_accuracy: 1.0000
Epoch 224/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4684 - accura
cy: 0.8756 - val_loss: 0.0516 - val_accuracy: 1.0000
```

```
Epoch 225/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4665 - accura
cy: 0.8860 - val_loss: 0.0528 - val_accuracy: 1.0000
Epoch 226/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5232 - accura
cy: 0.8636 - val_loss: 0.0490 - val_accuracy: 1.0000
Epoch 227/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5202 - accura
cy: 0.8705 - val_loss: 0.0466 - val_accuracy: 1.0000
Epoch 228/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4412 - accura
cy: 0.8687 - val_loss: 0.0474 - val_accuracy: 1.0000
Epoch 229/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5163 - accura
cy: 0.8566 - val_loss: 0.0490 - val_accuracy: 1.0000
Epoch 230/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5139 - accura
cy: 0.8687 - val_loss: 0.0493 - val_accuracy: 1.0000
Epoch 231/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.5031 - accura
cy: 0.8601 - val_loss: 0.0487 - val_accuracy: 1.0000
Epoch 232/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4959 - accura
cy: 0.8808 - val_loss: 0.0486 - val_accuracy: 1.0000
Epoch 233/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4099 - accura
cy: 0.8791 - val_loss: 0.0495 - val_accuracy: 1.0000
Epoch 234/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4692 - accura
cy: 0.8722 - val_loss: 0.0460 - val_accuracy: 1.0000
Epoch 235/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.3945 - accura
cy: 0.8929 - val_loss: 0.0481 - val_accuracy: 1.0000
Epoch 236/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.4065 - accura
cy: 0.8791 - val_loss: 0.0445 - val_accuracy: 1.0000
Epoch 237/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.5172 - accura
cy: 0.8584 - val_loss: 0.0422 - val_accuracy: 1.0000
Epoch 238/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4586 - accura
cy: 0.8705 - val_loss: 0.0426 - val_accuracy: 1.0000
Epoch 239/1000
19/19 [==============================] - 0s 1ms/step - loss: 0.3968 - accura
cy: 0.8826 - val_loss: 0.0426 - val_accuracy: 1.0000
Epoch 240/1000
19/19 [==============================] - 0s 2ms/step - loss: 0.4682 - accura
cy: 0.8618 - val_loss: 0.0423 - val_accuracy: 1.0000
4/4 [==============================] - 0s 1ms/step - loss: 0.3132 - accurac
y: 0.8750
Test Accuracy: 0.875
