Clasificación de tumores de mama utilizando MLP con TensorFlow-Keras

Integrantes:

Ing. Dewins Murillo García

Introducción

¡Bienvenido a mi Google Colab! En esta presentación, vamos a explorar el emocionante campo de la clasificación de tumores de mama utilizando una Red Neuronal Multicapa (MLP) implementada con TensorFlow-Keras. Para este propósito, utilizaremos el conocido conjunto de datos del "Wisconsin Diagnostic Breast Cancer" (WDBC).

El cáncer de mama es una enfermedad grave que afecta a millones de mujeres en todo el mundo. La capacidad de diagnosticar con precisión si un tumor es benigno o maligno es crucial para determinar el tratamiento adecuado. Aquí es donde entra en juego el aprendizaje automático. Mediante el análisis de características de tumores de mama, podemos entrenar un modelo de MLP para realizar predicciones precisas sobre la malignidad de un tumor.

Objetivos

Los objetivos de esta presentación son los siguientes:

1. Familiarizarse con el conjunto de datos WDBC y su estructura.
2. Preprocesar los datos para su uso en un modelo de MLP.
3. Diseñar, entrenar y evaluar una red neuronal MLP utilizando TensorFlow-Keras.
4. Utilizar el modelo entrenado para realizar predicciones sobre nuevos casos de tumores de mama.
5. Evaluar la precisión y rendimiento del modelo y discutir posibles mejoras.

Utilizaremos el poder de TensorFlow-Keras para construir una red neuronal profunda y aprovecharemos las capacidades del conjunto de datos WDBC para lograr resultados significativos.

Marco teórico

El cáncer de mama es una enfermedad maligna que se origina en las células del tejido mamario. Es una de las principales causas de muerte en mujeres en todo el mundo. El diagnóstico temprano y

preciso del cáncer de mama es esencial para un tratamiento efectivo y mejores resultados para los pacientes.

El aprendizaje automático, específicamente la clasificación de tumores de mama utilizando redes neuronales, ha demostrado ser una herramienta prometedora en la detección y diagnóstico de esta enfermedad. Las redes neuronales artificiales son modelos computacionales inspirados en el funcionamiento del cerebro humano, capaces de aprender patrones y realizar predicciones a partir de datos.

En este contexto, el conjunto de datos del "Wisconsin Diagnostic Breast Cancer" (WDBC) se ha utilizado ampliamente en la investigación sobre la clasificación de tumores de mama. Este conjunto de datos contiene mediciones de características obtenidas a partir de imágenes digitalizadas de aspiraciones con aguja fina (FNA) de tumores mamarios. Las características incluyen propiedades del núcleo de las células, como el tamaño, la forma y la textura.

El preprocesamiento de los datos es una etapa crítica en el desarrollo de modelos de aprendizaje automático. En este caso, será necesario realizar tareas como la normalización de características, la codificación de etiquetas y la división del conjunto de datos en conjuntos de entrenamiento y prueba. Estas etapas aseguran que los datos sean adecuados para el entrenamiento y la evaluación de la red neuronal.

La red neuronal multicapa (MLP) es un tipo de arquitectura de red neuronal ampliamente utilizada en clasificación. Consiste en capas de neuronas interconectadas, donde cada neurona está conectada a las neuronas de la capa siguiente mediante pesos sinápticos. Durante el entrenamiento, los pesos de las conexiones se ajustan para minimizar una función de pérdida, lo que permite a la red aprender a clasificar correctamente los tumores de mama como benignos o malignos.

TensorFlow-Keras es una biblioteca de código abierto ampliamente utilizada para la implementación de redes neuronales. Proporciona una interfaz fácil de usar para definir y entrenar modelos de aprendizaje profundo, lo que facilita la implementación de la MLP para la clasificación de tumores de mama.

Una vez entrenado el modelo MLP, se puede utilizar para realizar predicciones sobre nuevos casos de tumores de mama. Estas predicciones proporcionan una estimación de la malignidad de un tumor, lo que ayuda a los médicos a tomar decisiones informadas sobre el tratamiento y la atención de los pacientes.

La evaluación del modelo es esencial para comprender su rendimiento y precisión. Se pueden utilizar métricas como la precisión, la sensibilidad y la especificidad para medir la capacidad del modelo para clasificar correctamente los tumores de mama. Además, se pueden realizar análisis adicionales para identificar posibles mejoras en el modelo, como el ajuste de hiperparámetros o la exploración de otras arquitecturas de redes neuronales.

Descripción del problema

El problema a solucionar en este caso es la clasificación precisa de tumores de mama como benignos o malignos. El cáncer de mama es una enfermedad grave que afecta a millones de mujeres en todo el mundo, y la capacidad de diagnosticar con precisión la malignidad de un tumor es crucial para determinar el tratamiento adecuado.

