ÍNDICE DE CONTENIDOS

01 Ejercicio: Problema de modelización de la pérdida de clientes

En este ejercicio tomaremos como punto de partida el caso visto en el Notebook '02_Introducción a las RNA en TensorFlow 2.0'. Partiendo del mismo conjunto de datos, una muestra de 10.000 clientes, programar una estructura de red neuronal artificial con 4 capas ocultas y 3 capas dropout utilizando el proceso de validación cruzada k-fold en la etapa de entrenamiento con el objetivo de identificar si tenemos problemas de sesgo y/o varianza. El resto de parámetros son los que aparecen fijados aunque podéis modificarlos para ver cómo varían los resultados.

Recordad que las fases básicas para implementar dicho algoritmo de aprendizaje profundo son las siguientes:

1. Procesado datos entrada red neuronal artificial
2. Definición del modelo de red neuornal artificial
3. Configuración del proceso de aprendizaje de una RNA
4. Entrenamiento del modelo de red neuronal artificial
5. Evaluación del modelo de red neuronal artificial

01 Solución ejercicio: Problema de modelización de la pérdida de clientes

```
# Tenemos que instalar unas dependencias previamente (tenemos que
hacerlo en cada sesión que queramos utilizar la librería scikeras)
!python -m pip install scikeras

Collecting scikeras
  Downloading scikeras-0.13.0-py3-none-any.whl.metadata (3.1 kB)
Requirement already satisfied: keras>=3.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikeras) (3.4.1)
Requirement already satisfied: scikit-learn>=1.4.2 in
/usr/local/lib/python3.10/dist-packages (from scikeras) (1.5.2)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(1.4.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
```

```
(1.26.4)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (13.8.1)
Requirement already satisfied: namex in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (3.11.0)
Requirement already satisfied: optree in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(0.12.1)
Requirement already satisfied: ml-dtypes in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(0.4.1)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(24.1)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras) (3.5.0)
Requirement already satisfied: typing-extensions>=4.5.0 in
/usr/local/lib/python3.10/dist-packages (from optree->keras>=3.2.0->scikeras) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->scikeras) (0.1.2)
Downloading scikeras-0.13.0-py3-none-any.whl (26 kB)
Installing collected packages: scikeras
Successfully installed scikeras-0.13.0
```

```python
# Importamos las librerías necesarias para realizar dicho ejercicio
import keras
import scikeras
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import confusion_matrix, mean_squared_error,
precision_score, recall_score, f1_score
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score
from sklearn.model_selection import cross_validate
from sklearn.utils import class_weight

from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score

import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Sincronizamos Google Colab con Google Drive

try:
    from google.colab import drive
    drive.mount('/content/drive')
except Exception:
    print("No estás en el entorno de Google Colab.")
```

```
No estás en el entorno de Google Colab.
```

```
# Cargamos el conjunto de datos
dataset = pd.read_csv('./Churn_Modelling.csv')

# Definimos las variables independientes
x = dataset.iloc[:, 3:13].values

# Definimos la variable que queremos explicar (dependiente)
y = dataset.iloc[:, 13].values
print(dataset['Exited'].value_counts())
```

```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

```
# Realizamos la transformación para cada una de las variables que nos
interesan
# Transformación de la columna 1 (país) en variable dummy
labelencoder_x_1 = LabelEncoder()
x[:, 1] = labelencoder_x_1.fit_transform(x[:, 1])

# Comprobamos que se ha realizado correctamente
x
```

```
array([[619, 0, 'Female', ..., 1, 1, 101348.88],
       [608, 2, 'Female', ..., 0, 1, 112542.58],
       [502, 0, 'Female', ..., 1, 0, 113931.57],
       ...,
       [709, 0, 'Female', ..., 0, 1, 42085.58],
       [772, 1, 'Male', ..., 1, 0, 92888.52],
       [792, 0, 'Female', ..., 1, 0, 38190.78]], dtype=object)

# Cuando estamos considerando más de 3 categorías y queremos crear variables dummmies
# para no caer en problemas de multicolinealidad debido al exceso de variables creadas artificialmente
# tenemos que eliminar siempre 1 columna. Para ello utilizaremos las funciones OneHotEncoder y ColumnTransformer
transformer = ColumnTransformer(
    transformers=[
        ("Churn_Modelling",        # Un nombre de la transformación
         OneHotEncoder(categories='auto'), # La clase a la que transformar
         [1]            # Las columnas a transformar.
         )
    ], remainder='passthrough'
)

x = transformer.fit_transform(x) # aplicamos la función transformer
x = x[:, 1:] # eliminamos la columna 1º

# Comprobamos que se ha realizado correctamente
x[:, 0:3]

array([[0.0, 0.0, 619],
       [0.0, 1.0, 608],
       [0.0, 0.0, 502],
       ...,
       [0.0, 0.0, 709],
       [1.0, 0.0, 772],
       [0.0, 0.0, 792]], dtype=object)

# Transformación de la columna 2 (género) en variable dummy
labelencoder_x_2 = LabelEncoder()
x[:, 3] = labelencoder_x_2.fit_transform(x[:, 3])

# Definimos los conjuntos de train-test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)

print("Forma de x_train:", x_train.shape)
print("Forma de y_train:", y_train.shape)

Forma de x_train: (8000, 11)
Forma de y_train: (8000,)
```

Estandarízación

```
# Estandarizamos las variables con la función StandardScaler
sc_x = StandardScaler()

# Variables independientes entrenamiento estandarizadas
x_train = sc_x.fit_transform(x_train)

# Variables independientes testing estandarizadas
x_test = sc_x.transform(x_test)


# Averiguamos el número de features para definir la capa de entrada
print("Forma de x_train:", x_train.shape)
print("Forma de y_train:", y_train.shape)

Forma de x_train: (8000, 11)
Forma de y_train: (8000,)

# Creamos un Dataframe para ir guardando los resultados
results = pd.DataFrame(columns=['Accuracy', 'Variance', 'Precision',
'Recall', 'F1'])


#TODO Creamos una función para implementar la estructura de RNA con:
"""
-4 capas ocultas
-3 capas de dropout
-Proceso de validación cruzada k-fold en la etapa de entrenamiento
para identificar si hay problemas de sesgo o varianza.
"""

def build_rna() -> Sequential:

    """
    ¿Qué utilizas y por qué?
    -Puesto que tenemos 11 features, añadimos el mismo número de nodos
en la capa de entrada.
    -Para las capas intermedias utiliamos la media entre la capa ca de
entrada y salida.
    -Un solo nodo para la salida y activación sigmoide al ser una
clasificación binaria.
    -Reducimos el peso de las capas de dropout, ya que hay sospechas
de que están afectando negativamente al modelo.
    -Cambiamos el inicializador de pesos de uniforma he_uniform
(recomendado para la activación reul)
    -Cambiamos el inicializador "uniform" a "glorot_unifor" para la
capa de salida, recomendado para las sigmoides.
    -En la capa de salida, añadimos como métricas de validación el
recall y precisión.

    Estos útimos dos cambios mejoran sustancialmente el recall y
```

```
precisión.
    """
    rna = Sequential()
    rna.add(Dense(units = 11, kernel_initializer = "he_uniform",
activation = "relu", input_dim = 11))
    rna.add(Dense(units = 8, kernel_initializer = "he_uniform",
activation = "relu"))
    rna.add(Dropout(0.1))
    rna.add(Dense(units = 8, kernel_initializer = "he_uniform",
activation = "relu"))
    rna.add(Dropout(0.15))
    rna.add(Dense(units = 8, kernel_initializer = "he_uniform",
activation = "relu"))
    rna.add(Dropout(0.20))
    rna.add(Dense(units = 1, kernel_initializer = "glorot_uniform",
activation = "sigmoid"))
    rna.compile(optimizer = "adam", loss = "binary_crossentropy",
metrics = [ "Recall", "Precision"])
    return rna

# Preparamos la RNA al conjunto de entrenamiento para poder utilizar
el k-fold cv
# Al aumentar el tamaño del lote (de 50 a 80), también vamos a
aumentar el número de épocas.
rna = KerasClassifier(build_fn = build_rna, batch_size = 80, epochs =
130)
```

Alternativa al cross_val_score

En el código propuesto para el ejercicio, se utiliza el cross_val_score, tal que así:

```
                        accuracies = cross_val_score(
                            estimator=rna,
                            X = x_train, y = y_train,
                            cv = 10,
                            n_jobs=-1,
                            verbose = 1)

                        accuracy = accuracies.mean()
                        variance = accuracies.std()

                        print("Media del accuracy: ", accuracy) #
sesgo / bias
                        print("Varianza", variance) # varianza
```

Durante los primeros entrenamientos, aún teniendo una accuracy relativamente alto, fue posible advertir que la clasificación no estaba siendo del todo correcta, y es que este enfoque no tiene en cuenta que la muestra está completamente desblanceada (hay pocos clientes que se van a ir).

Así que como alternativa propoonemos un cross_validate, que además del accuracy, nos permitirá tener en cuenta la precisión, el recall y el f1 de nuestra validación cruzada, entrenando la red neuronal de una forma mucho más eficiente.

```
"""
Con un cv=10 -punto del que partíamos- se está produnciendo una
división de 10 pliegues.
Esto quiere decir que cada pliegue tendrá 800 muestras.

Sin emabargo, el dataset contiene un 20% de positivos, lo que
significa que quizás con 10 pliegues
no capture demasiado bien la clase minoritaria. Para ajustar el
modelo, lo reduciremos a solo 5 pliegues.
"""

# Realizamos la validación cruzada con múltiples métricas
scoring = ['accuracy', 'precision', 'recall', 'f1']

# Como vemos, únicamente añadimos más parámetros de medición a la
validación cruzada
scores = cross_validate(
    estimator=rna,
    X=x_train, y=y_train,
    cv=5,
    scoring=scoring,
    n_jobs=-1, # Paralelización, afecta al rendimiento. NO TOCAR
    verbose=2 # 1 para pocos mensajes por consola, 2 para tener
información detallada
)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent
workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:   35.6s
remaining:   53.4s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   37.8s finished
```

Funcionamiento de cross_validate

La validación cruzada de scores (cross_validate) nos devuelve un diccionario donde cada clave del diccionario contendrá un array con tantos índices como pliegues le hayamos indicado dentro del cv. Para obtener las métricas de validación, será necesario calcular la media o la desviación estándar de cada clave.

```
accuracy = scores['test_accuracy'].mean()
variance = scores['test_accuracy'].std()
precision = scores['test_precision'].mean()
recall = scores['test_recall'].mean()
f1 = scores['test_f1'].mean()
```

```python
# Crear un DataFrame con la nueva fila de resultados
new_row = pd.DataFrame({
    'Accuracy': [accuracy],
    'Variance': [variance],
    'Precision': [precision],
    'F1': [f1],
    'Recall': [recall]
})

# Concatenar el nuevo DataFrame al anterior
results = pd.concat([results, new_row], ignore_index=True)

# Mostrar el DataFrame resultante
results
```

	Accuracy	Variance	Precision	Recall	F1
0	0.856625	0.005025	0.743078	0.457132	0.565185
1	0.850125	0.007020	0.718834	0.436305	0.542542
2	0.854625	0.003945	0.732700	0.456554	0.559824
3	0.854875	0.005948	0.732893	0.454723	0.559672

```python
scores
```

```
{'fit_time': array([65.04563856, 65.14037657, 71.00909948,
73.51092243, 73.33638859,
        74.19309735, 70.94028211, 75.69607711, 39.43485093,
38.64616632]),
 'score_time': array([0.47971129, 0.52858758, 0.51362586, 0.36602092,
0.42985034,
        0.33011937, 0.59042192, 0.34108901, 0.11316228, 0.17384601]),
 'test_accuracy': array([0.84625, 0.8375 , 0.8375 , 0.83625, 0.8475 ,
0.825  , 0.79625,
        0.84875, 0.8125 , 0.8675 ]),
 'test_precision': array([0.65151515, 0.73239437, 0.68539326, 0.75
, 0.6952381 ,
        0.70175439, 0.        , 0.69444444, 0.62962963, 0.73387097]),
 'test_recall': array([0.52760736, 0.3190184 , 0.37423313, 0.29447853,
0.44785276,
        0.24539877, 0.        , 0.4601227 , 0.20731707, 0.55487805]),
 'test_f1': array([0.58305085, 0.44444444, 0.48412698, 0.42290749,
0.54477612,
        0.36363636, 0.        , 0.55350554, 0.31192661, 0.63194444])}
```

Reconsiderando la validación cruzada

Hasta aquí las métricas de bias y varianza son bastante buenas.... ¿Pero realmente están interpretando bien los datos?

1. Si el modelo tiene un alto sesgo (es decir, el accuracy es bajo) (lo que significa que tiene un error de entrenamiento alto), esto podría sugerir que tu modelo es demasiado simple y no puede capturar la complejidad de los datos. Esto se conoce como underfitting. Esto sucedería si en la media del accuracy tuviésemos un valor bajo, como del 0.70 o 0.60.

2. Si el modelo tiene una alta varianza (lo que significa que tiene una gran diferencia entre el error de entrenamiento y el error de prueba), es posible que haya overfitting. Cuánto más alta sea la varianza, más es la distancia que hay entre el valor de la predicción y el valor real.

Nosotros no nos enfrentamos a ninguna de estas situaciones, ya que el problema está en la clasificación de falsos positivos y falsos negativos. Así que vamos a probar con un nuevo enfoque.

Para seguir profundizando en esta problemática, descartamos la validación cruzada y vamos a obtener las métricas trabajando directamente con el modelo.

```
# Entrenamos el modelo
rna.fit(x_train, y_train)

# Hacemos las predicciones sobre los datos de prueba
y_pred = rna.predict(x_test)

# Convertimos las predicciones y los datos reales a un formato
adecuado
y_pred = np.array(y_pred).flatten()
y_test = np.array(y_test).flatten()
```

```
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning:
``build_fn`` will be renamed to ``model`` in a future release, at
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - Accuracy: 0.4106 -
Precision: 0.1849 - Recall: 0.5482 - loss: 0.8489
Epoch 2/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.7650 -
Precision: 0.1831 - Recall: 0.0541 - loss: 0.5373
```

```
Epoch 3/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.7960 -
Precision: 0.1900 - Recall: 0.0059 - loss: 0.4851
Epoch 4/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.7973 -
Precision: 0.3192 - Recall: 0.0053 - loss: 0.4676
Epoch 5/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.7935 -
Precision: 0.3273 - Recall: 0.0036 - loss: 0.4626
Epoch 6/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.7894 -
Precision: 0.4600 - Recall: 0.0023 - loss: 0.4630
Epoch 7/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.7925 -
Precision: 0.6508 - Recall: 0.0073 - loss: 0.4560
Epoch 8/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8004 -
Precision: 0.7553 - Recall: 0.0052 - loss: 0.4421
Epoch 9/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8010 -
Precision: 0.8667 - Recall: 0.0141 - loss: 0.4355
Epoch 10/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.7993 -
Precision: 0.8760 - Recall: 0.0247 - loss: 0.4441
Epoch 11/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8017 -
Precision: 0.8113 - Recall: 0.0302 - loss: 0.4371
Epoch 12/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8094 -
Precision: 0.8166 - Recall: 0.0369 - loss: 0.4223
Epoch 13/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8050 -
Precision: 0.8310 - Recall: 0.0402 - loss: 0.4305
Epoch 14/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8049 -
Precision: 0.8870 - Recall: 0.0657 - loss: 0.4301
Epoch 15/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8084 -
Precision: 0.7702 - Recall: 0.0817 - loss: 0.4270
Epoch 16/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8107 -
Precision: 0.7156 - Recall: 0.0889 - loss: 0.4172
Epoch 17/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8114 -
Precision: 0.7898 - Recall: 0.0866 - loss: 0.4095
Epoch 18/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8091 -
Precision: 0.8113 - Recall: 0.0901 - loss: 0.4158
Epoch 19/120
```

```
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8140 -
Precision: 0.8317 - Recall: 0.0874 - loss: 0.4054
Epoch 20/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8188 -
Precision: 0.8681 - Recall: 0.1101 - loss: 0.4012
Epoch 21/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8164 -
Precision: 0.8784 - Recall: 0.1104 - loss: 0.3910
Epoch 22/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8133 -
Precision: 0.8466 - Recall: 0.0960 - loss: 0.4021
Epoch 23/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8234 -
Precision: 0.8855 - Recall: 0.1287 - loss: 0.3852
Epoch 24/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8126 -
Precision: 0.8496 - Recall: 0.1060 - loss: 0.3930
Epoch 25/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8202 -
Precision: 0.8952 - Recall: 0.1432 - loss: 0.3875
Epoch 26/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8276 -
Precision: 0.9217 - Recall: 0.1280 - loss: 0.3861
Epoch 27/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8190 -
Precision: 0.8867 - Recall: 0.1388 - loss: 0.3891
Epoch 28/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8162 -
Precision: 0.9039 - Recall: 0.1242 - loss: 0.3837
Epoch 29/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8178 -
Precision: 0.8835 - Recall: 0.1231 - loss: 0.3805
Epoch 30/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8224 -
Precision: 0.8478 - Recall: 0.1283 - loss: 0.3754
Epoch 31/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8156 -
Precision: 0.9251 - Recall: 0.1328 - loss: 0.3774
Epoch 32/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8159 -
Precision: 0.8869 - Recall: 0.1192 - loss: 0.3832
Epoch 33/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8227 -
Precision: 0.8543 - Recall: 0.1299 - loss: 0.3832
Epoch 34/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8174 -
Precision: 0.8689 - Recall: 0.1376 - loss: 0.3836
Epoch 35/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8179 -
```

```
Precision: 0.9156 - Recall: 0.1296 - loss: 0.3752
Epoch 36/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8177 -
Precision: 0.8732 - Recall: 0.1386 - loss: 0.3714
Epoch 37/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8265 -
Precision: 0.8809 - Recall: 0.1303 - loss: 0.3764
Epoch 38/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8180 -
Precision: 0.9069 - Recall: 0.1221 - loss: 0.3701
Epoch 39/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8236 -
Precision: 0.8873 - Recall: 0.1269 - loss: 0.3742
Epoch 40/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8205 -
Precision: 0.9089 - Recall: 0.1215 - loss: 0.3779
Epoch 41/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8173 -
Precision: 0.9000 - Recall: 0.1276 - loss: 0.3704
Epoch 42/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8186 -
Precision: 0.9152 - Recall: 0.1240 - loss: 0.3746
Epoch 43/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8061 -
Precision: 0.8569 - Recall: 0.1122 - loss: 0.3816
Epoch 44/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8231 -
Precision: 0.9116 - Recall: 0.1125 - loss: 0.3639
Epoch 45/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8210 -
Precision: 0.9182 - Recall: 0.1362 - loss: 0.3630
Epoch 46/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8277 -
Precision: 0.8827 - Recall: 0.1206 - loss: 0.3635
Epoch 47/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8239 -
Precision: 0.8801 - Recall: 0.1364 - loss: 0.3690
Epoch 48/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8225 -
Precision: 0.8758 - Recall: 0.1283 - loss: 0.3635
Epoch 49/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8230 -
Precision: 0.8770 - Recall: 0.1278 - loss: 0.3634
Epoch 50/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8163 -
Precision: 0.8994 - Recall: 0.1435 - loss: 0.3753
Epoch 51/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8142 -
Precision: 0.8289 - Recall: 0.1668 - loss: 0.3822
```

```
Epoch 52/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8292 -
Precision: 0.8453 - Recall: 0.1915 - loss: 0.3647
Epoch 53/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8523 -
Precision: 0.7464 - Recall: 0.4663 - loss: 0.3780
Epoch 54/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8528 -
Precision: 0.7238 - Recall: 0.4468 - loss: 0.3646
Epoch 55/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8556 -
Precision: 0.7416 - Recall: 0.4800 - loss: 0.3781
Epoch 56/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8539 -
Precision: 0.7162 - Recall: 0.4719 - loss: 0.3690
Epoch 57/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8505 -
Precision: 0.7321 - Recall: 0.4772 - loss: 0.3794
Epoch 58/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8582 -
Precision: 0.7340 - Recall: 0.4897 - loss: 0.3668
Epoch 59/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8558 -
Precision: 0.7254 - Recall: 0.4758 - loss: 0.3691
Epoch 60/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8524 -
Precision: 0.7131 - Recall: 0.4694 - loss: 0.3662
Epoch 61/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8555 -
Precision: 0.7219 - Recall: 0.4846 - loss: 0.3539
Epoch 62/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8581 -
Precision: 0.7419 - Recall: 0.4708 - loss: 0.3677
Epoch 63/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8594 -
Precision: 0.7455 - Recall: 0.4799 - loss: 0.3625
Epoch 64/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8549 -
Precision: 0.7029 - Recall: 0.4904 - loss: 0.3569
Epoch 65/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8621 -
Precision: 0.7510 - Recall: 0.4896 - loss: 0.3547
Epoch 66/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8626 -
Precision: 0.7511 - Recall: 0.4988 - loss: 0.3660
Epoch 67/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8577 -
Precision: 0.7414 - Recall: 0.4673 - loss: 0.3617
Epoch 68/120
```