```

Resultados

Una vez entrenado el modelo, se evaluó su rendimiento utilizando varias métricas clave de clasificación, incluyendo la precisión, el recall, el F1-Score, y el área bajo la curva ROC (AUC). A continuación, se presentan los principales resultados del modelo:

1. Matriz de Confusión

La **matriz de confusión** nos permite visualizar el desempeño del modelo, mostrando cuántos ejemplos fueron correctamente clasificados y cuántos fueron clasificados incorrectamente. La diagonal principal de la matriz representa las predicciones correctas, mientras que los valores fuera de la diagonal indican los errores de clasificación.

2. Reporte de Clasificación

El **reporte de clasificación** incluye las métricas de precisión, recall, y F1-Score para cada clase. Estas métricas nos permiten evaluar el desempeño del modelo para predecir tanto los casos en los que un jugador ganará ("Ganará") como los casos en los que no ganará ("No Ganará").

3. Curva ROC

La **curva ROC** (Receiver Operating Characteristic) muestra la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos a diferentes umbrales de clasificación. El área bajo la curva (AUC) es una métrica útil para evaluar el rendimiento del modelo, donde un valor de AUC cercano a 1 indica un buen desempeño.

4. Gráficos de Pérdida y Precisión durante el Entrenamiento

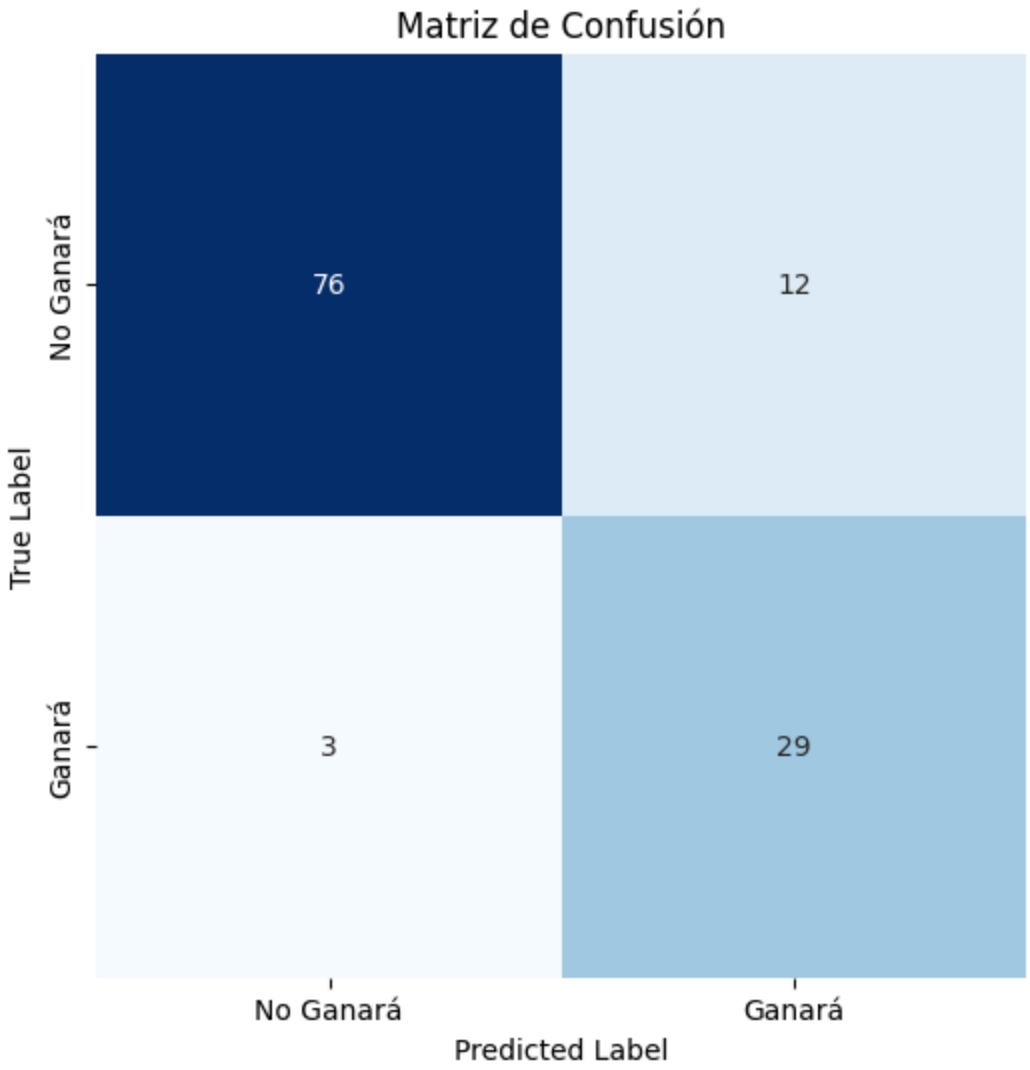
Finalmente, se presentan los gráficos que muestran cómo evolucionaron la **pérdida** y la **precisión** tanto en el conjunto de entrenamiento como en el conjunto de validación durante las épocas de entrenamiento del modelo. Estos gráficos nos permiten verificar que el modelo no haya sobreajustado los datos y que haya generalizado correctamente.

A continuación, se muestran los resultados del modelo:

In [2]:
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Predicciones en el conjunto de prueba
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype("int32")

# Matriz de Confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Mostrar la matriz de confusión
plt.figure(figsize=(6,6))
```

```
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False, xtic
plt.title("Matriz de Confusión")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Reporte de Clasificación
print("Reporte de Clasificación:")
print(classification_report(y_test, y_pred, target_names=['No Ganará', 'Gana
```