Actualmente, los médicos utilizan diferentes técnicas y pruebas, como la biopsia y el análisis de imágenes, para determinar si un tumor es benigno o maligno. Sin embargo, estas técnicas pueden ser subjetivas y dependientes de la experiencia del médico, lo que puede llevar a errores de diagnóstico.

El aprendizaje automático y las redes neuronales ofrecen una oportunidad de mejorar la precisión en la clasificación de tumores de mama. Al utilizar un conjunto de datos como el "Wisconsin Diagnostic Breast Cancer" (WDBC), que contiene mediciones de características de tumores mamarios, es posible entrenar un modelo de redes neuronales para aprender patrones y realizar predicciones precisas sobre la malignidad de un tumor.

La solución propuesta consiste en diseñar, entrenar y evaluar una red neuronal multicapa (MLP) implementada con TensorFlow-Keras utilizando el conjunto de datos WDBC. La red neuronal aprenderá a reconocer patrones en las características de los tumores de mama y realizará predicciones sobre la malignidad de nuevos casos.

Al lograr una clasificación más precisa de los tumores de mama, se espera mejorar el proceso de diagnóstico y tratamiento. Los médicos podrán contar con un apoyo adicional para tomar decisiones informadas sobre el tratamiento de los pacientes, lo que puede conducir a una detección temprana del cáncer de mama y a una mejor supervivencia y calidad de vida para los pacientes.

Planteamiento de la solución

Para abordar el problema de la clasificación de tumores de mama, utilizando una Red Neuronal Multicapa (MLP) implementada con TensorFlow-Keras, se propone el siguiente planteamiento de solución:

1. **Familiarización con el conjunto de datos WDBC**:

- Analizar la estructura del conjunto de datos WDBC, comprendiendo las características disponibles y las etiquetas de clasificación (benigno/maligno).

- Realizar una exploración inicial de los datos para identificar posibles desequilibrios de clases, valores atípicos u otros desafíos que puedan afectar la calidad del modelo.

2. **Preprocesamiento de los datos**:

- Realizar un proceso de limpieza de datos, tratando valores faltantes, eliminando características irrelevantes o redundantes si es necesario.
- Realizar la normalización o estandarización de las características para asegurar que todas tengan la misma escala y no se vea afectada la eficiencia del modelo.
- Codificar las etiquetas de clasificación (benigno/maligno) en valores numéricos.

3. **Diseño y entrenamiento de la red neuronal MLP**:

- Definir la arquitectura de la red neuronal MLP, incluyendo el número de capas ocultas, el número de neuronas en cada capa, las funciones de activación, entre otros hiperparámetros.
- Dividir el conjunto de datos en conjuntos de entrenamiento y prueba.
- Entrenar la red neuronal utilizando el conjunto de entrenamiento, ajustando los pesos sinápticos mediante el algoritmo de retropropagación y minimizando una función de pérdida adecuada.
- Evaluar el rendimiento de la red neuronal durante el entrenamiento, monitoreando métricas como la precisión, la pérdida y la precisión por clase.

4. **Evaluación del modelo y predicciones**:

- Evaluar el rendimiento final del modelo utilizando el conjunto de prueba, calculando métricas de evaluación, como la precisión, la sensibilidad, la especificidad y la curva ROC.
- Utilizar el modelo entrenado para realizar predicciones sobre nuevos casos de tumores de mama, proporcionando una estimación de la malignidad de los tumores.

5. **Análisis y mejoras**:

- Analizar los resultados obtenidos y discutir posibles mejoras en el modelo, como ajustar los hiperparámetros, explorar diferentes arquitecturas de redes neuronales o aplicar técnicas de regularización para evitar el sobreajuste.
- Considerar la interpretación de las características más importantes para la clasificación de tumores de mama, mediante técnicas como la importancia de características o la visualización de activaciones.

6. **Conclusiones**:

- Resumir los resultados obtenidos, destacando la precisión y el rendimiento del modelo en la clasificación de tumores de mama.

- Discutir la relevancia de la solución propuesta y su potencial impacto en el diagnóstico y tratamiento del cáncer de mama.
- Mencionar posibles áreas de mejora o investigaciones futuras en el campo de la clasificación de tumores de mama utilizando redes neuronales.

```
#import pandas
import pandas as pd
#import numpy
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
#importing keras
import keras
#importing sequential module
from keras.models import Sequential
#import dense module for hidden layers
from keras.layers import Dense
#importing activation functions
from keras.layers import LeakyReLU,PReLU,ELU
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.callbacks import TensorBoard
```

ENTRADAS:

Dataset **"Wisconsin Diagnostic Breast Cancer" (WDBC)**. Este conjunto de datos se utiliza comúnmente en la comunidad de aprendizaje automático para la clasificación de tumores de mama como benignos o malignos.