```
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8582 -
Precision: 0.7332 - Recall: 0.4889 - loss: 0.3597
Epoch 69/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8605 -
Precision: 0.7291 - Recall: 0.5016 - loss: 0.3522
Epoch 70/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8548 -
Precision: 0.7227 - Recall: 0.4736 - loss: 0.3646
Epoch 71/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8647 -
Precision: 0.7311 - Recall: 0.5184 - loss: 0.3540
Epoch 72/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8583 -
Precision: 0.7362 - Recall: 0.5104 - loss: 0.3648
Epoch 73/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8601 -
Precision: 0.7477 - Recall: 0.4963 - loss: 0.3592
Epoch 74/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8623 -
Precision: 0.7485 - Recall: 0.4897 - loss: 0.3534
Epoch 75/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8562 -
Precision: 0.7322 - Recall: 0.4915 - loss: 0.3604
Epoch 76/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8543 -
Precision: 0.7210 - Recall: 0.4856 - loss: 0.3621
Epoch 77/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8583 -
Precision: 0.7344 - Recall: 0.4866 - loss: 0.3661
Epoch 78/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8498 -
Precision: 0.7209 - Recall: 0.4708 - loss: 0.3766
Epoch 79/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8618 -
Precision: 0.7186 - Recall: 0.4938 - loss: 0.3478
Epoch 80/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8572 -
Precision: 0.7275 - Recall: 0.4857 - loss: 0.3609
Epoch 81/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8524 -
Precision: 0.7330 - Recall: 0.4870 - loss: 0.3514
Epoch 82/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8592 -
Precision: 0.7389 - Recall: 0.4624 - loss: 0.3524
Epoch 83/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8607 -
Precision: 0.7400 - Recall: 0.5182 - loss: 0.3600
Epoch 84/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8561 -
```

```
Precision: 0.7223 - Recall: 0.4952 - loss: 0.3614
Epoch 85/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8616 -
Precision: 0.7313 - Recall: 0.5028 - loss: 0.3539
Epoch 86/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8632 -
Precision: 0.7606 - Recall: 0.5025 - loss: 0.3463
Epoch 87/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8583 -
Precision: 0.7356 - Recall: 0.4919 - loss: 0.3585
Epoch 88/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8622 -
Precision: 0.7362 - Recall: 0.4840 - loss: 0.3475
Epoch 89/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8552 -
Precision: 0.7080 - Recall: 0.5072 - loss: 0.3589
Epoch 90/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8614 -
Precision: 0.7299 - Recall: 0.4966 - loss: 0.3479
Epoch 91/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8610 -
Precision: 0.7326 - Recall: 0.5137 - loss: 0.3563
Epoch 92/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8604 -
Precision: 0.7367 - Recall: 0.5106 - loss: 0.3593
Epoch 93/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8608 -
Precision: 0.7182 - Recall: 0.4997 - loss: 0.3554
Epoch 94/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8589 -
Precision: 0.7324 - Recall: 0.4883 - loss: 0.3550
Epoch 95/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8508 -
Precision: 0.7275 - Recall: 0.4930 - loss: 0.3610
Epoch 96/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8587 -
Precision: 0.7232 - Recall: 0.4854 - loss: 0.3477
Epoch 97/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8609 -
Precision: 0.7254 - Recall: 0.4963 - loss: 0.3579
Epoch 98/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8650 -
Precision: 0.7298 - Recall: 0.4952 - loss: 0.3432
Epoch 99/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8564 -
Precision: 0.7194 - Recall: 0.5072 - loss: 0.3559
Epoch 100/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8506 -
Precision: 0.7013 - Recall: 0.4728 - loss: 0.3613
```

```
Epoch 101/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8603 -
Precision: 0.7159 - Recall: 0.5141 - loss: 0.3595
Epoch 102/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8671 -
Precision: 0.7449 - Recall: 0.5193 - loss: 0.3567
Epoch 103/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8550 -
Precision: 0.7156 - Recall: 0.4742 - loss: 0.3581
Epoch 104/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8557 -
Precision: 0.7422 - Recall: 0.4903 - loss: 0.3615
Epoch 105/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8594 -
Precision: 0.7439 - Recall: 0.4878 - loss: 0.3584
Epoch 106/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8546 -
Precision: 0.7187 - Recall: 0.4673 - loss: 0.3600
Epoch 107/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8598 -
Precision: 0.7200 - Recall: 0.5156 - loss: 0.3561
Epoch 108/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8586 -
Precision: 0.7313 - Recall: 0.5054 - loss: 0.3545
Epoch 109/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8574 -
Precision: 0.7363 - Recall: 0.4811 - loss: 0.3592
Epoch 110/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8596 -
Precision: 0.7191 - Recall: 0.5104 - loss: 0.3501
Epoch 111/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8570 -
Precision: 0.7295 - Recall: 0.5042 - loss: 0.3644
Epoch 112/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8567 -
Precision: 0.7065 - Recall: 0.4979 - loss: 0.3507
Epoch 113/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8620 -
Precision: 0.7392 - Recall: 0.4992 - loss: 0.3518
Epoch 114/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8603 -
Precision: 0.7307 - Recall: 0.4724 - loss: 0.3538
Epoch 115/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8622 -
Precision: 0.7286 - Recall: 0.5096 - loss: 0.3530
Epoch 116/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Accuracy: 0.8596 -
Precision: 0.7103 - Recall: 0.4932 - loss: 0.3490
Epoch 117/120
```

```
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8525 -
Precision: 0.7364 - Recall: 0.4838 - loss: 0.3634
Epoch 118/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8603 -
Precision: 0.7174 - Recall: 0.5055 - loss: 0.3504
Epoch 119/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8640 -
Precision: 0.7129 - Recall: 0.5055 - loss: 0.3365
Epoch 120/120
160/160 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Accuracy: 0.8661 -
Precision: 0.7440 - Recall: 0.5215 - loss: 0.3462
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 967us/step
```

```
accuracy = accuracy_score(y_test, y_pred)
variance = np.var(y_pred)

precision = precision_score(y_test, y_pred, zero_division=1)
recall = recall_score(y_test, y_pred, zero_division=1)
f1 = f1_score(y_test, y_pred, zero_division=1)

# Crear un DataFrame con la nueva fila de resultados
new_row = pd.DataFrame({
    'Accuracy': [accuracy],
    'Variance': [variance],
    'Precision': [precision],
    'F1': [f1],
    'Recall': [recall]
})

# Concatenar el nuevo DataFrame al anterior
results = pd.concat([results, new_row], ignore_index=True)

# Mostrar el DataFrame resultante
results
```

	Accuracy	Variance	Precision	Recall	F1
0	0.8535	0.008782	0.742369	0.440562	0.550056
1	0.8570	0.004815	0.746024	0.454676	0.564146
2	0.8535	0.134400	0.675000	0.533333	0.595862

Matriz de confusión y curva ROC

El modelo tiene claros problemas para identificar positivos (es decir, del total de clientes que pretenden marcharse, ¿cuántos están siendo identificados correctamente?). Con un 53% como mejor resultado hasta ahora, estaríamos perdiendo a la mitad de los clientes que se van a ir.

por su parte, tenemos una precisión que ronda el 70%, esto quiere decir que algunos clientes que se no pretenden irse, están siendo clasificados como clientes que se van a ir (una situación que no es tan grave como la anterior).

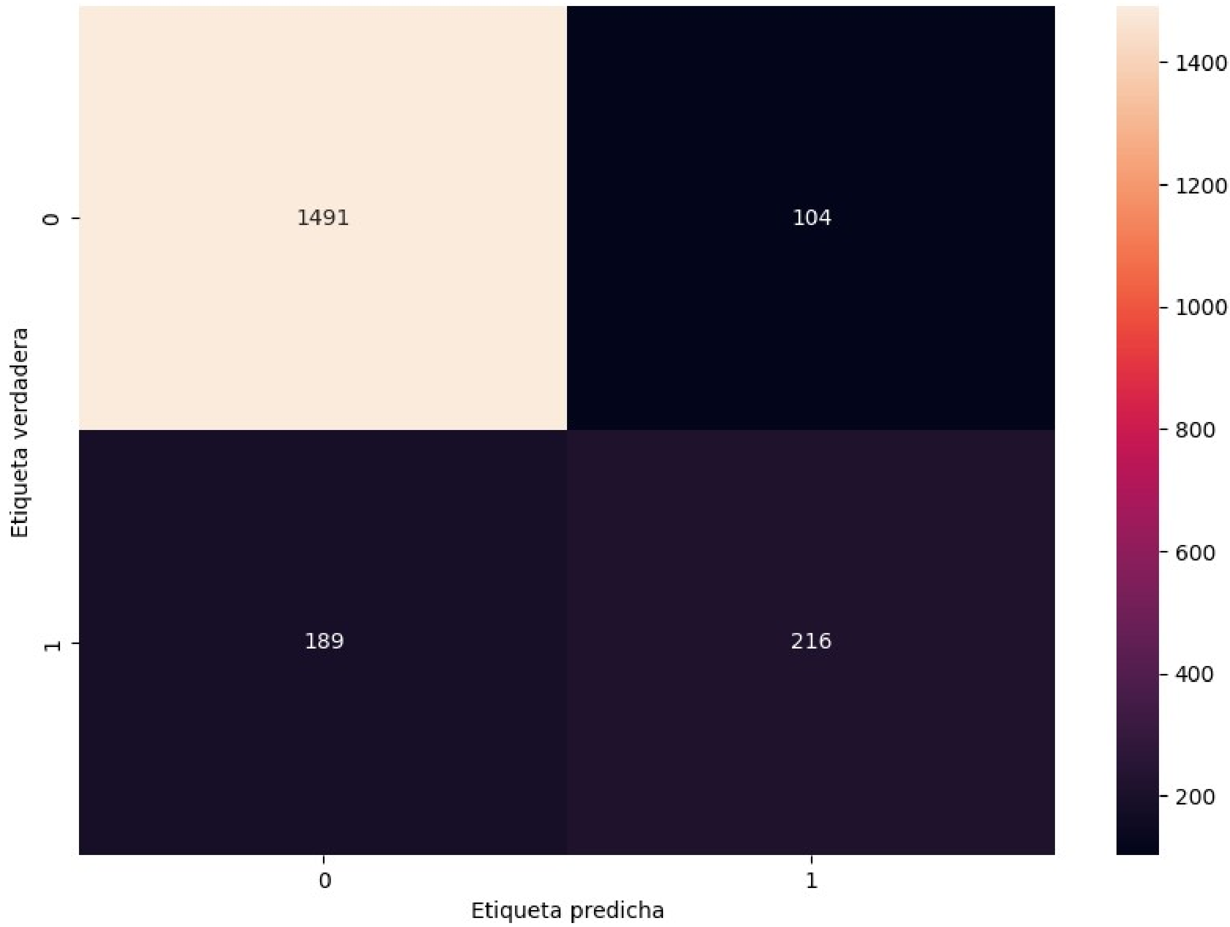
A falta de ideas sobre qué está fallando en el modelo, vamos a profundizar más en las métricas obteniendo la matriz de confusión y la curva de ROC.

NOTA

1. Precisión (precision): De todos los que fueron predichos como positivos, ¿cuántos realmente lo son?
2. Recall (sensibilidad): De TODOS los positivos del modelo, ¿cuántos fueron caputrados como positivos?
3. F1: La media de una cosa y la otra. Es decir, la media armónica entre la precisión y el recall.

```python
# Creamos la matriz de confusión
cm = confusion_matrix(y_test, y_pred)
print(cm)
# Visualización de la matriz de confusión
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.ylabel('Etiqueta verdadera')
plt.xlabel('Etiqueta predicha')
plt.show()
```

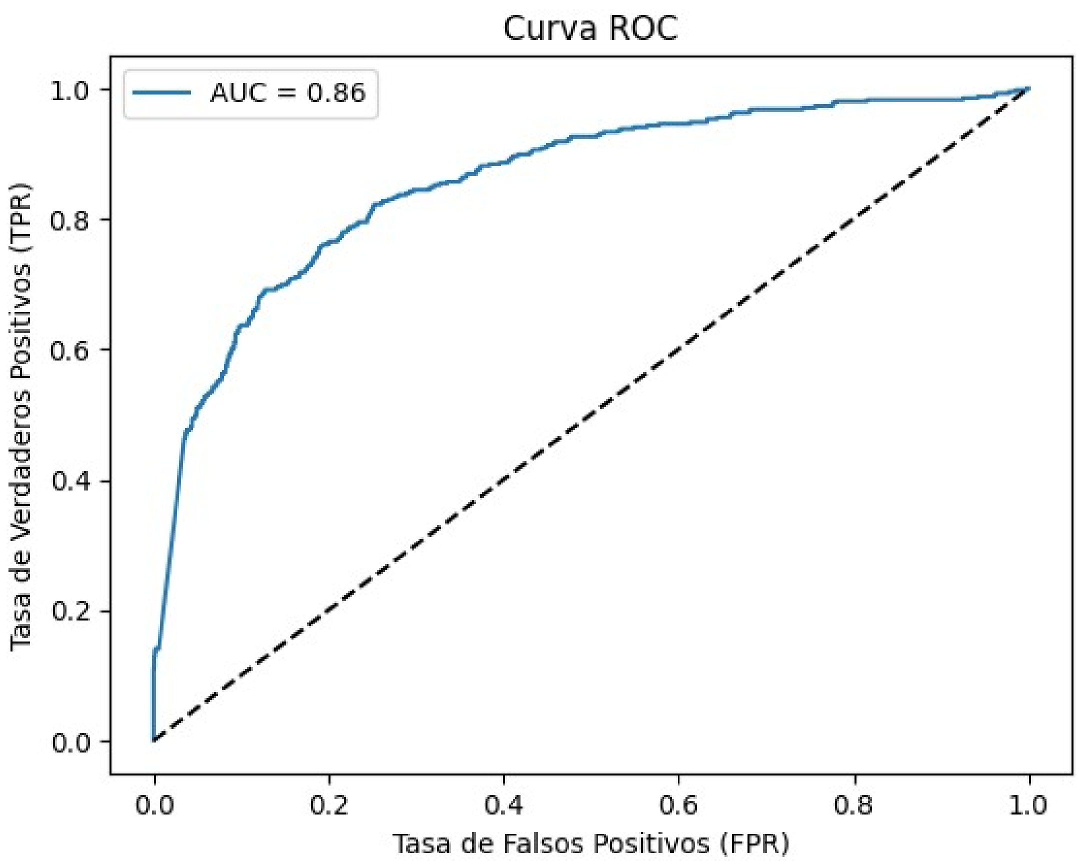
```
[[1491  104]
 [ 189  216]]
```



Curva de ROC

```
# Predicciones probabilísticas
y_pred_proba = rna.predict_proba(x_test)[:, 1]

# Cálculo de la curva ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Cálculo del AUC
auc = roc_auc_score(y_test, y_pred_proba)
print(f"AUC: {auc}")

# Gráfico de la curva ROC
plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')  # Línea de referencia (clasificación
aleatoria)
plt.xlabel("Tasa de Falsos Positivos (FPR)")
plt.ylabel("Tasa de Verdaderos Positivos (TPR)")
plt.title("Curva ROC")
plt.legend(loc="best")
plt.show()
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
AUC: 0.857019234490499
```



Obtención "manual" de las métricas

Con el objetivo de ver si podemos mejorar el recall del problema, vamos a crear la red neuronal manualmente y vamos a prescindir de la validación cruzada.

Además, vamos a remuestrear la clase minoritaria para ver si conseguimos detectar mejor los positivos.

```
rna_2 = Sequential()
rna_2.add(Dense(units = 11, kernel_initializer = "he_uniform",
activation = "relu", input_dim = 11))
rna_2.add(Dense(units = 6, kernel_initializer = "he_uniform",
activation = "relu"))
rna_2.add(Dropout(0.05))
rna_2.add(Dense(units = 6, kernel_initializer = "he_uniform",
activation = "relu"))
rna_2.add(Dropout(0.1))
rna_2.add(Dense(units = 6, kernel_initializer = "he_uniform",
activation = "relu"))
```

```python
rna_2.add(Dropout(0.15))
rna_2.add(Dense(units = 1, kernel_initializer = "glorot_uniform",
activation = "sigmoid"))
rna_2.compile(optimizer = "adam", loss = "binary_crossentropy",
metrics = ["accuracy", "Recall", "Precision"])
```

```
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```python
from imblearn.over_sampling import SMOTE

"""
Vamos a utilizar una librería externa, Imbalanced Learn, para realizar
un sobremuestreo de la clase minoritaria,
que es lo que parece que está provocando la baja precisión y recall.

"""

smote = SMOTE()
x_train_res, y_train_res = smote.fit_resample(x_train, y_train)

history = rna_2.fit(x_train_res, y_train_res, validation_data=(x_test,
y_test), epochs=100, batch_size=50)
```

```
Epoch 1/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - Precision: 0.5061 - Recall:
0.6918 - accuracy: 0.5034 - loss: 0.8062 - val_Precision: 0.2628 -
val_Recall: 0.5432 - val_accuracy: 0.5990 - val_loss: 0.6897
Epoch 2/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.5678 - Recall:
0.6694 - accuracy: 0.5783 - loss: 0.6833 - val_Precision: 0.2992 -
val_Recall: 0.8444 - val_accuracy: 0.5680 - val_loss: 0.6944
Epoch 3/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.6228 - Recall:
0.7431 - accuracy: 0.6477 - loss: 0.6435 - val_Precision: 0.3758 -
val_Recall: 0.7580 - val_accuracy: 0.6960 - val_loss: 0.6468
Epoch 4/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7040 - Recall:
0.6702 - accuracy: 0.6951 - loss: 0.6107 - val_Precision: 0.4483 -
val_Recall: 0.7284 - val_accuracy: 0.7635 - val_loss: 0.5989
Epoch 5/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7492 - Recall:
0.6576 - accuracy: 0.7178 - loss: 0.5857 - val_Precision: 0.4708 -
val_Recall: 0.7358 - val_accuracy: 0.7790 - val_loss: 0.5668
```

```
Epoch 6/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7793 - Recall: 0.6692 - accuracy: 0.7388 - loss: 0.5508 - val_Precision: 0.5493 - val_Recall: 0.7012 - val_accuracy: 0.8230 - val_loss: 0.5053
Epoch 7/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7940 - Recall: 0.6463 - accuracy: 0.7376 - loss: 0.5429 - val_Precision: 0.5439 - val_Recall: 0.7037 - val_accuracy: 0.8205 - val_loss: 0.4899
Epoch 8/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8010 - Recall: 0.6779 - accuracy: 0.7555 - loss: 0.5170 - val_Precision: 0.5236 - val_Recall: 0.7136 - val_accuracy: 0.8105 - val_loss: 0.4870
Epoch 9/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8073 - Recall: 0.6841 - accuracy: 0.7564 - loss: 0.5063 - val_Precision: 0.5138 - val_Recall: 0.7333 - val_accuracy: 0.8055 - val_loss: 0.4788
Epoch 10/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7858 - Recall: 0.6814 - accuracy: 0.7536 - loss: 0.5071 - val_Precision: 0.5195 - val_Recall: 0.7235 - val_accuracy: 0.8085 - val_loss: 0.4633
Epoch 11/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7980 - Recall: 0.7058 - accuracy: 0.7597 - loss: 0.4957 - val_Precision: 0.4936 - val_Recall: 0.7605 - val_accuracy: 0.7935 - val_loss: 0.4768
Epoch 12/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7966 - Recall: 0.7137 - accuracy: 0.7650 - loss: 0.4931 - val_Precision: 0.5172 - val_Recall: 0.7432 - val_accuracy: 0.8075 - val_loss: 0.4620
Epoch 13/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7964 - Recall: 0.7037 - accuracy: 0.7611 - loss: 0.4949 - val_Precision: 0.4845 - val_Recall: 0.7728 - val_accuracy: 0.7875 - val_loss: 0.4806
Epoch 14/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8001 - Recall: 0.7095 - accuracy: 0.7684 - loss: 0.4916 - val_Precision: 0.5118 - val_Recall: 0.7481 - val_accuracy: 0.8045 - val_loss: 0.4577
Epoch 15/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8044 - Recall: 0.7170 - accuracy: 0.7717 - loss: 0.4834 - val_Precision: 0.5051 - val_Recall: 0.7358 - val_accuracy: 0.8005 - val_loss: 0.4556
Epoch 16/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8038 - Recall: 0.7331 - accuracy: 0.7782 - loss: 0.4799 - val_Precision: 0.5114 - val_Recall: 0.7210 - val_accuracy: 0.8040 - val_loss: 0.4487
Epoch 17/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8018 - Recall: 0.7249 - accuracy: 0.7719 - loss: 0.4877 - val_Precision: 0.4983 - val_Recall: 0.7383 - val_accuracy: 0.7965 - val_loss: 0.4544
Epoch 18/100
```