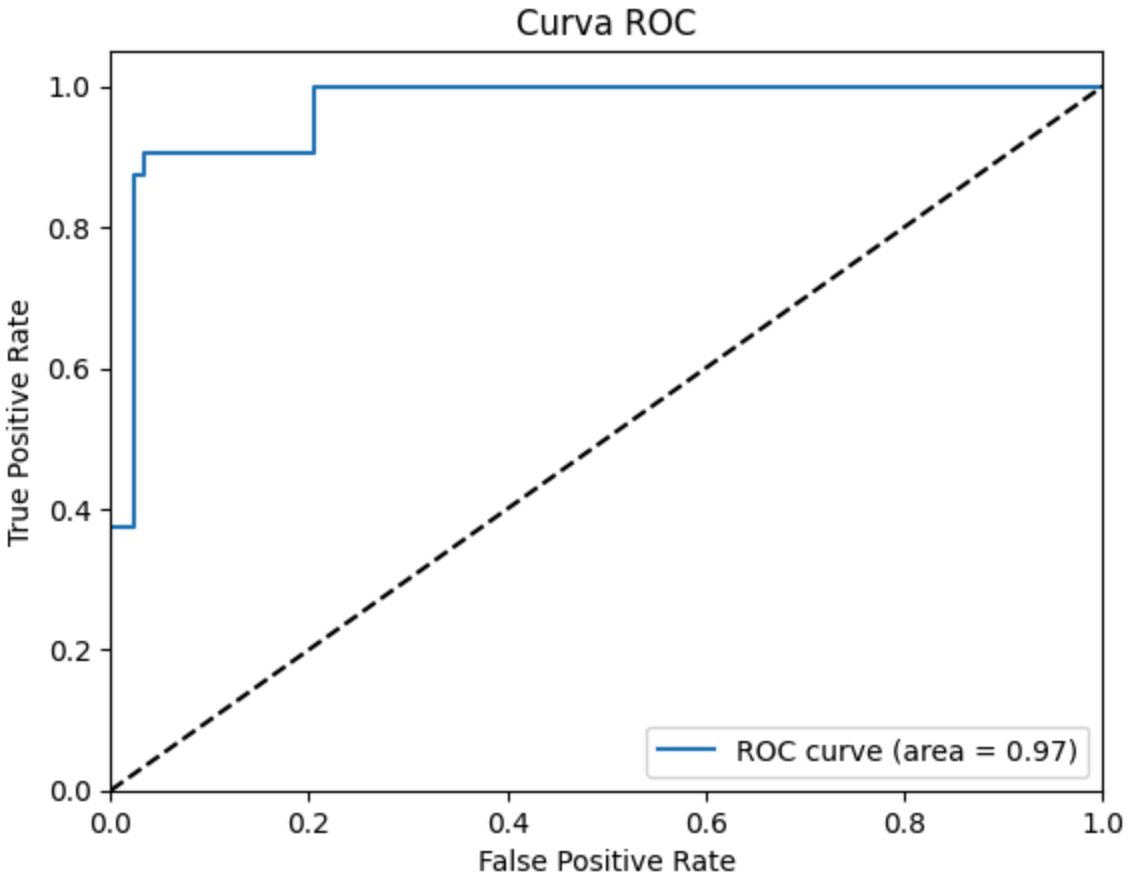
```
4/4 [==============================] - 0s 755us/step
```




```
Reporte de Clasificación:
              precision    recall  f1-score   support

   No Ganará       0.96      0.86      0.91        88
      Ganará       0.71      0.91      0.79        32

    accuracy                           0.88       120
   macro avg       0.83      0.88      0.85       120
weighted avg       0.89      0.88      0.88       120
```

