



# UNIVERSIDAD NACIONAL DE TRUJILLO

## FACULTAD DE INGENIERÍA PROGRAMA DE INGENIERÍA MECATRÓNICA

---

### SEMANA 9 - REDES NEURONALES

---

### PROCESAMIENTO DIGITAL DE SEÑALES E IMÁGENES

**ESTUDIANTE(S)** :  
CORTEZ GOMEZ, BRAJAN LEONEL  
ESPINOZA LEÓN KARL ALEJANDRO  
GUTIÉRREZ GUTIÉRREZ ITALO AARÓN

**DOCENTE** :  
MS. ING. EMERSON MÁXIMO ASTO RODRIGUEZ

**CICLO** :  
2023 - II

Trujillo, Perú  
2023

## **RESUMEN**

En el presente informe de laboratorio, se utilizó un programa, el cual permitía entrenar una red neuronal artificial completamente conectada para resolver “El problema de clasificación de iris”, para determinar principalmente la eficiencia de dicha red neuronal durante 10 entrenamientos, utilizando hiperparámetros que ajustaban la eficiencia. Se analizó la recolección de datos y realizó un análisis de estos mismos. Además, se revisó la posible presencia de Overfitting en esta experiencia. Por último, se obtuvieron las conclusiones generales de este entrenamiento.

## ÍNDICE

**Objetivos**

**Implementación del código**

**Prueba del código**

**Cuadro de hiperparámetros**

**Observaciones**

**Verificación de overfitting**

**Conclusiones**

**Bibliografía**

## **OBJETIVOS**

- Entrenar una red neuronal para que prediga la especie de flor iris y solucione el conjunto de datos iris.
- Completar el cuadro de hiperparámetros con los datos obtenidos de la red neuronal.
- Analizar los datos obtenidos.
- Detectar la existencia de overfitting en los modelos.

## UTILIZACIÓN DEL CÓDIGO

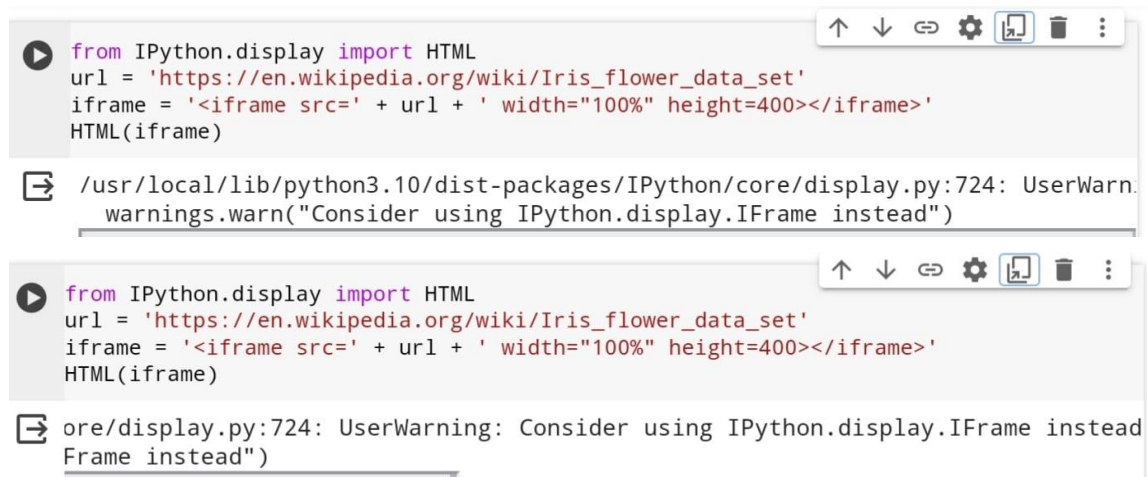
Para utilizar un código en lenguaje Python que permita entrenar una red neuronal artificial completamente conectada para resolver “El problema de clasificación de iris”, cuyo funcionamiento y cuya eficiencia dependan de parámetros como el número de épocas, se debe entender el flujo del mismo. Para ello, realizamos lo siguiente:

Importamos la biblioteca “TensorFlow” como ‘tf’ e imprimimos su versión a ejecutar.

```
[ ] import tensorflow as tf
    print(tf.__version__)

2.15.0
```

Reslizamos una visualización de una página web dentro de un entorno interactivo. Este sería la base de datos de las flores Iris.

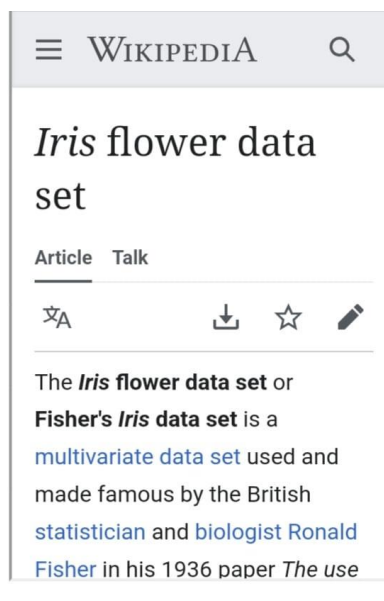


```
from IPython.display import HTML
url = 'https://en.wikipedia.org/wiki/Iris_flower_data_set'
iframe = '<iframe src=' + url + ' width="100%" height=400></iframe>'
HTML(iframe)

/usr/local/lib/python3.10/dist-packages/IPython/core/display.py:724: UserWarn.
warnings.warn("Consider using IPython.display.IFrame instead")

from IPython.display import HTML
url = 'https://en.wikipedia.org/wiki/Iris_flower_data_set'
iframe = '<iframe src=' + url + ' width="100%" height=400></iframe>'
HTML(iframe)

ore/display.py:724: UserWarning: Consider using IPython.display.IFrame instead
Frame instead")
```



Importamos las siguientes librerías y clases que vamos a utilizar. Además, establecemos las características de la data.

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
```

<pre>data = load_iris() features = data["data"] labels = data["target"]  data.keys()  dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])  data.target_names  array(['setosa', 'versicolor', 'virginica'], dtype='&lt;U10')  data.feature_names  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']</pre>	<pre>labels  array([[0, 1, 2, 2])</pre>
---	---

Definimos las variables de entrenamiento y prueba.

```
input_train, input_test, output_train, output_test = train_test_split(
    features, labels, test_size=0.2)
```

Preprocesamos la data.

```
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')

mean = input_train.mean(axis=0)
input_train -= mean

std = input_train.std(axis=0)
input_train /= std

input_test -= mean
input_test /= std
```

```

from tensorflow import keras

print("Antes de la codificación one-hot", output_train)

output_train = keras.utils.to_categorical(output_train, 3)
output_test = keras.utils.to_categorical(output_test, 3)

print("Despues de la codificación one-hot:", output_train[0])

```

Antes de la codificación one-hot [1 2 2 0 1 0 0 1 0 1 0 2 1 0 2 0 0 0 2 2 0 2  
1 2 1 0 1 2 0 0 0 2 1 2 2 2 2 2 2 2 1 1 1 2 1 2 0 1 1 0 2 0 2 1 0 1 1 0  
0 1 1 1 2 2 1 2 2 1 0 0 2 2 2 0 0 2 1 2 2 1 0 2 1 1 2 2 0 0 0 1 2 1 1 2 1  
0 2 0 1 0 1 0 2 2]

Despues de la codificación one-hot: [0. 1. 0.]

