Segunda parte: validación

> ⚠️ Nota ⚠️
>
> Las conclusiones que sacamos en este archivo se encuentran comentadas en el notebook principal, de modo que no afecta al entrenamiento y/o comparación de los modelos futuros.

Revisando la celda que entrena el modelo, vemos que hay un error:

```
# Entrenamos el modelo
history = network.fit(
    X_train, train_labels,
    epochs=100,
    batch_size=5,
    validation_data=(X_test, test_labels),
    shuffle=True
)
```

En *validation_data* deberían estar el segmento de datos de validación, no los de test. Esto se debe hacer así porque sino el modelo conoce los datos de test antes de hacer el propio test (data leakage) y puede afectar a los resultados del modelo.

Ahora, podemos aplicar, principalmente, dos tipos de validación:

- Validación simple.
- Validación cruzada (Cross-validation).

Validación simple

Podemos utilizar varios tipos de validación simple. En esta práctica he optado por ver estos dos "tipos":

- El método *train_valid_test_split.*
- Segmentar los datos a mano.

Método train_valid_test_split

Es una librería de *fast_ml* que realiza la división de forma directa, solo le tenemos que indicar:

- X
- target

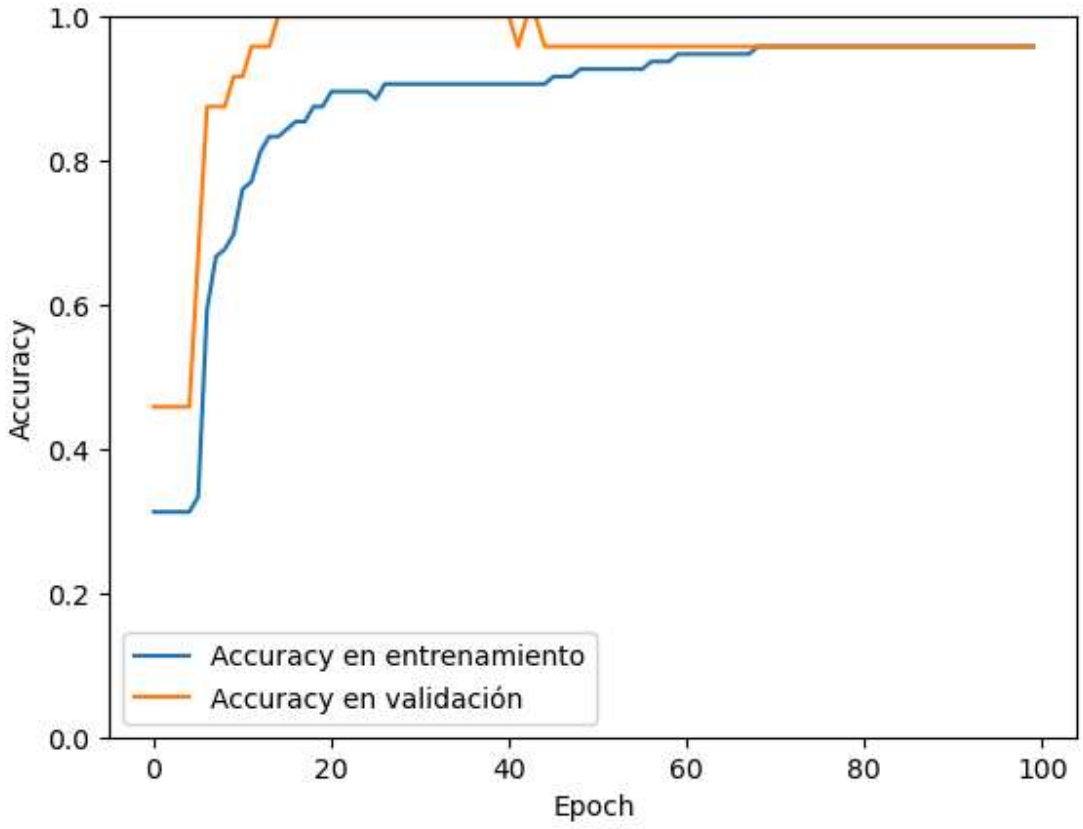
- Tamaño de train
- Tamaño de validation
- Tamaño de test