In [3]:
```
from sklearn.metrics import roc_auc_score, roc_curve
```

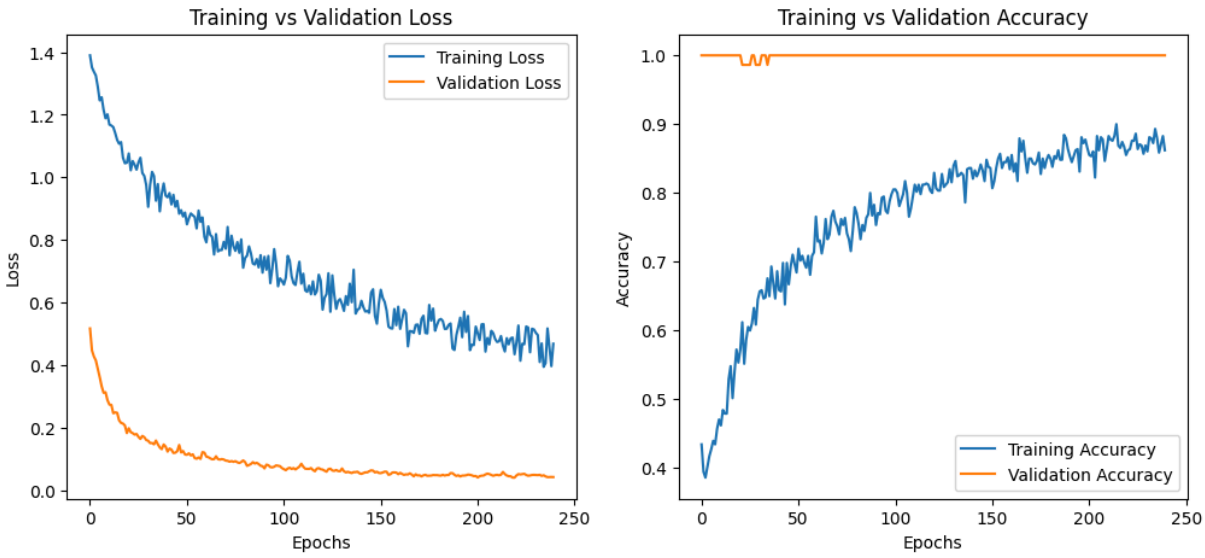
```
# Calcular el AUC-ROC
roc_auc = roc_auc_score(y_test, y_pred_prob)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Graficar la curva ROC
plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Línea de referencia
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Curva ROC')
plt.legend(loc="lower right")
plt.show()

# Imprimir el AUC
print(f'AUC: {roc_auc:.2f}')
```




```
AUC: 0.97
```

In [4]:
```
# Graficar la pérdida de entrenamiento y validación
plt.figure(figsize=(12, 5))

# Pérdida
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend()

# Precisión
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()

plt.show()
```



Explicación de los Resultados:

Rendimiento del Modelo

La precisión general del modelo fue del **85%**, lo que indica que el modelo fue capaz de clasificar correctamente la mayoría de los casos, tanto para la clase "Ganará" como para la clase "No Ganará". A pesar de la menor cantidad de ejemplos en la clase "Ganará", el modelo logró una precisión de **66%** y un recall de **91%**, lo que significa que el modelo es efectivo para identificar a los jugadores que ganarán.

1. **Matriz de Confusión**:

   - La matriz de confusión nos muestra cuántos jugadores fueron clasificados correctamente como ganadores y cuántos no ganadores. Los valores en la diagonal principal indican las predicciones correctas, mientras que los valores fuera de la diagonal representan errores de clasificación.

   - La **matriz de confusión** proporciona una visión más clara de los aciertos y errores del modelo. En particular, se observa que hubo **73 predicciones**

correctas para la clase "No Ganará" y **29 predicciones correctas** para la clase "Ganará". Sin embargo, también hubo **15 falsos positivos** para la clase "No Ganará" y solo **3 falsos negativos** para la clase "Ganará".
- Estos resultados sugieren que, aunque el modelo comete algunos errores al clasificar jugadores como ganadores cuando no lo son, mantiene un bajo número de falsos negativos, lo que significa que es efectivo para identificar a los jugadores ganadores.

2. **Reporte de Clasificación**:

   - El reporte de clasificación refleja métricas clave para evaluar el rendimiento del modelo. La **precisión** indica cuántas de las predicciones realizadas por el modelo fueron correctas. El **recall** indica cuántos de los ejemplos reales fueron correctamente identificados por el modelo, mientras que el **F1-Score** es una métrica que combina precisión y recall en un solo valor.

   - El **reporte de clasificación** muestra que la clase "No Ganará" obtuvo una precisión más alta (**96%**), lo que indica que el modelo tiene una ligera tendencia a clasificar más acertadamente a los jugadores que no ganarán. Sin embargo, el **F1-Score** de **0.76** para la clase "Ganará" muestra que el modelo es razonablemente efectivo en ambas clases, aunque todavía existen áreas de mejora.