```
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8080 - Recall: 0.7379 - accuracy: 0.7780 - loss: 0.4729 - val_Precision: 0.5042 - val_Recall: 0.7358 - val_accuracy: 0.8000 - val_loss: 0.4503
Epoch 19/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7975 - Recall: 0.7365 - accuracy: 0.7737 - loss: 0.4814 - val_Precision: 0.4829 - val_Recall: 0.7679 - val_accuracy: 0.7865 - val_loss: 0.4733
Epoch 20/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7977 - Recall: 0.7332 - accuracy: 0.7717 - loss: 0.4780 - val_Precision: 0.4967 - val_Recall: 0.7432 - val_accuracy: 0.7955 - val_loss: 0.4560
Epoch 21/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8081 - Recall: 0.7359 - accuracy: 0.7837 - loss: 0.4728 - val_Precision: 0.4843 - val_Recall: 0.7605 - val_accuracy: 0.7875 - val_loss: 0.4679
Epoch 22/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8003 - Recall: 0.7479 - accuracy: 0.7779 - loss: 0.4700 - val_Precision: 0.4943 - val_Recall: 0.7457 - val_accuracy: 0.7940 - val_loss: 0.4544
Epoch 23/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8017 - Recall: 0.7310 - accuracy: 0.7794 - loss: 0.4706 - val_Precision: 0.4926 - val_Recall: 0.7407 - val_accuracy: 0.7930 - val_loss: 0.4517
Epoch 24/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8071 - Recall: 0.7447 - accuracy: 0.7797 - loss: 0.4737 - val_Precision: 0.4984 - val_Recall: 0.7481 - val_accuracy: 0.7965 - val_loss: 0.4549
Epoch 25/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8069 - Recall: 0.7340 - accuracy: 0.7790 - loss: 0.4707 - val_Precision: 0.4959 - val_Recall: 0.7383 - val_accuracy: 0.7950 - val_loss: 0.4514
Epoch 26/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8065 - Recall: 0.7455 - accuracy: 0.7828 - loss: 0.4642 - val_Precision: 0.4827 - val_Recall: 0.7580 - val_accuracy: 0.7865 - val_loss: 0.4674
Epoch 27/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8000 - Recall: 0.7473 - accuracy: 0.7803 - loss: 0.4686 - val_Precision: 0.4718 - val_Recall: 0.7654 - val_accuracy: 0.7790 - val_loss: 0.4735
Epoch 28/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8013 - Recall: 0.7501 - accuracy: 0.7805 - loss: 0.4672 - val_Precision: 0.5218 - val_Recall: 0.7383 - val_accuracy: 0.8100 - val_loss: 0.4397
Epoch 29/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8093 - Recall: 0.7388 - accuracy: 0.7830 - loss: 0.4640 - val_Precision: 0.4967 - val_Recall: 0.7506 - val_accuracy: 0.7955 - val_loss: 0.4550
Epoch 30/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7991 - Recall:
```

```
0.7403 - accuracy: 0.7779 - loss: 0.4687 - val_Precision: 0.5249 -
val_Recall: 0.7284 - val_accuracy: 0.8115 - val_loss: 0.4400
Epoch 31/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8082 - Recall:
0.7463 - accuracy: 0.7816 - loss: 0.4689 - val_Precision: 0.4984 -
val_Recall: 0.7481 - val_accuracy: 0.7965 - val_loss: 0.4499
Epoch 32/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7979 - Recall:
0.7444 - accuracy: 0.7788 - loss: 0.4650 - val_Precision: 0.5008 -
val_Recall: 0.7358 - val_accuracy: 0.7980 - val_loss: 0.4475
Epoch 33/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8033 - Recall:
0.7406 - accuracy: 0.7826 - loss: 0.4595 - val_Precision: 0.4693 -
val_Recall: 0.7556 - val_accuracy: 0.7775 - val_loss: 0.4673
Epoch 34/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8039 - Recall:
0.7546 - accuracy: 0.7830 - loss: 0.4660 - val_Precision: 0.4886 -
val_Recall: 0.7383 - val_accuracy: 0.7905 - val_loss: 0.4527
Epoch 35/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7928 - Recall:
0.7413 - accuracy: 0.7752 - loss: 0.4701 - val_Precision: 0.4992 -
val_Recall: 0.7407 - val_accuracy: 0.7970 - val_loss: 0.4488
Epoch 36/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8061 - Recall:
0.7481 - accuracy: 0.7867 - loss: 0.4608 - val_Precision: 0.4911 -
val_Recall: 0.7531 - val_accuracy: 0.7920 - val_loss: 0.4555
Epoch 37/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8085 - Recall:
0.7484 - accuracy: 0.7847 - loss: 0.4622 - val_Precision: 0.5008 -
val_Recall: 0.7407 - val_accuracy: 0.7980 - val_loss: 0.4477
Epoch 38/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8018 - Recall:
0.7347 - accuracy: 0.7798 - loss: 0.4649 - val_Precision: 0.5041 -
val_Recall: 0.7506 - val_accuracy: 0.8000 - val_loss: 0.4524
Epoch 39/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8098 - Recall:
0.7451 - accuracy: 0.7841 - loss: 0.4625 - val_Precision: 0.5238 -
val_Recall: 0.7333 - val_accuracy: 0.8110 - val_loss: 0.4350
Epoch 40/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7958 - Recall:
0.7339 - accuracy: 0.7721 - loss: 0.4697 - val_Precision: 0.4950 -
val_Recall: 0.7407 - val_accuracy: 0.7945 - val_loss: 0.4542
Epoch 41/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7950 - Recall:
0.7598 - accuracy: 0.7835 - loss: 0.4649 - val_Precision: 0.5102 -
val_Recall: 0.7407 - val_accuracy: 0.8035 - val_loss: 0.4432
Epoch 42/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8053 - Recall:
0.7460 - accuracy: 0.7823 - loss: 0.4652 - val_Precision: 0.5060 -
```

```
val_Recall: 0.7309 - val_accuracy: 0.8010 - val_loss: 0.4467
Epoch 43/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8132 - Recall:
0.7463 - accuracy: 0.7870 - loss: 0.4575 - val_Precision: 0.5051 -
val_Recall: 0.7309 - val_accuracy: 0.8005 - val_loss: 0.4466
Epoch 44/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8025 - Recall:
0.7534 - accuracy: 0.7874 - loss: 0.4610 - val_Precision: 0.4959 -
val_Recall: 0.7457 - val_accuracy: 0.7950 - val_loss: 0.4562
Epoch 45/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8084 - Recall:
0.7528 - accuracy: 0.7899 - loss: 0.4592 - val_Precision: 0.4903 -
val_Recall: 0.7481 - val_accuracy: 0.7915 - val_loss: 0.4569
Epoch 46/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7972 - Recall:
0.7459 - accuracy: 0.7799 - loss: 0.4622 - val_Precision: 0.4872 -
val_Recall: 0.7531 - val_accuracy: 0.7895 - val_loss: 0.4641
Epoch 47/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8047 - Recall:
0.7548 - accuracy: 0.7868 - loss: 0.4590 - val_Precision: 0.4864 -
val_Recall: 0.7531 - val_accuracy: 0.7890 - val_loss: 0.4637
Epoch 48/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8077 - Recall:
0.7625 - accuracy: 0.7864 - loss: 0.4607 - val_Precision: 0.4935 -
val_Recall: 0.7556 - val_accuracy: 0.7935 - val_loss: 0.4579
Epoch 49/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8122 - Recall:
0.7712 - accuracy: 0.7933 - loss: 0.4537 - val_Precision: 0.5096 -
val_Recall: 0.7210 - val_accuracy: 0.8030 - val_loss: 0.4426
Epoch 50/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8053 - Recall:
0.7595 - accuracy: 0.7895 - loss: 0.4533 - val_Precision: 0.4772 -
val_Recall: 0.7506 - val_accuracy: 0.7830 - val_loss: 0.4622
Epoch 51/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8083 - Recall:
0.7667 - accuracy: 0.7933 - loss: 0.4508 - val_Precision: 0.5133 -
val_Recall: 0.7160 - val_accuracy: 0.8050 - val_loss: 0.4396
Epoch 52/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8155 - Recall:
0.7450 - accuracy: 0.7893 - loss: 0.4542 - val_Precision: 0.4967 -
val_Recall: 0.7333 - val_accuracy: 0.7955 - val_loss: 0.4500
Epoch 53/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8042 - Recall:
0.7617 - accuracy: 0.7882 - loss: 0.4496 - val_Precision: 0.4878 -
val_Recall: 0.7432 - val_accuracy: 0.7900 - val_loss: 0.4565
Epoch 54/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8085 - Recall:
0.7759 - accuracy: 0.7955 - loss: 0.4492 - val_Precision: 0.5035 -
val_Recall: 0.7185 - val_accuracy: 0.7995 - val_loss: 0.4427
```

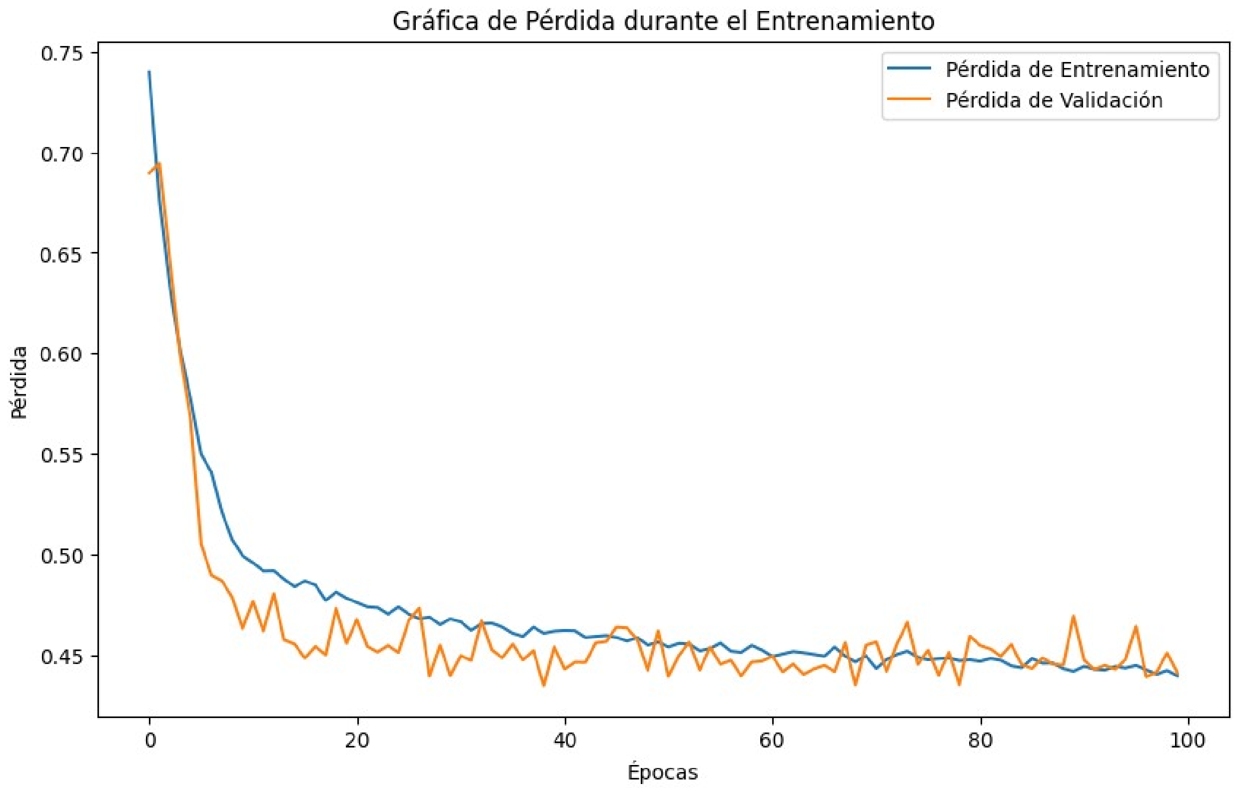
```
Epoch 55/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8098 - Recall:
0.7583 - accuracy: 0.7891 - loss: 0.4508 - val_Precision: 0.4910 -
val_Recall: 0.7432 - val_accuracy: 0.7920 - val_loss: 0.4540
Epoch 56/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8106 - Recall:
0.7678 - accuracy: 0.7922 - loss: 0.4494 - val_Precision: 0.4958 -
val_Recall: 0.7259 - val_accuracy: 0.7950 - val_loss: 0.4456
Epoch 57/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8026 - Recall:
0.7701 - accuracy: 0.7906 - loss: 0.4519 - val_Precision: 0.4950 -
val_Recall: 0.7358 - val_accuracy: 0.7945 - val_loss: 0.4476
Epoch 58/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8098 - Recall:
0.7664 - accuracy: 0.7928 - loss: 0.4487 - val_Precision: 0.5087 -
val_Recall: 0.7185 - val_accuracy: 0.8025 - val_loss: 0.4398
Epoch 59/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8105 - Recall:
0.7573 - accuracy: 0.7889 - loss: 0.4521 - val_Precision: 0.5000 -
val_Recall: 0.7333 - val_accuracy: 0.7975 - val_loss: 0.4467
Epoch 60/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8112 - Recall:
0.7611 - accuracy: 0.7926 - loss: 0.4525 - val_Precision: 0.5008 -
val_Recall: 0.7284 - val_accuracy: 0.7980 - val_loss: 0.4472
Epoch 61/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8126 - Recall:
0.7573 - accuracy: 0.7889 - loss: 0.4509 - val_Precision: 0.4926 -
val_Recall: 0.7407 - val_accuracy: 0.7930 - val_loss: 0.4496
Epoch 62/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8025 - Recall:
0.7628 - accuracy: 0.7879 - loss: 0.4540 - val_Precision: 0.5061 -
val_Recall: 0.7210 - val_accuracy: 0.8010 - val_loss: 0.4419
Epoch 63/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8086 - Recall:
0.7637 - accuracy: 0.7925 - loss: 0.4518 - val_Precision: 0.4983 -
val_Recall: 0.7358 - val_accuracy: 0.7965 - val_loss: 0.4458
Epoch 64/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8088 - Recall:
0.7682 - accuracy: 0.7945 - loss: 0.4450 - val_Precision: 0.5026 -
val_Recall: 0.7259 - val_accuracy: 0.7990 - val_loss: 0.4406
Epoch 65/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8156 - Recall:
0.7788 - accuracy: 0.7994 - loss: 0.4412 - val_Precision: 0.5051 -
val_Recall: 0.7358 - val_accuracy: 0.8005 - val_loss: 0.4433
Epoch 66/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8219 - Recall:
0.7776 - accuracy: 0.8029 - loss: 0.4412 - val_Precision: 0.5026 -
val_Recall: 0.7259 - val_accuracy: 0.7990 - val_loss: 0.4451
Epoch 67/100
```

```
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8102 - Recall: 0.7583 - accuracy: 0.7927 - loss: 0.4601 - val_Precision: 0.5034 - val_Recall: 0.7235 - val_accuracy: 0.7995 - val_loss: 0.4420
Epoch 68/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8113 - Recall: 0.7717 - accuracy: 0.7957 - loss: 0.4519 - val_Precision: 0.4686 - val_Recall: 0.7358 - val_accuracy: 0.7775 - val_loss: 0.4563
Epoch 69/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8085 - Recall: 0.7785 - accuracy: 0.7992 - loss: 0.4464 - val_Precision: 0.5069 - val_Recall: 0.7210 - val_accuracy: 0.8015 - val_loss: 0.4353
Epoch 70/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8086 - Recall: 0.7717 - accuracy: 0.7938 - loss: 0.4510 - val_Precision: 0.4719 - val_Recall: 0.7457 - val_accuracy: 0.7795 - val_loss: 0.4550
Epoch 71/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8142 - Recall: 0.7808 - accuracy: 0.7998 - loss: 0.4451 - val_Precision: 0.4810 - val_Recall: 0.7481 - val_accuracy: 0.7855 - val_loss: 0.4566
Epoch 72/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8097 - Recall: 0.7775 - accuracy: 0.8007 - loss: 0.4473 - val_Precision: 0.5017 - val_Recall: 0.7210 - val_accuracy: 0.7985 - val_loss: 0.4421
Epoch 73/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8136 - Recall: 0.7683 - accuracy: 0.7960 - loss: 0.4514 - val_Precision: 0.4770 - val_Recall: 0.7432 - val_accuracy: 0.7830 - val_loss: 0.4558
Epoch 74/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8019 - Recall: 0.7579 - accuracy: 0.7849 - loss: 0.4586 - val_Precision: 0.4633 - val_Recall: 0.7630 - val_accuracy: 0.7730 - val_loss: 0.4664
Epoch 75/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8106 - Recall: 0.7818 - accuracy: 0.7982 - loss: 0.4452 - val_Precision: 0.4941 - val_Recall: 0.7284 - val_accuracy: 0.7940 - val_loss: 0.4456
Epoch 76/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8095 - Recall: 0.7651 - accuracy: 0.7913 - loss: 0.4610 - val_Precision: 0.4785 - val_Recall: 0.7407 - val_accuracy: 0.7840 - val_loss: 0.4525
Epoch 77/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8182 - Recall: 0.7763 - accuracy: 0.7986 - loss: 0.4501 - val_Precision: 0.4941 - val_Recall: 0.7259 - val_accuracy: 0.7940 - val_loss: 0.4401
Epoch 78/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8048 - Recall: 0.7650 - accuracy: 0.7904 - loss: 0.4536 - val_Precision: 0.4768 - val_Recall: 0.7358 - val_accuracy: 0.7830 - val_loss: 0.4514
Epoch 79/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8139 - Recall:
```

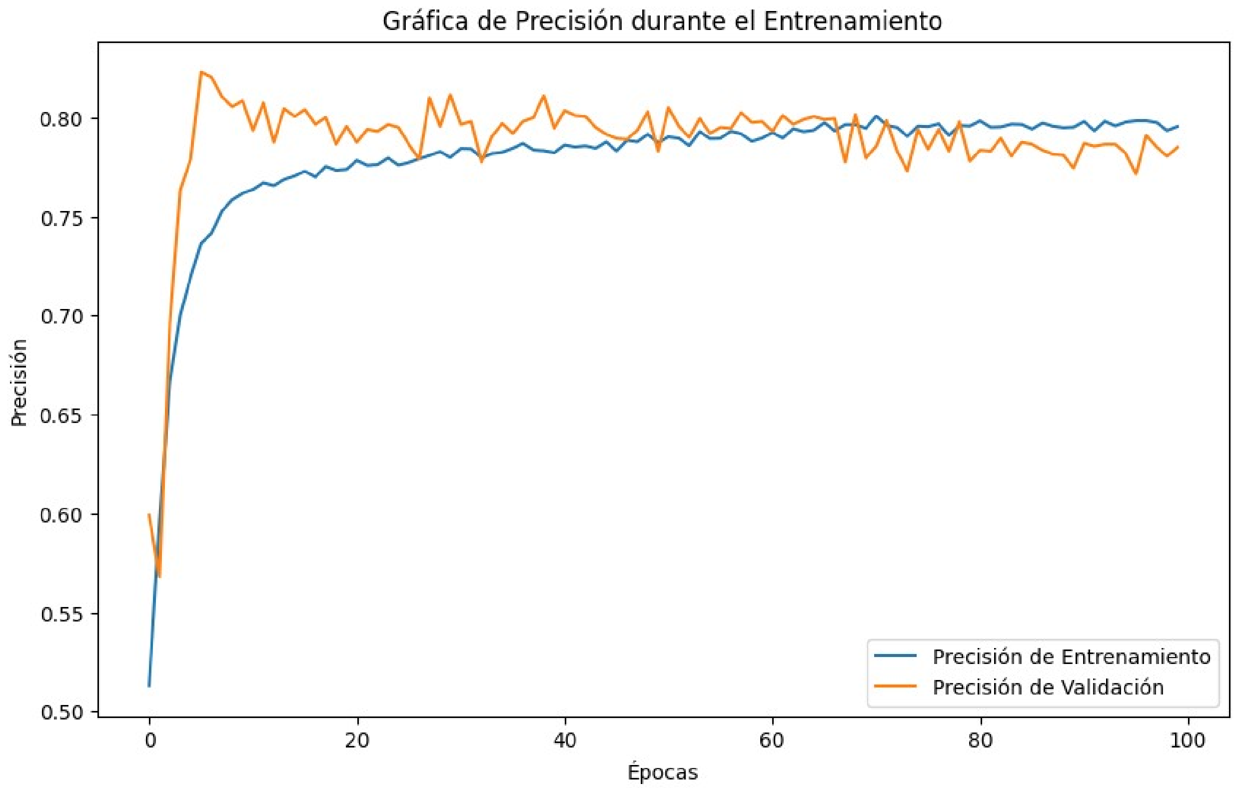
```
0.7847 - accuracy: 0.8000 - loss: 0.4444 - val_Precision: 0.5009 -
val_Recall: 0.7259 - val_accuracy: 0.7980 - val_loss: 0.4353
Epoch 80/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8132 - Recall:
0.7640 - accuracy: 0.7936 - loss: 0.4535 - val_Precision: 0.4704 -
val_Recall: 0.7654 - val_accuracy: 0.7780 - val_loss: 0.4596
Epoch 81/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8094 - Recall:
0.7814 - accuracy: 0.7968 - loss: 0.4496 - val_Precision: 0.4778 -
val_Recall: 0.7457 - val_accuracy: 0.7835 - val_loss: 0.4550
Epoch 82/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8010 - Recall:
0.7797 - accuracy: 0.7923 - loss: 0.4503 - val_Precision: 0.4772 -
val_Recall: 0.7481 - val_accuracy: 0.7830 - val_loss: 0.4531
Epoch 83/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8017 - Recall:
0.7787 - accuracy: 0.7964 - loss: 0.4497 - val_Precision: 0.4871 -
val_Recall: 0.7481 - val_accuracy: 0.7895 - val_loss: 0.4496
Epoch 84/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8148 - Recall:
0.7797 - accuracy: 0.8001 - loss: 0.4412 - val_Precision: 0.4735 -
val_Recall: 0.7506 - val_accuracy: 0.7805 - val_loss: 0.4554
Epoch 85/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8086 - Recall:
0.7826 - accuracy: 0.7983 - loss: 0.4435 - val_Precision: 0.4838 -
val_Recall: 0.7358 - val_accuracy: 0.7875 - val_loss: 0.4455
Epoch 86/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8108 - Recall:
0.7838 - accuracy: 0.8007 - loss: 0.4413 - val_Precision: 0.4823 -
val_Recall: 0.7383 - val_accuracy: 0.7865 - val_loss: 0.4435
Epoch 87/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8068 - Recall:
0.7670 - accuracy: 0.7892 - loss: 0.4556 - val_Precision: 0.4776 -
val_Recall: 0.7383 - val_accuracy: 0.7835 - val_loss: 0.4487
Epoch 88/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8075 - Recall:
0.7755 - accuracy: 0.7972 - loss: 0.4443 - val_Precision: 0.4748 -
val_Recall: 0.7432 - val_accuracy: 0.7815 - val_loss: 0.4457
Epoch 89/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8036 - Recall:
0.7817 - accuracy: 0.7936 - loss: 0.4465 - val_Precision: 0.4735 -
val_Recall: 0.7284 - val_accuracy: 0.7810 - val_loss: 0.4452
Epoch 90/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8012 - Recall:
0.7795 - accuracy: 0.7930 - loss: 0.4374 - val_Precision: 0.4655 -
val_Recall: 0.7654 - val_accuracy: 0.7745 - val_loss: 0.4696
Epoch 91/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8119 - Recall:
0.7911 - accuracy: 0.8030 - loss: 0.4380 - val_Precision: 0.4833 -
```