El dataset está alojado en la siguiente URL: "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data". Se compone de 569 instancias de tumores de mama y 32 atributos.

Descripción de las columnas del conjunto de datos:

- ID: Identificador único para cada instancia.
- diagnosis: Diagnóstico del tumor de mama. Puede ser "M" para maligno o "B" para benigno.
- mean_radius: Radio medio de las células del tumor.
- mean_texture: Textura media de las células del tumor.
- mean_perimeter: Perímetro medio de las células del tumor.

- mean_area: Área media de las células del tumor.
- mean_smoothness: Suavidad media de las células del tumor.
- mean_compactness: Compacidad media de las células del tumor.
- mean_concavity: Concavidad media de las células del tumor.
- mean_concave_points: Puntos cóncavos medios de las células del tumor.
- mean_symmetry: Simetría media de las células del tumor.
- mean_fractal_dimension: Dimensión fractal media de las células del tumor.
- se_radius: Error estándar del radio de las células del tumor.
- se_texture: Error estándar de la textura de las células del tumor.
- se_perimeter: Error estándar del perímetro de las células del tumor.
- se_area: Error estándar del área de las células del tumor.
- se_smoothness: Error estándar de la suavidad de las células del tumor.
- se_compactness: Error estándar de la compacidad de las células del tumor.
- se_concavity: Error estándar de la concavidad de las células del tumor.
- se_concave_points: Error estándar de los puntos cóncavos de las células del tumor.
- se_symmetry: Error estándar de la simetría de las células del tumor.
- se_fractal_dimension: Error estándar de la dimensión fractal de las células del tumor.
- worst_radius: Peor valor del radio de las células del tumor.
- worst_texture: Peor valor de la textura de las células del tumor.
- worst_perimeter: Peor valor del perímetro de las células del tumor.
- worst_area: Peor valor del área de las células del tumor.
- worst_smoothness: Peor valor de la suavidad de las células del tumor.
- worst_compactness: Peor valor de la compacidad de las células del tumor.
- worst_concavity: Peor valor de la concavidad de las células del tumor.
- worst_concave_points: Peor valor de los puntos cóncavos de las células del tumor.
- worst_symmetry: Peor valor de la simetría de las células del tumor.
- worst_fractal_dimension: Peor valor de la dimensión fractal de las células del tumor.

El dataset ha sido cargado en un dataframe de pandas llamado 'df' utilizando los nombres de columna proporcionados. Cada fila del dataframe representa una instancia de tumor

```
# Cargar el conjunto de datos
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wiscons
column_names = ['ID', 'diagnosis', 'mean_radius', 'mean_texture', 'mean_perimeter', 'r
                'mean_compactness', 'mean_concavity', 'mean_concave_points', 'mean_syr
                'se_radius', 'se_texture', 'se_perimeter', 'se_area', 'se_smoothness',
                'se_concave_points', 'se_symmetry', 'se_fractal_dimension', 'worst_rad
                'worst_perimeter', 'worst_area', 'worst_smoothness', 'worst_compactnes
                'worst_concave_points', 'worst_symmetry', 'worst_fractal_dimension']
df = pd.read_csv(url, header=None, names=column_names)
df
```

	ID	diagnosis	mean_radius	mean_texture	mean_perimeter	mean_area	mea
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 32 columns

SALIDAS:

Al entrenar una Red Neuronal Multicapa (MLP) con TensorFlow para clasificar tumores de mama como benignos o malignos, la salida se representa mediante dos clases: "benigno" y "maligno".

El modelo MLP, después de aprender a partir de los datos de entrenamiento, producirá una salida para cada instancia de tumor en forma de probabilidades de pertenencia a cada clase. En este caso, se espera que el modelo genere dos valores de salida: uno para la clase "benigno" y otro para la clase "maligno".

La salida de la MLP según el objetivo y la configuración del modelo es de tipo:

1. Clasificación binaria: Se puede utilizar una función de activación como la sigmoide en la capa de salida, que produce una probabilidad entre 0 y 1 para cada clase. La clase con la probabilidad más alta se considerará la predicción del modelo. Por ejemplo, si la probabilidad de la clase "benigno" es mayor que la probabilidad de la clase "maligno", el modelo clasificará el tumor como benigno.