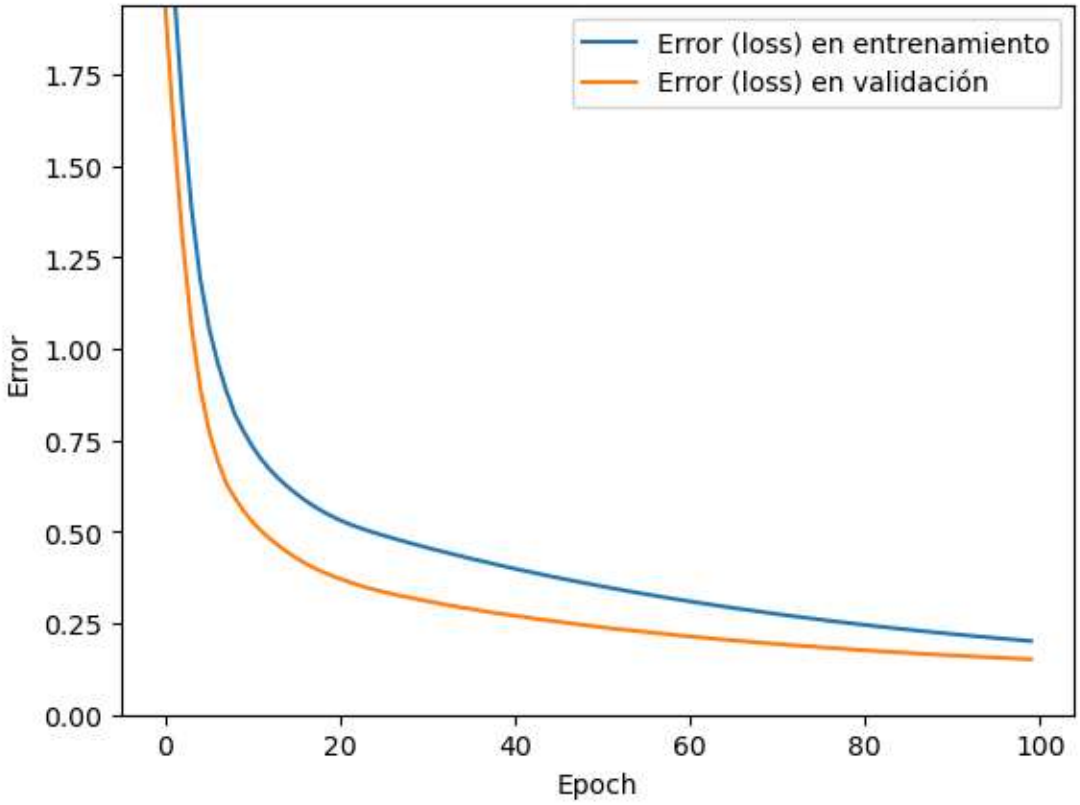
También podemos añadir "method" con el que podemos aleatorizar los datos o no, e indicar una "seed" para poder replicar los resultados.