```

hot [1 2 2 0 1 0 0 1 0 1 0 2 1 0 2 0 0 0 2 2 0 2 0 1 2 0 0 1 1 0 0 1 0 2 0 0 0  

2 2 2 2 2 1 1 1 2 1 2 0 1 1 0 2 0 2 1 0 1 1 0  

2 0 0 2 1 2 2 1 0 2 1 1 2 2 0 0 0 1 2 1 1 2 1  

e-hot: [0. 1. 0.]

```

Realizamos el modelamiento de la red neuronal. Empezamos creando 1 capa oculta con 50 neuronas.

```

model = Sequential([
    Dense(1000, activation="relu", input_shape=(4,)),
    Dense(50, activation="relu"),
    Dense(3, activation="softmax")
])

model.summary()

```

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 1000)	5000
dense_7 (Dense)	(None, 50)	50050
dense_8 (Dense)	(None, 3)	153

=====  
dense\_8 (Dense) (None, 3) 153  
=====

Total params: 55203 (215.64 KB)  
Trainable params: 55203 (215.64 KB)  
Non-trainable params: 0 (0.00 Byte)

Cabe resaltar que, para el cuadro de hiperparámetros, se contabilizará únicamente las capas ocultas dentro de la columna de Capas. Las capas de entrada y de salida solo se asumirán que ya están dentro del código.

Capa de entrada:

Dense(1000, activation="relu", input\_shape=(4,)),

Capa de salida:

Dense(3, activation="softmax")

Decidimos usar la función de pérdida “Categorical Crossentropy” y el optimizador “Adam” para empezar. Además, la tasa de aprendizaje, en este caso, será de 0.001.

```
0 s ▶ model.compile(loss=keras.losses.categorical_crossentropy,
                    optimizer=keras.optimizers.Adam(learning_rate=0.001),
                    metrics=['accuracy'])
```

Entrenamos la red neuronal estableciendo que la cantidad de épocas sea 10 al inicio.

```
▶ history = model.fit(input_train, output_train,
                      epochs=10,
                      validation_split=0.1)
```

Epoch 1/10  
4/4 [=====] - 0s 27ms/step - loss: 0.1565 - accuracy: 0.9537 - val\_loss: 0.0183 - val\_accuracy: 1.0000  
Epoch 2/10  
4/4 [=====] - 0s 11ms/step - loss: 0.1434 - accuracy: 0.9630 - val\_loss: 0.0175 - val\_accuracy: 1.0000  
Epoch 3/10  
4/4 [=====] - 0s 18ms/step - loss: 0.1302 - accuracy: 0.9630 - val\_loss: 0.0122 - val\_accuracy: 1.0000  
Epoch 4/10  
4/4 [=====] - 0s 22ms/step - loss: 0.1208 - accuracy: 0.9444 - val\_loss: 0.0107 - val\_accuracy: 1.0000  
Epoch 5/10  
4/4 [=====] - 0s 23ms/step - loss: 0.1115 - accuracy: 0.9352 - val\_loss: 0.0067 - val\_accuracy: 1.0000  
Epoch 6/10  
4/4 [=====] - 0s 21ms/step - loss: 0.1040 - accuracy: 0.9537 - val\_loss: 0.0052 - val\_accuracy: 1.0000  
Epoch 7/10  
4/4 [=====] - 0s 22ms/step - loss: 0.1006 - accuracy: 0.9630 - val\_loss: 0.0057 - val\_accuracy: 1.0000  
Epoch 8/10  
4/4 [=====] - 0s 20ms/step - loss: 0.0909 - accuracy: 0.9722 - val\_loss: 0.0055 - val\_accuracy: 1.0000  
Epoch 9/10  
4/4 [=====] - 0s 21ms/step - loss: 0.0850 - accuracy: 0.9630 - val\_loss: 0.0054 - val\_accuracy: 1.0000  
Epoch 10/10  
4/4 [=====] - 0s 22ms/step - loss: 0.0812 - accuracy: 0.9722 - val\_loss: 0.0044 - val\_accuracy: 1.0000

Mostramos los resultados en base a gráficas de “Pérdida (variable ‘loss’) vs Épocas” y “Exactitud (variable ‘accuracy’) vs Épocas”.

```
2 s ▶ loss = history.history['loss']
acc = history.history['accuracy']
val_loss = history.history['val_loss']
val_acc = history.history['val_accuracy']
epochs = range(len(loss))

f, axarr = plt.subplots(1, 2, figsize=(12, 6))
p0 = axarr[0]
p1 = axarr[1]

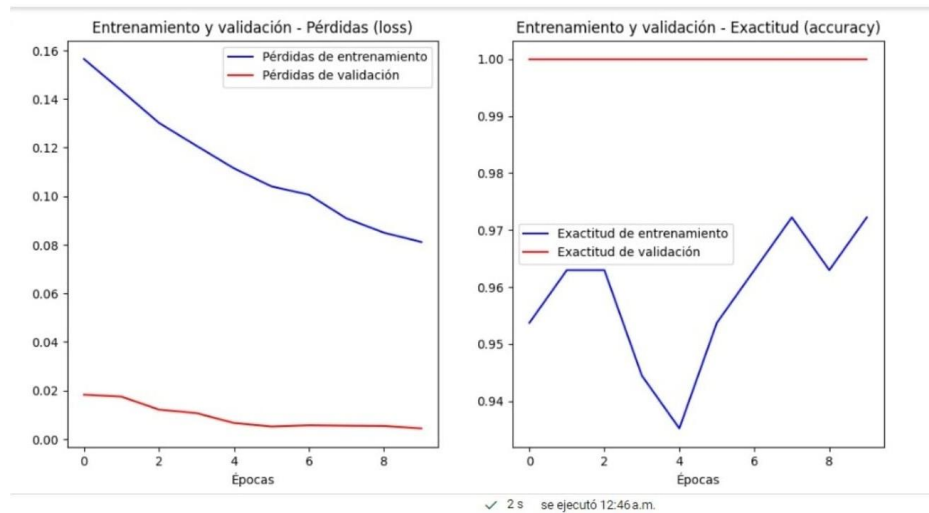
p0.set_title("Entrenamiento y validación - Pérdidas (loss)")
p0.set_xlabel('Épocas')
p1.set_title("Entrenamiento y validación - Exactitud (accuracy)")
p1.set_xlabel('Épocas')

p0l0 = p0.plot(epochs, loss, "-b", label="Pérdidas de entrenamiento")
p0l1 = p0.plot(epochs, val_loss, "-r", label="Pérdidas de validación")

p1l0 = p1.plot(epochs, acc, "-b", label="Exactitud de entrenamiento")
p1l1 = p1.plot(epochs, val_acc, "-r", label="Exactitud de validación")

legend0 = p0.legend()
legend1 = p1.legend()
```





Evaluamos el modelo de la siguiente manera:

```
✓ 0 s [17] score = model.evaluate(input_test, output_test)

print('Test Loss:', score[0])
print('Test Accuracy:', score[1])
```

1/1 [=====] - 0s 24ms/step - loss: 0.0259 - accuracy: 1.0000  
Test Loss: 0.02594398520886898  
Test Accuracy: 1.0

```
✓ 1 s ▶ predictions = model.predict(input_test[0:])

true_labels = output_test[0:]

for i in range(len(predictions)):
    print("Predicción:", data["target_names"][predictions[i].argmax()], ",",
          "Etiqueta real:", data["target_names"][true_labels[i].argmax()])
```