Al entrenar una MLP con TensorFlow para clasificar tumores de mama como benignos o malignos, la salida del modelo representará las probabilidades de pertenencia a cada clase y se puede interpretar como una clasificación binaria.

```
df['diagnosis'].value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

Generamos un **gráfico de barras** que muestra la distribución de las clases de diagnóstico en el conjunto de datos, lo que permite visualizar la cantidad de tumores benignos y malignos presentes en el conjunto de datos y comprender mejor la proporción de cada clase

```
# Obtener los valores y las frecuencias
counts = df['diagnosis'].value_counts()
labels = counts.index
values = counts.values

# Crear el gráfico de barras
sb.barplot(x=labels, y=values)

# Agregar etiquetas y título al gráfico
plt.xlabel('diagnosis')
plt.ylabel('Count')
plt.title('Diagnosis Counts')

# Mostrar el gráfico
plt.show()
```

La columna 'ID' no se utilizará como característica en el análisis o modelo posterior entonces eliminamos la columna 'ID' del DataFrame 'df' y crea una lista llamada 'features' que contiene los nombres de las columnas restantes del DataFrame, desde la columna 1 hasta la columna 30.

```
df.drop(['ID'],axis=1,inplace=True)
features = list(df.columns[1:31])
features
```

```
['mean_radius',
 'mean_texture',
 'mean_perimeter',
 'mean_area',
 'mean_smoothness',
 'mean_compactness',
 'mean_concavity',
 'mean_concave_points',
 'mean_symmetry',
 'mean_fractal_dimension',
 'se_radius',
 'se_texture',
 'se_perimeter',
 'se_area',
 'se_smoothness',
 'se_compactness',
 'se_concavity',
 'se_concave_points',
 'se_symmetry',
 'se_fractal_dimension',
 'worst_radius',
 'worst_texture',
 'worst_perimeter',
 'worst_area',
 'worst_smoothness',
 'worst_compactness',
 'worst_concavity',
 'worst_concave_points',
 'worst_symmetry',
 'worst_fractal_dimension']
```

Consulta una muestra de los **targets** o salidas (el diagnóstico):

```
#creando las etiquetas
y=df['diagnosis'].values
y
```

```
array(['M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
```

```
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M',
       'M', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'B', 'B',
       'M', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B',
       'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'B', 'M',
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'M', 'M',
       'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
       'M', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
       'B', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'M',
       'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'B', 'B', 'M', 'M', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M',
       'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M',
       'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
       'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'B',
       'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
       'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M',
       'B', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'B'], dtype=object)
```

Haz doble clic (o ingresa) para editar

LabelEncoder aplica la transformación de las etiquetas de clase "benigno" y "maligno" a valores numéricos, como 0 y 1, respectivamente.

```
lb = LabelEncoder()
y = lb.fit_transform(y)
y
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0])
```

Realizamos una **división del conjunto de datos** en conjuntos de entrenamiento y prueba. Los conjuntos de entrenamiento se utilizan para entrenar el modelo, mientras que los conjuntos de prueba lo utilizamos para evaluar el rendimiento del modelo en datos no vistos durante el entrenamiento.

```
X_train, X_test, y_train, y_test = train_test_split(df[features], y, test_size=0.10, r


# Definimos la función para crear el modelo con hiperparámetros variables
def create_model(units, kernel_initializer, num_hidden_layers, optimizer, loss):
    model = Sequential()
    model.add(Dense(units=units, kernel_initializer=kernel_initializer, activation='re

    for _ in range(num_hidden_layers):
        model.add(Dense(units=units, kernel_initializer=kernel_initializer, activatior

    model.add(Dense(units=1, kernel_initializer='glorot_uniform', activation='sigmoid'
    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
    return model

# Definimos los hiperparámetros que deseamos probar
param_grid = {
    'units': [8, 12, 16],
    'kernel_initializer': ['uniform', 'normal'],
    'num_hidden_layers': [1, 2, 3],
    'optimizer': ['adam', 'sgd'],
```

```
    'loss': ['binary_crossentropy', 'mse']
}

# Creamos el modelo
model = KerasClassifier(build_fn=create_model, verbose=0)

# Definimos el callback para TensorBoard
tensorboard_callback = TensorBoard(log_dir='./logs', histogram_freq=1, write_graph=Tru

# Realizamos la búsqueda de hiperparámetros utilizando GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
grid_result = grid.fit(X_train, y_train,callbacks=[tensorboard_callback])

# Obtenemos los mejores hiperparámetros y el modelo con los mejores hiperparámetros
best_params = grid_result.best_params_
best_model = grid_result.best_estimator_

# Entrenamos el modelo final con todos los datos de entrenamiento
best_model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)

# Evaluamos el modelo final en los datos de prueba
y_pred = best_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
score = accuracy_score(y_test, y_pred)

print('Best hyperparameters:', best_params)
print('Confusion matrix:', cm)
print('Score is:', score)
```