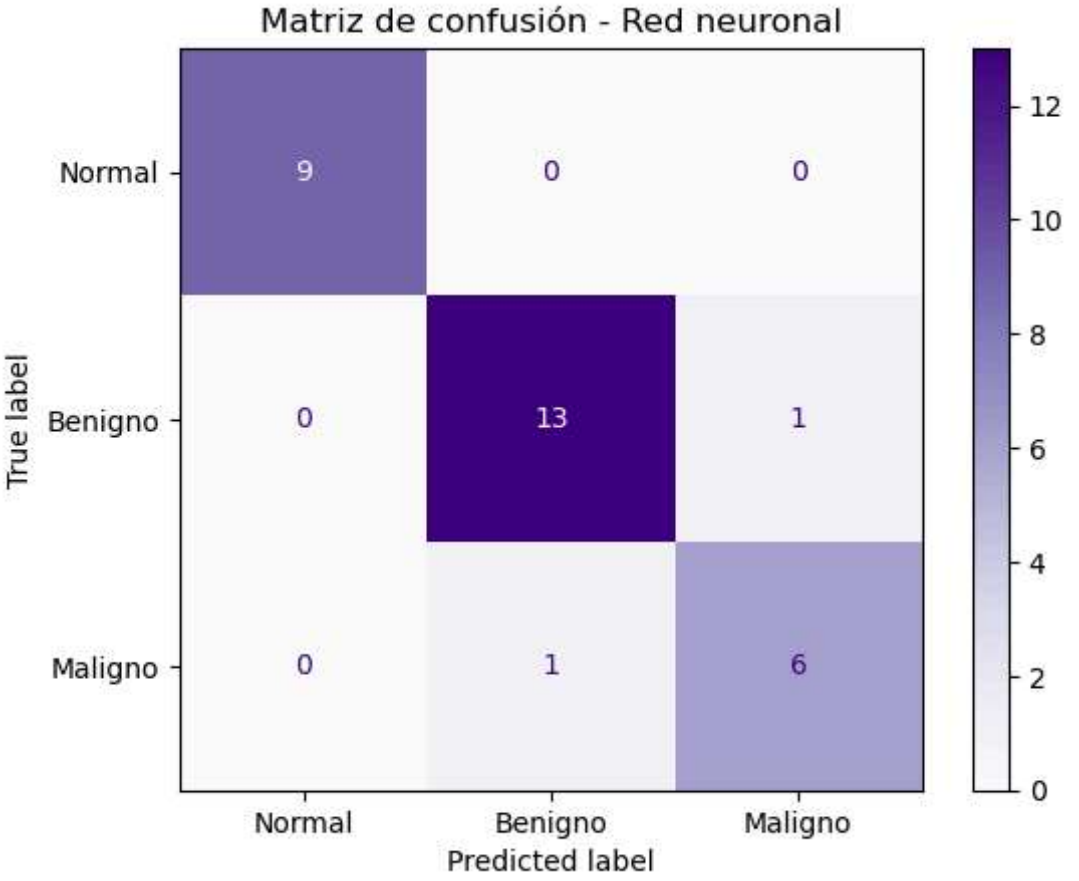
Debemos usar el df que genera `pandas` y no los array de `numpy` que se usan en el programa original.

> Lo malo de usar *train_valid_test_split* es que no existe el parámetro *stratify*, así existe la posibilidad de generar un bloque que solo tenga un tipo de tumor

Usando esta librería hemos obtenido los siguientes resultados con el modelo base:







	precision	recall	f1-score	support
Normal	1.00	1.00	1.00	9
Benigno	0.93	0.93	0.93	14
Maligno	0.86	0.86	0.86	7

```
    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

Segmentar los datos a mano

Esta opción es más recomendable en mi opinión porque no tienes que importar una librería para una sola función.

Para ello realizamos un primer *train_test_split* sobre **X** e **y** (suponiendo que ya están definidas las variables de objetivo) de modo que obtenemos:

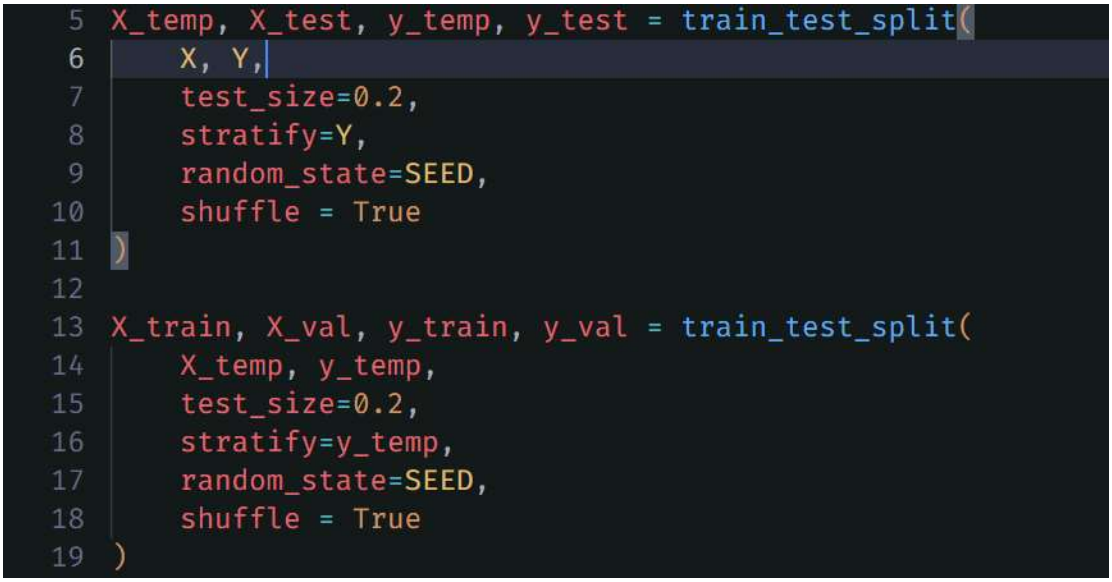