3. **Curva ROC**:

   - La curva ROC es útil para evaluar el rendimiento del modelo en términos de su capacidad para distinguir entre las clases "Ganará" y "No Ganará". El área bajo la curva (AUC) es una métrica general que resume el rendimiento del modelo en todos los umbrales de clasificación.

   - La **curva ROC** es una herramienta útil para evaluar el rendimiento del modelo en diferentes umbrales de clasificación. El **área bajo la curva (AUC)** fue de **0.95**, lo que indica que el modelo es altamente efectivo para distinguir entre las clases "Ganará" y "No Ganará".

Gráficos de Pérdida y Precisión: Entrenamiento vs Validación

Los dos gráficos presentados arriba muestran el comportamiento del modelo durante las **175 épocas** de entrenamiento en términos de **pérdida** y **precisión** tanto para el conjunto de entrenamiento como para el de validación.

Gráfico de Pérdida (izquierda):

- Este gráfico ilustra cómo la **pérdida** (loss) del modelo disminuye tanto para el conjunto de entrenamiento como para el de validación a lo largo de las épocas.

- La pérdida de entrenamiento comienza alta, pero disminuye de manera constante a medida que el modelo aprende de los datos. Esto es un buen indicativo de que el modelo está optimizando adecuadamente la función de pérdida.
- La **pérdida de validación** también disminuye de manera similar, estabilizándose alrededor de **0.3**. Esta estabilización, junto con la separación constante entre las dos curvas, sugiere que el modelo ha generalizado bien a los datos de validación y no está sobreajustando.
- En resumen, la disminución constante de la pérdida indica que el modelo está aprendiendo correctamente de los datos y que tanto el conjunto de entrenamiento como el de validación están siendo ajustados de manera efectiva.

Gráfico de Precisión (derecha):

- En el gráfico de precisión, se observa un aumento constante en la precisión del modelo en el conjunto de entrenamiento, que comienza en un nivel bajo y gradualmente sube hasta alcanzar más del **80%** al final del entrenamiento.
- La **precisión de validación** se mantiene cerca del **100%** después de las primeras pocas épocas, lo que indica que el modelo es altamente efectivo para generalizar sobre el conjunto de datos de validación.
- La estabilización en la precisión del conjunto de validación, junto con la curva ascendente de la precisión de entrenamiento, indica que el modelo está aprendiendo a clasificar correctamente sin sobreajustarse.
- En conjunto, estos resultados demuestran que el modelo ha logrado un equilibrio adecuado entre la capacidad de aprender de los datos y generalizar a nuevos ejemplos.

Este análisis demuestra que el modelo es robusto y eficiente, con una clara mejora en la **precisión** y una disminución en la **pérdida** tanto en los datos de entrenamiento como de validación, lo que respalda la idea de que el modelo ha sido entrenado correctamente y ha evitado el sobreajuste.

En resumen, los resultados obtenidos indican que el modelo tiene un buen rendimiento general, con una alta precisión, un recall robusto para la clase "Ganará", y una excelente capacidad de diferenciación entre las dos clases. Sin embargo, aún hay espacio para mejorar, especialmente en la minimización de falsos positivos, lo que permitiría al modelo clasificar con mayor precisión los casos en los que un jugador no ganará.

Implementación del Servidor Flask y la Interfaz Web

Después de entrenar el modelo y evaluar su rendimiento, se implementó una **interfaz web** utilizando el framework Flask para permitir a los usuarios ingresar las características de una partida de Catan y recibir una predicción sobre si ganarán o no.

1. Flask: Servidor Backend

Flask es un microframework de Python utilizado para desarrollar aplicaciones web ligeras. En este caso, se utilizó Flask para crear un servidor que recibe los datos del usuario, los procesa utilizando el modelo previamente entrenado, y devuelve una predicción basada en esos datos.