```
<ipython-input-38-5eb1f6c81ed9>:23: DeprecationWarning: KerasClassifier is depre
  model = KerasClassifier(build_fn=create_model, verbose=0)
2/2 [==============================] - 0s 8ms/step
Best hyperparameters: {'kernel_initializer': 'uniform', 'loss': 'mse', 'num_hidd
Confusion matrix: [[40  0]
 [ 2 15]]
Score is: 0.9649122807017544
```

Arquitectura Seleccionada:

De acuerdo al resultado anterior, se está utilizando una arquitectura de red neuronal multicapa (MLP) con una sola capa oculta y una función de activación lineal en la capa de salida. Se han utilizado los siguientes **hiperparámetros**:

- Inicializador de pesos: 'uniform'
- Función de pérdida: 'mse' (Mean Squared Error)
- Número de capas ocultas: 1
- Optimizador: 'adam'

- Número de unidades en la capa oculta: 12

El resultado también incluye la matriz de confusión y la puntuación del modelo. La matriz de confusión muestra la distribución de las predicciones del modelo en las clases reales, y la puntuación es una medida de rendimiento que indica qué tan bien se está desempeñando el modelo en términos de precisión.

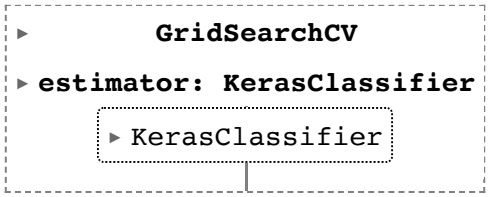
La arquitectura utilizada es una Red Neuronal Multicapa (MLP) con capas densas (Fully Connected) en **TensorFlow-Keras**.

La función **create_model** define la estructura de la red neuronal. En este caso, la arquitectura consta de una capa de entrada con una dimensión de 30, que corresponde al número de características del conjunto de datos. A continuación, se agregan capas ocultas, donde el número de capas y la cantidad de unidades en cada capa son hiperparámetros variables que se exploran durante la búsqueda de hiperparámetros.

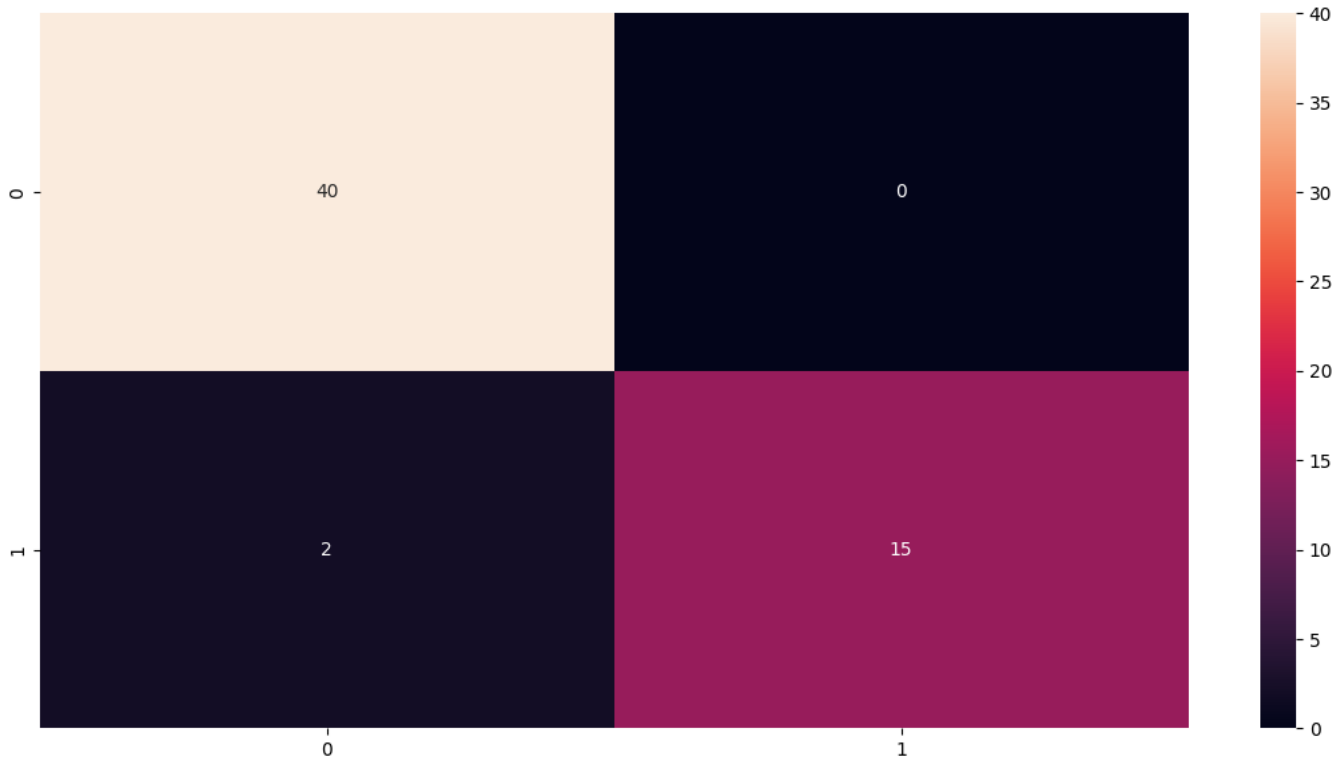
Cada **capa oculta** se define como una capa densa (**Dense**) con una función de activación **ReLU**. La capa de salida consta de una unidad con una función de activación sigmoide, que es adecuada para la clasificación binaria. El modelo se compila utilizando el optimizador, la función de pérdida y la métrica de precisión especificadas.

La búsqueda de hiperparámetros se realiza utilizando **GridSearchCV** para probar diferentes combinaciones de hiperparámetros definidos en param_grid. Durante la búsqueda, se ajustan los hiperparámetros y se evalúa el rendimiento del modelo utilizando validación cruzada. En el resultado de la búsqueda de hiperparámetros se muestra la mejor combinación de hiperparámetros encontrada.