```
1/1 [=====] - 0s 186ms/step
Predicción: versicolor , Etiqueta real: versicolor
Predicción: versicolor , Etiqueta real: versicolor
Predicción: versicolor , Etiqueta real: versicolor
Predicción: setosa , Etiqueta real: setosa
Predicción: versicolor , Etiqueta real: versicolor
Predicción: virginica , Etiqueta real: virginica
Predicción: setosa , Etiqueta real: setosa
Predicción: versicolor , Etiqueta real: versicolor
Predicción: setosa , Etiqueta real: setosa
Predicción: setosa , Etiqueta real: setosa
Predicción: versicolor , Etiqueta real: versicolor
Predicción: virginica , Etiqueta real: virginica
Predicción: virginica , Etiqueta real: virginica
Predicción: versicolor , Etiqueta real: versicolor
Predicción: virginica , Etiqueta real: virginica
Predicción: virginica , Etiqueta real: virginica
Predicción: versicolor , Etiqueta real: versicolor
Predicción: setosa , Etiqueta real: setosa
Predicción: virginica , Etiqueta real: virginica
Predicción: setosa , Etiqueta real: setosa
Predicción: setosa , Etiqueta real: setosa
Predicción: virginica , Etiqueta real: virginica
Predicción: virginica , Etiqueta real: virginica
Predicción: virginica , Etiqueta real: virginica
Predicción: versicolor , Etiqueta real: versicolor
Predicción: virginica , Etiqueta real: virginica
Predicción: setosa , Etiqueta real: setosa
Predicción: setosa , Etiqueta real: setosa
Predicción: virginica , Etiqueta real: virginica
Predicción: virginica , Etiqueta real: virginica
```

NOTA: Este procedimiento se realizó como primer entrenamiento (Entrenamiento 1) de la red neuronal, cuyos datos se agregarán en el cuadro de hiperparámetros.

## PRUEBA DEL CÓDIGO

Utilizando esta red neuronal, se realizaron 10 entrenamientos en total, de los cuales recolectamos los siguientes parámetros:

- Cantidad de épocas
- Número de capas ocultas
- Cantidad de neuronas por capa
- Función de activación de la capa
- Función de optimización
- Tasa de aprendizaje
- Exactitud (%)

Estos entrenamientos se realizaron de la siguiente manera:

### Entrenamiento 1:

Mismos datos que el procedimiento anterior:

```
0 s ▶ model = Sequential([
    Dense(1000, activation="relu", input_shape=(4,)),
    Dense(50, activation="relu"),
    Dense(3, activation="softmax")
])

model.summary()
```

```
0 s ▶ Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 1000)	5000
dense_7 (Dense)	(None, 50)	50050
dense_8 (Dense)	(None, 3)	153

```
=====
Total params: 55203 (215.64 KB)
Trainable params: 55203 (215.64 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
0 s ▶ model.compile(loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy'])
```

```

▶ history = model.fit(input_train, output_train,
                      epochs=10,
                      validation_split=0.1)

```

```

Epoch 1/10
4/4 [=====] - 0s 27ms/step - loss: 0.1565 - accuracy: 0.9537 - val_loss: 0.0183 - val_accuracy: 1.0000
Epoch 2/10
4/4 [=====] - 0s 11ms/step - loss: 0.1434 - accuracy: 0.9630 - val_loss: 0.0175 - val_accuracy: 1.0000
Epoch 3/10
4/4 [=====] - 0s 18ms/step - loss: 0.1302 - accuracy: 0.9630 - val_loss: 0.0122 - val_accuracy: 1.0000
Epoch 4/10
4/4 [=====] - 0s 22ms/step - loss: 0.1208 - accuracy: 0.9444 - val_loss: 0.0107 - val_accuracy: 1.0000
Epoch 5/10
4/4 [=====] - 0s 23ms/step - loss: 0.1115 - accuracy: 0.9352 - val_loss: 0.0067 - val_accuracy: 1.0000
Epoch 6/10
4/4 [=====] - 0s 21ms/step - loss: 0.1040 - accuracy: 0.9537 - val_loss: 0.0052 - val_accuracy: 1.0000
Epoch 7/10
4/4 [=====] - 0s 22ms/step - loss: 0.1006 - accuracy: 0.9630 - val_loss: 0.0057 - val_accuracy: 1.0000
Epoch 8/10
4/4 [=====] - 0s 20ms/step - loss: 0.0909 - accuracy: 0.9722 - val_loss: 0.0055 - val_accuracy: 1.0000
Epoch 9/10
4/4 [=====] - 0s 21ms/step - loss: 0.0850 - accuracy: 0.9630 - val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 10/10
4/4 [=====] - 0s 22ms/step - loss: 0.0812 - accuracy: 0.9722 - val_loss: 0.0044 - val_accuracy: 1.0000

```

Cantidad de épocas: 10

Número de capas ocultas: 1

Cantidad de neuronas por capa: 500

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.001

Exactitud: 97.22 %

### Entrenamiento 2:

Para este entrenamiento, la capa de entrada ahora tiene 500 neuronas, a diferencia del entrenamiento 1, cuya capa de entrada tenía 1000 neuronas. Este detalle (500 neuronas en la capa de entrada) se considerará hasta el entrenamiento 6.

```

▶ model = Sequential([
    Dense(500, activation="relu", input_shape=(4,)),
    Dense(100, activation="relu"),
    Dense(100, activation="relu"),
    Dense(100, activation="relu"),
    Dense(3, activation="softmax")
])

```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 500)	2500
dense_1 (Dense)	(None, 100)	50100
dense_2 (Dense)	(None, 100)	10100
dense_3 (Dense)	(None, 100)	10100
dense_4 (Dense)	(None, 3)	303

```

=====
Total params: 73103 (285.56 KB)
Trainable params: 73103 (285.56 KB)
Non-trainable params: 0 (0.00 Byte)

```

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])
```

```
history = model.fit(input_train, output_train,
                    epochs=15,
                    validation_split=0.1)
```

Epoch 1/15  
 4/4 [=====] - 3s 100ms/step - loss: 1.0479 - accuracy: 0.5648 - val\_loss: 0.9489 - val\_accuracy: 0.7500  
 Epoch 2/15  
 4/4 [=====] - 0s 14ms/step - loss: 0.8881 - accuracy: 0.6574 - val\_loss: 0.7392 - val\_accuracy: 0.7500  
 Epoch 3/15  
 4/4 [=====] - 0s 15ms/step - loss: 0.6997 - accuracy: 0.6852 - val\_loss: 0.5111 - val\_accuracy: 0.7500  
 Epoch 4/15  
 4/4 [=====] - 0s 13ms/step - loss: 0.5464 - accuracy: 0.7222 - val\_loss: 0.3916 - val\_accuracy: 0.8333  
 Epoch 5/15  
 4/4 [=====] - 0s 19ms/step - loss: 0.4637 - accuracy: 0.7778 - val\_loss: 0.3382 - val\_accuracy: 0.8333  
 Epoch 6/15  
 4/4 [=====] - 0s 14ms/step - loss: 0.4002 - accuracy: 0.8148 - val\_loss: 0.2876 - val\_accuracy: 1.0000  
 Epoch 7/15  
 4/4 [=====] - 0s 13ms/step - loss: 0.3322 - accuracy: 0.8426 - val\_loss: 0.2367 - val\_accuracy: 1.0000  
 Epoch 8/15  
 4/4 [=====] - 0s 14ms/step - loss: 0.2813 - accuracy: 0.9444 - val\_loss: 0.1886 - val\_accuracy: 1.0000  
 Epoch 9/15  
 4/4 [=====] - 0s 14ms/step - loss: 0.2323 - accuracy: 0.9352 - val\_loss: 0.1183 - val\_accuracy: 1.0000  
 Epoch 10/15  
 4/4 [=====] - 0s 12ms/step - loss: 0.1898 - accuracy: 0.9352 - val\_loss: 0.1178 - val\_accuracy: 1.0000  
 Epoch 11/15  
 4/4 [=====] - 0s 15ms/step - loss: 0.1551 - accuracy: 0.9537 - val\_loss: 0.0661 - val\_accuracy: 1.0000  
 Epoch 12/15  
 4/4 [=====] - 0s 17ms/step - loss: 0.1322 - accuracy: 0.9444 - val\_loss: 0.0504 - val\_accuracy: 1.0000  
 Epoch 13/15  
 4/4 [=====] - 0s 13ms/step - loss: 0.1199 - accuracy: 0.9638 - val\_loss: 0.0893 - val\_accuracy: 1.0000  
 Epoch 14/15  
 4/4 [=====] - 0s 13ms/step - loss: 0.0957 - accuracy: 0.9638 - val\_loss: 0.0385 - val\_accuracy: 1.0000  
 Epoch 15/15  
 4/4 [=====] - 0s 12ms/step - loss: 0.0769 - accuracy: 0.9815 - val\_loss: 0.0304 - val\_accuracy: 1.0000  
 6s se ejecutó 5:51 a.m.

Cantidad de épocas: 15

Número de capas ocultas: 3

Cantidad de neuronas por capa: 100

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.001

Exactitud: 98.15 %

Entrenamiento 3:

```
model = Sequential([
    Dense(500, activation="relu", input_shape=(4,)),
    Dense(120, activation="relu"),
    Dense(120, activation="relu"),
    Dense(120, activation="relu"),
    Dense(3, activation="softmax")
])

model.summary()
```

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 500)	2500
dense_11 (Dense)	(None, 120)	60120
dense_12 (Dense)	(None, 120)	14520
dense_13 (Dense)	(None, 120)	14520
dense_14 (Dense)	(None, 3)	363

=====  
 Total params: 92023 (359.46 KB)  
 Trainable params: 92023 (359.46 KB)  
 Non-trainable params: 0 (0.00 Byte)

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(learning_rate=0.002),
              metrics=['accuracy'])
```

```
history = model.fit(input_train, output_train,
                    epochs=15,
                    validation_split=0.1)

Epoch 1/15
4/4 [=====] - 2s 80ms/step - loss: 1.0106 - accuracy: 0.6111 - val_loss: 0.7774 - val_accuracy: 0.9167
Epoch 2/15
4/4 [=====] - 0s 14ms/step - loss: 0.6274 - accuracy: 0.8056 - val_loss: 0.3416 - val_accuracy: 0.9167
Epoch 3/15
4/4 [=====] - 0s 13ms/step - loss: 0.4099 - accuracy: 0.8241 - val_loss: 0.2183 - val_accuracy: 1.0000
Epoch 4/15
4/4 [=====] - 0s 13ms/step - loss: 0.2807 - accuracy: 0.8981 - val_loss: 0.2132 - val_accuracy: 0.9167
Epoch 5/15
4/4 [=====] - 0s 13ms/step - loss: 0.2468 - accuracy: 0.9074 - val_loss: 0.0835 - val_accuracy: 1.0000
Epoch 6/15
4/4 [=====] - 0s 12ms/step - loss: 0.1803 - accuracy: 0.9167 - val_loss: 0.0905 - val_accuracy: 1.0000
Epoch 7/15
4/4 [=====] - 0s 13ms/step - loss: 0.1628 - accuracy: 0.9444 - val_loss: 0.0248 - val_accuracy: 1.0000
Epoch 8/15
4/4 [=====] - 0s 12ms/step - loss: 0.1253 - accuracy: 0.9630 - val_loss: 0.0187 - val_accuracy: 1.0000
Epoch 9/15
4/4 [=====] - 0s 13ms/step - loss: 0.0773 - accuracy: 0.9630 - val_loss: 0.0175 - val_accuracy: 1.0000
Epoch 10/15
4/4 [=====] - 0s 13ms/step - loss: 0.0708 - accuracy: 0.9722 - val_loss: 0.0111 - val_accuracy: 1.0000
Epoch 11/15
4/4 [=====] - 0s 12ms/step - loss: 0.0572 - accuracy: 0.9815 - val_loss: 0.0063 - val_accuracy: 1.0000
Epoch 12/15
4/4 [=====] - 0s 13ms/step - loss: 0.0587 - accuracy: 0.9815 - val_loss: 0.0598 - val_accuracy: 1.0000
Epoch 13/15
4/4 [=====] - 0s 13ms/step - loss: 0.0581 - accuracy: 0.9815 - val_loss: 0.0384 - val_accuracy: 1.0000
Epoch 14/15
4/4 [=====] - 0s 16ms/step - loss: 0.0518 - accuracy: 0.9722 - val_loss: 0.0059 - val_accuracy: 1.0000
Epoch 15/15
4/4 [=====] - 0s 13ms/step - loss: 0.0395 - accuracy: 0.9907 - val_loss: 0.0285 - val_accuracy: 1.0000
```

Cantidad de épocas: 15

Número de capas ocultas: 3

Cantidad de neuronas por capa: 120

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.002

Exactitud: 99.07 %

Entrenamiento 4:

```
history = model.fit(input_train, output_train,
                    epochs=15,
                    validation_split=0.1)

Epoch 1/15
4/4 [=====] - 2s 69ms/step - loss: 1.1173 - accuracy: 0.1852 - val_loss: 1.1043 - val_accuracy: 0.2500
Epoch 2/15
4/4 [=====] - 0s 12ms/step - loss: 1.1135 - accuracy: 0.1852 - val_loss: 1.1015 - val_accuracy: 0.2500
Epoch 3/15
4/4 [=====] - 0s 12ms/step - loss: 1.1098 - accuracy: 0.2037 - val_loss: 1.0986 - val_accuracy: 0.2500
Epoch 4/15
4/4 [=====] - 0s 12ms/step - loss: 1.1059 - accuracy: 0.2222 - val_loss: 1.0957 - val_accuracy: 0.3333
Epoch 5/15
4/4 [=====] - 0s 12ms/step - loss: 1.1020 - accuracy: 0.2315 - val_loss: 1.0926 - val_accuracy: 0.4167
Epoch 6/15
4/4 [=====] - 0s 18ms/step - loss: 1.0981 - accuracy: 0.2593 - val_loss: 1.0895 - val_accuracy: 0.4167
Epoch 7/15
4/4 [=====] - 0s 13ms/step - loss: 1.0942 - accuracy: 0.3056 - val_loss: 1.0869 - val_accuracy: 0.4167
Epoch 8/15
4/4 [=====] - 0s 18ms/step - loss: 1.0902 - accuracy: 0.3519 - val_loss: 1.0840 - val_accuracy: 0.4167
Epoch 9/15
4/4 [=====] - 0s 13ms/step - loss: 1.0864 - accuracy: 0.4259 - val_loss: 1.0810 - val_accuracy: 0.5000
Epoch 10/15
4/4 [=====] - 0s 13ms/step - loss: 1.0826 - accuracy: 0.5185 - val_loss: 1.0784 - val_accuracy: 0.6667
Epoch 11/15
4/4 [=====] - 0s 13ms/step - loss: 1.0787 - accuracy: 0.6944 - val_loss: 1.0751 - val_accuracy: 0.9167
Epoch 12/15
4/4 [=====] - 0s 12ms/step - loss: 1.0748 - accuracy: 0.7963 - val_loss: 1.0722 - val_accuracy: 0.9167
Epoch 13/15
4/4 [=====] - 0s 12ms/step - loss: 1.0710 - accuracy: 0.8148 - val_loss: 1.0690 - val_accuracy: 0.9167
Epoch 14/15
4/4 [=====] - 0s 13ms/step - loss: 1.0670 - accuracy: 0.8148 - val_loss: 1.0657 - val_accuracy: 0.9167
Epoch 15/15
4/4 [=====] - 0s 12ms/step - loss: 1.0632 - accuracy: 0.8056 - val_loss: 1.0623 - val_accuracy: 0.9167
```