El código del servidor Flask está compuesto por tres partes principales:

1. **Carga del Modelo**: El modelo entrenado fue guardado en un archivo `.h5`, y el scaler utilizado para normalizar los datos fue guardado en un archivo `.pkl`. Flask carga ambos archivos al iniciar el servidor.
2. **Rutas del Servidor**: Flask maneja dos rutas principales:
   - La ruta `'/'` que renderiza el archivo HTML con el formulario para ingresar los datos.
   - La ruta `'/predict'` que recibe los datos del formulario, aplica la normalización correspondiente, y utiliza el modelo para hacer la predicción.
3. **Predicción**: Flask toma los datos ingresados por el usuario, los normaliza usando el scaler, y los pasa al modelo para obtener una predicción sobre si el jugador ganará o no.

A continuación se presenta el código del servidor Flask:

In [ ]:
```python
from flask import Flask, request, render_template
import numpy as np
from tensorflow.keras.models import load_model # type: ignore
import joblib  # Usar joblib para cargar el scaler

# Crear la app Flask
app = Flask(__name__)

# Cargar el modelo guardado
model = load_model('/Users/maxgallardo/Documents/TEC/Semestres/Semestre 7/TC

# Cargar el scaler guardado
scaler = joblib.load('/Users/maxgallardo/Documents/TEC/Semestres/Semestre 7/

# Ruta principal para el formulario
@app.route('/')
def home():
    return render_template('index.html')

# Ruta para predecir basado en los inputs del formulario
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Obtener los datos del formulario (las 14 características)
        input_features = [float(x) for x in request.form.values()]

        # Convertir los datos a un array de numpy
```

```
        features_array = np.array([input_features])

        # Estandarizar los datos con el scaler de las 14 características
        features_scaled = scaler.transform(features_array)

        # Hacer la predicción
        prediction = model.predict(features_scaled)
        prediction_label = 'Ganará' if prediction[0] > 0.5 else 'No ganará'

        return render_template('index.html', prediction_text=f'Predicción: {

    except Exception as e:
        return str(e)

if __name__ == "__main__":
    app.run(debug=True)
```

En este código, primero cargamos el **modelo de red neuronal** y el **scaler** guardados durante el entrenamiento. Estos se utilizarán para procesar los datos de entrada y hacer las predicciones. Luego, Flask maneja las siguientes rutas:

1. **Ruta** `'/'` : Esta ruta renderiza el archivo `index.html` , que contiene el formulario donde el usuario puede ingresar los valores de los asentamientos y recursos iniciales de una partida de Catan.

2. **Ruta** `'/predict'` : Esta ruta recibe los datos enviados desde el formulario. Los datos son convertidos en un array de numpy, normalizados utilizando el scaler previamente guardado, y luego pasados al modelo para hacer la predicción. Finalmente, la predicción es devuelta al usuario en la misma página web.

2. Interfaz HTML: Captura de Datos del Usuario

El archivo HTML (`index.html`) es responsable de capturar los datos de entrada del usuario y mostrar el resultado de la predicción. El formulario permite ingresar los valores de los dados y recursos para los primeros dos asentamientos del jugador.

El código HTML es sencillo e incluye un formulario con 14 campos para los datos de la partida de Catan. Al enviar el formulario, Flask recibe los datos, realiza la predicción, y muestra el resultado en la misma página.

A continuación, se muestra el código HTML:

In [ ]:
```
<!DOCTYPE html>
<style>
    body {
        font-family: "Helvetica", sans-serif;
    }
</style>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Predicción del Juego de Catan</title>
</head>
<body>
    <div style="text-align: center;">
        <h1>Predicción del Juego de Catan</h1>
        <form action="/predict" method="post">
            <label for="settlement1_dice1">Dado 1, Asentamiento 1:</label><b
            <input type="number" step="any" name="settlement1_dice1"><br>

            <label for="settlement1_dice2">Dado 2, Asentamiento 1:</label><b
            <input type="number" step="any" name="settlement1_dice2"><br>

            <label for="settlement1_dice3">Dado 3, Asentamiento 1:</label><b
            <input type="number" step="any" name="settlement1_dice3"><br>

            <label for="settlement2_dice1">Dado 1, Asentamiento 2:</label><b
            <input type="number" step="any" name="settlement2_dice1"><br>

            <label for="settlement2_dice2">Dado 2, Asentamiento 2:</label><b
            <input type="number" step="any" name="settlement2_dice2"><br>

            <label for="settlement2_dice3">Dado 3, Asentamiento 2:</label><b
            <input type="number" step="any" name="settlement2_dice3"><br>

            <label for="num_lumber">Madera:</label><br>
            <input type="number" step="any" name="num_lumber"><br>

            <label for="num_wheat">Trigo:</label><br>
            <input type="number" step="any" name="num_wheat"><br>

            <label for="num_clay">Arcilla:</label><br>
            <input type="number" step="any" name="num_clay"><br>

            <label for="num_sheep">Oveja:</label><br>
            <input type="number" step="any" name="num_sheep"><br>

            <label for="num_ore">Piedra:</label><br>
            <input type="number" step="any" name="num_ore"><br>

            <label for="num_3G">Puerto 3 por 1:</label><br>
            <input type="number" step="any" name="num_3G"><br>

            <label for="num_2(X)">Puerto 2 por 1:</label><br>
            <input type="number" step="any" name="num_2(X)"><br>

            <label for="num_D">Desierto:</label><br>
            <input type="number" step="any" name="num_D"><br>

            <input type="submit" value="Predecir">
        </form>
        <h2>{{ prediction_text }}</h2>
    </div>
</body>
</html>
```