```
val_Recall: 0.7481 - val_accuracy: 0.7870 - val_loss: 0.4476
Epoch 92/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.7994 - Recall:
0.7806 - accuracy: 0.7913 - loss: 0.4442 - val_Precision: 0.4806 -
val_Recall: 0.7333 - val_accuracy: 0.7855 - val_loss: 0.4431
Epoch 93/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8073 - Recall:
0.7835 - accuracy: 0.7988 - loss: 0.4424 - val_Precision: 0.4821 -
val_Recall: 0.7333 - val_accuracy: 0.7865 - val_loss: 0.4451
Epoch 94/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8071 - Recall:
0.7792 - accuracy: 0.7940 - loss: 0.4468 - val_Precision: 0.4823 -
val_Recall: 0.7383 - val_accuracy: 0.7865 - val_loss: 0.4432
Epoch 95/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8087 - Recall:
0.7870 - accuracy: 0.8027 - loss: 0.4343 - val_Precision: 0.4756 -
val_Recall: 0.7457 - val_accuracy: 0.7820 - val_loss: 0.4480
Epoch 96/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8074 - Recall:
0.7738 - accuracy: 0.7932 - loss: 0.4543 - val_Precision: 0.4608 -
val_Recall: 0.7556 - val_accuracy: 0.7715 - val_loss: 0.4644
Epoch 97/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8095 - Recall:
0.7924 - accuracy: 0.8043 - loss: 0.4383 - val_Precision: 0.4892 -
val_Recall: 0.7284 - val_accuracy: 0.7910 - val_loss: 0.4393
Epoch 98/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8091 - Recall:
0.7791 - accuracy: 0.7971 - loss: 0.4432 - val_Precision: 0.4798 -
val_Recall: 0.7333 - val_accuracy: 0.7850 - val_loss: 0.4418
Epoch 99/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - Precision: 0.8054 - Recall:
0.7892 - accuracy: 0.7968 - loss: 0.4373 - val_Precision: 0.4731 -
val_Recall: 0.7383 - val_accuracy: 0.7805 - val_loss: 0.4512
Epoch 100/100
255/255 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8107 - Recall:
0.7869 - accuracy: 0.8041 - loss: 0.4316 - val_Precision: 0.4799 -
val_Recall: 0.7383 - val_accuracy: 0.7850 - val_loss: 0.4412
```

```python
# Graficar la pérdida
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Gráfica de Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()
```




```
# Graficar la precisión
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Precisión de
Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión de
Validación')
plt.title('Gráfica de Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()
plt.show()
```




```
results = rna_2.evaluate(x_test, y_test)
print(f'Pérdida: {results[0]}')
print(f'Precisión: {results[1]}')
```

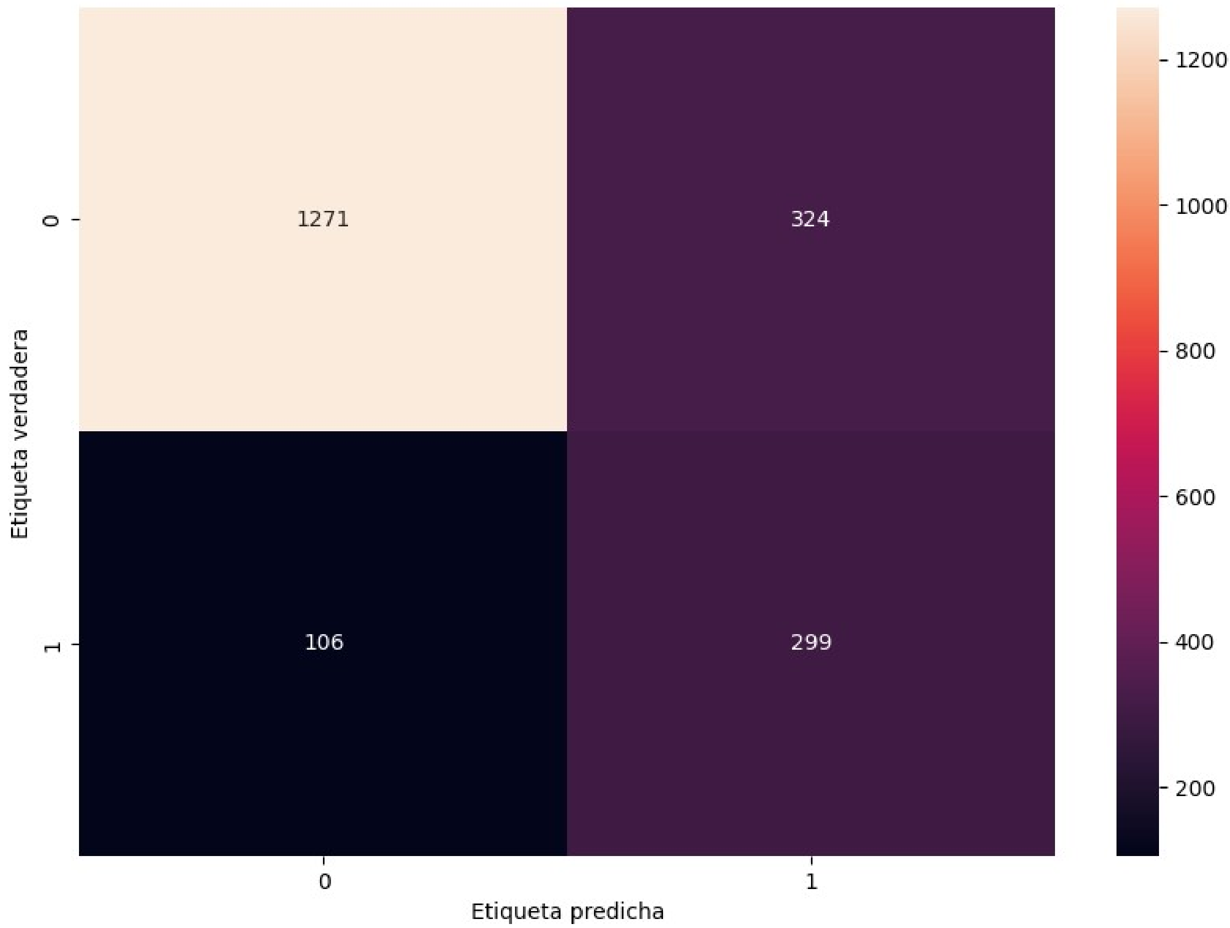
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5056 - Recall:
0.7832 - accuracy: 0.7952 - loss: 0.4337
Pérdida: 0.441192090511322
Precisión: 0.7850000262260437
```

No parece que con más de 100 epochs el modelo vaya a mejorar. Lo cuál nos lleva a pensar que el problema no es de falta de entrenamiento.

Sin embargo, con una precisión que supera el 85%, nos podemos dar por satisfechos.

```
y_pred_2 = rna_2.predict(x_test)

cm = confusion_matrix(y_test, y_pred_2 > 0.5)

print(cm)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.ylabel('Etiqueta verdadera')
plt.xlabel('Etiqueta predicha')
plt.show()
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
[[1271  324]
 [ 106  299]]
```



Finalmente obtenemos un resultado más satisfactorio. Efectivamente, hemos pedido un poco de accuracy y la función de pérdida aumenta ligeramente. Sin embargo, hemos conseguido detectar a un mayor número de verdaderos positivos. En este caso, hemos perdido precisión, ya que también tenemos muchos más falsos positivos.

Sin embargo, teniendo en cuenta las características del problema (perder el menor número posible de clientes), quizás esta sea la mejor resolución posible con los datos que tenemos disponibles.

02 Ejercicio: Problema de clasificación multiclase de diferentes especies de flores

En este ejercicio utilizaremos el conjunto de datos de flores denominado *iris* que utilizamos también en la asignatura de análisis estadístico. Este conjunto de datos está bien estudiado y es un buen problema para practicar con redes neuronales ya que las 4 variables de entrada son numéricas y tienen la misma escala en centímetros. Cada observación describe las propiedades de las medidas de una flor observada y la variable de salida será la especie específica de iris.

Se trata de un problema de clasificación multiclase, lo que significa que hay más de dos clases que predecir, de hecho, vamos a considerar tres especies de flores. Se trata de un tipo de problema importante en el que practicar con redes neuronales porque los valores de las tres clases requieren un manejo especializado. El objetivo será proponer la estructura de una red neuronal artificial que proporcione una precisión elevada del conjunto de prueba (> 85%).

02 Solución ejercicio: Problema de clasificación multiclase de diferentes especies de flores

```
# Tenemos que instalar unas dependencias previamente (tenemos que
hacerlo en cada sesión que queramos utilizar la librería scikeras)
!python -m pip install scikeras

# Importamos las librerías necesarias para realizar dicho ejercicio
import scikeras
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
from tensorflow.python.keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline


try:
    drive.mount('/content/drive')
    from google.colab import drive
except Exception:
    print("No estás en el entorno de Google Colab ;)")
```

```
No estás en el entorno de Google Colab ;)
```

```
# Cargamos el conjunto de datos
dataset = pd.read_csv('./iris.csv')

# Definimos las variables independientes
x = dataset.iloc[:, 0:4].values

# Comprobamos que hemos realizado correctamente la selección
print("x: ", x)
print("x ndim: ", x.ndim)
print("x shape:", x.shape)
print("x size: ", x.size)
print("x dtype: ", x.dtype)
```

```
x:  [[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
```

```
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.1 1.5 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
```

```
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
 [5.9 3.  4.2 1.5]
 [6.  2.2 4.  1. ]
 [6.1 2.9 4.7 1.4]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.4 1.4]
 [5.6 3.  4.5 1.5]
 [5.8 2.7 4.1 1. ]
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
 [6.1 2.8 4.  1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3.  5.  1.7]
 [6.  2.9 4.5 1.5]
 [5.7 2.6 3.5 1. ]
 [5.5 2.4 3.8 1.1]
 [5.5 2.4 3.7 1. ]
 [5.8 2.7 3.9 1.2]
 [6.  2.7 5.1 1.6]
 [5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
 [6.3 2.3 4.4 1.3]
 [5.6 3.  4.1 1.3]
 [5.5 2.5 4.  1.3]
 [5.5 2.6 4.4 1.2]
 [6.1 3.  4.6 1.4]
 [5.8 2.6 4.  1.2]
 [5.  2.3 3.3 1. ]
 [5.6 2.7 4.2 1.3]
 [5.7 3.  4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
 [5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
 [6.3 3.3 6.  2.5]
```

```
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
x ndim:  2
```

```
x shape: (149, 4)
x size:  596
x dtype:  float64
```

```python
# Definimos la variable dependiente
y = dataset.iloc[:, 4].values

# Comprobamos que hemos realizado correctamente la selección
print("y: ", y)
print("y ndim: ", y.ndim)
print("y shape:", y.shape)
print("y size: ", y.size)
print("y dtype: ", y.dtype)
```

```
y:  ['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-
setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-
versicolor'
```

```
 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica']
y ndim:  1
y shape: (149,)
y size:  149
y dtype:  object

# Codificamos los valores de la clase como enteros
encoder = LabelEncoder()
encoder.fit(y)
encoded_y = encoder.transform(y)

# Convertimos los enteros en variables ficticias
dummy_y = np_utils.to_categorical(encoded_y)

# Comprobamos que hemos realizado correctamente la selección
print("dummy_y: ", dummy_y)
print("dummy_y ndim: ", dummy_y.ndim)
print("dummy_y shape:", dummy_y.shape)
print("dummy_y size: ", dummy_y.size)
print("dummy_y dtype: ", dummy_y.dtype)

dummy_y:  [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
```

```
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
```

```
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
```

```
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]]
dummy_y ndim:  2
dummy_y shape: (149, 3)
dummy_y size:  447
dummy_y dtype:  float32

# Definimos la arquitectura del modelo de RNA
def base_model():
    """
    ¿Qué utilizas y por qué?

    -Tenemos 4 variables independientes, así que hay 4 nodos de 
entrada.

    -En las capas intermedias comenzamos usando 4 nodos (la media 
entre la entrada y la salida).
```

```
    -Comenzamos sin ninguna capa oculta para ver cómo se comporta el
modelo. En caso de overfitting, añadiríamos alguna de dropout.

    -En la capa de salida hay 3 posibles categorías, así que solo
necesitamos 3 nodos.

    -Este no es un problema de clasificación binario como el anterior,
así que utilizamos softmax en lugar de la sigmoide.
    En caso de usar la sigmoide, no podríamos garantizar que los pesos
de cada categoría sumen 1. La softmax nos garantiza esto.


    -El dataset de iris está muy balanceado (todo lo contrario que el
ejercicio anterior), así que en un principio no sería necesario
incluir
    la sensabilidad y la precisión. Aún así, y solamente para evaluar,
se ha decidido incluir.
    """
    rna = Sequential()
    rna.add(Dense(units = 4, input_dim=4, activation='relu'))
    rna.add(Dense(units = 4, activation='relu'))
    rna.add(Dense(units = 4, activation='relu'))
    rna.add(Dense(units = 3, activation='softmax'))
    rna.compile(optimizer='adam', loss='categorical_crossentropy',
metrics = ["accuracy", "Recall", "Precision"])
    return rna

# Realizamos la fase de entrenamiento con k-fold cv (k=10)
model = KerasClassifier(build_fn = base_model, batch_size = 5, epochs
= 100)
```

Analizando el dataset

El dataset de Iris, como podemos ver, solamente tiene 150 muestras. Esto puede ser insuficiente y ha resultado ser especialmente conflictivo a la hora de aplicar el crossvalidation. Para disminuir este efecto, se ha reducido el número de pliegues a solamente 3 (esto nos daría 3 lotes de 50).