- X_train
- y_train
- X_test
- y_test

Ahora realizamos otro *train_test_split* sobre **X_train** e **y_train** (preferiblemente con las mismas proporciones que hemos hecho la primera división), de modo que obtenemos:

- X_train
- y_train
- X_val
- y_val

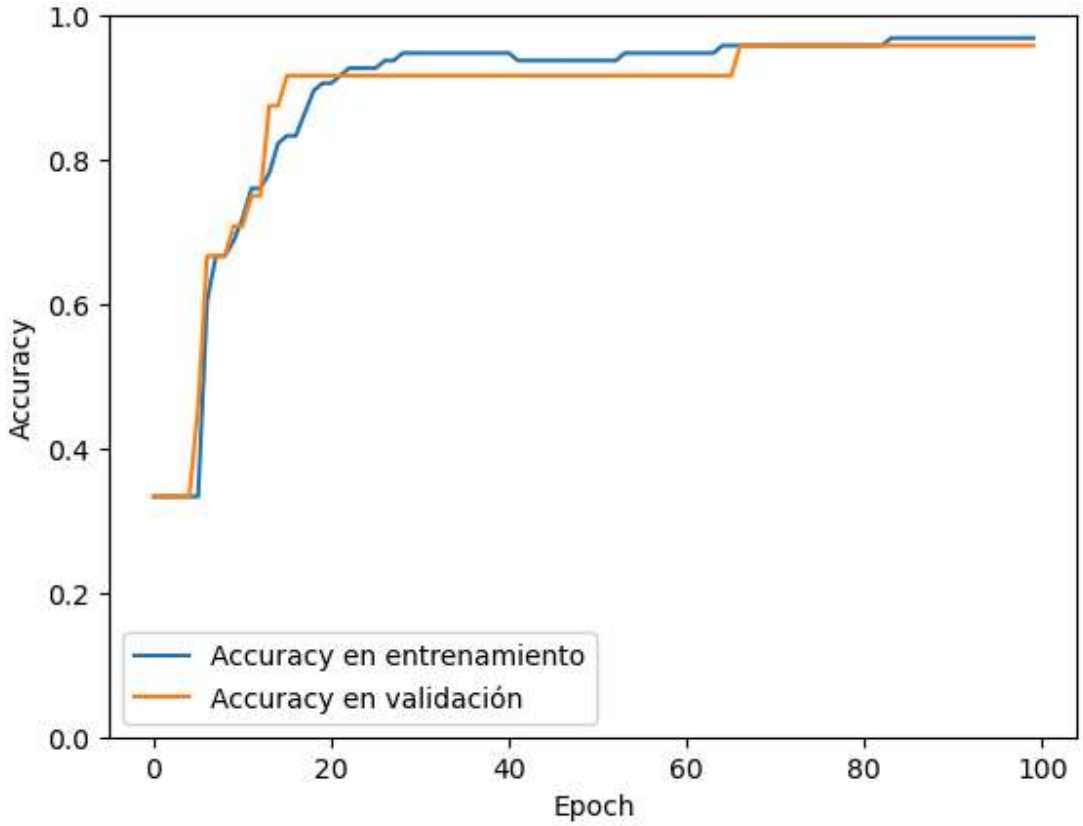
Y una vez hechas estas dos divisiones nos quedamos con las siguientes variables:

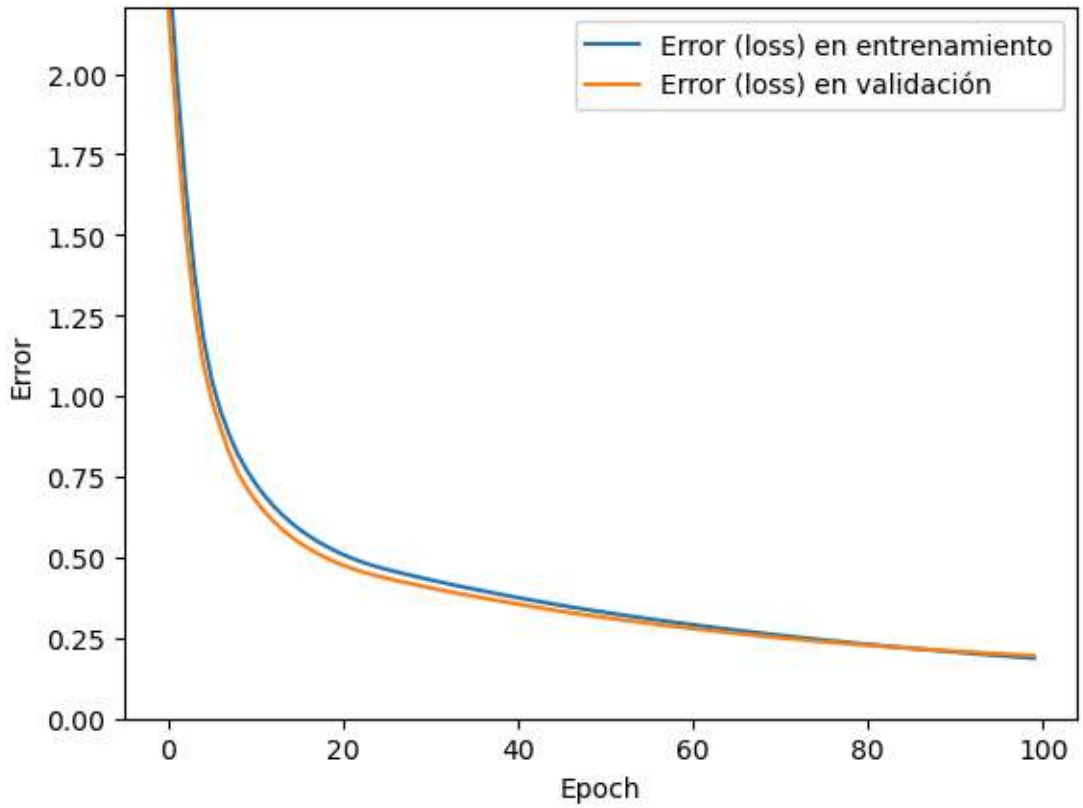
- X_train
- y_train
- X_val
- y_val
- X_test
- y_test