Cantidad de épocas: 15

Número de capas ocultas: 3

Cantidad de neuronas por capa: 200

Función de activación de la capa: ReLU

Función de optimización: Adadelta

Tasa de aprendizaje: 0.01

Exactitud: 80.56 %

### Entrenamiento 5:

```
history = model.fit(input_train, output_train,
                    epochs=15,
                    validation_split=0.1)

Epoch 1/15
4/4 [=====] - 1s 77ms/step - loss: 1.0999 - accuracy: 0.2222 - val_loss: 1.0952 - val_accuracy: 0.1667
Epoch 2/15
4/4 [=====] - 0s 12ms/step - loss: 1.0961 - accuracy: 0.3056 - val_loss: 1.0911 - val_accuracy: 0.1667
Epoch 3/15
4/4 [=====] - 0s 12ms/step - loss: 1.0923 - accuracy: 0.4259 - val_loss: 1.0869 - val_accuracy: 0.5833
Epoch 4/15
4/4 [=====] - 0s 12ms/step - loss: 1.0885 - accuracy: 0.7407 - val_loss: 1.0830 - val_accuracy: 0.7500
Epoch 5/15
4/4 [=====] - 0s 12ms/step - loss: 1.0847 - accuracy: 0.8426 - val_loss: 1.0796 - val_accuracy: 0.8333
Epoch 6/15
4/4 [=====] - 0s 13ms/step - loss: 1.0812 - accuracy: 0.8148 - val_loss: 1.0758 - val_accuracy: 0.9167
Epoch 7/15
4/4 [=====] - 0s 12ms/step - loss: 1.0776 - accuracy: 0.7963 - val_loss: 1.0726 - val_accuracy: 0.9167
Epoch 8/15
4/4 [=====] - 0s 14ms/step - loss: 1.0742 - accuracy: 0.8333 - val_loss: 1.0686 - val_accuracy: 0.8333
Epoch 9/15
4/4 [=====] - 0s 13ms/step - loss: 1.0705 - accuracy: 0.8426 - val_loss: 1.0651 - val_accuracy: 0.9167
Epoch 10/15
4/4 [=====] - 0s 12ms/step - loss: 1.0670 - accuracy: 0.8333 - val_loss: 1.0614 - val_accuracy: 0.8333
Epoch 11/15
4/4 [=====] - 0s 13ms/step - loss: 1.0633 - accuracy: 0.8426 - val_loss: 1.0579 - val_accuracy: 0.8333
Epoch 12/15
4/4 [=====] - 0s 13ms/step - loss: 1.0593 - accuracy: 0.8426 - val_loss: 1.0542 - val_accuracy: 0.8333
Epoch 13/15
4/4 [=====] - 0s 13ms/step - loss: 1.0553 - accuracy: 0.8426 - val_loss: 1.0504 - val_accuracy: 0.8333
Epoch 14/15
4/4 [=====] - 0s 12ms/step - loss: 1.0513 - accuracy: 0.8426 - val_loss: 1.0453 - val_accuracy: 0.8333
Epoch 15/15
4/4 [=====] - 0s 13ms/step - loss: 1.0467 - accuracy: 0.8426 - val_loss: 1.0406 - val_accuracy: 0.8333
```

Cantidad de épocas: 15

Número de capas ocultas: 5

Cantidad de neuronas por capa: 180

Función de activación de la capa: ReLU

Función de optimización: Adadelta

Tasa de aprendizaje: 0.02

Exactitud: 84.26 %

### Entrenamiento 6:



```

✓ 34 history = model.fit(input_train, output_train,
                        epochs=12,
                        validation_split=0.1)

Epoch 1/12
4/4 [=====] - 3s 83ms/step - loss: 1.0104 - accuracy: 0.5185 - val_loss: 0.3370 - val_accuracy: 0.8333
Epoch 2/12
4/4 [=====] - 0s 12ms/step - loss: 0.4353 - accuracy: 0.8241 - val_loss: 0.4837 - val_accuracy: 0.8333
Epoch 3/12
4/4 [=====] - 0s 12ms/step - loss: 0.2467 - accuracy: 0.8981 - val_loss: 0.0970 - val_accuracy: 1.0000
Epoch 4/12
4/4 [=====] - 0s 11ms/step - loss: 0.1417 - accuracy: 0.9444 - val_loss: 0.4558 - val_accuracy: 0.8333
Epoch 5/12
4/4 [=====] - 0s 12ms/step - loss: 0.3125 - accuracy: 0.9259 - val_loss: 0.1859 - val_accuracy: 0.9167
Epoch 6/12
4/4 [=====] - 0s 12ms/step - loss: 0.1368 - accuracy: 0.9537 - val_loss: 0.0133 - val_accuracy: 1.0000
Epoch 7/12
4/4 [=====] - 0s 12ms/step - loss: 0.2848 - accuracy: 0.9630 - val_loss: 0.0101 - val_accuracy: 1.0000
Epoch 8/12
4/4 [=====] - 0s 13ms/step - loss: 0.0582 - accuracy: 0.9722 - val_loss: 0.0486 - val_accuracy: 1.0000
Epoch 9/12
4/4 [=====] - 0s 12ms/step - loss: 0.1670 - accuracy: 0.9352 - val_loss: 0.1540 - val_accuracy: 0.9167
Epoch 10/12
4/4 [=====] - 0s 13ms/step - loss: 0.1943 - accuracy: 0.9444 - val_loss: 0.0191 - val_accuracy: 1.0000
Epoch 11/12
4/4 [=====] - 0s 12ms/step - loss: 0.1153 - accuracy: 0.9444 - val_loss: 0.0206 - val_accuracy: 1.0000
Epoch 12/12
4/4 [=====] - 0s 12ms/step - loss: 0.0575 - accuracy: 0.9815 - val_loss: 0.0129 - val_accuracy: 1.0000

```

Cantidad de épocas: 12

Número de capas ocultas: 2

Cantidad de neuronas por capa: 150

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.02

Exactitud: 98.15 %

### Entrenamiento 7:

A partir de este entrenamiento, la capa de entrada ahora tendrá 800 neuronas, ya no 500 neuronas. Este detalle se considerará hasta el último entrenamiento.

```

✓ 35 history = model.fit(input_train, output_train,
                        epochs=5,
                        validation_split=0.1)

Epoch 1/5
4/4 [=====] - 2s 68ms/step - loss: 0.1826 - accuracy: 0.8981 - val_loss: 0.0405 - val_accuracy: 1.0000
Epoch 2/5
4/4 [=====] - 0s 13ms/step - loss: 0.0929 - accuracy: 0.9444 - val_loss: 0.0342 - val_accuracy: 1.0000
Epoch 3/5
4/4 [=====] - 0s 13ms/step - loss: 0.0883 - accuracy: 0.9537 - val_loss: 0.0031 - val_accuracy: 1.0000
Epoch 4/5
4/4 [=====] - 0s 12ms/step - loss: 0.0506 - accuracy: 0.9815 - val_loss: 0.0141 - val_accuracy: 1.0000
Epoch 5/5
4/4 [=====] - 0s 13ms/step - loss: 0.1921 - accuracy: 0.9815 - val_loss: 5.6298e-04 - val_accuracy: 1.0000