Eso sí, de las pruebas realizadas, y como veremos más adelante, las mejores predicciones se realizan cuando no hay una validaciónc cruzada como tal, sino que directamente se hace la comprobación con el conjunto de test.

```
#Aquí podemos ver que quizás no hay suficientes datos para hacer
demasiados pliegues
x.shape
```

```
(149, 4)
```

```
# Hacmeos la validación con solo 3 pliegues
results = cross_val_score(model, x, dummy_y, cv=3)
```

```
Epoch 1/100

c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning:
``build_fn`` will be renamed to ``model`` in a future release, at
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3110 - loss: 1.0958
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5017 - loss: 1.0853
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4716 - loss: 1.0752
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4938 - loss: 1.0654
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5558 - loss: 1.0560
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5257 - loss: 1.0469
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4688 - loss: 1.0384
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5172 - loss: 1.0296
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5633 - loss: 1.0210
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4727 - loss: 1.0137
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5984 - loss: 1.0053
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4800 - loss: 0.9985
```

```
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5304 - loss: 0.9910
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5179 - loss: 0.9841
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5007 - loss: 0.9775
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4910 - loss: 0.9710
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5295 - loss: 0.9644
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5432 - loss: 0.9582
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5388 - loss: 0.9523
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4816 - loss: 0.9469
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4795 - loss: 0.9412
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4967 - loss: 0.9358
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5253 - loss: 0.9303
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4650 - loss: 0.9258
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5455 - loss: 0.9202
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5924 - loss: 0.9149
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5396 - loss: 0.9108
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4878 - loss: 0.9069
Epoch 29/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.6427 - loss: 0.9011
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4923 - loss: 0.8983
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5141 - loss: 0.8939
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4309 - loss: 0.8910
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5118 - loss: 0.8862
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5233 - loss: 0.8823
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5058 - loss: 0.8788
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4817 - loss: 0.8755
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4915 - loss: 0.8720
Epoch 38/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5112 - loss: 0.8684
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5930 - loss: 0.8641
Epoch 40/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4535 - loss: 0.8630
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5868 - loss: 0.8581
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4476 - loss: 0.8567
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5310 - loss: 0.8528
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4469 - loss: 0.8508
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
```

```
Recall: 0.0000e+00 - accuracy: 0.5339 - loss: 0.8470
Epoch 46/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5276 - loss: 0.8444
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4506 - loss: 0.8428
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5709 - loss: 0.8386
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4411 - loss: 0.8380
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4641 - loss: 0.8349
Epoch 51/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5710 - loss: 0.8313
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4688 - loss: 0.8303
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5117 - loss: 0.8273
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4401 - loss: 0.8262
Epoch 55/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4948 - loss: 0.8232
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5056 - loss: 0.8209
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4649 - loss: 0.8194
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4783 - loss: 0.8172
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5179 - loss: 0.8146
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5121 - loss: 0.8128
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5100 - loss: 0.8109
```

```
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4510 - loss: 0.8102
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5694 - loss: 0.8065
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4238 - loss: 0.8071
Epoch 65/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4458 - loss: 0.8048
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4817 - loss: 0.8026
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5437 - loss: 0.7998
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5902 - loss: 0.7973
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5032 - loss: 0.7973
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5029 - loss: 0.7959
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4665 - loss: 0.7950
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5082 - loss: 0.7928
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5215 - loss: 0.7911
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4636 - loss: 0.7907
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5822 - loss: 0.7873
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5404 - loss: 0.7865
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5224 - loss: 0.7855
Epoch 78/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4439 - loss: 0.7858
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5305 - loss: 0.7828
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5484 - loss: 0.7812
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4892 - loss: 0.7811
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5082 - loss: 0.7795
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5422 - loss: 0.7776
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5201 - loss: 0.7769
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5247 - loss: 0.7756
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4975 - loss: 0.7750
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5448 - loss: 0.7731
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5327 - loss: 0.7721
Epoch 89/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5722 - loss: 0.7705
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4833 - loss: 0.7710
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5088 - loss: 0.7695
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3959 - loss: 0.7704
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4583 - loss: 0.7683
Epoch 94/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
```

```
Recall: 0.0000e+00 - accuracy: 0.5624 - loss: 0.7659
Epoch 95/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4877 - loss: 0.7660
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4650 - loss: 0.7654
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5231 - loss: 0.7636
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4700 - loss: 0.7637
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4380 - loss: 0.7631
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5146 - loss: 0.7611
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Epoch 1/100

c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning:
``build_fn`` will be renamed to ``model`` in a future release, at
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.0060 - loss: 1.3192
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.1464 - loss: 1.1406
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4082 - loss: 1.0695
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4380 - loss: 1.0314
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.7723 - loss: 0.9965
Epoch 6/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.9533 - loss: 0.9610
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.9449 - loss: 0.9205
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.2857 - Recall:
0.0098 - accuracy: 0.9253 - loss: 0.8964
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.2239 - accuracy: 0.8613 - loss: 0.8614
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.5035 - accuracy: 0.9107 - loss: 0.8125
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4617 - accuracy: 0.9134 - loss: 0.7944
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.5026 - accuracy: 0.9256 - loss: 0.7342
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4107 - accuracy: 0.9199 - loss: 0.7433
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4954 - accuracy: 0.9481 - loss: 0.6638
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9960 - Recall:
0.4652 - accuracy: 0.9460 - loss: 0.6440
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9799 - Recall:
0.4874 - accuracy: 0.9821 - loss: 0.6004
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9823 - Recall:
0.5317 - accuracy: 0.9880 - loss: 0.5411
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9895 - Recall:
0.5065 - accuracy: 0.9948 - loss: 0.5274
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9950 - Recall:
0.5259 - accuracy: 0.9960 - loss: 0.4861
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9495 - Recall:
0.4635 - accuracy: 0.9985 - loss: 0.5089
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9806 - Recall:
0.4939 - accuracy: 0.9899 - loss: 0.4764
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9971 - Recall:
```

```
0.4944 - accuracy: 0.9985 - loss: 0.4574
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9981 - Recall:
0.5319 - accuracy: 0.9990 - loss: 0.4138
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9828 - Recall:
0.4140 - accuracy: 0.9922 - loss: 0.4915
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9436 - Recall:
0.4568 - accuracy: 0.9748 - loss: 0.4603
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9949 - Recall:
0.4852 - accuracy: 0.9974 - loss: 0.4088
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.4405 - accuracy: 0.9940 - loss: 0.4323
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9763 - Recall:
0.5327 - accuracy: 0.9870 - loss: 0.3669
Epoch 29/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9939 - Recall:
0.5164 - accuracy: 0.9962 - loss: 0.3905
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9968 - Recall:
0.9880 - accuracy: 0.9969 - loss: 0.3771
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.3488
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.3895
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9969 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.3302
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.3240
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9886 - Recall:
0.9886 - accuracy: 0.9886 - loss: 0.3423
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9653 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.3998
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.3357
Epoch 38/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9899 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.3121
```

```
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9795 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.3255
Epoch 40/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9931 - Recall:
0.9931 - accuracy: 0.9931 - loss: 0.3306
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.3077
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9990 - Recall:
0.9990 - accuracy: 0.9990 - loss: 0.2767
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.3088
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.3008
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.2524
Epoch 46/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9899 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.2440
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.2676
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.2025
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9931 - Recall:
0.9931 - accuracy: 0.9931 - loss: 0.2316
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.2595
Epoch 51/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.2434
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.2187
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.2554
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9969 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.1944
Epoch 55/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9899 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.2077
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9899 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.2344
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9899 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.2102
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9886 - Recall:
0.9886 - accuracy: 0.9886 - loss: 0.1905
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9985 - Recall:
0.9985 - accuracy: 0.9985 - loss: 0.1918
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9827 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.2206
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.1934
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.2142
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.2374
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.1849
Epoch 65/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.1967
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.2309
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9931 - accuracy: 0.9931 - loss: 0.1550
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.1654
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9985 - accuracy: 0.9985 - loss: 0.1734
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.1832
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
```

```
0.9974 - accuracy: 0.9974 - loss: 0.1639
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.1876
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.1844
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.1435
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.1492
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - Precision: 1.0000 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.1370
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.1711
Epoch 78/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.1622
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.1711
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.1415
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9974 - accuracy: 0.9974 - loss: 0.1627
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.1346
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.1859
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.1382
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.1528
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9990 - accuracy: 0.9990 - loss: 0.1172
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.1539
```

```
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.1200
Epoch 89/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.1328
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.1472
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9985 - accuracy: 0.9985 - loss: 0.1189
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9886 - accuracy: 0.9886 - loss: 0.1247
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.1293
Epoch 94/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.1314
Epoch 95/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.1120
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.1564
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.1728
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.1199
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.1101
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9899 - accuracy: 0.9899 - loss: 0.1154
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Epoch 1/100

c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning:
``build_fn`` will be renamed to ``model`` in a future release, at
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
```

```
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.2219 - loss: 1.1035
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4663 - loss: 1.0750
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4203 - loss: 1.0529
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4961 - loss: 1.0220
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5146 - loss: 0.9719
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4888 - loss: 0.9068
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5626 - loss: 0.8447
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5169 - loss: 0.7893
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4876 - Recall:
0.1036 - accuracy: 0.5125 - loss: 0.7664
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5056 - Recall:
0.1351 - accuracy: 0.5194 - loss: 0.7653
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6348 - Recall:
0.3818 - accuracy: 0.5834 - loss: 0.7442
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.1789 - accuracy: 0.7988 - loss: 0.7171
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9524 - Recall:
0.4305 - accuracy: 0.4409 - loss: 0.7280
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9983 - Recall:
0.4629 - accuracy: 0.6808 - loss: 0.6835
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9887 - Recall:
0.7908 - accuracy: 0.9475 - loss: 0.6954
Epoch 16/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.7415
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.6721
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9975 - Recall:
0.9975 - accuracy: 0.9975 - loss: 0.6029
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.5626
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9975 - Recall:
0.9975 - accuracy: 0.9975 - loss: 0.5243
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.5331
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.4371
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.4835
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.3682
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9827 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.3434
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9795 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.3186
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9932 - Recall:
0.9932 - accuracy: 0.9932 - loss: 0.2065
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9900 - Recall:
0.9900 - accuracy: 0.9900 - loss: 0.1962
Epoch 29/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.1366
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.1230
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.2305
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9922 - Recall:
```

```
0.9922 - accuracy: 0.9922 - loss: 0.1182
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9900 - Recall:
0.9900 - accuracy: 0.9900 - loss: 0.1159
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9985 - Recall:
0.9985 - accuracy: 0.9985 - loss: 0.0592
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.0639
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.1328
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.0721
Epoch 38/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.0612
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.1043
Epoch 40/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9900 - Recall:
0.9900 - accuracy: 0.9900 - loss: 0.0849
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.0988
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.0610
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9969 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.0435
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9969 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.0469
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9653 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.2021
Epoch 46/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.0457
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9922 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.0598
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.0478
```

```
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9932 - Recall:
0.9932 - accuracy: 0.9932 - loss: 0.0536
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9922 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.0587
Epoch 51/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9962 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.0411
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.0436
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9990 - Recall:
0.9990 - accuracy: 0.9990 - loss: 0.0242
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9795 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.1114
Epoch 55/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.0567
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.0267
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9975 - Recall:
0.9975 - accuracy: 0.9975 - loss: 0.0274
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9975 - Recall:
0.9975 - accuracy: 0.9975 - loss: 0.0276
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.0257
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.0539
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9975 - Recall:
0.9975 - accuracy: 0.9975 - loss: 0.0287
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.0710
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.1177
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.0775
Epoch 65/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9653 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.1541
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9900 - Recall:
0.9900 - accuracy: 0.9900 - loss: 0.0554
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9922 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.0465
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.0314
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9962 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.0292
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.0715
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.0476
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9795 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.0919
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.0726
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9827 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.0768
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9932 - Recall:
0.9932 - accuracy: 0.9932 - loss: 0.0408
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.0624
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9827 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.0780
Epoch 78/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9900 - Recall:
0.9900 - accuracy: 0.9900 - loss: 0.0501
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9653 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.1411
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9795 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.0882
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
```

```
0.9851 - accuracy: 0.9851 - loss: 0.0669
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9827 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.0770
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.0293
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9975 - Recall:
0.9975 - accuracy: 0.9975 - loss: 0.0209
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.0308
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.1030
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.0298
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9900 - Recall:
0.9900 - accuracy: 0.9900 - loss: 0.0486
Epoch 89/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.0654
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.0188
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9922 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.0396
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.0573
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9975 - Recall:
0.9975 - accuracy: 0.9975 - loss: 0.0202
Epoch 94/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.0652
Epoch 95/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9962 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.0237
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.0414
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.0428
```

```
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall: 
0.9980 - accuracy: 0.9980 - loss: 0.0193
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall: 
0.9948 - accuracy: 0.9948 - loss: 0.0305
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall: 
0.9911 - accuracy: 0.9911 - loss: 0.0420
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step

accuracy = results.mean()*100
variance = results.std()*100

print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, 
results.std()*100))
print("Accuracy: ", accuracy)
print("Variance: ", variance)


Baseline: 0.67% (0.94%)
Accuracy:  0.6666666666666667
Variance:  0.9428090415820634
```

Con un cv=3, conseguimos un accuracy del 0.67. Este es un resultado que dista mucho del objetivo del 85% que nos habíamos propuesto. Vamos a intentar afinar un poco más. Para ello, vamos a aplicar el Cross Validate.

El Cross Validate nos proporcionará las métricas de recall y precisión, que resultarán bastante útiles para ver cuál es el origen del problema.