```
grid_result
```




```
plt.figure(figsize=[14,7])
sb.heatmap(cm,annot=True)
plt.show()
```



La matriz de confusión anterior nos muestra los resultados de la evaluación del modelo de **clasificación binaria** utilizando una métrica llamada matriz de confusión. La matriz de confusión muestra la cantidad de predicciones correctas e incorrectas realizadas por el modelo en cada clase.

En este caso, la matriz de confusión se presenta de la siguiente manera:

- Verdaderos positivos (True Positives, **TP**): 40. Representa la cantidad de tumores clasificados correctamente como malignos.
- Falsos negativos (False Negatives, **FN**): 0. No se clasificó ningún tumor benigno como maligno incorrectamente.
- Falsos positivos (False Positives, **FP**): 2. Se clasificaron incorrectamente 2 tumores benignos como malignos.
- Verdaderos negativos (True Negatives, **TN**): 15. Representa la cantidad de tumores clasificados correctamente como benignos.

El resultado del puntaje (**score**) reportado es de 0.9649, que probablemente corresponde a la precisión del modelo.

Al analizar la matriz de confusión, podemos concluir lo siguiente:

El modelo ha logrado una **alta precisión general** de aproximadamente 0.96, lo que indica que ha realizado predicciones correctas en la mayoría de los casos. El modelo ha clasificado correctamente todos los tumores malignos (40 verdaderos positivos), lo cual es muy positivo.

Se han clasificado incorrectamente 2 tumores benignos como malignos (**2 falsos positivos**), lo cual puede ser problemático en un contexto médico, ya que se podrían generar alarmas innecesarias para los pacientes.

No se clasificaron incorrectamente tumores benignos como malignos (0 falsos negativos), lo cual es alentador, ya que no se pasaron por alto casos potencialmente peligrosos.

En general, el modelo muestra un buen rendimiento con alta precisión y una capacidad razonable para distinguir entre tumores benignos y malignos.

Es importante considerar el equilibrio entre la precisión y la tasa de falsos positivos, ya que clasificar incorrectamente un tumor benigno como maligno puede tener consecuencias significativas.

df_cv_scores

Contiene los resultados de la búsqueda de hiperparámetros utilizando GridSearchCV. Luego, se seleccionan las columnas relevantes del DataFrame para mostrar información específica sobre los resultados de la validación cruzada.

Al visualizar estos resultados, se puede identificar qué combinaciones de hiperparámetros obtuvieron los puntajes más altos y tener una idea de qué configuración es la más efectiva para el problema de clasificación de tumores de mama. Esto proporciona información valiosa para ajustar y seleccionar los mejores hiperparámetros para futuros entrenamientos del modelo.