```

Cantidad de épocas: 5

Número de capas ocultas: 4

Cantidad de neuronas por capa: 160

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.01

Exactitud: 98.15 %



## Entrenamiento 8:

```
history = model.fit(input_train, output_train,
                    epochs=5,
                    validation_split=0.1)
```

```
Epoch 1/5
4/4 [=====] - 0s 30ms/step - loss: 0.1163 - accuracy: 0.9444 - val_loss: 0.0346 - val_accuracy: 1.0000
Epoch 2/5
4/4 [=====] - 0s 13ms/step - loss: 0.1062 - accuracy: 0.9722 - val_loss: 0.0625 - val_accuracy: 1.0000
Epoch 3/5
4/4 [=====] - 0s 12ms/step - loss: 0.0914 - accuracy: 0.9630 - val_loss: 0.0416 - val_accuracy: 1.0000
Epoch 4/5
4/4 [=====] - 0s 12ms/step - loss: 0.0689 - accuracy: 0.9630 - val_loss: 0.0102 - val_accuracy: 1.0000
Epoch 5/5
4/4 [=====] - 0s 12ms/step - loss: 0.0877 - accuracy: 0.9815 - val_loss: 0.0065 - val_accuracy: 1.0000
```

Cantidad de épocas: 5

Número de capas ocultas: 4

Cantidad de neuronas por capa: 120

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.002

Exactitud: 98.15 %

## Entrenamiento 9:

```
history = model.fit(input_train, output_train,
                    epochs=10,
                    validation_split=0.1)
```

```
Epoch 1/10
4/4 [=====] - 2s 68ms/step - loss: 1.0214 - accuracy: 0.6852 - val_loss: 0.8625 - val_accuracy: 0.8333
Epoch 2/10
4/4 [=====] - 0s 12ms/step - loss: 0.8038 - accuracy: 0.8426 - val_loss: 0.6178 - val_accuracy: 0.8333
Epoch 3/10
4/4 [=====] - 0s 12ms/step - loss: 0.5646 - accuracy: 0.8704 - val_loss: 0.4738 - val_accuracy: 0.8333
Epoch 4/10
4/4 [=====] - 0s 12ms/step - loss: 0.4089 - accuracy: 0.8611 - val_loss: 0.4115 - val_accuracy: 0.8333
Epoch 5/10
4/4 [=====] - 0s 13ms/step - loss: 0.3146 - accuracy: 0.8611 - val_loss: 0.3362 - val_accuracy: 0.7500
Epoch 6/10
4/4 [=====] - 0s 12ms/step - loss: 0.2617 - accuracy: 0.8889 - val_loss: 0.3101 - val_accuracy: 0.7500
Epoch 7/10
4/4 [=====] - 0s 12ms/step - loss: 0.2220 - accuracy: 0.8981 - val_loss: 0.2388 - val_accuracy: 0.8333
Epoch 8/10
4/4 [=====] - 0s 13ms/step - loss: 0.1844 - accuracy: 0.9259 - val_loss: 0.1632 - val_accuracy: 0.9167
Epoch 9/10
4/4 [=====] - 0s 12ms/step - loss: 0.1522 - accuracy: 0.9352 - val_loss: 0.0889 - val_accuracy: 1.0000
Epoch 10/10
4/4 [=====] - 0s 12ms/step - loss: 0.1293 - accuracy: 0.9722 - val_loss: 0.0799 - val_accuracy: 1.0000
```

Cantidad de épocas: 10

Número de capas ocultas: 3

Cantidad de neuronas por capa: 100

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.001

Exactitud: 97.22 %

## Entrenamiento 10:

```
✓ 25 history = model.fit(input_train, output_train,
                        epochs=10,
                        validation_split=0.1)

Epoch 1/10
4/4 [=====] - 1s 70ms/step - loss: 0.2227 - accuracy: 0.9074 - val_loss: 0.3906 - val_accuracy: 0.8333
Epoch 2/10
4/4 [=====] - 0s 12ms/step - loss: 0.3975 - accuracy: 0.8611 - val_loss: 0.2031 - val_accuracy: 0.8333
Epoch 3/10
4/4 [=====] - 0s 12ms/step - loss: 0.2584 - accuracy: 0.9074 - val_loss: 0.0597 - val_accuracy: 1.0000
Epoch 4/10
4/4 [=====] - 0s 12ms/step - loss: 0.2444 - accuracy: 0.8889 - val_loss: 0.0445 - val_accuracy: 1.0000
Epoch 5/10
4/4 [=====] - 0s 12ms/step - loss: 0.1265 - accuracy: 0.9537 - val_loss: 0.1058 - val_accuracy: 1.0000
Epoch 6/10
4/4 [=====] - 0s 13ms/step - loss: 0.1665 - accuracy: 0.8981 - val_loss: 0.0972 - val_accuracy: 1.0000
Epoch 7/10
4/4 [=====] - 0s 12ms/step - loss: 0.1373 - accuracy: 0.9444 - val_loss: 0.0305 - val_accuracy: 1.0000
Epoch 8/10
4/4 [=====] - 0s 12ms/step - loss: 0.0822 - accuracy: 0.9815 - val_loss: 0.0274 - val_accuracy: 1.0000
Epoch 9/10
4/4 [=====] - 0s 12ms/step - loss: 0.0934 - accuracy: 0.9722 - val_loss: 0.0177 - val_accuracy: 1.0000
Epoch 10/10
4/4 [=====] - 0s 12ms/step - loss: 0.0691 - accuracy: 0.9815 - val_loss: 0.0217 - val_accuracy: 1.0000
```

Cantidad de épocas: 10

Número de capas ocultas: 3

Cantidad de neuronas por capa: 120

Función de activación de la capa: ReLU

Función de optimización: Adam

Tasa de aprendizaje: 0.002

Exactitud: 98.15 %

A partir de ahora, ubicamos los parámetros mencionados anteriormente, de cada entrenamiento, dentro del cuadro de hiperparámetros.

## CUADRO DE HIPERPARÁMETROS

Antes de completar el cuadro de hiperparámetros, cabe resaltar que se tuvo en cuenta que la columna “Capas” se refiere a “Cantidad de capas ocultas”. Además, dentro de un mismo entrenamiento, se decidió considerar solo un tipo de “Función de activación” para cada grupo de capas.

Entonces, procedimos a completar dicho cuadro de la siguiente forma:

E	Épocas	Capas	Neuronas por capa	Función de activación	Función de optimización	Tasa de aprendizaje	Exactitud (%)
1	10	1	50	ReLU	Adam	0.001	97.22
2	15	3	100	ReLU	Adam	0.001	98.15
3	15	3	120	ReLU	Adam	0.002	99.07
4	15	3	200	ReLU	Adadelata	0.01	80.56
5	15	5	180	ReLU	Adadelata	0.02	84.26
6	12	2	150	ReLU	Adam	0.02	98.15
7	5	4	160	ReLU	Adam	0.01	98.15
8	5	4	120	ReLU	Adam	0.002	98.15
9	10	3	100	ReLU	Adam	0.001	97.22
10	10	3	120	ReLU	Adam	0.002	98.15

Donde E es Entrenamiento.