Evaluación de métricas con Cross Validate

```
#Creamos un dataframe para ir guardando los resultados y comparar
resultados = pd.DataFrame(columns=['Accuracy', 'Variance', 
'Precision', 'Recall', 'F1'])

model = KerasClassifier(build_fn = base_model, batch_size = 5, epochs 
= 100)

scoring = {
    'precision': 'precision_macro',
    'recall': 'recall_macro',
    'f1': 'f1_macro',
}

results = cross_validate(model, x, dummy_y, cv=3, scoring=scoring)

Epoch 1/100
```

```
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning:
``build_fn`` will be renamed to ``model`` in a future release, at
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Precision: 0.5839 - Recall:
0.5143 - accuracy: 0.5143 - loss: 1.1188
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5850 - Recall:
0.5573 - accuracy: 0.5573 - loss: 0.9900
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5162 - Recall:
0.5009 - accuracy: 0.5009 - loss: 0.9926
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5013 - Recall:
0.5013 - accuracy: 0.5013 - loss: 0.9429
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4924 - Recall:
0.4924 - accuracy: 0.4924 - loss: 0.9112
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4476 - Recall:
0.4476 - accuracy: 0.4476 - loss: 0.9170
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5575 - Recall:
0.5575 - accuracy: 0.5575 - loss: 0.8141
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5420 - Recall:
0.5420 - accuracy: 0.5420 - loss: 0.8042
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4703 - Recall:
0.4703 - accuracy: 0.4703 - loss: 0.8323
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5624 - Recall:
0.5624 - accuracy: 0.5624 - loss: 0.7604
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4740 - Recall:
0.4740 - accuracy: 0.4740 - loss: 0.7962
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4399 - Recall:
0.4399 - accuracy: 0.4399 - loss: 0.7943
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5488 - Recall:
```

```
0.5488 - accuracy: 0.5488 - loss: 0.7236
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5424 - Recall:
0.5424 - accuracy: 0.5424 - loss: 0.7264
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4604 - Recall:
0.4604 - accuracy: 0.4604 - loss: 0.7479
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4498 - Recall:
0.4498 - accuracy: 0.4498 - loss: 0.7416
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4831 - Recall:
0.4831 - accuracy: 0.4831 - loss: 0.7227
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5411 - Recall:
0.5411 - accuracy: 0.5411 - loss: 0.6984
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4433 - Recall:
0.4433 - accuracy: 0.4433 - loss: 0.7254
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4909 - Recall:
0.4909 - accuracy: 0.4909 - loss: 0.7046
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4884 - Recall:
0.4884 - accuracy: 0.4884 - loss: 0.7041
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4870 - Recall:
0.4870 - accuracy: 0.4870 - loss: 0.6975
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4348 - Recall:
0.4348 - accuracy: 0.4348 - loss: 0.7106
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4711 - Recall:
0.4711 - accuracy: 0.4711 - loss: 0.6924
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5944 - Recall:
0.5944 - accuracy: 0.5944 - loss: 0.6689
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5319 - Recall:
0.5319 - accuracy: 0.5319 - loss: 0.6776
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4854 - Recall:
0.4803 - accuracy: 0.4803 - loss: 0.6887
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4411 - Recall:
0.4406 - accuracy: 0.4406 - loss: 0.6942
Epoch 29/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5078 - Recall:
0.4913 - accuracy: 0.4913 - loss: 0.6822
```

```
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6050 - Recall:
0.6050 - accuracy: 0.6050 - loss: 0.6539
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5452 - Recall:
0.5452 - accuracy: 0.5452 - loss: 0.6674
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4938 - Recall:
0.4710 - accuracy: 0.4710 - loss: 0.6805
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5066 - Recall:
0.4957 - accuracy: 0.4957 - loss: 0.6790
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5103 - Recall:
0.5103 - accuracy: 0.5103 - loss: 0.6736
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4713 - Recall:
0.4697 - accuracy: 0.4697 - loss: 0.6843
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4700 - Recall:
0.4673 - accuracy: 0.4673 - loss: 0.6766
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4600 - Recall:
0.4551 - accuracy: 0.4551 - loss: 0.6873
Epoch 38/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5618 - Recall:
0.5471 - accuracy: 0.5471 - loss: 0.6663
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4452 - Recall:
0.4384 - accuracy: 0.4404 - loss: 0.6943
Epoch 40/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4768 - Recall:
0.4572 - accuracy: 0.4572 - loss: 0.6808
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4718 - Recall:
0.4718 - accuracy: 0.4718 - loss: 0.6767
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5571 - Recall:
0.5571 - accuracy: 0.5571 - loss: 0.6631
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4663 - Recall:
0.4663 - accuracy: 0.4663 - loss: 0.6773
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4639 - Recall:
0.4525 - accuracy: 0.4525 - loss: 0.6776
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4351 - Recall:
0.4099 - accuracy: 0.4800 - loss: 0.6832
Epoch 46/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5354 - Recall:
0.5354 - accuracy: 0.5354 - loss: 0.6610
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5444 - Recall:
0.5444 - accuracy: 0.5444 - loss: 0.6590
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4934 - Recall:
0.4753 - accuracy: 0.4842 - loss: 0.6715
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5123 - Recall:
0.4898 - accuracy: 0.4918 - loss: 0.6661
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5167 - Recall:
0.5075 - accuracy: 0.5075 - loss: 0.6655
Epoch 51/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - Precision: 0.5441 - Recall:
0.5410 - accuracy: 0.5410 - loss: 0.6555
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5483 - Recall:
0.5353 - accuracy: 0.5353 - loss: 0.6576
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5209 - Recall:
0.5107 - accuracy: 0.5107 - loss: 0.6648
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6096 - Recall:
0.6027 - accuracy: 0.6027 - loss: 0.6500
Epoch 55/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5614 - Recall:
0.5302 - accuracy: 0.5402 - loss: 0.6568
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5782 - Recall:
0.5782 - accuracy: 0.5782 - loss: 0.6610
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5362 - Recall:
0.5313 - accuracy: 0.5313 - loss: 0.6633
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5825 - Recall:
0.5797 - accuracy: 0.5797 - loss: 0.6549
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4764 - Recall:
0.4764 - accuracy: 0.4764 - loss: 0.6771
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5788 - Recall:
0.5698 - accuracy: 0.5812 - loss: 0.6494
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4992 - Recall:
0.4992 - accuracy: 0.4992 - loss: 0.6690
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4938 - Recall:
```

```
0.4908 - accuracy: 0.4908 - loss: 0.6700
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5792 - Recall:
0.5599 - accuracy: 0.5921 - loss: 0.6599
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5025 - Recall:
0.4985 - accuracy: 0.4985 - loss: 0.6647
Epoch 65/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.4897 - Recall:
0.4897 - accuracy: 0.4897 - loss: 0.6638
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5383 - Recall:
0.5383 - accuracy: 0.5383 - loss: 0.6592
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5813 - Recall:
0.5604 - accuracy: 0.5952 - loss: 0.6423
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5611 - Recall:
0.5591 - accuracy: 0.5628 - loss: 0.6477
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5968 - Recall:
0.5929 - accuracy: 0.5998 - loss: 0.6473
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6186 - Recall:
0.6002 - accuracy: 0.6062 - loss: 0.6565
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6009 - Recall:
0.5996 - accuracy: 0.6021 - loss: 0.6466
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5256 - Recall:
0.5256 - accuracy: 0.5256 - loss: 0.6559
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5765 - Recall:
0.5744 - accuracy: 0.5744 - loss: 0.6571
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.6406 - Recall:
0.6387 - accuracy: 0.6401 - loss: 0.6367
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5800 - Recall:
0.5775 - accuracy: 0.5820 - loss: 0.6368
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5873 - Recall:
0.5777 - accuracy: 0.5777 - loss: 0.6443
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5824 - Recall:
0.5758 - accuracy: 0.5784 - loss: 0.6486
Epoch 78/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5356 - Recall:
0.5341 - accuracy: 0.5341 - loss: 0.6502
```

```
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5888 - Recall:
0.5888 - accuracy: 0.5888 - loss: 0.6367
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5929 - Recall:
0.5842 - accuracy: 0.5991 - loss: 0.6404
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5070 - Recall:
0.4895 - accuracy: 0.4895 - loss: 0.6612
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.5871 - Recall:
0.5871 - accuracy: 0.5871 - loss: 0.6494
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5606 - Recall:
0.5606 - accuracy: 0.5606 - loss: 0.6460
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5924 - Recall:
0.5842 - accuracy: 0.5842 - loss: 0.6615
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5873 - Recall:
0.5873 - accuracy: 0.5873 - loss: 0.6380
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5893 - Recall:
0.5718 - accuracy: 0.5970 - loss: 0.6355
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5919 - Recall:
0.5878 - accuracy: 0.5920 - loss: 0.6573
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.6096 - Recall:
0.5966 - accuracy: 0.5966 - loss: 0.6313
Epoch 89/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.6087 - Recall:
0.6043 - accuracy: 0.6111 - loss: 0.6440
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6873 - Recall:
0.6753 - accuracy: 0.6753 - loss: 0.6446
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6081 - Recall:
0.5987 - accuracy: 0.6136 - loss: 0.6495
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5882 - Recall:
0.5852 - accuracy: 0.5904 - loss: 0.6354
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5572 - Recall:
0.5477 - accuracy: 0.5497 - loss: 0.6427
Epoch 94/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6002 - Recall:
0.5953 - accuracy: 0.5953 - loss: 0.6288
Epoch 95/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6291 - Recall: 
0.6188 - accuracy: 0.6288 - loss: 0.6231
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6411 - Recall: 
0.6411 - accuracy: 0.6411 - loss: 0.6469
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7023 - Recall: 
0.6996 - accuracy: 0.6996 - loss: 0.6223
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6131 - Recall: 
0.6131 - accuracy: 0.6131 - loss: 0.6113
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.6608 - Recall: 
0.6448 - accuracy: 0.6448 - loss: 0.6361
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.6868 - Recall: 
0.6781 - accuracy: 0.6850 - loss: 0.6344
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Epoch 1/100

c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\sklearn\metrics\_classification.py:1517: 
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 
in labels with no predicted samples. Use `zero_division` parameter to 
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", 
len(result))
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\sklearn\metrics\_classification.py:1517: 
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in 
labels with no true samples. Use `zero_division` parameter to control 
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", 
len(result))
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning: 
``build_fn`` will be renamed to ``model`` in a future release, at 
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87: 
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a 
layer. When using Sequential models, prefer using an `Input(shape)` 
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, 
**kwargs)

20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Precision: 0.4678 - Recall: 
0.4678 - accuracy: 0.4678 - loss: 1.3525
Epoch 2/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4874 - Recall:
0.4874 - accuracy: 0.4874 - loss: 1.1440
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5482 - Recall:
0.5482 - accuracy: 0.5482 - loss: 1.0011
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5011 - Recall:
0.5011 - accuracy: 0.5011 - loss: 1.0034
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.3923 - Recall:
0.3916 - accuracy: 0.3916 - loss: 1.0540
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5422 - Recall:
0.5346 - accuracy: 0.5346 - loss: 0.8618
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5070 - Recall:
0.4626 - accuracy: 0.4626 - loss: 0.8800
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.5805 - Recall:
0.5050 - accuracy: 0.5050 - loss: 0.8126
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7309 - Recall:
0.5349 - accuracy: 0.5349 - loss: 0.7350
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8503 - Recall:
0.4705 - accuracy: 0.4705 - loss: 0.7602
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.8708 - Recall:
0.4681 - accuracy: 0.4681 - loss: 0.7164
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9615 - Recall:
0.5151 - accuracy: 0.5151 - loss: 0.6312
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9971 - Recall:
0.5458 - accuracy: 0.5677 - loss: 0.5587
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9704 - Recall:
0.4346 - accuracy: 0.5768 - loss: 0.6124
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9819 - Recall:
0.5122 - accuracy: 0.8740 - loss: 0.5309
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.5246 - accuracy: 0.9541 - loss: 0.4923
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4490 - accuracy: 0.9990 - loss: 0.5141
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
```

```
0.4535 - accuracy: 0.9922 - loss: 0.5003
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4504 - accuracy: 0.9962 - loss: 0.4845
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.5383 - accuracy: 0.9795 - loss: 0.4155
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.5315 - accuracy: 0.9886 - loss: 0.4004
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4989 - accuracy: 0.9931 - loss: 0.4071
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4909 - accuracy: 0.9990 - loss: 0.3916
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.4811 - accuracy: 0.9827 - loss: 0.4074
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.5339 - accuracy: 0.9969 - loss: 0.3988
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.3394
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.4068
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9985 - accuracy: 0.9985 - loss: 0.3253
Epoch 29/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 1.0000 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.2859
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.3038
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9990 - accuracy: 0.9990 - loss: 0.3147
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9931 - accuracy: 0.9931 - loss: 0.2982
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.2983
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9985 - accuracy: 0.9985 - loss: 0.2821
```

```
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.2904
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.2770
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.2921
Epoch 38/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.3248
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9827 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.2734
Epoch 40/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.2839
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9931 - Recall:
0.9931 - accuracy: 0.9931 - loss: 0.2403
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.2652
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9886 - Recall:
0.9886 - accuracy: 0.9886 - loss: 0.2915
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9962 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.2465
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.2290
Epoch 46/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9886 - Recall:
0.9886 - accuracy: 0.9886 - loss: 0.2374
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.2563
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.2080
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.2059
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9653 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.2651
Epoch 51/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.2305
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9962 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.2013
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9985 - Recall:
0.9985 - accuracy: 0.9985 - loss: 0.2133
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9795 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.2188
Epoch 55/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9940 - Recall:
0.9940 - accuracy: 0.9940 - loss: 0.1944
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
0.9911 - accuracy: 0.9911 - loss: 0.1698
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9886 - Recall:
0.9886 - accuracy: 0.9886 - loss: 0.1924
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.2087
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.1901
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9962 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.1611
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.1829
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9886 - Recall:
0.9886 - accuracy: 0.9886 - loss: 0.1690
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9795 - Recall:
0.9795 - accuracy: 0.9795 - loss: 0.2093
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9653 - Recall:
0.9653 - accuracy: 0.9653 - loss: 0.2210
Epoch 65/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9748 - Recall:
0.9748 - accuracy: 0.9748 - loss: 0.1954
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9962 - Recall:
0.9962 - accuracy: 0.9962 - loss: 0.1754
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall:
```

```
0.9911 - accuracy: 0.9911 - loss: 0.1823
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9990 - Recall:
0.9990 - accuracy: 0.9990 - loss: 0.1463
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.1750
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9931 - Recall:
0.9931 - accuracy: 0.9931 - loss: 0.1538
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9948 - Recall:
0.9948 - accuracy: 0.9948 - loss: 0.1557
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9974 - Recall:
0.9974 - accuracy: 0.9974 - loss: 0.1232
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9931 - Recall:
0.9931 - accuracy: 0.9931 - loss: 0.1418
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9851 - Recall:
0.9851 - accuracy: 0.9851 - loss: 0.1677
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9969 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.1460
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9969 - Recall:
0.9969 - accuracy: 0.9969 - loss: 0.1396
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9955 - Recall:
0.9955 - accuracy: 0.9955 - loss: 0.1469
Epoch 78/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9870 - Recall:
0.9870 - accuracy: 0.9870 - loss: 0.1575
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9922 - Recall:
0.9922 - accuracy: 0.9922 - loss: 0.1404
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.1197
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9980 - Recall:
0.9980 - accuracy: 0.9980 - loss: 0.1299
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9974 - Recall:
0.9974 - accuracy: 0.9974 - loss: 0.1164
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9827 - Recall:
0.9827 - accuracy: 0.9827 - loss: 0.1552
```

```
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall: 0.9980 - accuracy: 0.9980 - loss: 0.1238
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9985 - Recall: 0.9985 - accuracy: 0.9985 - loss: 0.1209
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9795 - Recall: 0.9795 - accuracy: 0.9795 - loss: 0.1516
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9911 - Recall: 0.9911 - accuracy: 0.9911 - loss: 0.1444
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall: 0.9980 - accuracy: 0.9980 - loss: 0.1021
Epoch 89/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9922 - Recall: 0.9922 - accuracy: 0.9922 - loss: 0.1170
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9827 - Recall: 0.9827 - accuracy: 0.9827 - loss: 0.1181
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9948 - Recall: 0.9948 - accuracy: 0.9948 - loss: 0.1005
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9980 - Recall: 0.9980 - accuracy: 0.9980 - loss: 0.0966
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9911 - Recall: 0.9911 - accuracy: 0.9911 - loss: 0.1062
Epoch 94/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9886 - Recall: 0.9886 - accuracy: 0.9886 - loss: 0.1331
Epoch 95/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - Precision: 0.9940 - Recall: 0.9940 - accuracy: 0.9940 - loss: 0.1181
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9980 - Recall: 0.9980 - accuracy: 0.9980 - loss: 0.0833
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9922 - Recall: 0.9922 - accuracy: 0.9922 - loss: 0.1160
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.9948 - Recall: 0.9948 - accuracy: 0.9948 - loss: 0.0936
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9955 - Recall: 0.9955 - accuracy: 0.9955 - loss: 0.0881
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9931 - Recall:
```

```
0.9931 - accuracy: 0.9931 - loss: 0.0970
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Epoch 1/100

c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\sklearn\metrics\_classification.py:1517:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\sklearn\metrics\_classification.py:1517:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning:
``build_fn`` will be renamed to ``model`` in a future release, at
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4829 - loss: 1.0964
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4387 - loss: 1.0873
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4802 - loss: 1.0772
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4331 - loss: 1.0688
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4835 - loss: 1.0571
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4854 - loss: 1.0484
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
```

```
Recall: 0.0000e+00 - accuracy: 0.5204 - loss: 1.0425
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4493 - loss: 1.0314
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5760 - loss: 1.0241
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5225 - loss: 1.0207
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4874 - loss: 1.0078
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4649 - loss: 1.0018
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4751 - loss: 0.9982
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4825 - loss: 0.9858
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5255 - loss: 0.9807
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4379 - loss: 0.9774
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4645 - loss: 0.9694
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4513 - loss: 0.9617
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4288 - loss: 0.9647
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5400 - loss: 0.9499
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4819 - loss: 0.9498
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5158 - loss: 0.9394
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4607 - loss: 0.9448
```

```
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5890 - loss: 0.9268
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5488 - loss: 0.9252
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5490 - loss: 0.9171
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5211 - loss: 0.9178
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5011 - loss: 0.9191
Epoch 29/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4761 - loss: 0.9063
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5149 - loss: 0.9154
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4935 - loss: 0.9062
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5659 - loss: 0.8935
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4598 - loss: 0.8924
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5750 - loss: 0.9049
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5171 - loss: 0.8838
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4748 - loss: 0.8838
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4397 - loss: 0.8884
Epoch 38/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3956 - loss: 0.8844
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4414 - loss: 0.8767
Epoch 40/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4554 - loss: 0.8987
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5427 - loss: 0.8718
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5022 - loss: 0.8672
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4634 - loss: 0.8664
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4566 - loss: 0.8748
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4894 - loss: 0.8509
Epoch 46/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4708 - loss: 0.8585
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5312 - loss: 0.8467
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4480 - loss: 0.8458
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5141 - loss: 0.8461
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4717 - loss: 0.8405
Epoch 51/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5208 - loss: 0.8414
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5355 - loss: 0.8496
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4727 - loss: 0.8363
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5066 - loss: 0.8420
Epoch 55/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4832 - loss: 0.8425
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
```

```
Recall: 0.0000e+00 - accuracy: 0.4546 - loss: 0.8391
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4403 - loss: 0.8263
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4866 - loss: 0.8316
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4505 - loss: 0.8305
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5127 - loss: 0.8388
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5194 - loss: 0.8477
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5776 - loss: 0.8243
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5140 - loss: 0.8147
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5547 - loss: 0.8140
Epoch 65/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5566 - loss: 0.8170
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4243 - loss: 0.8236
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4917 - loss: 0.8087
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5272 - loss: 0.8039
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5406 - loss: 0.8075
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5445 - loss: 0.7997
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5674 - loss: 0.8066
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4580 - loss: 0.8108
```

```
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4723 - loss: 0.8109
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4146 - loss: 0.8133
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4732 - loss: 0.8035
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5832 - loss: 0.7974
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4492 - loss: 0.7933
Epoch 78/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5357 - loss: 0.8084
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4821 - loss: 0.7899
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4767 - loss: 0.8196
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5099 - loss: 0.8059
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4960 - loss: 0.8026
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4880 - loss: 0.7870
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4699 - loss: 0.7951
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4580 - loss: 0.7981
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5105 - loss: 0.7854
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4669 - loss: 0.7890
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5025 - loss: 0.7923
Epoch 89/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.6052 - loss: 0.7779
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4562 - loss: 0.8371
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5671 - loss: 0.7864
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5477 - loss: 0.7923
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5187 - loss: 0.8169
Epoch 94/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4669 - loss: 0.7813
Epoch 95/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4191 - loss: 0.7901
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4691 - loss: 0.7841
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5675 - loss: 0.7719
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4923 - loss: 0.7724
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.5261 - loss: 0.7758
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.4591 - loss: 0.7751
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step

c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\sklearn\metrics\_classification.py:1517:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\sklearn\metrics\_classification.py:1517:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
```

```
len(result))
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\sklearn\metrics\_classification.py:1517:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in
labels with no true nor predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
precision = results['test_precision'].mean()
recall = results['test_recall'].mean()
f1 = results['test_f1'].mean()

new_row = pd.DataFrame({
    'Accuracy': [accuracy],
    'Variance': [variance],
    'Precision': [precision],
    'Recall': [recall],
    'F1': [f1]
})

resultados = pd.concat([resultados, new_row], ignore_index=True)
resultados = resultados.round(2)
```

```
resultados
```

```
C:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Temp\
ipykernel_14792\1210053740.py:13: FutureWarning: The behavior of
DataFrame concatenation with empty or all-NA entries is deprecated. In
a future version, this will no longer exclude empty or all-NA columns
when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  resultados = pd.concat([resultados, new_row], ignore_index=True)
```

```
   Accuracy  Variance  Precision  Recall   F1
0      0.51      0.22        0.0    0.11  0.0
```

Medició del accuracy sobre el conjunto de prueba

Como vemos, el cross validate parece tener bastantes problemas en clasificar correctamente. Nuevamente, el partir de un dataset de tan solo 150 registros nos debería hacer sospechar que es nesario cambiar el enfoque. Para ello, vamos a calcuar las métricas directamente sobre el conjunto de prueba.

```
x_train, x_test, y_train, y_test = train_test_split(x, dummy_y,
test_size = 0.2, random_state = 0)
```

```python
model = KerasClassifier(build_fn = base_model, batch_size = 5, epochs
= 100)

# Entrenamos
model.fit(x_train, y_train, epochs=100, batch_size=10)

# Hacemos las predicciones sobre los datos de prueba
y_pred = model.predict(x_test)
```

```
Epoch 1/100

c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\scikeras\wrappers.py:925: UserWarning:
``build_fn`` will be renamed to ``model`` in a future release, at
which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - Precision: 0.2884 - Recall:
0.2884 - accuracy: 0.2884 - loss: 1.6500
Epoch 2/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.3646 - Recall:
0.3646 - accuracy: 0.3646 - loss: 1.3968
Epoch 3/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.3177 - Recall:
0.3160 - accuracy: 0.3160 - loss: 1.3812
Epoch 4/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4594 - Recall:
0.3774 - accuracy: 0.3774 - loss: 1.1829
Epoch 5/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.4371 - Recall:
0.3027 - accuracy: 0.3186 - loss: 1.1734
Epoch 6/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.2579 - Recall:
0.1191 - accuracy: 0.3074 - loss: 1.1328
Epoch 7/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3257 - loss: 1.0941
Epoch 8/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3309 - loss: 1.0793
Epoch 9/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
```

```
Recall: 0.0000e+00 - accuracy: 0.2515 - loss: 1.0397
Epoch 10/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3224 - loss: 1.0398
Epoch 11/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3453 - loss: 1.0396
Epoch 12/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3757 - loss: 1.0227
Epoch 13/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3246 - loss: 1.0441
Epoch 14/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3119 - loss: 1.0289
Epoch 15/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.0000e+00 -
Recall: 0.0000e+00 - accuracy: 0.3242 - loss: 1.0247
Epoch 16/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.2308 - Recall:
0.0020 - accuracy: 0.3073 - loss: 1.0225
Epoch 17/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6923 - Recall:
0.0273 - accuracy: 0.3725 - loss: 0.9881
Epoch 18/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9231 - Recall:
0.0624 - accuracy: 0.3879 - loss: 0.9716
Epoch 19/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.0414 - accuracy: 0.3632 - loss: 0.9760
Epoch 20/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8462 - Recall:
0.0519 - accuracy: 0.3818 - loss: 0.9682
Epoch 21/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.1499 - accuracy: 0.3814 - loss: 0.9473
Epoch 22/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.2532 - accuracy: 0.4287 - loss: 0.9300
Epoch 23/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.2207 - accuracy: 0.3767 - loss: 0.9482
Epoch 24/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.1994 - accuracy: 0.4634 - loss: 0.9398
Epoch 25/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.2596 - accuracy: 0.5025 - loss: 0.9164
```

```
Epoch 26/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - Precision: 0.9862 - Recall:
0.3005 - accuracy: 0.5124 - loss: 0.9098
Epoch 27/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 1.0000 - Recall:
0.1837 - accuracy: 0.4488 - loss: 0.9343
Epoch 28/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9687 - Recall:
0.3213 - accuracy: 0.6224 - loss: 0.8763
Epoch 29/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8669 - Recall:
0.4294 - accuracy: 0.6036 - loss: 0.8426
Epoch 30/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8404 - Recall:
0.3323 - accuracy: 0.5858 - loss: 0.8769
Epoch 31/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9631 - Recall:
0.3220 - accuracy: 0.6274 - loss: 0.8680
Epoch 32/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9569 - Recall:
0.3326 - accuracy: 0.6625 - loss: 0.8539
Epoch 33/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8413 - Recall:
0.3648 - accuracy: 0.6936 - loss: 0.8283
Epoch 34/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7972 - Recall:
0.3485 - accuracy: 0.6685 - loss: 0.8258
Epoch 35/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8045 - Recall:
0.3500 - accuracy: 0.7052 - loss: 0.8053
Epoch 36/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8108 - Recall:
0.3735 - accuracy: 0.6845 - loss: 0.7855
Epoch 37/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7523 - Recall:
0.3432 - accuracy: 0.6423 - loss: 0.7927
Epoch 38/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7738 - Recall:
0.2819 - accuracy: 0.6477 - loss: 0.7979
Epoch 39/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8184 - Recall:
0.3729 - accuracy: 0.6993 - loss: 0.7548
Epoch 40/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7388 - Recall:
0.3715 - accuracy: 0.6897 - loss: 0.7456
Epoch 41/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7422 - Recall:
0.3241 - accuracy: 0.6617 - loss: 0.7476
Epoch 42/100
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7677 - Recall: 0.3421 - accuracy: 0.6935 - loss: 0.7308
Epoch 43/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7334 - Recall: 0.3870 - accuracy: 0.7180 - loss: 0.7033
Epoch 44/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6989 - Recall: 0.3435 - accuracy: 0.7045 - loss: 0.7191
Epoch 45/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.6525 - Recall: 0.3433 - accuracy: 0.6547 - loss: 0.7180
Epoch 46/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8298 - Recall: 0.6454 - accuracy: 0.7624 - loss: 0.6593
Epoch 47/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7618 - Recall: 0.6103 - accuracy: 0.7074 - loss: 0.6750
Epoch 48/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8018 - Recall: 0.6331 - accuracy: 0.7020 - loss: 0.6616
Epoch 49/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8063 - Recall: 0.6167 - accuracy: 0.7433 - loss: 0.6639
Epoch 50/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8835 - Recall: 0.6979 - accuracy: 0.7787 - loss: 0.6206
Epoch 51/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7683 - Recall: 0.6167 - accuracy: 0.7180 - loss: 0.6446
Epoch 52/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8130 - Recall: 0.6424 - accuracy: 0.7470 - loss: 0.6301
Epoch 53/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8544 - Recall: 0.7269 - accuracy: 0.7970 - loss: 0.6152
Epoch 54/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7929 - Recall: 0.6563 - accuracy: 0.7478 - loss: 0.6215
Epoch 55/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7416 - Recall: 0.6089 - accuracy: 0.7136 - loss: 0.6230
Epoch 56/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8314 - Recall: 0.6707 - accuracy: 0.7544 - loss: 0.5950
Epoch 57/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8520 - Recall: 0.6885 - accuracy: 0.7931 - loss: 0.5740
Epoch 58/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8362 - Recall:
```

```
0.6763 - accuracy: 0.8059 - loss: 0.5700
Epoch 59/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8373 - Recall:
0.7300 - accuracy: 0.8057 - loss: 0.5516
Epoch 60/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.7902 - Recall:
0.6820 - accuracy: 0.7777 - loss: 0.5638
Epoch 61/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8328 - Recall:
0.6592 - accuracy: 0.8491 - loss: 0.5742
Epoch 62/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8121 - Recall:
0.6578 - accuracy: 0.7891 - loss: 0.5603
Epoch 63/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8225 - Recall:
0.7193 - accuracy: 0.8281 - loss: 0.5427
Epoch 64/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8164 - Recall:
0.7341 - accuracy: 0.8186 - loss: 0.5268
Epoch 65/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8663 - Recall:
0.7338 - accuracy: 0.8504 - loss: 0.5311
Epoch 66/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8717 - Recall:
0.7807 - accuracy: 0.8567 - loss: 0.5147
Epoch 67/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8017 - Recall:
0.7436 - accuracy: 0.8045 - loss: 0.5380
Epoch 68/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8531 - Recall:
0.7954 - accuracy: 0.8393 - loss: 0.5241
Epoch 69/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8133 - Recall:
0.7341 - accuracy: 0.8259 - loss: 0.5340
Epoch 70/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8571 - Recall:
0.8287 - accuracy: 0.8372 - loss: 0.5108
Epoch 71/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8364 - Recall:
0.8198 - accuracy: 0.8300 - loss: 0.5154
Epoch 72/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8414 - Recall:
0.8065 - accuracy: 0.8382 - loss: 0.4957
Epoch 73/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8247 - Recall:
0.8105 - accuracy: 0.8248 - loss: 0.5056
Epoch 74/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8422 - Recall:
0.8137 - accuracy: 0.8207 - loss: 0.5004
```

```
Epoch 75/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8998 - Recall:
0.8471 - accuracy: 0.8930 - loss: 0.4826
Epoch 76/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8504 - Recall:
0.8365 - accuracy: 0.8405 - loss: 0.4905
Epoch 77/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8727 - Recall:
0.8496 - accuracy: 0.8566 - loss: 0.4755
Epoch 78/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8310 - Recall:
0.7915 - accuracy: 0.8195 - loss: 0.5202
Epoch 79/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8848 - Recall:
0.8479 - accuracy: 0.8647 - loss: 0.4863
Epoch 80/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8464 - Recall:
0.8261 - accuracy: 0.8297 - loss: 0.4757
Epoch 81/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8448 - Recall:
0.8271 - accuracy: 0.8328 - loss: 0.4897
Epoch 82/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9109 - Recall:
0.8745 - accuracy: 0.8781 - loss: 0.4547
Epoch 83/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8913 - Recall:
0.8695 - accuracy: 0.8941 - loss: 0.4735
Epoch 84/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8735 - Recall:
0.8625 - accuracy: 0.8645 - loss: 0.4765
Epoch 85/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8805 - Recall:
0.8482 - accuracy: 0.8764 - loss: 0.4699
Epoch 86/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8470 - Recall:
0.8448 - accuracy: 0.8461 - loss: 0.4660
Epoch 87/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8574 - Recall:
0.8223 - accuracy: 0.8423 - loss: 0.4782
Epoch 88/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9025 - Recall:
0.8739 - accuracy: 0.8860 - loss: 0.4600
Epoch 89/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8745 - Recall:
0.8598 - accuracy: 0.8727 - loss: 0.4304
Epoch 90/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8953 - Recall:
0.8738 - accuracy: 0.8878 - loss: 0.4418
Epoch 91/100
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8691 - Recall:
0.8538 - accuracy: 0.8667 - loss: 0.4306
Epoch 92/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8728 - Recall:
0.8665 - accuracy: 0.8665 - loss: 0.4396
Epoch 93/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8789 - Recall:
0.8731 - accuracy: 0.8777 - loss: 0.4184
Epoch 94/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9168 - Recall:
0.9083 - accuracy: 0.9176 - loss: 0.4104
Epoch 95/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8885 - Recall:
0.8852 - accuracy: 0.8888 - loss: 0.4377
Epoch 96/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9323 - Recall:
0.9222 - accuracy: 0.9250 - loss: 0.4136
Epoch 97/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9126 - Recall:
0.8898 - accuracy: 0.8898 - loss: 0.4115
Epoch 98/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.8628 - Recall:
0.8628 - accuracy: 0.8628 - loss: 0.4302
Epoch 99/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9097 - Recall:
0.9023 - accuracy: 0.9023 - loss: 0.3942
Epoch 100/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - Precision: 0.9442 - Recall:
0.8994 - accuracy: 0.9193 - loss: 0.3862
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

# Convertimos las predicciones y los datos reales a un formato
adecuado
y_pred_flatten = np.array(y_pred).flatten()
y_test_flatten = np.array(y_test).flatten()

accuracy = accuracy_score(y_test_flatten, y_pred_flatten)
variance = np.var(y_pred_flatten)

precision = precision_score(y_test_flatten, y_pred_flatten,
zero_division=1)
recall = recall_score(y_test_flatten, y_pred_flatten, zero_division=1)
f1 = f1_score(y_test_flatten, y_pred_flatten, zero_division=1)

# Crear un DataFrame con la nueva fila de resultados
new_row = pd.DataFrame({
    'Accuracy': [accuracy],
    'Variance': [variance],
    'Precision': [precision],
    'F1': [f1],
```

```
    'Recall': [recall]
})

resultados = pd.concat([resultados, new_row], ignore_index=True)
resultados = resultados.round(2)


resultados
```