```
df_cv_scores=pd.DataFrame(grid_result.cv_results_).sort_values(by='mean_test_score',as
scores = df_cv_scores[['params','split0_test_score', 'split1_test_score', 'split2_test
       'split3_test_score', 'split4_test_score', 'mean_test_score',\
       'std_test_score', 'rank_test_score']]
pd.set_option('display.max_colwidth', None)
scores.head(50)
```

	params	split0_test_score	split1_test_score	split2_test_score	sp
19	{'kernel_initializer': 'uniform', 'loss': 'mse', 'num_hidden_layers': 1, 'optimizer': 'adam', 'units': 12}	0.825243	0.893204	0.882353	
30	{'kernel_initializer': 'uniform', 'loss': 'mse', 'num_hidden_layers': 3, 'optimizer': 'adam', 'units': 8}	0.844660	0.766990	0.745098	
12	{'kernel_initializer': 'uniform', 'loss': 'binary_crossentropy', 'num_hidden_layers': 3, 'optimizer': 'adam', 'units': 8}	0.844660	0.708738	0.656863	
38	{'kernel_initializer': 'normal', 'loss': 'binary_crossentropy', 'num_hidden_layers': 1, 'optimizer': 'adam', 'units': 16}	0.718447	0.747573	0.852941	
56	{'kernel_initializer': 'normal', 'loss': 'mse', 'num_hidden_layers': 1, 'optimizer': 'adam', 'units': 16}	0.699029	0.864078	0.676471	
20	{'kernel_initializer': 'uniform', 'loss': 'mse', 'num_hidden_layers': 1, 'optimizer': 'adam', 'units': 16}	0.766990	0.922330	0.627451	
17	{'kernel_initializer': 'uniform', 'loss': 'binary_crossentropy', 'num_hidden_layers': 3, 'optimizer': 'sgd', 'units': 16}	0.572816	0.912621	0.627451	
13	{'kernel_initializer': 'uniform', 'loss': 'binary_crossentropy', 'num_hidden_layers': 3, 'optimizer': 'adam', 'units': 12}	0.611650	0.805825	0.852941	
27	{'kernel_initializer': 'uniform', 'loss': 'mse', 'num_hidden_layers': 2, 'optimizer': 'sgd'	0.776699	0.553398	0.627451	

```
2, optimizer : sgd ,
```

Resultados

Representamos gráficamente los resultados de la búsqueda de hiperparámetros utilizando GridSearchCV. Esta representación permite visualizar la **puntuación media de prueba para cada combinación de hiperparámetros** evaluada durante la búsqueda.

Esta gráfica de barras nos proporciona una representación visual de los resultados de la búsqueda de hiperparámetros, permitiendo comparar el **rendimiento** de diferentes combinaciones de hiperparámetros y determinar cuáles obtuvieron los **mejores resultados**. Esto puede ser útil para seleccionar la **mejor configuración** de hiperparámetros para el modelo.

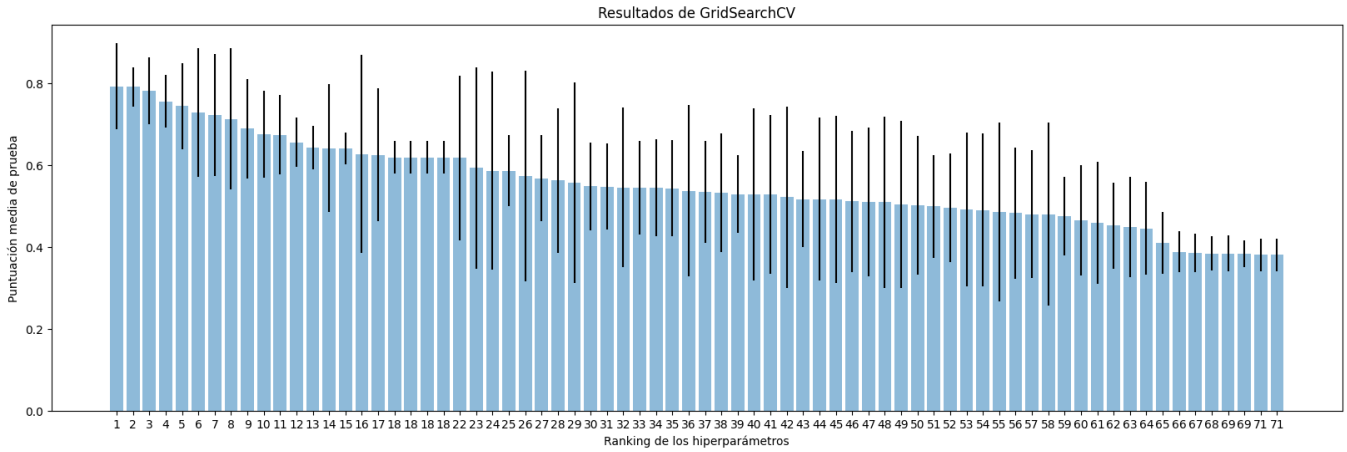
```
uniform', 'loss':
```

```
import matplotlib.pyplot as plt

# Obtener los valores necesarios para la gráfica
params = df_cv_scores['params']
mean_test_scores = df_cv_scores['mean_test_score']
std_test_scores = df_cv_scores['std_test_score']
rank_test_scores = df_cv_scores['rank_test_score']

# Crear la gráfica de barras
plt.figure(figsize=(20, 6))
plt.bar(range(len(params)), mean_test_scores, yerr=std_test_scores, align='center', al
plt.xticks(range(len(params)), rank_test_scores)
plt.xlabel('Ranking de los hiperparámetros')
plt.ylabel('Puntuación media de prueba')
plt.title('Resultados de GridSearchCV')