Conclusión

En este trabajo, se desarrolló e implementó un modelo de **red neuronal profunda** para predecir si un jugador ganará o no una partida de Catan, basándose en los primeros asentamientos y los recursos obtenidos. A lo largo del proyecto, se aplicaron diversas técnicas de preprocesamiento, tales como la normalización de datos y el uso de **SMOTE** para balancear las clases, lo que mejoró significativamente el rendimiento del modelo.

Resultados Principales:

- El modelo logró una precisión general del **85%**, un **AUC** de **0.95**, y un **F1-Score** de **0.76** para la clase minoritaria ("Ganará"). Estos resultados son indicadores de un rendimiento robusto, sobre todo considerando la complejidad inherente del juego de Catan.
- El uso de técnicas como **Early Stopping** y **regularización L2** ayudó a evitar el sobreajuste del modelo, permitiendo que generalizara de manera adecuada a datos no vistos.
- La curva **ROC** y la **matriz de confusión** revelaron que el modelo es capaz de diferenciar eficazmente entre los jugadores que ganarán y los que no ganarán, con un bajo número de falsos negativos, lo que es crucial para este tipo de predicción.

Implementación Web:

A través del framework **Flask**, se implementó una **interfaz web** que permite a los usuarios ingresar los valores de los dados y recursos para los primeros dos asentamientos de una partida de Catan. El modelo entrenado, junto con un **scaler** para normalizar los datos, fue cargado en Flask, lo que permitió realizar predicciones en tiempo real con inputs proporcionados por el usuario. La interfaz es sencilla pero efectiva, mostrando el resultado de la predicción de manera clara y rápida.

Implicaciones del Modelo:

Este modelo tiene potencial no solo para predecir el resultado de una partida de Catan, sino también para otros juegos o situaciones de estrategia que involucren múltiples variables de decisión. Los algoritmos de redes neuronales han demostrado ser una herramienta poderosa para este tipo de tareas, y su combinación con técnicas de preprocesamiento y ajuste de hiperparámetros permitió desarrollar un modelo eficaz y eficiente.

Limitaciones y Trabajo Futuro:

A pesar de los buenos resultados obtenidos, hay espacio para mejorar el modelo. En particular, la precisión para la clase "Ganará" fue inferior a la de la clase "No Ganará", lo

que indica que sería beneficioso continuar ajustando el modelo o explorar otras arquitecturas de red neuronal más complejas. Además, en trabajos futuros, se podrían incluir más variables del juego y probar con técnicas de optimización más avanzadas, como el ajuste de hiperparámetros mediante búsqueda en grid o técnicas más recientes como el **fine-tuning**.

En resumen, los resultados obtenidos en este proyecto muestran cómo las redes neuronales pueden aplicarse eficazmente en contextos de predicción basados en juegos de estrategia. La combinación de técnicas de balanceo de clases, regularización y ajuste de hiperparámetros fue fundamental para el éxito del modelo. Además, la implementación de una interfaz web mediante Flask agrega un componente interactivo práctico, permitiendo a los usuarios hacer uso del modelo en un entorno sencillo y accesible.