	Accuracy	Variance	Precision	Recall	F1
0	0.51	0.22	0.00	0.11	0.00
1	0.96	0.22	0.93	0.93	0.93

Resultados

En las primeras versiones del código, la matriz de confusión tenía el siguiente aspecto:

[[12 0 0] [0 0 10] [0 0 8]]

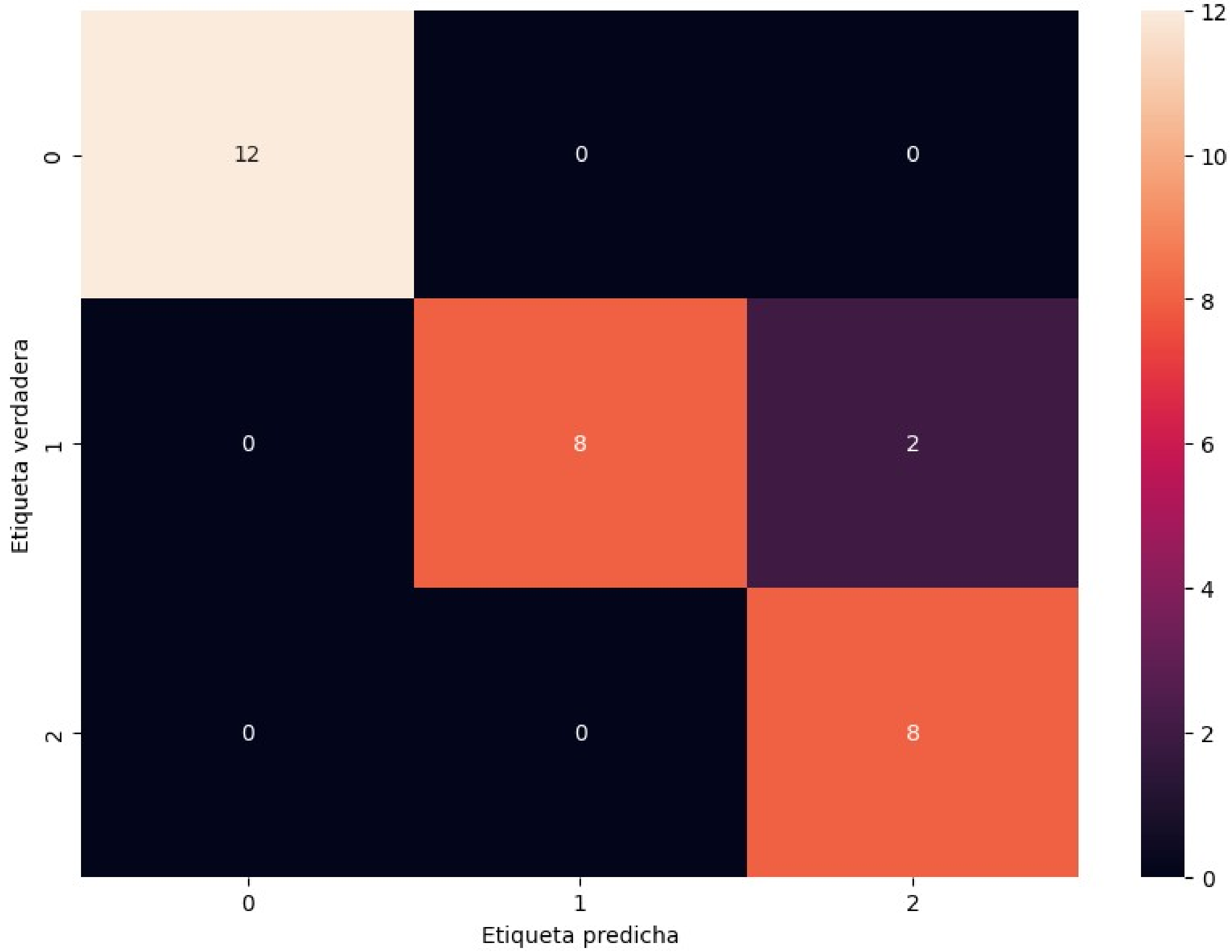
Esta matriz revela que efectivamente al principio podía clasificar bien la etiquetas 0 y 2 (setosa y virginica). Sin embargo es incapaz de detectar la versicolor y la clasifica como virginica. Esto nos lleva a pensar que el modelo tiene un sesgo hacia la clase virginica.

Tras realizar varias pruebas, logramos obtener un accuracy del 0.96. Este resultado es bastante bueno, y nos permite afirmar que el modelo es capaz de clasificar correctamente el 96% de las flores. Teniendo en cuenta que las muestras son bastante balanceadas, esta métrica debería ser suficiente.

Aún así, y solo para garantizar que el modelo funciona, haremos una matriz de confusión en la que podemos ver cómo está clasificando las flores.

```
# Pequeño print para ver que todo está en orden y no se me han movido
las variables
print(x_test.shape, y_test.shape,  y_pred.shape, x_test.shape)

(30, 4) (30, 3) (30, 3) (30, 4)


# Como la salida son probabilidades (recordamos que la salida de la
rna es softmax), convertimos a etiquetas
try:
    y_pred = np.argmax(y_pred, axis=1)
    y_test = np.argmax(y_test, axis=1)

    # Verificamos los valores únicos
    print("Clases únicas en y_pred:", np.unique(y_pred))
    print("Clases únicas en y_test:", np.unique(y_test))

except ValueError:
    print("Ya están en formato de etiquetas, no necesitas calcular el
```

```
máximo valor")


Ya están en formato de etiquetas, no necesitas calcular el máximo
valor



# Matriz de confusión
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Visualización de la matriz
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.ylabel('Etiqueta verdadera')
plt.xlabel('Etiqueta predicha')
plt.show()

[[12  0  0]
 [ 0  8  2]
 [ 0  0  8]]
```



Dado el siguiente objetivo para el ejercicio:

"El objetivo será proponer la estructura de una red neuronal artificial que proporcione una precisión elevada del conjunto de prueba (> 85%)".

Podemos afirmar que el modelo propuesto cumple.

Nota Los resultados pueden variar debido a la naturaleza estocástica del algoritmo o del procedimiento de evaluación, o a las diferencias en la precisión numérica. Considerad la posibilidad de ejecutar el ejercicio varias veces y comparad el resultado medio.

03 Ejercicio: Problema de clasificación multiclase de diferentes artículos de ropa y calzados

En este ejercicio utilizaremos el conjunto de datos de *Fashion-MNIST* que viene precargado en la librería de Keras. Os dejo el enlace al repositorio de GitHub https://github.com/zalandoresearch/fashion-mnist.

Fashion-MNIST es un conjunto de datos de las imágenes de los artículos de Zalando, una tienda de moda online alemana especializada en venta de ropa y zapatos. EL conjunto de datos contiene 70000 imágenes en escala de grises en 10 categorías. Las imágenes muestran prendas individuales de ropa en baja resolución (28x28 píxeles). Se van a utilizar 60000 imágenes para entrenar la red y 10000 imágenes para evaluar la precisión con la que la red aprende a clasificar las imágenes.

Por tanto, se trata de un problema de clasificación multiclase, lo que significa que hay más de dos clases que predecir, de hecho, vamos a considerar diez clases de artículos de ropa. El objetivo será proponer la estructura de una red neuronal de convolución que proporcione una precisión elevada del conjunto de prueba (> 80%). En el caso de que no se alcance en la primera aproximación tendréis que tomar medidas para mejorar el proceso de diseño y entrenamiento de la red en cuestión hasta alcanzar dicho objetivo.

03 Solución ejercicio: Problema de clasificación multiclase de diferentes artículos de ropa y calzados

Veamos paso a paso como resolvemos dicho ejercicio.

Paso 1: Preparación de los datos

Como siempre, antes de empezar a programar nuestra red neuronal debemos importar todas las librerías que se van a requerir (y asegurarnos de que estamos ejecutando la versión correcta de TensorFlow en nuestro Colab).

```
# Cargamos las librerías necesarias
# %tensorflow_version 2.x

import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

```
2.17.0
```

```
# Cargamos el conjunto de datos precargados en Keras
fashion_mnist = keras.datasets.fashion_mnist
```

Preparación del dataset

El método load_data() devuelve dos tuplas, una para el conjunto de entrenamiento y otra para el de test.

1. La tupla del conjunto de entrenamiento se compone de dos arrays: uno con las imágenes que va a entrenar y otra con las etiquetas de las imágenes reales que servirán durante el proceso de retropropagación para comprobar si la red está aprendiendo correctamente.

2. La segunda tupla contiene el dataset con las imágenes de test y las etiquetas reales de las imágenes de test. Cuando llegue el momento de hacer las predicciones, una vez el modelo esté entrenado, bastará con confrontar la predicción vs la etiqueta de test real.

Preparación de un dataset con imágenes

En caso de que nosotros querásemos preparar nuestro propio dataset de imágenes, utilizaríamos el ImageGenarator facilitado por Keras:

````
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```
````

Nota: para más detalles sobre como clasificar imágenes, consultar los datasets de Animals.

```python
# Obtenemos el conjunto de train y test preparado

(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()
```

Como podéis observar la carga del conjunto de datos devuelve cuatro matrices Numpy. Las matrices *train_images* y *train_labels* son el conjunto de entrenamiento. Las matrices *test_images* y *test_labels* son el conjunto de prueba para evaluar la precisión del modelo.

Las imágenes son matrices NumPy de 28x28 píxeles, con valores que van de 0 a 255. Las etiquetas son una matriz de enteros, que van de 0 a 9. Estos corresponden a la clase de ropa que representa la imagen:

Clase	Tipo
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

```
# Dado que los nombres de clase no se incluyen con el conjunto de
datos, podemos crear una lista con ellos para usarlos más adelante al
visualizar las imágenes:
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Vamos a escalar los valores de entrada en el rango 0-1
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')

train_images = train_images / 255.0
test_images = test_images / 255.0


# Recordar que es una buena práctica comprobar que los datos tienen la
forma que esperamos

print("train_images.shape:",train_images.shape)
print("len(train_labels:",len(train_labels))
```

```
print("test_images.shape:",test_images.shape)
print("len(test_labels):",len(test_labels))
```

```
train_images.shape: (60000, 28, 28)
len(train_labels: 60000
test_images.shape: (10000, 28, 28)
len(test_labels): 10000
```

```
# y que las muestras y etiquetas son los valores que esperamos
train_labels
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

```
# Visualización las 50 primeras imágenes del conjunto de datos
Fashion-MNIST
plt.figure(figsize=(12,12))
for i in range(50):
    plt.subplot(10,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Paso 2: Definimos la arquitectura de la red neuronal

Tened en cuenta que Keras nos facilita el paso de reconvertir las muestras de entrada de 28×28 a un vector (array) de 784 números (concatenando fila a fila) con el uso de la capa *keras.layers.Flatten()*. Podemos comprobar con el método *summary()* que esta capa no requiere parámetros para aplicar la transformación (columna Param #). En general, siempre usaremos esta capa del modelo para hacer esta operación en lugar de redimensionar el tensor de datos antes de la entrada.

```
# Cargamos las librerías necesarias para configurar la red
import keras
from keras.models import Sequential
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Conv2D
from keras.layers import MaxPooling2D

# Vamos a comenzar con una red neuronal sencilla
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(6, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compilamos el modelo con SGD (requisito del ejercicio)
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Hacemos un summary de la red considerada
model.summary()
```

```
Model: "sequential_5"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ flatten_16 (Flatten)             │ (None, 784)             │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ dense_21 (Dense)                 │ (None, 6)               │
4,710 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ dense_22 (Dense)                 │ (None, 10)              │
70 │
└──────────────────────────────────┴─────────────────────────┴──────────
─────┘

 Total params: 4,780 (18.67 KB)

 Trainable params: 4,780 (18.67 KB)

 Non-trainable params: 0 (0.00 B)
```

Paso 4: Entrenamiento del modelo de red neuronal utilizado

Ahora el modelo ya está listo para entrenar mediante el método fit(), actualizando los parámetros de tal manera que aprenda a asociar imágenes a etiquetas. Como se puede observar, a medida que el modelo entrena, se muestran las métricas de loss y accuracy.

En este caso (pueden cambiar los valores cuando ustedes lo probéis) este modelo alcanza una precisión de, aproximadamente, 0.7951 (o 79.5 %) en los datos de entrenamiento, pasando todas las imágenes por la red neuronal 5 veces (5 épocas, o epochs).

```
# Realizamos el proceso de entrenamiento sobre el conjunto de train
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.5492 - loss:
1.2986
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7816 - loss:
0.6558
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8049 - loss:
0.5690
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8128 - loss:
0.5375
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8210 - loss:
0.5136
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8274 - loss:
0.4978
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8277 - loss:
0.4890
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8323 - loss:
0.4804
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8354 - loss:
0.4697
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8361 - loss:
0.4627

<keras.src.callbacks.history.History at 0x1ae862cca40>
```

Paso 5: Evaluación del modelo de red neuronal utilizado

El siguiente paso es comparar el rendimiento del modelo en el conjunto de datos de prueba. Vemos que es aproximadamente la misma precisión que en los datos de entrenamiento. Buenas noticias!! No existe el sobreajuste.

```
# Realizamos el proceso de validación sobre el conjunto de test con
model.evaluate para obtener pérdida y accuracy
loss, accuracy = model.evaluate(test_images,  test_labels)

313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8273 - loss:
0.4879

# Obtenemos por pantalla el resultado
print('Accuracy:', accuracy)
print('Loss:', loss)

Accuracy: 0.8245000243186951
Loss: 0.49527764320373535
```

Paso 6: Predicciones del modelo de red neuronal utilizado

Con el modelo entrenado, podemos empezar a usarlo para hacer predicciones sobre algunas imágenes (usemos por comodidad alguna de las imágenes de prueba que ya tenemos cargadas en el notebook). En predictions vamos a almacenar la predicción de la etiqueta para cada imagen en el conjunto de prueba. Echemos un vistazo a la primera predicción:

```
# Guardamos las predicciones realizadas sobre el conjunto de test
predictions = model.predict(test_images)

"""
Esto nos devuelve un array que a su vez contiene un array para cada
indice
 con las probabilidades asignadas a cada etiqueta.
"""

predictions[1]

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step

array([6.3356420e-04, 1.5785750e-07, 8.1239539e-01, 3.7995583e-06,
       2.7996181e-02, 1.7570055e-12, 1.5889896e-01, 2.6847005e-19,
       7.1903989e-05, 1.1912237e-12], dtype=float32)

# Se puede ver qué etiqueta tiene el valor de confianza más alto con
la función argmax


# Obtenemos la información sobre una de las predicciones obtenidas

predictions[5]
"""
argmax nos devuelve el índice de la etiqueta con la probabilidad más
alta (para no tener que buscarlo manualmente).
"""
mas_probable = np.argmax(predictions[5])
```

```
print(mas_probable, '--->' ,class_names[mas_probable])
print ("valor real: ", test_labels[5], class_names[test_labels[5]] )
```

```
1 ---> Trouser
valor real:  1 Trouser
```

El modelo está más seguro de que esta imagen son unos pantalones (Trouser) ya que nos reporta una clase igual a 1. Al examinar la etiqueta que le corresponde muestra que esta clasificación es correcta ya que es igual a 1 también.

A continuación, vamos a aprovechar igualmente para graficar hasta 50 predicciones realizadas para ver qué tal se comporta.

```
plt.figure(figsize=(12,12))

for index in range (50):
    mas_probable = np.argmax(predictions[index])
    plt.subplot(10,5,index+1)
    plt.title("Etiqueta real " + class_names[test_labels[index]])
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[index], cmap=plt.cm.binary)
    plt.xlabel(f"Prediccion:  {class_names[mas_probable]}")
plt.tight_layout()
plt.show()
```

Paso 7: Mejora del modelo de red neuronal utilizado

Podemos observar que la precisión obtenida de este modelo para estos datos (que suele rondar el 75-80 %) dista mucho de ser la mejor de las que podemos obtener. Tener en cuenta que no hay una solución única para todos los problemas, sino que cada problema requiere su propia solución. Intentemos, por ejemplo, cambiar el optimizador usado.

Recordemos que el optimizador es el algoritmo usado por el modelo para actualizar los pesos de cada una de sus capas en el proceso de entrenamiento. Una elección bastante habitual es el optimizador *sgd*, pero hay más como sabemos, como por ejemplo el optimizador *Adam*, que a veces puede hacer converger mejor el proceso de optimización. Vamos a probar.