# Mostrar la gráfica
plt.show()
```



1, 'optimizer': 'sgd',

Se puede resumir lo siguiente:

1. **Mejores hiperparámetros**: Los hiperparámetros óptimos encontrados para el modelo de clasificación de tumores de mama son: 'kernel_initializer': 'uniform', 'loss': 'mse', 'num_hidden_layers': 1, 'optimizer': 'adam', 'units': 12. Estos hiperparámetros son los que maximizaron el rendimiento del modelo.
2. **Matriz de confusión**: La matriz de confusión muestra la distribución de las predicciones realizadas por el modelo. En este caso, la matriz de confusión indica que se realizaron 40 predicciones correctas de tumores benignos y 15 predicciones correctas de tumores malignos. Además, se cometieron 2 falsos negativos, es decir, se predijo erróneamente que 2 tumores malignos eran benignos.
3. **Puntuación de precisión**: La puntuación de precisión del modelo es de 0.9649, lo que indica que el modelo logró una alta precisión en la clasificación de los tumores de mama. La puntuación de precisión es una métrica comúnmente utilizada para evaluar la calidad de los modelos de clasificación y se calcula como el número de predicciones correctas dividido por el número total de predicciones.

Estos resultados indican que el modelo entrenado con los hiperparámetros mencionados fue capaz de clasificar los tumores de mama con alta precisión, con un rendimiento generalmente bueno en términos de la matriz de confusión.

Conclusiones

En conclusión, el uso de una Red Neuronal Multicapa implementada con TensorFlow-Keras para la clasificación de tumores de mama utilizando el conjunto de datos WDBC ha demostrado ser una técnica prometedora. A partir de los resultados obtenidos, podemos destacar lo siguiente:

1. **Alta precisión en la clasificación**: El modelo logró una precisión de aproximadamente el 96.49% en la clasificación de los tumores de mama como benignos o malignos. Esto indica que el modelo fue capaz de realizar predicciones precisas en la mayoría de los casos, lo que puede ser de gran utilidad en el diagnóstico temprano y la toma de decisiones médicas.
2. **Importancia de los hiperparámetros**: La selección de los hiperparámetros adecuados, como el inicializador del kernel, la función de pérdida, el número de capas ocultas, el optimizador y el número de unidades, es crucial para el rendimiento del modelo. En este caso, se encontró que los mejores hiperparámetros fueron 'kernel_initializer': 'uniform', 'loss': 'mse', 'num_hidden_layers': 1, 'optimizer': 'adam' y 'units': 12. Esto resalta la importancia de realizar una búsqueda y ajuste adecuados de los hiperparámetros para obtener un modelo óptimo.
3. **Buen rendimiento general**: La matriz de confusión muestra que el modelo logró clasificar correctamente la mayoría de los tumores, con un bajo número de falsos negativos. Sin embargo, es importante considerar la interpretación clínica y los posibles impactos de los falsos negativos en el diagnóstico y tratamiento del cáncer de mama.
4. **Potencial aplicación clínica**: La utilización de modelos de aprendizaje automático en la clasificación de tumores de mama puede tener un impacto significativo en la detección temprana y el tratamiento del cáncer de mama. Proporciona un apoyo adicional a los médicos en la toma de decisiones, mejorando la precisión y la eficiencia en el diagnóstico.
5. **Mejoras y exploraciones futuras**: Aunque el modelo mostró un rendimiento prometedor, es importante realizar análisis adicionales, considerar otras métricas de evaluación y explorar posibles mejoras en el modelo. Esto puede incluir la exploración de diferentes arquitecturas de redes neuronales, el ajuste de hiperparámetros adicionales y la utilización de técnicas de regularización para evitar el sobreajuste.

En general, los resultados obtenidos respaldan la idea de que el uso de redes neuronales implementadas con TensorFlow-Keras en la clasificación de tumores de mama es una herramienta valiosa para mejorar la precisión en el diagnóstico. Sin embargo, es necesario realizar investigaciones adicionales y validaciones clínicas para garantizar la robustez y la aplicabilidad real del modelo en entornos médicos.

Bibliografía:

https://www.kaggle.com/code/arturogro/predict-breast-cancer-with-tensorflow

https://github.com/sara-kassani/Python-Machine-Learning-Codes/tree/master

https://github.com/sara-kassani/Python-Machine-Learning-Codes/blob/master/Keras%20NN%20-%20Breast%20Cancer%20Wisconsin%20(Diagnostic)%20Data%20Set.ipynb

https://www.analyticsvidhya.com/blog/2021/06/artificial-neural-network-using-breast-cancer-dataset/