```python
X_temp, X_test, y_temp, y_test = train_test_split(
    X, Y,
    test_size=0.2,
    stratify=Y,
    random_state=SEED,
    shuffle = True
)

X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp,
    test_size=0.2,
    stratify=y_temp,
    random_state=SEED,
    shuffle = True
)
```

Con la segmentación hecha a mano conseguimos los siguientes resutaldos:





	precision	recall	f1-score	support
Normal	1.00	1.00	1.00	10
Benigno	1.00	0.80	0.89	10

Maligno	0.83	1.00	0.91	10
accuracy			0.93	30
macro avg	0.94	0.93	0.93	30
weighted avg	0.94	0.93	0.93	30

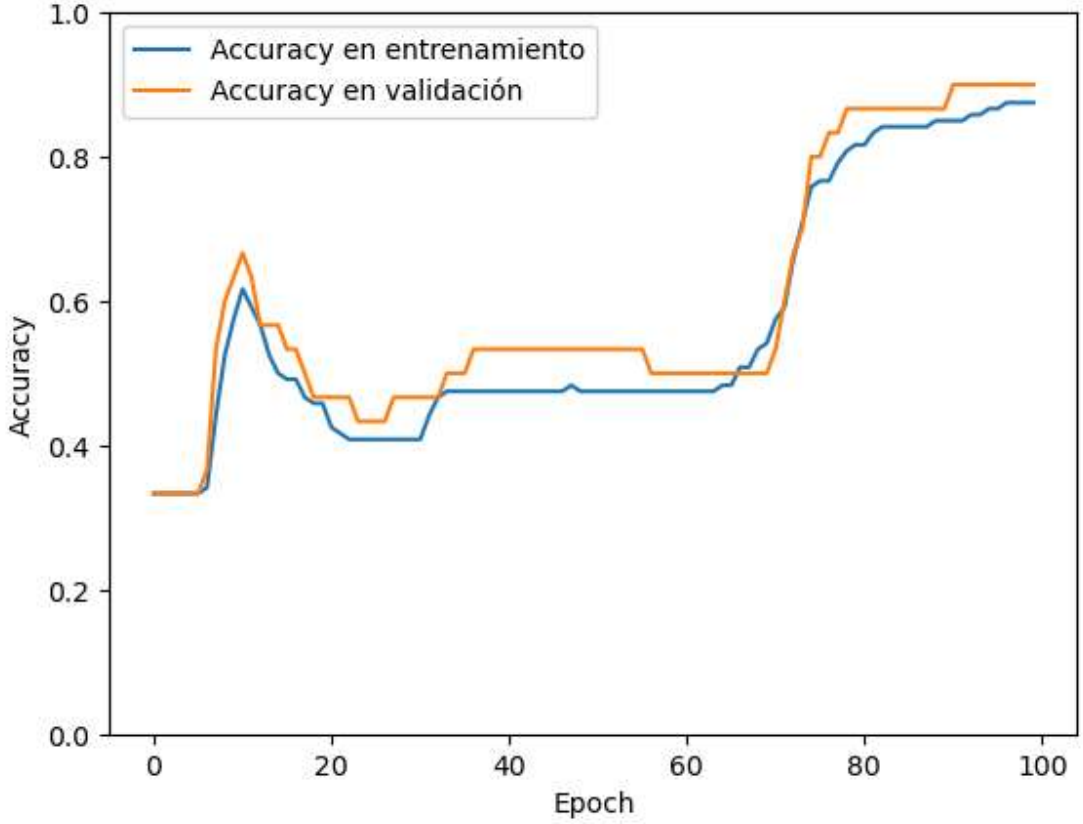
Validación cruzada (Cross-validation)

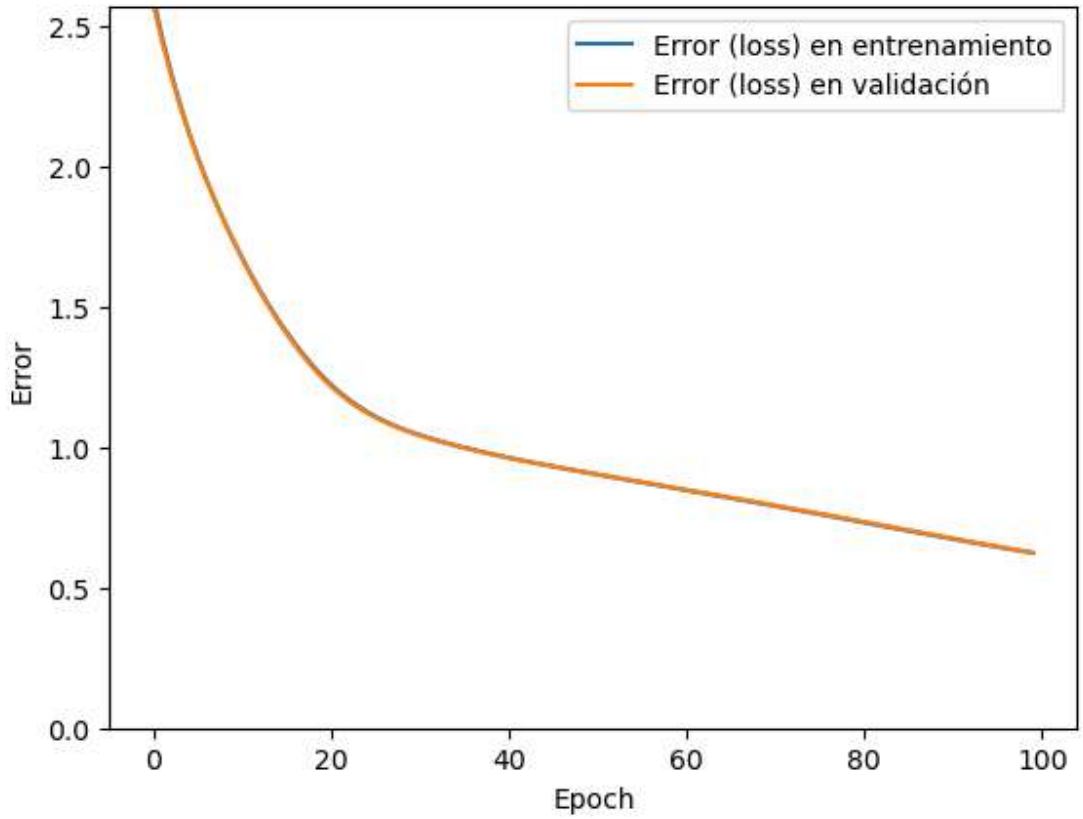
Haremos validación cruzada utilizando StratifiedKFold.

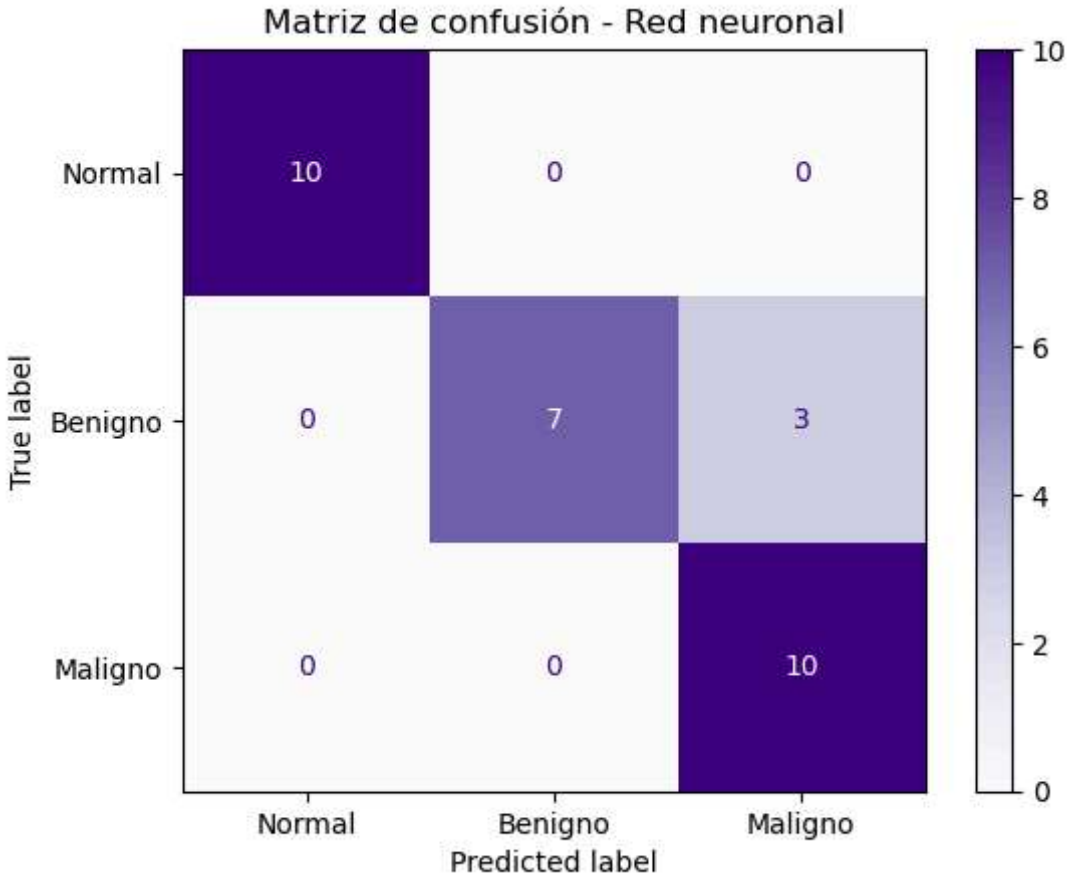
```python
# Validación cruzada (StratifiedKFold)
skf = StratifiedKFold(
    n_splits=5,
    shuffle=True,
    random_state=SEED
)

# entrenamos el modelo
for i, (train_index, test_index) in enumerate(skf.split(X, Y)):
    # dividimos los datos
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = Y[train_index], Y[test_index]

    # conversion de tipos de dato
    y_train = [classes[elem] for elem in y_train]
    y_test = [classes[elem] for elem in y_test]
    train_labels = to_categorical(y_train)
    test_labels = to_categorical(y_test)

    # creamos y entrenamos el modelo
    model = baseline_model()
    history = model.fit(
        X_train, train_labels,
        validation_data=(X_test, test_labels),
        epochs=100,
        batch_size=16,
        verbose=0
    )
```

Utilizamos 5 splits para imitar la proporción 80/20.

Con la validación cruzada hemos obtenido los siguientes resultados:







	precision	recall	f1-score	support
Normal	1.00	1.00	1.00	10
Benigno	1.00	0.70	0.82	10
Maligno	0.77	1.00	0.87	10
accuracy			0.90	30
macro avg	0.92	0.90	0.90	30
weighted avg	0.92	0.90	0.90	30

Vemos que los valores no son tan prometedores.

> Sin embargo, simplemente aumentando el número de neuronas de la capa oculta del modelo y cambiando la función de activación de la capa de salida obtenemos un modelo casi perfecto.