```
"""
Reproducimos el modelo anterior y solamente cambiamos el optimizador a
Adam
"""

# Vamos a comenzar con una red neuronal sencilla
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(6, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compilamos el modelo con SGD (requisito del ejercicio)
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
c:\Users\Administrador.CRISASUSESTUDIO\AppData\Local\Programs\Python\
Python312\Lib\site-packages\keras\src\layers\reshaping\flatten.py:37:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
# Realizamos el proceso de entrenamiento sobre el conjunto de train
considerando el nuevo modelo de red neuronal
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.7796 - loss:
0.6214
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8129 - loss:
0.5372
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8230 - loss:
0.5157
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8233 - loss:
0.5077
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8284 - loss:
0.4974
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8340 - loss:
0.4842
Epoch 7/10
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8332 - loss:
0.4794
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8365 - loss:
0.4733
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8356 - loss:
0.4723
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8392 - loss:
0.4619

<keras.src.callbacks.history.History at 0x1ae80fd3f20>

# Realizamos el proceso de validación sobre el conjunto de test el
nuevo modelo de red neuronal
test_loss, test_acc = model.evaluate(test_images,  test_labels)

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8306 - loss:
0.4860

# Obtenemos por pantalla el resultado
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)


Test accuracy: 0.8259999752044678

Test loss: 0.5034791827201843
```

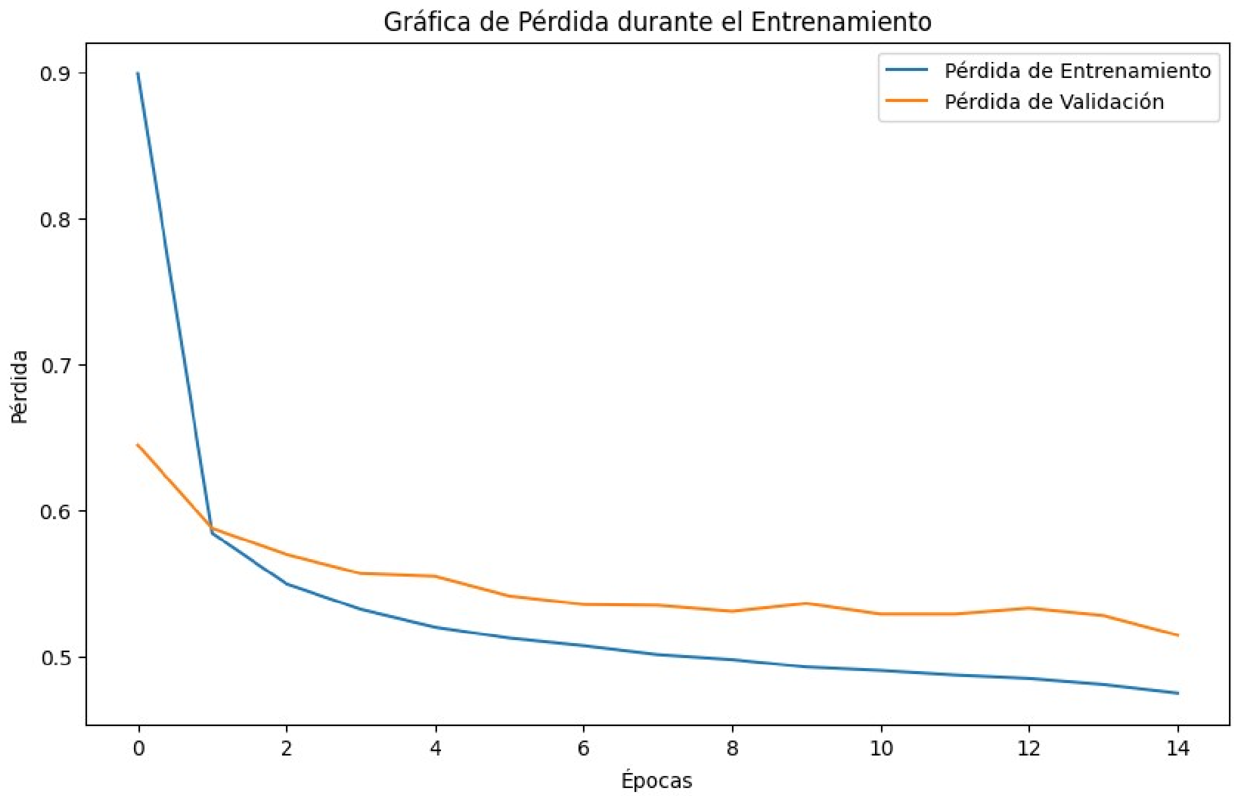
En realidad, parece que haber cambiado el optimizador a Adam tampoco parece haber cambiado demasiado nuestra precisión. Quizás podríamos entrenar un poco más el modelo para ver si mejora.

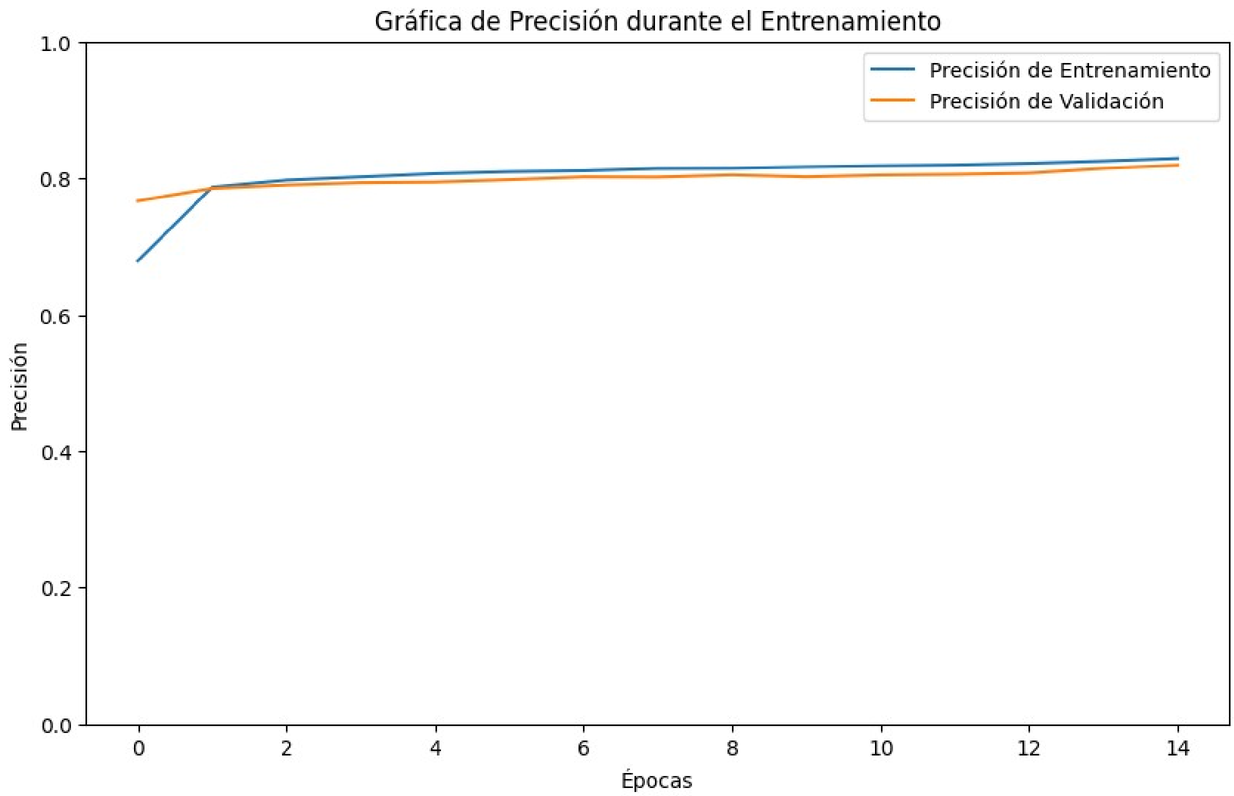
```
history = model.fit(train_images, train_labels,
validation_data=(test_images, test_labels), epochs=15)

# Graficar la pérdida
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Gráfica de Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()

# Graficar la precisión
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Precisión de
```

```
Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión de
Validación')
plt.title('Gráfica de Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.ylim([0, 1])
plt.legend()
plt.show()
```

```
Epoch 1/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.5373 - loss:
1.2819 - val_accuracy: 0.7668 - val_loss: 0.6444
Epoch 2/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7818 - loss:
0.6007 - val_accuracy: 0.7851 - val_loss: 0.5873
Epoch 3/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7967 - loss:
0.5530 - val_accuracy: 0.7901 - val_loss: 0.5695
Epoch 4/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8021 - loss:
0.5323 - val_accuracy: 0.7935 - val_loss: 0.5564
Epoch 5/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8097 - loss:
0.5161 - val_accuracy: 0.7944 - val_loss: 0.5544
Epoch 6/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8058 - loss:
0.5188 - val_accuracy: 0.7979 - val_loss: 0.5411
Epoch 7/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8111 - loss:
0.5065 - val_accuracy: 0.8020 - val_loss: 0.5352
Epoch 8/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8152 - loss:
0.4999 - val_accuracy: 0.8017 - val_loss: 0.5346
Epoch 9/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8159 - loss:
0.4947 - val_accuracy: 0.8049 - val_loss: 0.5305
Epoch 10/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8176 - loss:
0.4892 - val_accuracy: 0.8021 - val_loss: 0.5359
Epoch 11/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8196 - loss:
0.4881 - val_accuracy: 0.8047 - val_loss: 0.5286
Epoch 12/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8184 - loss:
0.4882 - val_accuracy: 0.8057 - val_loss: 0.5286
Epoch 13/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8199 - loss:
0.4868 - val_accuracy: 0.8076 - val_loss: 0.5326
Epoch 14/15
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8247 - loss: 0.4765 - val_accuracy: 0.8145 - val_loss: 0.5276
Epoch 15/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8303 - loss: 0.4706 - val_accuracy: 0.8191 - val_loss: 0.5142
```





Mejorando nuestra red neuronal

Aumentar el entrenamiento o cambiar el optimizador no mejora nuestro reconocimiento de prendas de ropa, así que vamos a intentar profundizar un poco más en la materia para ver qué podemos hacer.

Posiblemente, si las imágenes tuviesen mayor resolución, podríamos intententar algún algoritmo como el de VGG16. Pero no es el caso, optaremos por intentar "copiar" una red neuronal que ya exista y que sepamos que puede funcionar. Concretamente, vamos a imitar la propuesta vista en la UF_5(Redes neuronales y Deep Learning) de AlexNet, que tiene las siguientes características:

1. Utilia funciones de activación con 5 capas convolucionales.
2. Utiliza 3 capas de pooling que se intercalan con la capa convolucional 1, 2 y 5.
3. Utiliza 3 capas densas para la clasificación al final de la red.
4. 1 capa de aplanado -flatten- y 2 capas de dropout al 0.5.
5. Utiliza un clasificador de softmax.

En resumen, si quisiéramos imitar la arquitectura de AlexNet propuesta en los apuntes de la UF-5, deberíamos plantear lo siguiente:

1. Convolution 11x11 kernel +4 stride
2. Maxpooling 3x3 kernel +2 stride
3. Convolution 5x5 kernel +2 pad
4. Maxpooling 3x3 kernel +2 stride

5. Convolution 3x3 kernel +1 pad
6. Convolution 3x3 kernel +1 pad
7. Convolution 3x3 kernel +1 pad
8. Maxpooling 3x3 kernel +2 stride
9. Flatten
10. Dense 4096 -> ReLu
11. Dropout 0.5
12. Dense 4096 -> ReLu
13. Dropout 0.5
14. Dense 1000 -> Softmax

Sin embargo, dado el alto coste computacional de usar esta red neuronal (que estamos entrando en un entorno local y con imágenes de por sí de baja resolución), vamos simplificar esta arquitectura.

Concretamente, haremos lo siguiente:

1. Limitaremos el número de capas convolucionales (en lugar de 5, solamente usaremos 3 para entrenar más rápido).
2. Utilizaremos solamente 2 capas densas (una ReLu y la Softmax de la salida) y solo 1 de dropout.
3. Para simplificar el código, no vamos a parametrizar el stride y no vamos a tocar el padding.
4. Al mismo tiempo, contemplaremos un número de epoc que irá entre 5 y 10, para no sobrecargar el entrenamiento.

```
model_2 = Sequential()

# Primera capa convolucional (vamos a trabajar con un kernel de 3x3 
para todas las capas)
model_2.add(Conv2D(32, (3, 3), activation='relu', padding='same', 
input_shape=(28, 28, 1)))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

# Segunda capa convolucional
model_2.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

# Tercera capa convolucional
model_2.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

# Aplanamiento antes de llegar a las capas densas y que llegue un 
array de una sola dimensión
model_2.add(Flatten())

# Primera capa densa
model_2.add(Dense(128, activation='relu'))
```

```
model_2.add(Dropout(0.5))

# Capa de salida con 10 unidades (para las 10 clases en Fashion MNIST)
model_2.add(Dense(10, activation='softmax'))

# Compilación del modelo con adam y únicamente usamos la métrica de
accuracy
model_2.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Explicación de esta red neuronal

¿Qué haríamos con cada capa?

-Las capas convolucionales se encargarían de capturar las características de las imágenes (bordes, texturas, etc.).

-Las capas de pooling se utilizan para reducir la dimensionalidad de los datos de entrada y quedarnos con las características más importantes.

-La capa de flatten se utiliza para convertir los datos de entrada en un vector unidimensional. Este paso siempre lo hacemos antes de llegar a las capas densas.

-Las capas densas se utilizan para clasificar las imágenes en las diferentes categorías. Concretamente, la función ReLu se utiliza para introducir no linealidades en la red neuronal, mientras que la función Softmax se utiliza para obtener la probabilidad de que una imagen pertenezca a una determinada categoría.

-El Dropout, finalmente, contribuirá a evitar el sobreajuste de la red neuronal.

Parámetros de las capas

-el kernel (normalmente 3x3 o 5x5) es una pequeña matriz de pesos que se utiliza para detectar características específicas en la entrada, como bordes, texturas, patrones, y otros detalles relevantes de la imagen.

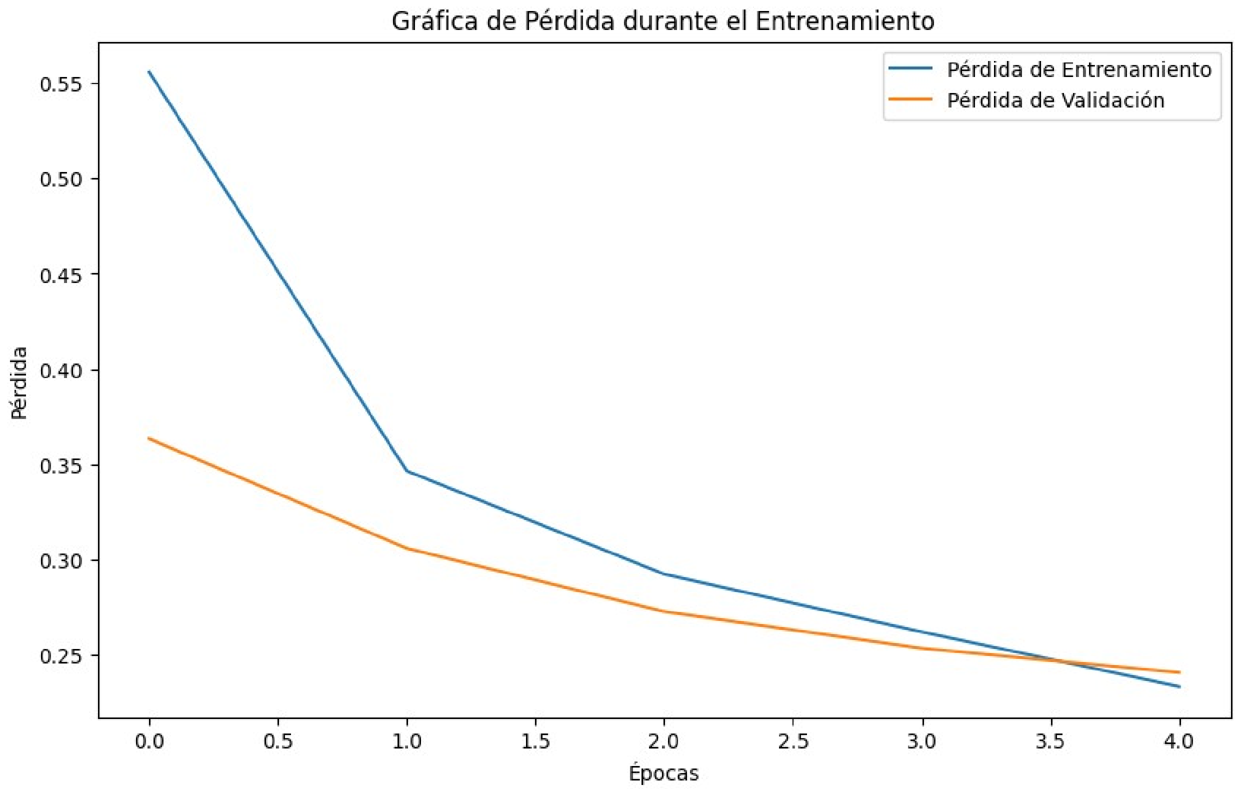
-El stride (que no hemos incluido) es el número de píxeles que el filtro se mueve a medida que se aplica sobre la entrada.

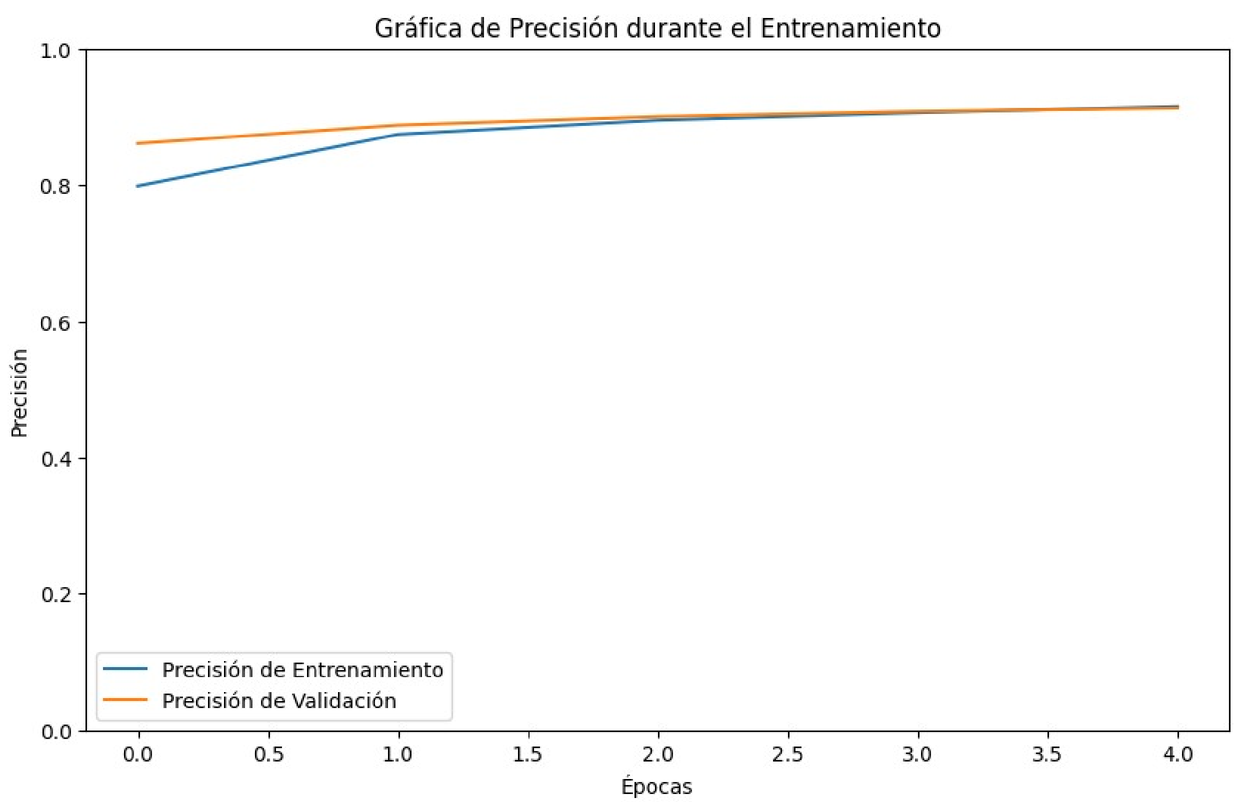
-El padding (que dejamos en "same") se refiere a agregar píxeles adicionales (generalmente ceros) alrededor de la imagen de entrada antes de aplicar el filtro.

```
history_2 = model_2.fit(train_images, train_labels,
validation_data=(test_images, test_labels), epochs=5)

# Graficar la pérdida
plt.figure(figsize=(10, 6))
plt.plot(history_2.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history_2.history['val_loss'], label='Pérdida de Validación')
```

```python
plt.title('Gráfica de Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()

# Graficar la precisión
plt.figure(figsize=(10, 6))
plt.plot(history_2.history['accuracy'], label='Precisión de
Entrenamiento')
plt.plot(history_2.history['val_accuracy'], label='Precisión de
Validación')
plt.title('Gráfica de Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.ylim([0, 1])
plt.legend()
plt.show()
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 31s 15ms/step - accuracy: 0.7113 -
loss: 0.7902 - val_accuracy: 0.8611 - val_loss: 0.3632
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.8660 -
loss: 0.3666 - val_accuracy: 0.8878 - val_loss: 0.3058
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 31s 16ms/step - accuracy: 0.8928 -
loss: 0.3010 - val_accuracy: 0.9003 - val_loss: 0.2726
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.9057 -
loss: 0.2617 - val_accuracy: 0.9080 - val_loss: 0.2533
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 29s 15ms/step - accuracy: 0.9153 -
loss: 0.2319 - val_accuracy: 0.9129 - val_loss: 0.2406
```


Gráfica de Pérdida durante el Entrenamiento
Pérdida de Entrenamiento
Pérdida de Validación
Pérdida
Épocas
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Gráfica de Precisión durante el Entrenamiento
Precisión de Entrenamiento
Precisión de Validación
Precisión
Épocas
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Resultados

Las primeras versiones del entrenamiento o resultaban demasiado costosas, o no daban resultados muy precisos. Sin embargo, tras simplificar la red neuronal de AlexNet, obtenemos un accuracy del 91%, lo cuál no está nada mal para el ejercicio propuesto.