## OBSERVACIONES

Analizando el cuadro de hiperparámetros, podemos darnos cuenta de que la eficiencia de la red neuronal creció desde el primer entrenamiento hasta el entrenamiento 10, pero tuvo algunos altibajos durante cada entrenamiento realizado. Además la variación entre eficiencias contiguas no fue tan considerable en la mayoría de casos, incluso se presentó una constancia desde el entrenamiento 6 hasta el entrenamiento 8, a pesar de las variaciones que se realizaron a los parámetros del cuadro.

Debemos resaltar que, utilizando la función de optimización “Adam”, la red neuronal presentó mayor eficiencia, a diferencia de la función “Adadelata”, que es verdad que hizo que la red tuviera una eficiencia alta, pero no tanto como la de “Adam”.

En varios entrenamientos, la red tuvo una eficiencia de 98.15%, coincidentemente es la eficiencia del último entrenamiento.

La cantidad de épocas no fue tan influyente para que la eficiencia presentara variaciones representativas, eso lo podemos notar en los pares de entrenamientos 6-7 y 8-9, algo diferente con la cantidad de neuronas y sobretodo la tasa de aprendizaje, que influía en varias ocasiones.

El punto más alto de eficiencia fue de 99.07 % en el entrenamiento 3, esto constata al enunciado anterior con respecto a la tasa de aprendizaje, este parámetro en el entrenamiento 3 aumentó respecto del anterior más cercano.

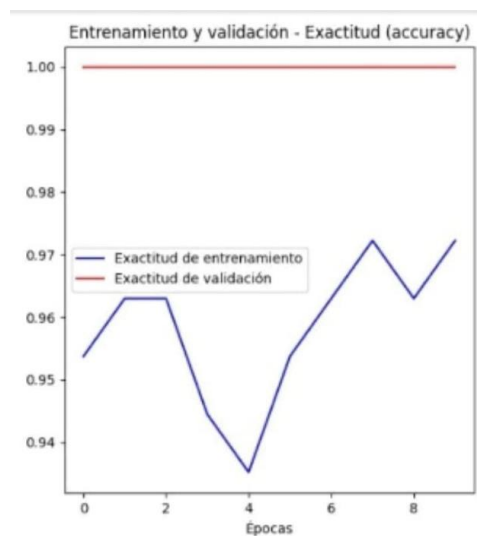
Se decidió usar la misma función de activación “ReLU” para todos los entrenamientos, esto para tener un punto de análisis más específico.

## VERIFICACIÓN DE OVERFITTING

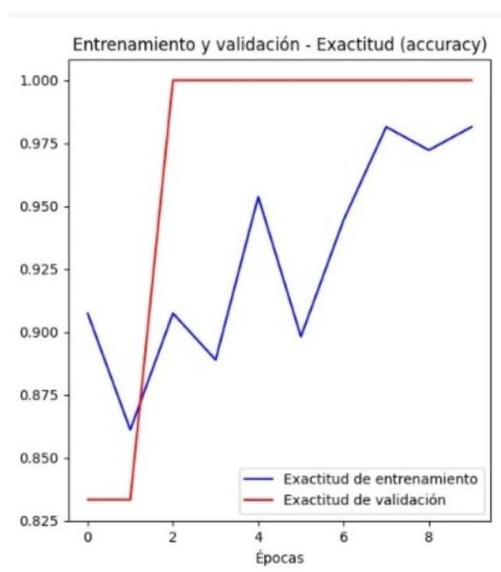
Durante esta serie de entrenamientos, no hubo presencia de Overfitting en promedio, ya que la diferencia entre la exactitud de entrenamiento y la exactitud de validación fue baja numéricamente. Esto lo podemos observar en las gráficas obtenidas de cada entrenamiento.

NOTA: Para el Overfitting, debemos notar una variación numérica considerable. En las gráficas se ve cierta lejanía entre ambas gráficas, pero es debido al enfoque que se le hace al espacio del diagrama únicamente.

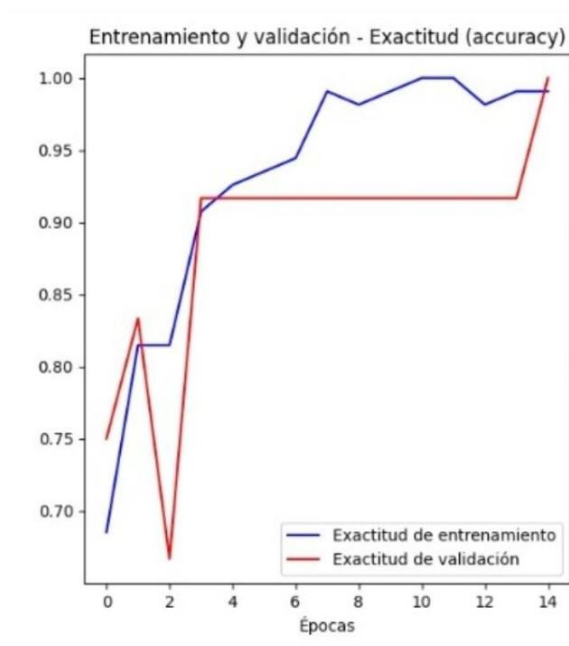
### Entrenamiento 1:



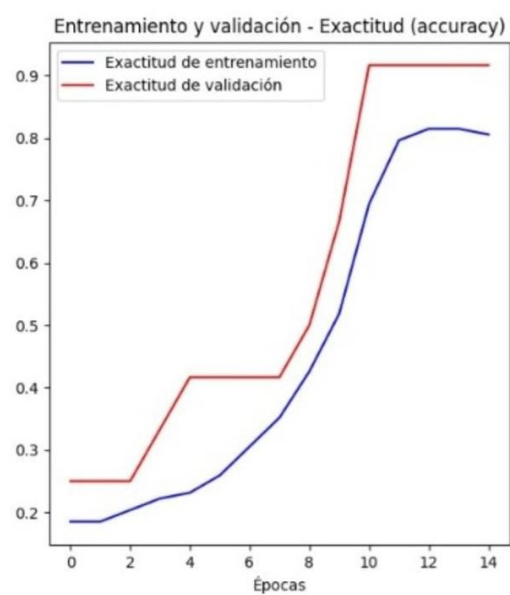
### Entrenamiento 2:



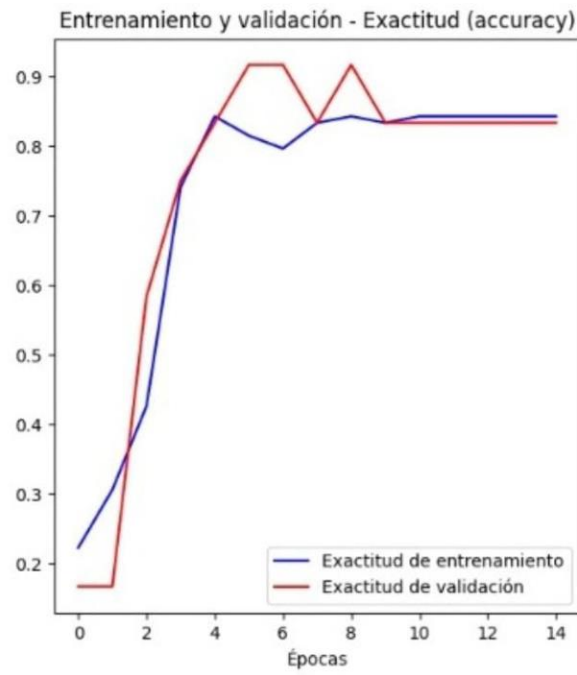
### Entrenamiento 3:



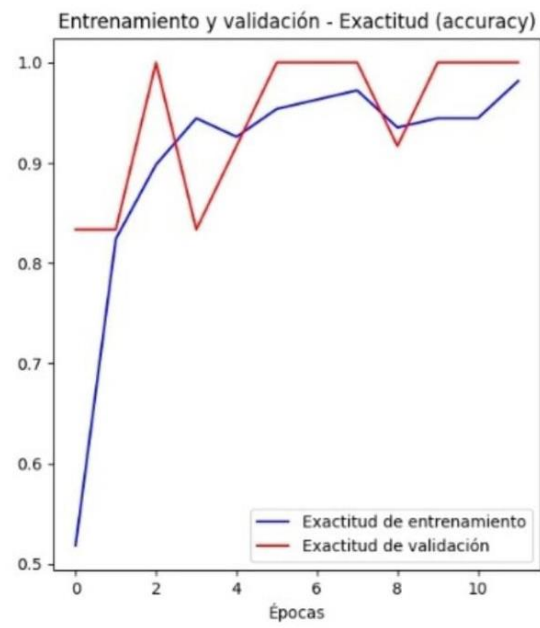
### Entrenamiento 4:



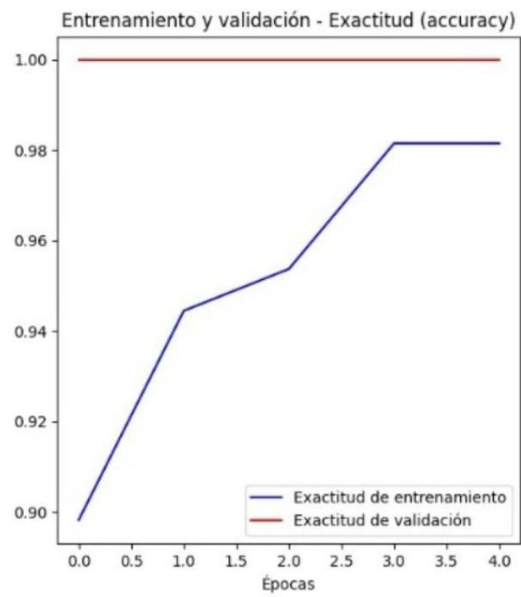
### Entrenamiento 5:



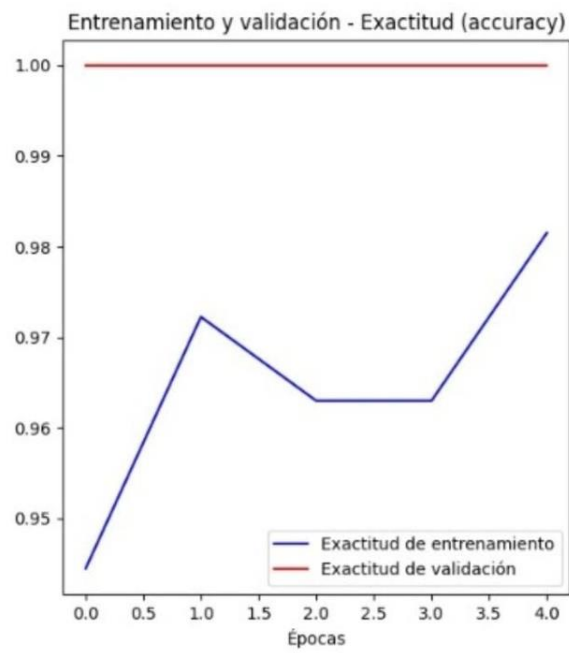
Entrenamiento 6:



Entrenamiento 7:

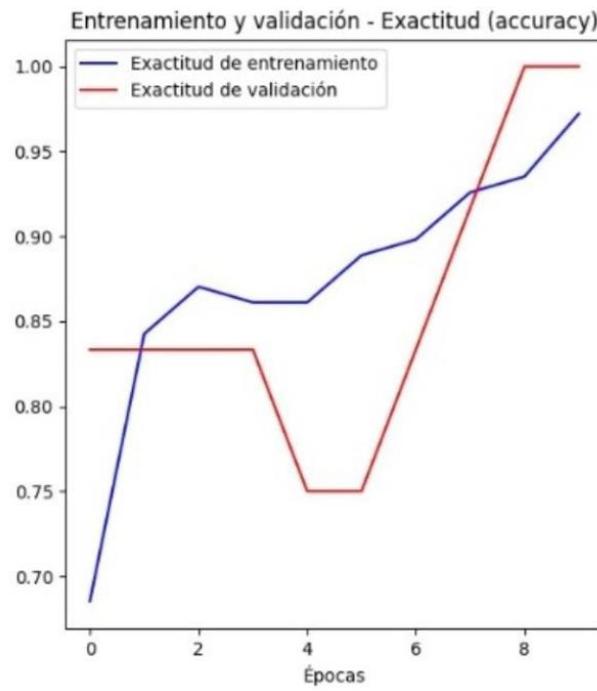


Entrenamiento 8:

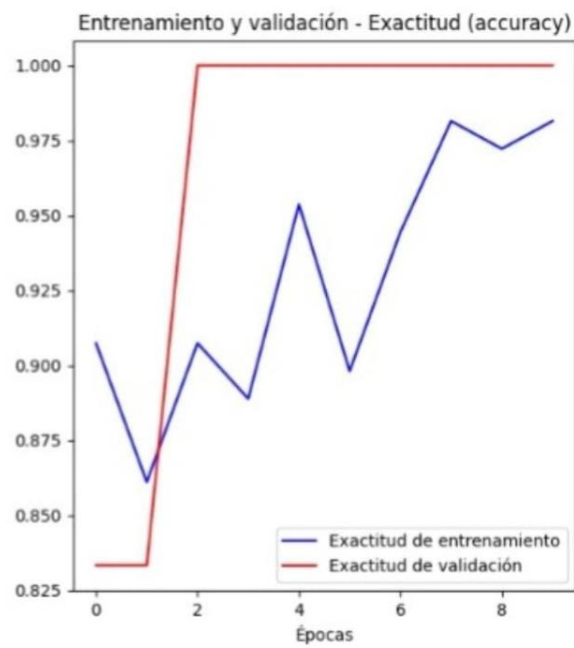


Entrenamiento 9:





Entrenamiento 10:



## CONCLUSIONES

- Se realizaron 10 entrenamientos, los cuales fueron fundamentales para determinar la eficiencia de esta red neuronal, que sirva para predecir la especie de flor iris y solucionar el conjunto de datos iris.
- El cuadro de hiperparámetros fue completado gracias a la ubicación de resultados del programa y a los datos ingresados, por el grupo, dentro del código.
- Lo que se puede decir sobre los datos obtenidos, es que la eficiencia que más se repetía en el entrenamiento era 98.15 % y que fue necesario el uso de la función de optimización “Adam” para la obtención de una mayor eficiencia.
- Con respecto al Overfitting, no hubo presencia de este caso en esta serie de entrenamientos.

## **BIBLIOGRAFÍA**

Oppenheim, A. V., Willsky, A. S., & Young, I. T. (1983). *Signals and systems*. Englewood Cliffs, N.J: Prentice-Hall.

Kamen, Edward W., y Bonnie S. Heck. (2008). Fundamentos de señales y sistemas usando la Web y MATLAB® PEARSON EDUCACIÓN, México, ISBN: 978-970-26-1187-5

## **ANEXOS**

Link del colab en donde se encuentra el código usado:

[https://drive.google.com/drive/folders/17tOR\\_CC7mnAwuHDDCx-qtPxxgtKgy0EP](https://drive.google.com/drive/folders/17tOR_CC7mnAwuHDDCx-qtPxxgtKgy0EP)